Package ‘addScales’

October 12, 2022

Type Package

Title Adds Labeled Center Line and Scale Lines/Regions to Trellis Plots

Version 1.0-1

Author Bert Gunter

Maintainer Bert Gunter<`bgunter.4567@gmail.com`>

Description Modifies trellis objects by adding horizontal and/or vertical reference lines or shaded regions that provide visual scaling information. This is mostly useful in multi-panel plots that use the relation = 'free' option in their 'scales' argument list.

Depends lattice (>= 0.20-38), R(>= 3.5.0)

Imports grid, stats, grDevices

Suggests knitr, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-06-11 04:30:02 UTC

R topics documented:

addScales-package *Adds Labeled Center Line and Scale Lines/Regions to Trellis Plots*

Description

Modifies trellis objects by adding horizontal and/or vertical reference lines or shaded regions that provide visual scaling information. This is mostly useful in multi-panel plots that use the relation = 'free' option in their 'scales' argument list.

Details

The DESCRIPTION file:

Package:	addScales
Type:	Package
Title:	Adds Labeled Center Line and Scale Lines/Regions to Trellis Plots
Version:	1.0-1
Author:	Bert Gunter
Maintainer:	Bert Gunter<bgunter.4567@gmail.com>
Description:	Modifies trellis objects by adding horizontal and/or vertical reference lines or shaded regions that provide
Depends:	lattice (>= 0.20-38), R(>= 3.5.0)
Imports:	grid, stats, grDevices
Suggests:	knitr, rmarkdown
License:	MIT + file LICENSE
Encoding:	UTF-8
LazyData:	true
VignetteBuilder:	knitr

Index of help topics:

```
CHITemps                Daily Chicago High and Low Temperatures in °F
NYCTemps                Daily New York City High and Low Temperatures
                        in °F
SFTemps                 Daily San Francisco High and Low Temperatures
                        in °F
USAcrime                USA Property and Violent Crime Data, 1960 -
                        2014
addScales               Add Scaling Information to Panels in
                        Multi-Panel Trellis Plots
addScales-package       Adds Labeled Center Line and Scale
                        Lines/Regions to Trellis Plots
```

`panel.addScales`	`Default panelFUN For addScales.trellis`
`prepanel.trim`	`Lattice Prepanel Function to Trim Panel Limits`
`revert`	`Revert A Scaled Trellis Plot To Its Previous Unscaled Form`
`scaleline`	`Extract scaleline list from 'scaledTrellis' object`
`update.scaledTrellis`	`Update Method for scaledTrellis Objects`

Further information is available in the following vignettes:

`addScales`	The addScales Package (source)

Trellised displays are powerful tools for exploring and comparing data. However, one challenge is how to handle plots in which the data in different panels have different locations and, more importantly, different scales/variability. Different locations can be dealt with via the `relation = "sliced"` option in lattice functions' `scales` argument. But different scaling sometimes causes plot details in panels with relatively little variability to be obscured because the data are “squashed” to accommodate the limits needed to show more variable data in others.

The `addScales` function unsquashes such data by varying the panel limits so that each panel fits just its data. This can easily be done by using the `relation = "free"` option in the `scales` argument. Unfortunately, with more then just a few panels, the separate axis scales for the different panels takes up too much display space and are difficult to read.

`addScales` addresses this problem by layering minimal location/scale information directly onto the panels. The intent is to save axis space without obscuring the panel data, but still allow the viewer to decode the information to compare the panels.

A simple API facilitates this task. In addition to the main `addScales` functions, a generic with an `addScales.trellis` method, there is an `update` method to allow the user to easily experiment with and customize the display. A few other functions provide some additional flexibility. Users desiring other capabilities should contact the author/maintainer with their requests.

Author(s)

Bert Gunter

Maintainer: Bert Gunter<bgunter.4567@gmail.com>

`addScales` *Add Scaling Information to Panels in Multi-Panel Trellis Plots*

Description

Adds a midline and upper and lower horizontal and/or vertical scale lines or shaded regions to all panels. Mostly useful when the `relation = "free"` option is used in the `scales` list to avoid loss of detail in plots from data that vary in location and scale from panel to panel.

Usage

```
    addScales(obj, ...)

## S3 method for class 'trellis'
addScales(obj,
         scaleline = list(h = TRUE, v = FALSE),
         legend = list(h = TRUE, v = TRUE),
         ndig.legend = c(h = 2, v = 2),
         legend.aes = list(),
         legend.loc = c("top","bottom","right","left"),
         panelFUN = panel.addScales,
         ...)
```

Arguments

`obj`	Object on which method dispatch is carried out. Currently only a `trellis` method exists.
`scaleline`	A two component list with component names "h" and "v". The "h" component is for (h)orizontal scale lines/regions and "v" is for(v)ertical. Each of these must be a single logical or numeric value (and can be mixed in the list, of course). `TRUE` means: use a calculated default; `FALSE` means: don't plot scale lines for this component; and a numeric value means: use this value. The calculated or user-supplied value is the distance between the midline and scale lines = 1/2 the height/width of the shaded region. A numeric value of 0 is interpreted as `FALSE` – don't draw. However, see the **Details** section below for abbreviated versions that are also accepted. **Note:** Scale lines/regions are only meaningful for `"numeric"` data as defined by `isTRUE(is.numeric())`. The corresponding `scaleline` component – "h" for y-axis data and "v" for x-axis data – is silently set to `FALSE` for anything else.
`legend`	A two component list as in `scaleline` for logical values; or non-logical values that must either be quoted UTF-8 character strings (which can include special characters such as ±, which is unicode U + 00B1, or °, unicode U + 00B0, math symbols, non-ascii language characters, etc.); or R `language` objects. See the **Legend Details** section for further details.
`ndig.legend`	Named or unnamed pair of integer arguments, or a single integer that will be replicated. The names must be (and are assumed to be if unnamed) "h" and "v" *in that order* and give the number of *significant* digits to show in the default legend for the corresponding scale lines/region. Non-integer values are rounded to integer, and values outside the range of 0 to 15 digits are converted to 2.
`legend.aes`	List of *aesthetics* of the legend text: `cex`, `font`, `fontface`, `col`, `etc.`. See `panel.text` for details.
`legend.loc`	One of `"top"`, `"bottom"`, `"left"`, or `"right"` specifying where the legend will be placed *outside* the the trellised panels on the page. See the `legend` section of the `xyplot` man page for details, but note that `addScales` will turn the legend argument into a list of the required form, so that part of the specification on the man page can be ignored.

`panelFUN`	The function used to add scaling details to the panels. Should use standard trellis/grid functionality.
`...`	Further arguments, controlling *aesthetics* of the lines, labels, and/or fill regions such as color, line width, color palette, line type, etc., passed down to the `panelFUN` function. See `panel.addScales` for details for the default panel function.

Details

As a convenience, abbreviated versions of `scaleline` and `legend` logical arguments can be used instead of the full versions described above.The abbreviated versions will be translated into the full versions for use by other functions such as `scaleline` and `update.scaledTrellis`.

Specifically, a single value of `TRUE` or `FALSE` is replicated to both components of the argument. Thus `scaleline = FALSE` aborts the function, since it says not to add scales in either direction. `legend = FALSE` is fine, because it specifies only that legends be omitted. See the **Legend Specification Details** section below for why this might be useful.

If an unnamed list with two components (of the correct form) are given, they are assumed to be in the order `c("h","v")`. If a single named component with name "h" or "v" is given, the missing component is assumed to be `FALSE`. Thus, `list(v = TRUE)`, `list(FALSE, TRUE)`, and `list(h = FALSE, v = TRUE)` are all equivalent. A list with a single unnamed component raises an error.

The default `scaleline` calculation assures that all lines/regions fall within the axis limits of all panels. A (typically user-supplied) `scaleline` that fails this criterion will raise a warning and result in some panels with missing scale lines when `scaleType = "line"`.

Value

An object of class `c("scaledTrellis","trellis")` if successful. Because it inherits from class `"trellis"`, it can be saved and/or (automatically) plotted as usual.

`NULL` invisibly if an error occurs.

The `scaledTrellis` object is the original `trellis` object list with its `panel` and `legend` components modified to add the scaling information. A new `addScales` component is also added that is itself a list with (at least) two components named "orig" and "args". The first of these contains the original `panel` and `legend` components of `obj`. The second contains either the names and values of the arguments in the call or the computed values of those arguments. The most important of these is the `scaleline` value, which can be extracted using the `scaleline` function by users who wish to construct their own scale line legends.The remaining values are used by the update method for `scaledTrellis` objects.

Legend Specification Details

The default legend is meant to be simple but serviceable. If there are scale lines in both directions, it will space them horizontally for `"top"` and `"bottom"` locations and vertically for `"right"` and `"left"` to minimize the space they occupy.

A user-supplied legend component can be given in two forms: either as a (quoted) UTF-8 character string, like this: "Scale lines are at ±10° "; or as a so-called `language` object. The latter allows the legend to use the (shortened to the `ndig.legend` number of digits) `scaleline` value. The former does not.

A detailed discussion of `language` objects is beyond the scope of this Help page, but a simple example provides a template that should usually suffice. Suppose, instead of the default, the desired legend is:

`Scale lines are at` ± xxx°,

where the `scaleline` value computed by `addScales` is to be substituted for the xxx. If xxx were available in the environment of the call (the `addScales` invocation), then one could use something like (as in the previous paragraph):

`paste0("Scale lines are at ± ",xxx," ° ")`

as the `legend` argument. But xxx is not known, because `addScales` hasn't calculated it yet. So instead, wrap the paste0 call by the `quote` function like this:

`quote(paste0("Scale lines are at ± ",sl," ° "))`

'sl' (unquoted) *must* be used to replace the not-yet-known `scaleline` value. The `quote` function will pass the whole unevaluated `paste0` expression into `addScales` where the `scaleline` value will be calculated and substituted for `sl` and the whole expression then evaluated. Of course, any R expression instead of `paste0...` can be used as long as `sl` is substituted wherever the actual `scaleline` value is wanted.

Another, perhaps slightly clumsier, way to do this – but which generalizes to arbitrary `scaleline` displays as text or graphical objects (so-called *grobs*) of any kind – is simply to run `addScales` with `legend = FALSE` and extract the `scaleline` value(s) from the resulting object with the `scaleline()` function. The value(s) can then be used in any construction the user wishes to create.

Author(s)

Bert Gunter <`bgunter.4567@gmail.com`>

See Also

`xyplot`, `panel.refline`, `panel.text`, `scaleline`, `panel.addScales`

Examples

```
###### Artificial example to show why addScales() might be useful and
###### how it works
##

###### Create a data set whose panels have different
###### centers and scales for y
x <- rep(0:10, 4)

scaling <- rep(c(1, 2, 5, 10), e = 11)

y <- sin(pi*x/10)*scaling + scaling ## add some structure

f <- factor(rep(LETTERS[1:4], e = 11))

## Now add noise proportional to data mean (= constant CV)
set.seed(91834)
```

```
y <- y + rnorm(44,sd = scaling/3)

## Plot this with the default "same" scaling and a loess curve

xplot(y ~ x| f, layout = c(4,1), col = "darkblue",
        scales = list(alternating = 1, tck = c(1,0)),
        panel = function(...){
           panel.points(...)
           panel.loess(..., span = .6, col.line = "blue")
        })

##
## Because of the different scaling, it's somewhat difficult to
## see the common panel data behavior. With relation = "free", it
## becomes clearer:
##
trellis.par.set(plot.symbol = list(col = "darkblue"),
               plot.line = list(col = "darkblue"))

xyplot(y ~ x| f, layout = c(4,1),
       scales = list(alternating = 1, tck = c(1,0),
                     y = list(relation = "free")),
       panel = function(...){
          panel.points(...)
          panel.loess(..., span = .6, col.line = "blue")
       })

##
##  Unfortunately, the y-scales take up a lot of space and are
##  difficult to read. With more panels, they would completely
##  mess things up. To avoid this, don't draw them and use addScales
##  to layer visual scaling onto the panels.
##
freeplot <- xyplot(y ~ x| f, layout = c(4,1),
          scales = list(alternating = 1, tck = c(1,0),
                        y = list(relation = "free", draw = FALSE)),
          panel = function(...){
             panel.points(...)
             panel.loess(..., span = .6, col.line = "blue")
          })

addScales(freeplot) ## using defaults

##
## The labeled midline allows location comparison among the panels.
## The fixed distance from the dashed scale lines to the midline are given
## by the legend at top. This allows scaling among the panels to be
## compared, because the more y varies within a panel, the closer together
## these fixed scale lines become.
##
## NOTE:
## The addScales object inherits from class "trellis", so can be
## saved and plotted in the same way as 'freeplot' was. That is, the
## following also works:
##
 enhanced <- addScales(freeplot)
 enhanced

## Further panel options, which we use the update() method to change,
## allow for color coded scale regions and midlines:
##
#### Warning: Nothing may display if your graphics device does not support
## alpha transparency

 update(enhanced, scaleType = "region", colCode = "r")

##
## cleanup
rm(scaling, x, y, f, freeplot, enhanced)
##
########  Some real examples   #############
############################################

## Historical daily temperatures for Chicago, New York, and San Francisco.
data(CHITemps, NYCTemps, SFTemps)

preprocess.temps <- function(d){
      meanTemp <- with(d, (TMAX + TMIN)/2)
      Month <- months(as.Date(d$DATE))
      z<- aggregate(meanTemp, list(
         Month = factor(Month, levels = unique(Month)),
         Year = as.numeric(substring(d$DATE,1,4))
      ), FUN = mean)
      names(z)[3] <- "meanTemp"
      z
}

## Create a list containing the preprocessed data for all 3 cities
plotdat <- lapply(
   list(CHI = CHITemps, NYC = NYCTemps, SF = SFTemps),
   preprocess.temps)

## Consider NYC. Because of monthly temperature variation, monthly temperature
## histories are mostly whitespace with the default relation = "same".
## Note also the use of the prepanel.trim function with defaults to remove
## extreme y values.
##
## Consider New York City
nyctemps <-
   xyplot(meanTemp ~ Year|Month, type = "l", layout = c(3,4),
          data = plotdat[[2]],
          as.table = TRUE,
          between = list(x=1, y=0),
          ## reduce strip size
          par.strip.text = list(lines = .8, cex = .7),
          ## remove blank space for top axis
          par.settings = list(layout.heights = list(axis.top = 0)),
          prepanel = prepanel.trim, ## to remove possible extreme values
          panel = function(...){
             panel.grid(v = -1, h = 0, col = "gray70")
             panel.xyplot(...)
             panel.loess(..., span = .5, col = "darkred",
             lwd = 1.5)
          },
          scales = list(axs = "i", alternating = 1, tck = c(1,0)),
          xlab = "Year",
          ylab = "Average Temperature (\u00B0F)",
          main = "Mean Monthly Historical Temperatures in NYC"
   )

nyctemps

## Now try it with y-scale = "free' and addScales
##
nyctemps <- update(nyctemps,
   scales = list(axs = "i", alternating = 1,
   tck = c(1,0),y = list(relation = "free", draw = FALSE)))

addScales(nyctemps)

## The historical temperature trend as the city
## built up and modernized (more concrete and asphalt,people,
## heat sources, etc.) is clearer and quantified by the
## legend and scale lines; and the scale lines also show
## that winter temperatures are clearly more variable than summer.
## This was almost undetectable in the previous plot.

## The same plot using region shading instead of scale lines.
## Warning: May not display if your graphics device does not support
## alpha transparency

addScales(nyctemps, scaleType = "region")

## ... and using color coding for midlines and regions to better visually
## distinguish their values...
##
addScales(nyctemps, scaleType = "region", colCode = "r")

## You can repeat the exercise with the other two cities if you like.
## cleanup
rm(nyctemps, preprocess.temps, plotdat)

#######  Historical Crime Data #####

data(USAcrime)

## We explore the relationship beween property and violent crime over time.
## Point transparency via the 'alpha' setting is used to code year
## and the violent vs. property crime relationship is trellised by state
## for a selection of states.
##
## First with scales = "same", the default..

state.smpl <- c("CA","TX","GA","CO","VA","FL","NY","OH","MO","UT","MA","TN")

wh <- USAcrime$State %in% state.smpl

pcols <- hcl.colors(55, rev = TRUE)

crm <-xyplot(allViolent ~ allProperty|State, data = USAcrime[wh,],
             subscripts = TRUE, as.table = TRUE,
             layout = c(4,3), type = c("p", "g"),
             cex= .75,  pch = 19,
             col = pcols[USAcrime[wh,'Year'] -1959],
             par.strip.text = list(lines = .8, cex = .7),
             between = list(x = 1),
             scales = list(axs="i",alternating =1, tck = c(1,0)),
             xlab = "Property Crime Rate (incidents/100,000 population)",
             ylab = "Violent Crime Rate (incidents/100,000 population)",
           main = paste0("Violent vs. Property Crime Rates from 1960-2014 For 12 States"),
             sub = "Point Darkness Codes Years: Darker = Later Years",
             panel = function(subscripts,col,...)
                panel.xyplot(col = col[subscripts],...)
)
crm
## remove the grid and update with
## "free" scales and no axes for both x and y
crm2 <- update(crm, type = "p",
            scales = list(axs="i", relation = "free", draw = FALSE))

## Add scales for both x and y and color code midlines
addScales(crm2, scaleline = TRUE, colCode = "m")

## Some features to note:
##  1. As one might expect, violent and property crime rates are
##   correlated.
##
##  2. Crime rates first increased, peaked, and then decreased over time.
##
##  3. For most states there appears to be a kind of 'hysteresis':
##  the trajectory of the crime decrease is shifted up (higher violent
##  crime rate for the same property rate) from when it increased.
##  This could have been due to a change in reporting procedures,
##  over time, for example.
##
##  4. The midline colors and labels show that NY has the highest
##  violent crime rate, but a modest property crime rate: Tennessee
##  has a middling violent crime rate but the lowest (with VA) property
##  crime rate.
##
##  cleanup
```

```
rm( state.smpl, wh, pcols, crm, crm2)
```

`CHITemps` *Daily Chicago High and Low Temperatures in °F*

Description

Chicago Midway Airport daily high and low temperatures from 1 March 1928 to 31 December 2019. There are some missing values, and the data are listed in chronological order.

Usage

```
CHITemps
```

Format

A data frame with 32152 observations on the following 3 variables.

`DATE`	character vector of form "YYYY-MM-DD"
`TMAX`	numeric, daily high temperature
`TMIN`	numeric, daily low temperature

Source

NOAA National Centers for Environmental Information: Climate Data Online

https://www.ncdc.noaa.gov/cdo-web/

`NYCTemps` *Daily New York City High and Low Temperatures in °F*

Description

New York City Central Park daily high and low temperatures from 1 January 1870 to 12 December 2019. There are no missing values, and the data are listed in chronological order.

Usage

```
NYCTemps
```

Format

A data frame with 54786 observations on the following 3 variables.

`DATE`	character vector of form "YYYY-MM-DD"
`TMAX`	numeric, daily high temperature
`TMIN`	numeric, daily low temperature

Source

NOAA National Centers for Environmental Information: Climate Data Online

https://www.ncdc.noaa.gov/cdo-web/

---

panel.addScales *Default panelFUN For addScales.trellis*

Description

Adds Labeled Midline and (Unlabeled) Lines or Shaded Regions Showing Plot Scaling to Trellis Panels.

Usage

```
panel.addScales(
scaleline = c(0,0),
scaleType = c("line", "region"),
ndig.midline = c(h = 2, v = 2),
col.midline = "red",
adj.midline = c(-0.1, -0.25),
midline.aes = list(lwd = if(colCode == "n") 1.5 else 3),
midline.label.aes =list(
   col = if(colCode == "n")col.midline else "black",
   fontface = "bold",
   cex = .8),
scaleline.aes = list(lty = "dashed", col = "purple"),
region.aes = list(fill = "tan", alpha = .20),
colCode = c("n", "m", "r"),
palette =  hcl.colors(n = 100, "Viridis"),
...
)
```

Arguments

`scaleline`	A numeric vector of length 1 or 2 giving the distance from the scalelines to the midline(s); or equivalently, the half width(s) of the shaded scale regions. Signs are ignored. If two values are given, the first is the (vertical) distance to the horizontal midline and the second is the (horizontal) distance to the vertical midline. If a single value is given,it is assumed to be the first and the second is 0. 0 or NA mean: don't add that scaleline/region and midline.
`scaleType`	Whether the scale distance from the midline is shown by scale lines or as a shaded region. Note: Use of shaded regions requires that the graphics device support alpha transparency. Regions will not be shown properly – or at all – if it does not.

`ndig.midline`	Named or unnamed pair of integer arguments, or a single integer that will be replicated. The names must be (and are assumed to be if unnamed) "h" and "v" *in that order* and give the number of *significant* digits to show in the midline labels for the corresponding midlines. Non-integer values are rounded to integer, and values outside the range of 0 to 15digits are converted to 2. However, see the **Details** section below for a caveat.
`col.midline`	Midline color. Can be given in any form suitable for the base `col` parameter of `par`.
`adj.midline`	`numeric`: The `adj` vector of length two that will be fed to `panel.text` to position the midline label with respect to the midline. `c(.5,.5)` centers the label on the midline and lower limit of the relevant axis (x-axis for a horizontal midline, so left-center; and y-axis for a vertical midline, so bottom-center). See the base `text` function for details.
`midline.aes`	List of *aesthetics* of midline: `lwd`, `lty`, `alpha`, but **not** `col`, as this is already specified in the `col.midline` argument.
`midline.label.aes`	List of *aesthetics* of the midline label: `col`, `cex`, `font`, `fontface`, `etc.`, but not `adj`, as this is handled by `adj.midline`. See `panel.text` for argument details.
`scaleline.aes`	List of *aesthetics* of the scale lines. Same as for `midline.aes` except that `col` can be specified.
`region.aes`	`fill` and `alpha` parameters for shaded scale region.
`colCode`	`character`: Should the midlines and possibly also the scale regions be color coded by the midline value in the range of all midline values? Doing this in addition to labeling their values can improve visual comparison of midline levels among the panels. "m" means color code just the midlines; "r" means color code both midlines and scale regions; "n", the default, means do not color code.
`palette`	A vector of colors to use for color coding from low to high values. See `heat.colors` for how to conveniently specify color palettes.
`...`	Additional arguments passed down to the panel function, mostly ignored here. But see the note below for an exception.

Details

Midline values with significant digits that change precision by “small” amounts relative to the `scaleline` values are *zapped* by the `zapsmall()` function to remove extra digits in the display to improve readability. For example, a `midline` value of 1.23 would be shown as 1 when `ndig.midline` = 2 and `scaleline` = 100.

Value

No value is returned. This panel function is added to the existing panel function component of the `trellis` object on which `addScales` dispatches. It adds midline(s) and scale lines or regions to the panels when they are plotted.

Note

A list, `all.panel.limits = list(h = obj$y.limits, v = obj$x.limits)` with these components of the trellis object list is always passed to the panel function (as part of the ... list). This makes available *all* panel limits to the panel function, not just those of its own panel. This list is used for color coding midlines and/or scale regions, but is ignored otherwise here. Alternative panel functions may choose to use this information in other ways.

Note

For the `xxx.aes` arguments, when the user explicitly specifies the list, any component not specified will default to its formal `panel.addScales` argument value if that exists, or to the relevant panel function default if not, i.e. `panel.refline` for the lines and `panel.text` for the labels.

Author(s)

Bert Gunter <`bgunter.4567@gmail.com`>

See Also

`colors`, `par`, `panel.refline`, `panel.text`, `text`

`prepanel.trim` *Lattice Prepanel Function to Trim Panel Limits*

Description

Trims numeric x and or y limits to specified quantiles. May be useful when unusual extreme values distort the scales and obscure informative features of the data. Scales for factors are not affected.

Usage

```
prepanel.trim(x, y, trim.x = 0, trim.y = 0.05, min.trim = 20, ...)
```

Arguments

`x, y`	x and y values, numeric or factor.
`trim.x, trim.y`	Numeric trimming proportions, p, with 0 <= p < .5 . `trim.x` and `trim.y` can be different.
`min.trim`	The minimum number of data values needed before trimming *after removing* '`NAs`' and '`Infs`'. Otherwise the range of the data is returned (min and max of all the remaining finite values).
`...`	Other arguments, usually ignored

Details

If the trimming proportion is p, the limits returned are essentially `quantile(p, 1-p, type = 8)`. See `quantile` for details. So, for example if p = .1, roughly 10% of the lowest and 10% of the highest values are removed, and the range of the middle 80% of the data are returned. More precisely (quoting from `xyplot`), "... the actual limits of the panels are guaranteed to include the limits returned by the prepanel function" – i.e., these quantiles.

Value

For numeric data, a numeric vector of length 2, as would be returned by `range`. For a factor, a list with components `yat` and `ylim`, as described in the `prepanel` section of `xyplot`

Note

No banking calculations are done.

Author(s)

Bert Gunter <`bgunter.4567@gmail.com`>

See Also

`xyplot`, `prepanel.default.xyplot`, `quantile`

`revert` *Revert A Scaled Trellis Plot To Its Previous Unscaled Form*

Description

S3 generic and scaledTrellis method to remove all scaling information from a `scaledTrellis` object, returning the prior unscaled trellis object.

Usage

```
revert(obj,...)
## S3 method for class 'scaledTrellis'
revert(obj, ...)
```

Arguments

`obj`	An object inheriting from class `scaledTrellis`.
`...`	Currently ignored

Details

Returns the last version of the `trellis` object with all `addScales` scales and legends removed. Note that this is *not* the original `trellis` object if that was subsequently modified by `update` calls. See the examples.

Value

A trellis object that can be printed/plotted as usual.

Author(s)

Bert Gunter <bgunter.4567@gmail.com>

See Also

update.scaledTrellis

Examples

```
## Using simple artificial data
set.seed (2233)
x <- rep(1:10,4)
y <- rnorm(40, mean = rep(seq(10, 25, by = 5), each = 10),
           sd = rep(1:4, each = 10))
f <- rep(c("AA","BB","CC","DD"), each = 10)
##
## trellis plot the data with "free" y axis sxaling
orig <- xyplot(y ~ x|f, type = c("l","p"), col.line = "black",
       scales = list(alternating =1,
                     y = list(relation = "free")),
       as.table = TRUE,
       layout = c(2,2),
       main = "revert() Example"
)
## Plot it
orig

## Remove the y axis scales and add horizontal scalelines
orig <- update(orig, scales = list(alternating =1,
            y = list(relation = "free", draw = FALSE)))
upd1 <- addScales(orig)
## Plot it
upd1
class(upd1)

## revert
upd2 <- revert(upd1)
## Plot it
upd2
class(upd2)

## clean up
rm(x, y, f, orig, upd1, upd2)
```

scaleline *Extract scaleline list from* scaledTrellis *object*

Description

Extracts the scaleline list from an object inheriting from class `scaledTrellis`. This is useful if the user wants to create their own legend rather than using that generated by `addScales`.

Usage

```
scaleline(obj, ...)
## S3 method for class 'scaledTrellis'
scaleline(obj, ...)
```

Arguments

`obj`	`scaledTrellis` object
`...`	Possible further arguments for future methods. Ignored at present.

Details

Extracts the `scaleline` list from a `scaledTrellis` object. Note that the actual calculated values are returned, not the rounded/formatted values that would be shown in the legend.

Value

A list with components:

h numeric: distance between horizontal scalelines and midline

v numeric: distance between vertical scalelines and midline

Author(s)

Bert Gunter <`bgunter.4567@gmail.com`>

See Also

`addScales`

Examples

```
set.seed(8763)
simp <- xyplot(rnorm(10) ~ runif(10))
## Plot it
simp

## Add horizontal and vertical scale lines
ad <-addScales(simp,
 scaleline = TRUE,,
 ndig.midline = 1  ## only 1 digit will be shown
)
## Plot it
ad

## But here are the actual values
## (shown to default number of digits given
## by "digits" argument of print.default)
scaleline(ad)

## cleanup
rm(simp, ad)
```

`SFTemps` *Daily San Francisco High and Low Temperatures in °F*

Description

San Francisco daily high and low temperatures from 1 March 1949 to 28 February 2020. There are some missing values, and the data are listed in chronological order.

Usage

```
SFTemps
```

Format

A data frame with 22406 observations on the following 3 variables.

`DATE`	character vector of form "YYYY-MM-DD"
`TMAX`	numeric, daily high temperature
`TMIN`	numeric, daily low temperature

Source

NOAA National Centers for Environmental Information: Climate Data Online

https://www.ncdc.noaa.gov/cdo-web/

`update.scaledTrellis` *Update Method for scaledTrellis Objects*

Description

Updates both `addScale` parameters, including those passed to the `panelFUN`, and components of the original `trellis` object. Note that this may produce undesirable results if the axis limits are changed via updating trellis parameters without updating scaleline limits. See the examples.

Usage

```
   ## S3 method for class 'scaledTrellis'
update(object, ...)
```

Arguments

`object`	`scaledTrellis` object: The object on which method dispatch is carried out.
`...`	Any number of name = value pairs giving arguments that will be used to update `object`

Details

Arguments to `addScales` and those to (the most recent version of) the original `trellis` object are separated and `update.trellis` is first called on the latter. This means that any `trellis` argument changes must satisfy the restrictions on what `update.trellis` can change, basically, anything but the data used for plotting. *All* `addScales` and `panelFUN` parameters can be changed as long as such changes are possible (e.g. no scale lines can be added for factors.

Value

The updated object of class `c("scaledTrellis","trellis")` if successful.
The unmodified object if an error occurs.

Author(s)

Bert Gunter <`bgunter.4567@gmail.com`>

See Also

`update.trellis`, `addScales`, `panel.addScales`

Examples

```
## Replicate the USAcrimes example in ?addScales
##
data(USAcrime)
state.smpl <- c("CA","TX","GA","CO","VA","FL","NY","OH","MO","UT","MA","TN")
wh <- USAcrime$State %in% state.smpl
```

```
pcols <- hcl.colors(n = 55, rev = TRUE)
crm <-xyplot(allViolent ~ allProperty|State, data = USAcrime[wh,],
             subscripts = TRUE,
             as.table = TRUE,
             layout = c(4,3), type = c("p", "g"),
             cex= .75,  pch = 19,
             col = pcols[USAcrime[wh,'Year'] -1959],
             par.strip.text = list(lines = .7, cex = .7),
             between = list(x = 1),
             scales = list(axs="i",relation = "free", draw = FALSE),
             xlab = "Property Crime Rate (incidents/100,000 population)",
             ylab = "Violent Crime Rate (incidents/100,000 population)",
           main = paste0("Violent vs. Property Crime Rates from 1960-2014 For 12 States"),
             sub = "Point Darkness Codes Years: Darker = Later Years",
             panel = function(subscripts,col,...)
                panel.xyplot(col = col[subscripts],...)
   )
crm
ads.1 <- addScales(crm, scaleline = TRUE)
ads.1 ## plot it

## Change the plotting symbol, add a fitted line to the panel,
## remove the grid, change the layout,
## color code the midline and use shaded scale regions instead
## of lines, and put the legend on the right.
##
## Note that the arguments can be given in any order.
## (automatically plotted since no return value)
update(ads.1, pch = 19,layout = c(3,4), type = "p",
       colCode = "m", scaleType = "reg", legend.loc = "right",
       panel = function(x, y,  ...){
          panel.xyplot(x, y,...)
          panel.abline(reg = lm(y ~ x), col = "darkred", lwd = 2)
       }
)

## example of problems that can occur when updating trellis scales without
## updating addScales.
##
## Example from addScales() help:
x <- rep(0:10, 4)
scaling <- rep(c(1, 2, 5, 10), e = 11)
y <- sin(pi*x/10)*scaling + scaling ## add some structure
f <- factor(rep(LETTERS[1:4], e = 11))

## Now add noise proportional to data mean (= constant CV)
set.seed(91834)
y <- y + rnorm(44,sd = scaling/3)

## Plot this with the default "same" scaling and a loess curve

samescale <- xyplot(y ~ x| f, layout = c(4,1), col = "darkblue",
                    scales = list(alternating = 1, tck = c(1,0)),
                    panel = function(...){
                        panel.points(...)
                        panel.loess(..., span = .6, col.line = "blue")
                    })
samescale

## Call addScales and then update scale. This uses update.scaledTrellis:

update(addScales(samescale, scaleType = "region"),
       scales =  list(y = list(relation = "free", draw = FALSE)))

## This will generate a warning message, shown after the examples complete,
## and useless scaleline regions.
##
## Repeat, but now update the scaleline argument of addScales() also:

update(addScales(samescale, scaleType = "region"),
       scaleline = list(h = TRUE, v = FALSE),
       scales =  list(y = list(relation = "free", draw = FALSE))
)

## The updated scale regions are now appropriate.
## This could also have been done by first updating the trellis object
## (which would use the update.trellis method) and **then** calling
## addScales() on that, i.e.

addScales(update(samescale, scales =
                     list(y = list(relation = "free", draw = FALSE))),
          scaleType = "region")
## cleanup
rm(ads.1, crm, pcols, wh, state.smpl, samescale,
   scaling, x, y, f)
```

USAcrime *USA Property and Violent Crime Data, 1960 - 2014*

Description

USA crime rates as incidents/100,000 population by state for several categories of property and violent crimes from 1960 - 2014 (except for New York, which starts in 1965).

Usage

```
data("USAcrime")
```

Format

A data frame with 2745 observations on the following 12 variables.

`Year` numeric

`State` a factor with `state.abb` as its levels

`Population` estimated population, numeric

`allViolent` overall estimated violent crime rate, numeric

`Homicide` numeric

`Rape` numeric

`Robbery` numeric

`Assault` numeric

`allProperty` overall estimated property crime rate, numeric

`Burglary` numeric

`Larceny` numeric

`vehicleTheft` numeric

Details

Note that these are estimates, and there are various caveats and inconsistencies in definitions and reporting methods among states and over time. Consult the source for details.

Note

The data are ordered by state and year within state. The levels of the (unordered) State factor are `state.abb` but in the full state name order of `state.name`. Hence, for example, AL will procede AK and AZ will precede AR in the default ordering of the levels, because Alabama precedes Alaska and Arizona precedes Arkansas in lexicographic order.

Source

FBI Uniform Crime Reporting Statistics `https://www.ucrdatatool.gov/Search/Crime/State/StatebyState.cfm`

Index