Package 'lattice'

April 24, 2011

Version 0.19-26

Date 2011/04/19

Priority recommended

Title Lattice Graphics

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

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

Description Lattice is a powerful and elegant high-level data visualization system, with an emphasis on multivariate data,that is sufficient for typical graphics needs, and is also flexible enough to handle most nonstandard requirements. See ?Lattice for an introduction.

Depends R (>= 2.11.0)

Suggests grid, KernSmooth, MASS

Imports grid, grDevices, graphics, stats, utils

Enhances chron

LazyLoad yes

LazyData yes

License GPL (>= 2)

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

Repository CRAN

Date/Publication 2011-04-24 06:55:43

R **topics documented:**

A_01_Lattice   *Lattice Graphics*

Description

The **lattice** add-on package is an implementation of Trellis graphics for R. It is a powerful and elegant high-level data visualization system with an emphasis on multivariate data. It is designed to meet most typical graphics needs with minimal tuning, but can also be easily extended to handle most nonstandard requirements.

Details

Trellis Graphics, originally developed for S and S-PLUS at the Bell Labs, is a framework for data visualization developed by R. A. Becker, W. S. Cleveland, et al, extending ideas presented in Cleveland's 1993 book *Visualizing Data*. The Lattice API is based on the original design in S, but extends it in many ways.

The Lattice user interface primarily consists of several 'high-level' generic functions (listed below in the "See Also" section), each designed to create a particular type of display by default. Although the functions produce different output, they share many common features, reflected in several common arguments that affect the resulting displays in similar ways. These arguments are extensively (sometimes only) documented in the help page for `xyplot`, which also includes a discussion of the important topics of *conditioning* and control of the Trellis layout. Features specific to other high-level functions are documented in their respective help pages.

Lattice employs an extensive system of user-controllable settings to determine the look and feel of the displays it produces. To learn how to use and customize the graphical parameters used by lattice, see `trellis.par.set`. For other settings, see `lattice.options`. The default graphical settings are (potentially) different for different graphical devices. To learn how to initialize new devices with the desired settings or change the settings of the current device, see `trellis.device`.

It is usually unnecessary, but sometimes important to be able to plot multiple lattice plots on a single page. Such capabilities are described in the `print.trellis` help page. See `update.trellis` to learn about manipulating a `"trellis"` object. Tools to augment lattice plots after they are drawn (including `locator`-like functionality) are described in the `trellis.focus` help page.

The online documentation accompanying the package is complete, and effort has been made to present the help pages in a logical sequence, so that one can learn how to use lattice by reading the PDF reference manual available at `http://cran.r-project.org/web/packages/lattice/`. However, the format in which the online documentation is written and the breadth of topics covered necessarily makes it somewhat terse and less than ideal as a first introduction. For a more gentle introduction, a book on lattice is available as part of Springer's 'Use R' series; see the "References" section below.

Note

High-level **lattice** functions like `xyplot` are different from traditional R graphics functions in that they do not perform any plotting themselves. Instead, they return an object, of class `"trellis"`, which has to be then `print`-ed or `plot`-ted to create the actual plot. Due to R's automatic printing rule, it is usually not necessary to explicitly carry out the second step, and **lattice** functions appear to behave like their traditional counterparts. However, the automatic plotting is suppressed when the high-level functions are called inside another function (most often `source`) or in other contexts where automatic printing is suppressed (e.g., `for` or `while` loops). In such situations, an explicit call to `print` or `plot` is required.

The **lattice** package is based on the Grid graphics engine and requires the **grid** add-on package. One consequence of this is that it is not (readily) compatible with traditional R graphics tools. In particular, changing `par()` settings usually has no effect on Lattice plots; **lattice** provides its own interface for querying and modifying an extensive set of graphical and non-graphical settings.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. ISBN: 978-0-387-75968-5 `http://lmdvr.r-forge.r-project.org/`

Cleveland, William .S. (1993) *Visualizing Data*, Hobart Press, Summit, New Jersey.

Becker, R. A. and Cleveland, W. S. and Shyu, M. J. (1996). "The Visual Design and Control of Trellis Display", *Journal of Computational and Graphical Statistics*, **5(2)**, 123–155.

Bell Lab's Trellis Page contains several documents outlining the use of Trellis graphics; these provide a holistic introduction to the Trellis paradigm: `http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/`

See Also

The following is a list of high-level functions in the **lattice** package and their default displays. In all cases, the actual display is produced by the so-called "panel" function, which has a suitable default, but can be substituted by an user defined function to create customized displays. In many cases, the default panel function will itself have many optional arguments to customize its output. The default

panel functions are named as "`panel.`" followed by the name of the corresponding high-level function; i.e., the default panel function for `xyplot` is `panel.xyplot`, the one for `histogram` is `panel.histogram`, etc. Each default panel function has a separate help page, linked from the help pages of the corresponding high-level function. Although documented separately, arguments to these panel functions can be supplied directly to the high-level functions, which will pass on the arguments appropriately.

Univariate:

`barchart`: Bar plots.

`bwplot`: Box-and-whisker plots.

`densityplot`: Kernel density estimates.

`dotplot`: Cleveland dot plots.

`histogram`: Histograms.

`qqmath`: Theretical quantile plots.

`stripplot`: One-dimensional scatterplots.

Bivariate:

`qq`: Quantile plots for comparing two distributions.

`xyplot`: Scatterplots and time-series plots (and potentially a lot more).

Trivariate:

`levelplot`: Level plots (similar to `image` plots).

`contourplot`: Contour plots.

`cloud`: Three-dimensional scatter plots.

`wireframe`: Three-dimensional surface plots (similar to `persp` plots).

Hypervariate:

`splom`: Scatterplot matrices.

`parallel`: Parallel coordinate plots.

Miscellaneous:

`rfs`: Residual and fitted value plots (also see `oneway`).

`tmd`: Tukey Mean-Difference plots.

In addition, there are several panel functions that do little by themselves, but can be useful components of custom panel functions. These are documented in `panel.functions`. Lattice also provides a collection of convenience functions that correspond to the traditional graphics primitives `lines`, `points`, etc. These are implemented using Grid graphics, but try to be as close to the traditional versions as possible in terms of their argument list. These functions have names like `llines` or `panel.lines` and are often useful when writing (or porting from S-PLUS code) nontrivial panel functions.

Finally, many useful enhancements that extend the Lattice system are available in the **latticeExtra** package.

Examples

```
## Not run:

## Show brief history of changes to lattice, including
## a summary of new features.

RShowDoc("NEWS", package = "lattice")

## End(Not run)
```

B_00_xyplot   *Common Bivariate Trellis Plots*

Description

This help page documents several commonly used high-level Lattice functions. xyplot produces
bivariate scatterplots or time-series plots, bwplot produces box-and-whisker plots, dotplot
produces Cleveland dot plots, barchart produces bar plots, and stripplot produces one-
dimensional scatterplots. All these functions, along with other high-level Lattice functions, respond
to a common set of arguments that control conditioning, layout, aspect ratio, legends, axis annota-
tion, and many other details in a consistent manner. These arguments are described extensively in
this help page, and should be used as the reference for other high-level functions as well.

For control and customization of the actual display in each panel, the help page of the respective
default panel function will often be more informative. In particular, these help pages describe many
arguments commonly used when calling the corresponding high-level function but are specific to
them.

Usage

```
xyplot(x, data, ...)
dotplot(x, data, ...)
barchart(x, data, ...)
stripplot(x, data, ...)
bwplot(x, data, ...)

## S3 method for class 'formula'
xyplot(x,
       data,
       allow.multiple = is.null(groups) || outer,
       outer = !is.null(groups),
       auto.key = FALSE,
       aspect = "fill",
       panel = lattice.getOption("panel.xyplot"),
       prepanel = NULL,
       scales = list(),
       strip = TRUE,
       groups = NULL,
```

```
        xlab,
        xlim,
        ylab,
        ylim,
        drop.unused.levels = lattice.getOption("drop.unused.levels"),
        ...,
        lattice.options = NULL,
        default.scales,
        default.prepanel = lattice.getOption("prepanel.default.xyplot"),
        subscripts = !is.null(groups),
        subset = TRUE)

## S3 method for class 'formula'
dotplot(x,
        data,
        panel = lattice.getOption("panel.dotplot"),
        default.prepanel = lattice.getOption("prepanel.default.dotplot"),
        ...)

## S3 method for class 'formula'
barchart(x,
        data,
        panel = lattice.getOption("panel.barchart"),
        default.prepanel = lattice.getOption("prepanel.default.barchart"),
        box.ratio = 2,
        ...)

## S3 method for class 'formula'
stripplot(x,
        data,
        panel = lattice.getOption("panel.stripplot"),
        default.prepanel = lattice.getOption("prepanel.default.stripplot"),
        ...)

## S3 method for class 'formula'
bwplot(x,
        data,
        allow.multiple = is.null(groups) || outer,
        outer = FALSE,
        auto.key = FALSE,
        aspect = "fill",
        panel = lattice.getOption("panel.bwplot"),
        prepanel = NULL,
        scales = list(),
        strip = TRUE,
        groups = NULL,
        xlab,
        xlim,
```

```
          ylab,
          ylim,
          box.ratio = 1,
          horizontal = NULL,
          drop.unused.levels = lattice.getOption("drop.unused.levels"),
          ...,
          lattice.options = NULL,
          default.scales,
          default.prepanel = lattice.getOption("prepanel.default.bwplot"),
          subscripts = !is.null(groups),
          subset = TRUE)
```

Arguments

x   All high-level function in **lattice** are generic. `x` is the object on which method
                dispatch is carried out.

                For the `"formula"` methods, `x` must be a formula describing the primary vari-
                ables (used for the per-panel display) and the optional conditioning variables
                (which define the subsets plotted in different panels) to be used in the plot. Con-
                ditioning is described in the "Details" section below.

                For the functions documented here, the formula is generally of the form `y ~ x
                | g1 * g2 * ...` (or equivalently, `y ~ x | g1 + g2 + ...`), indi-
                cating that plots of `y` (on the y-axis) versus `x` (on the x-axis) should be produced
                conditional on the variables `g1, g2, ...`. Here x and y are the primary
                variables, and `g1, g2, ...` are the conditioning variables. The conditioning
                variables may be omitted to give a formula of the form `y ~ x`, in which case
                the plot will consist of a single panel with the full dataset. The formula can also
                involve expressions, e.g., `sqrt()`, `log()`, etc. See the `data` argument below
                for rules regarding evaluation of the terms in the formula.

                With the exception of `xyplot`, the functions documented here may also be
                supplied a formula of the form `~ x | g1 * g2 * ...`. In that case, `y`
                defaults to `names(x)` if `x` is named, and a factor with a single level otherwise.

                Cases where `x` is not a formula is handled by appropriate methods. The `numeric`
                methods are equivalent to a call with no left hand side and no conditioning vari-
                ables in the formula. For `barchart` and `dotplot`, non-trivial methods exist
                for tables and arrays, documented at `barchart.table`.

                The conditioning variables `g1, g2, ...` must be either factors or shingles.
                Shingles provide a way of using numeric variables for conditioning; see the help
                page of `shingle` for details. Like factors, they have a `"levels"` attribute,
                which is used in producing the conditional plots. If necessary, numeric condi-
                tioning variables are converted to shingles using the `shingle` function; how-
                ever, using `equal.count` may be more appropriate in many cases. Character
                variables are coerced to factors.

                Extended formula interface: As a useful extension of the interface described
                above, the primary variable terms (both the LHS `y` and RHS `x`) may consist of
                multiple terms separated by a '+' sign, e.g., `y1 + y2 ~ x | a * b`. This
                formula would be taken to mean that the user wants to plot both `y1 ~ x | a`

`*` `b` and `y2 ~ x | a * b`, but with the `y1 ~ x` and `y2 ~ x` superposed in each panel. The two groups will be distinguished by different graphical parameters. This is essentially what the `groups` argument (see below) would produce, if `y1` and `y2` were concatenated to produce a longer vector, with the `groups` argument being an indicator of which rows come from which variable. In fact, this is exactly what is done internally using the [reshape](#) function. This feature cannot be used in conjunction with the `groups` argument.

To interpret `y1 + y2` as a sum, one can either set `allow.multiple=FALSE` or use `I(y1+y2)`.

A variation on this feature is when the `outer` argument is set to `TRUE`. In that case, the plots are not superposed in each panel, but instead separated into different panels (as if a new conditioning variable had been added).

Primary variables: The `x` and `y` variables should both be numeric in `xyplot`, and an attempt is made to coerce them if not. However, if either is a factor, the levels of that factor are used as axis labels. In the other four functions documented here, exactly one of `x` and `y` should be numeric, and the other a factor or shingle. Which of these will happen is determined by the `horizontal` argument — if `horizontal=TRUE`, then `y` will be coerced to be a factor or shingle, otherwise `x`. The default value of `horizontal` is `FALSE` if `x` is a factor or shingle, `TRUE` otherwise. (The functionality provided by `horizontal=FALSE` is not S-compatible.)

Note that the `x` argument used to be called `formula` in earlier versions (when the high-level functions were not generic and the formula method was essentially the only method). This is no longer allowed. It is recommended that this argument not be named in any case, but instead be the first (unnamed) argument.

`data` For the `formula` methods, a data frame (or more precisely, anything that is a valid `envir` argument in [eval](#), e.g., a list or an environment) containing values for any variables in the formula, as well as `groups` and `subset` if applicable. If not found in `data`, or if `data` is unspecified, the variables are looked for in the environment of the formula. For other methods (where `x` is not a formula), `data` is usually ignored, often with a warning if it is explicitly specified.

`allow.multiple`
  Logical flag specifying whether the extended formula interface described above should be in effect. Defaults to `TRUE` whenever sensible.

`outer` Logical flag controlling what happens with formulas using the extended interface described above (see the entry for `x` for details). Defaults to `FALSE`, except when `groups` is explicitly specified or grouping does not make sense for the default panel function.

`box.ratio` Applicable to `barchart` and `bwplot`. Specifies the ratio of the width of the rectangles to the inter-rectangle space. See also the `box.width` argument in the respective default panel functions.

`horizontal` Logical flag applicable to `bwplot`, `dotplot`, `barchart`, and `stripplot`. Determines which of `x` and `y` is to be a factor or shingle (`y` if TRUE, `x` otherwise). Defaults to `FALSE` if `x` is a factor or shingle, `TRUE` otherwise. This argument is used to process the arguments to these high-level functions, but more importantly, it is passed as an argument to the panel function, which is expected to use it as appropriate.

A potentially useful component of scales in this case may be abbreviate = TRUE, in which case long labels which would usually overlap will be abbreviated. scales could also contain a minlength argument in this case, which would be passed to the abbreviate function.

Common arguments: The following arguments are common to all the functions documented here, as well as most other high-level Trellis functions. These are not documented elsewhere, except to override the usage given here.

panel   Once the subset of rows defined by each unique combination of the levels of the grouping variables are obtained (see "Details"), the corresponding x and y variables (or other variables, as appropriate, in the case of other high-level functions) are passed on to be plotted in each panel. The actual plotting is done by the function specified by the panel argument. The argument may be a function object or a character string giving the name of a predefined function. Each high-level function has its own default panel function, named as "panel." followed by the name of the corresponding high-level function (e.g., panel.xyplot, panel.barchart, etc).

Much of the power of Trellis Graphics comes from the ability to define customized panel functions. A panel function appropriate for the functions described here would usually expect arguments named x and y, which would be provided by the conditioning process. It can also have other arguments. It is useful to know in this context that all arguments passed to a high-level Lattice function (such as xyplot) that are not recognized by it are passed through to the panel function. It is thus generally good practice when defining panel functions to allow a ... argument. Such extra arguments typically control graphical parameters, but other uses are also common. See documentation for individual panel functions for specifics.

Note that unlike in S-PLUS, it is not guaranteed that panel functions will be supplied only numeric vectors for the x and y arguments; they can be factors as well (but not shingles). Panel functions need to handle this case, which in most cases can be done by simply coercing them to numeric.

Technically speaking, panel functions must be written using Grid graphics functions. However, knowledge of Grid is usually not necessary to construct new custom panel functions, as there are several predefined panel functions which can help; for example, panel.grid, panel.loess, etc. There are also some grid-compatible replacements of commonly used traditional graphics functions useful for this purpose. For example, lines can be replaced by llines (or equivalently, panel.lines). Note that traditional graphics functions like lines will not work in a lattice panel function.

One case where a bit more is required of the panel function is when the groups argument is not NULL. In that case, the panel function should also accept arguments named groups and subscripts (see below for details). A useful panel function predefined for use in such cases is panel.superpose, which can be combined with different panel.groups functions to determine what is plotted for each group. See the "Examples" section for an interaction plot constructed in this way. Several other panel functions can also handle the groups argument, including the default ones for xyplot, barchart, dotplot, and stripplot.

Even when `groups` is not present, the panel function can have `subscripts` as a formal argument. In either case, the `subscripts` argument passed to the panel function are the indices of the `x` and `y` data for that panel in the original `data`, BEFORE taking into account the effect of the `subset` argument. Note that `groups` remains unaffected by any subsetting operations, so `groups[subscripts]` gives the values of `groups` that correspond to the data in that panel.

This interpretation of `subscripts` does not hold when the extended formula interface is in use (i.e., when `allow.multiple` is in effect). A comprehensive description would be too complicated (details can be found in the source code of the function `latticeParseFormula`), but in short, the extended interface works by creating an artificial grouping variable that is longer than the original data frame, and consequently, `subscripts` needs to refer to rows beyond those in the original data. To further complicate matters, the artificial grouping variable is created after any effect of `subset`, in which case `subscripts` may have no relationship with corresponding rows in the original data frame.

One can also use functions called `panel.number` and `packet.number`, representing panel order and packet order respectively, inside the panel function (as well as the strip function or while interacting with a lattice display using `trellis.focus` etc). Both provide a simple integer index indicating which panel is currently being drawn, but differ in how the count is calculated. The panel number is a simple incremental counter that starts with 1 and is incremented each time a panel is drawn. The packet number on the other hand indexes the combination of levels of the conditioning variables that is represented by that panel. The two indices coincide unless the order of conditioning variables is permuted and/or the plotting order of levels within one or more conditioning variables is altered (using `perm.cond` and `index.cond` respectively), in which case `packet.number` gives the index corresponding to the 'natural' ordering of that combination of levels of the conditioning variables.

`panel.xyplot` has an argument called `type` which is worth mentioning here because it is quite frequently used (and as mentioned above, can be passed to `xyplot` directly). In the event that a `groups` variable is used, `panel.xyplot` calls `panel.superpose`, arguments of which can also be passed directly to `xyplot`. Panel functions for `bwplot` and friends should have an argument called `horizontal` to account for the cases when `x` is the factor or shingle.

aspect   This argument controls the physical aspect ratio of the panels, which is usually the same for all the panels. It can be specified as a ratio (vertical size/horizontal size) or as a character string. In the latter case, legitimate values are `"fill"` (the default) which tries to make the panels as big as possible to fill the available space; `"xy"`, which computes the aspect ratio based on the 45 degree banking rule (see `banking`); and `"iso"` for isometric scales, where the relation between physical distance on the device and distance in the data scale are forced to be the same for both axes.

If a `prepanel` function is specified and it returns components `dx` and `dy`, these are used for banking calculations. Otherwise, values from the default prepanel function are used. Not all default prepanel functions produce sensible banking calculations.

groups   A variable or expression to be evaluated in `data`, expected to act as a group-
                  ing variable within each panel, typically used to distinguish different groups by
                  varying graphical parameters like color and line type. Formally, if `groups` is
                  specified, then `groups` along with `subscripts` is passed to the panel func-
                  tion, which is expected to handle these arguments. For high level functions
                  where grouping is appropriate, the default panel functions can handle grouping.

                  It is very common to use a key (legend) when a grouping variable is specified.
                  See entries for `key`, `auto.key` and `simpleKey` for how to draw a key.

auto.key   A logical, or a list containing components to be used as arguments to `simpleKey`.
                  `auto.key=TRUE` is equivalent to `auto.key=list()`, in which case `simpleKey`
                  is called with a set of default arguments (which may depend on the relevant high-
                  level function). Most valid components to the `key` argument can be specified
                  in this manner, as `simpleKey` will simply add unrecognized arguments to the
                  list it produces.

                  `auto.key` is typically used to automatically produce a suitable legend in con-
                  junction with a grouping variable. If `auto.key=TRUE`, a suitable legend will
                  be drawn if a `groups` argument is also provided, and not otherwise. In list
                  form, `auto.key` will modify the default legend thus produced. For exam-
                  ple, `auto.key=list(columns = 2)` will create a legend split into two
                  columns (`columns` is documented in the entry for `key`).

                  More precisely, if `auto.key` is not `FALSE`, `groups` is non-null, and there
                  is no `key` or `legend` argument specified in the call, a key is created with
                  `simpleKey` with `levels(groups)` as the first (`text`) argument. (Note:
                  this may not work in all high-level functions, but it does work for the ones
                  where grouping makes sense with the default panel function). If `auto.key` is
                  provided as a list and includes a `text` component, then that is used instead as
                  the text labels in the key, and the key is drawn even if `groups` is not specified.

                  Note that `simpleKey` uses the default settings (see `trellis.par.get`) to
                  determine the graphical parameters in the key, so the resulting legend will be
                  meaningful only if the same settings are used in the plot as well. The `par.settings`
                  argument may be useful to temporarily modify the default settings for this pur-
                  pose.

                  One disadvantage to using `key` (or even `simpleKey`) directly is that the graph-
                  ical parameters used in the key are absolutely determined at the time when the
                  `"trellis"` object is created. Consequently, if a plot once created is re-plot-
                  ted with different settings, the original parameter settings will be used for the
                  key even though the new settings are used for the actual display. However, with
                  `auto.key`, the key is actually created at plotting time, so the settings will
                  match.

prepanel   A function that takes the same arguments as the `panel` function and returns
                  a list, possibly containing components named `xlim`, `ylim`, `dx`, and `dy` (and
                  less frequently, `xat` and `yat`). The return value of a user-supplied prepanel
                  function need not contain all these components; in case some are missing, they
                  are replaced by the component-wise defaults.

                  The `xlim` and `ylim` components are similar to the high level `xlim` and `ylim`
                  arguments (i.e., they are usually a numeric vector of length 2 defining a range,
                  or a character vector representing levels of a factor). If the `xlim` and `ylim`

arguments are not explicitly specified (possibly as components in scales) in the high-level call, then the actual limits of the panels are guaranteed to include the limits returned by the prepanel function. This happens globally if the relation component of scales is "same", and on a per-panel basis otherwise.

The dx and dy components are used for banking computations in case aspect is specified as "xy". See documentation of banking for details.

If xlim or ylim is a character vector (which is appropriate when the corresponding variable is a factor), this implicitly indicates that the scale should include the first n integers, where n is the length of xlim or ylim, as the case may be. The elements of the character vector are used as the default labels for these n integers. Thus, to make this information consistent between panels, the xlim or ylim values should represent all the levels of the corresponding factor, even if some are not used within that particular panel.

In such cases, an additional component xat or yat may be returned by the prepanel function, which should be a subset of 1:n, indicating which of the n values (levels) are actually represented in the panel. This is useful when calculating the limits with relation="free" or relation="sliced" in scales.

The prepanel function is responsible for providing a meaningful return value when the x, y (etc.) variables are zero-length vectors. When nothing else is appropriate, values of NA should be returned for the xlim and ylim components.

strip   A logical flag or function. If FALSE, strips are not drawn. Otherwise, strips are drawn using the strip function, which defaults to strip.default. See documentation of strip.default to see the arguments that are available to the strip function. This description also applies to the strip.left argument (see ... below), which can be used to draw strips on the left of each panel (useful for wide short panels, e.g., in time-series plots).

xlab   Character or expression (or a "grob") giving label(s) for the x-axis. Generally defaults to the expression for x in the formula defining the plot. Can be specified as NULL to omit the label altogether. Finer control is possible, as described in the entry for main, with the modification that if the label component is omitted from the list, it is replaced by the default xlab.

ylab   Character or expression (or "grob") giving label for the y-axis. Generally defaults to the expression for y in the formula defining the plot. Finer control is possible, see entries for main and xlab.

scales Generally a list determining how the x- and y-axes (tick marks and labels) are drawn. The list contains parameters in name=value form, and may also contain two other lists called x and y of the same form (described below). Components of x and y affect the respective axes only, while those in scales affect both. When parameters are specified in both lists, the values in x or y are used. Note that certain high-level functions have defaults that are specific to a particular axis (e.g., bwplot has alternating=FALSE for the categorical axis only); these can only be overridden by an entry in the corresponding component of scales.

As a special exception, `scales` (or its `x` and `y` components) can also be a character string, in which case it is interpreted as the `relation` component.

The possible components are :

`relation` A character string that determines how axis limits are calculated for each panel. Possible values are `"same"` (default), `"free"` and `"sliced"`. For `relation="same"`, the same limits, usually large enough to encompass all the data, are used for all the panels. For `relation="free"`, limits for each panel is determined by just the points in that panel. Behavior for `relation="sliced"` is similar, except that the length (max - min) of the scales are constrained to remain the same across panels.

  The determination of what axis limits are suitable for each panel can be controlled by the `prepanel` function, which can be overridden by `xlim`, `ylim` or `scales$limits` (except when `relation="sliced"`, in which case explicitly specified limits are ignored with a warning). When `relation` is `"free"`, `xlim` or `ylim` can be a list, in which case it is treated as if its components were the limit values obtained from the prepanel calculations for each panel (after being replicated if necessary).

`tick.number` An integer, giving the suggested number of ticks. This is ignored for a factor, shingle or character vector, in which case there is no natural rule for leaving out some of the labels. But see `xlim`.

`draw` A logical flag, defaulting to `TRUE`, that determines whether to draw the axis at all.

`alternating` Usually a logical flag specifying whether axis labels should alternate from one side of the group of panels to the other. For finer control, `alternating` can also be a vector (replicated to be as long as the number of rows or columns per page) consisting of the following numbers

- 0: do not draw tick labels
- 1: bottom/left
- 2: top/right
- 3: both.

  `alternating` applies only when `relation="same"`. The default is `TRUE`, or equivalently, `c(1, 2)`

`limits` Same as `xlim` and `ylim`.

`at` The location of tick marks along the axis (in native coordinates), or a list as long as the number of panels describing tick locations for each panel.

`labels` Vector of labels (characters or expressions) to go along with `at`. Can also be a list like `at`.

`cex` A numeric multiplier to control character sizes for axis labels. Can be a vector of length 2, to control left/bottom and right/top labels separately.

`font`, `fontface`, `fontfamily` Specifies the font to be used for axis labels.

`lineheight` Specifies the line height parameter (height of line as a multiple of the size of text); relevant for multi-line labels. (This is currently ignored for `cloud`.)

`tck` Usually a numeric scalar controlling the length of tick marks. Can also be a vector of length 2, to control the length of left/bottom and right/top tick marks separately.

col Color of tick marks and labels.

rot Angle (in degrees) by which the axis labels are to be rotated. Can be a vector of length 2, to control left/bottom and right/top axes separately.

abbreviate A logical flag, indicating whether to abbreviate the labels using the abbreviate function. Can be useful for long labels (e.g., in factors), especially on the x-axis.

minlength Argument passed to abbreviate if abbreviate=TRUE.

log Controls whether the corresponding variable (x or y) will be log transformed before being passed to the panel function. Defaults to FALSE, in which case the data are not transformed. Other possible values are any number that works as a base for taking logarithm, TRUE (which is equivalent to 10), and "e" (for the natural logarithm). As a side effect, the corresponding axis is labeled differently. Note that this is in reality a transformation of the data, not the axes. Other than the axis labeling, using this feature is no different than transforming the data in the formula; e.g., scales=list(x = list(log = 2)) is equivalent to y ~ log2(x).

format The format to use for POSIXct variables. See strptime for description of valid values.

axs A character string, "r" (default) or "i". In the latter case, the axis limits are calculated as the exact data range, instead of being padded on either side. (May not always work as expected.)

Note that much of the function of scales is accomplished by pscales in splom.

subscripts A logical flag specifying whether or not a vector named subscripts should be passed to the panel function. Defaults to FALSE, unless groups is specified, or if the panel function accepts an argument named subscripts. This argument is useful if one wants the subscripts to be passed on even if these conditions do not hold; a typical example is when one wishes to augment a Lattice plot after it has been drawn, e.g., using panel.identify.

subset An expression that evaluates to a logical or integer indexing vector. Like groups, it is evaluated in data. Only the resulting rows of data are used for the plot. If subscripts is TRUE, the subscripts provided to the panel function will be indices referring to the rows of data prior to the subsetting. Whether levels of factors in the data frame that are unused after the subsetting will be dropped depends on the drop.unused.levels argument.

xlim Normally a numeric vector (or a DateTime object) of length 2 giving left and right limits for the x-axis, or a character vector, expected to denote the levels of x. The latter form is interpreted as a range containing c(1, length(xlim)), with the character vector determining labels at tick positions 1:length(xlim).

xlim could also be a list, with as many components as the number of panels (recycled if necessary), with each component as described above. This is meaningful only when scales$x$relation is "free", in which case these are treated as if they were the corresponding limit components returned by prepanel calculations.

ylim Similar to xlim, applied to the y-axis.

drop.unused.levels

A logical flag indicating whether the unused levels of factors will be dropped, usually relevant with a subsetting operation is performed or an `interaction` is created. Unused levels are usually dropped, but it is sometimes appropriate to suppress dropping to preserve a useful layout. For finer control, this argument could also be list containing components `cond` and `data`, both logical, indicating desired behavior for conditioning variables and primary variables respectively. The default is given by `lattice.getOption("drop.unused.levels")`, which is initially set to `TRUE` for both components. Note that this argument does not control dropping of levels of the `groups` argument.

default.scales

A list giving the default values of `scales` for a particular high-level function. This is rarely of interest to the end-user, but may be helpful when defining other functions that act as a wrapper to one of the high-level Lattice functions.

default.prepanel

A function or character string giving the name of a function that serves as the (component-wise) fallback prepanel function when the `prepanel` argument is not specified, or does not return all necessary components. The main purpose of this argument is to enable the defaults to be overridden through the use of `lattice.options`.

lattice.options

A list that could be supplied to `lattice.options`. These options are applied temporarily for the duration of the call, after which the settings revert back to what they were before. The options are retained along with the object and reused during plotting. This enables the user to attach options settings to the trellis object itself rather than change the settings globally. See also the `par.settings` argument described below for a similar treatment of graphical settings.

...   Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions. Such arguments can be broadly categorized into two types: those that affect all high-level Lattice functions in a similar manner, and those that are meant for the specific panel function being used.

The first group of arguments are processed by a common, unexported function called `trellis.skeleton`. These arguments affect all high-level functions, but are only documented here (except to override the behaviour described here). All other arguments specified in a high-level call, specifically those neither described here nor in the help page of the relevant high-level function, are passed unchanged to the panel function used. By convention, the default panel function used for any high-level function is named as "`panel.`" followed by the name of the high-level function; for example, the default panel function for `bwplot` is `panel.bwplot`. In practical terms, this means that in addition to the help page of the high-level function being used, the user should also consult the help page of the corresponding panel function for arguments that may be specified in the high-level call.

The effect of the first group of common arguments are as follows:

as.table: A logical flag that controls the order in which panels should be displayed: if `FALSE` (the default), panels are drawn left to right, bottom to top (as in a graph); if `TRUE`, left to right, top to bottom (as in a table).

between: A list with components x and y (both usually 0 by default), numeric vectors specifying the space between the panels (units are character heights). x and y are repeated to account for all panels in a page and any extra components are ignored. The result is used for all pages in a multi page display. In other words, it is not possible to use different between values for different pages.

key: A list that defines a legend to be drawn on the plot. This list is used as an argument to the draw.key function, which produces a "grob" (grid object) eventually plotted by the print method for "trellis" objects. The structure of the legend is constrained in the ways described below.

Although such a list can be and often is created explicitly, it is also possible to generate such a list using the simpleKey function; the latter is more convenient but less flexible. The auto.key argument can be even more convenient for the most common situation where legends are used, namely, in conjunction with a grouping variable. To use more than one legend, or to have arbitrary legends not constrained by the structure imposed by key, use the legend argument.

The position of the key can be controlled in either of two possible ways. If a component called space is present, the key is positioned outside the plot region, in one of the four sides, determined by the value of space, which can be one of "top", "bottom", "left" and "right". Alternatively, the key can be positioned inside the plot region by specifying components x, y and corner. x and y determine the location of the corner of the key given by corner, which is usually one of c(0,0), c(1,0), c(1,1) and c(0,1), which denote the corners of the unit square. Fractional values are also allowed, in which case x and y determine the position of an arbitrary point inside (or outside for values outside the unit interval) the key.

x and y should be numbers between 0 and 1, giving coordinates with respect to the "display area". Depending on the value of the "legend.bbox" option (see lattice.getOption), this can be either the full figure region ("full"), or just the region that bounds the panels and strips ("panel"). The key essentially consists of a number of columns, possibly divided into blocks, each containing some rows. The contents of the key are determined by (possibly repeated) components named "rectangles", "lines", "points" or "text". Each of these must be lists with relevant graphical parameters (see later) controlling their appearance. The key list itself can contain graphical parameters, these would be used if relevant graphical components are omitted from the other components.

The length (number of rows) of each such column (except "text"s) is taken to be the largest of the lengths of the graphical components, including the ones specified outside (see the entry for rep below for details on this). The "text" component must have a character or expression vector as its first component, to be used as labels. The length of this vector determines the number of rows.

The graphical components that can be included in key and also in the components named "text", "lines", "points" and "rectangles" (as appropriate) are:

- `cex=1` (text, lines, points)
- `col="black"` (text, rectangles, lines, points)
- `alpha=1` (text, rectangles, lines, points)
- `fill="transparent"` (lines, points)
- `lty=1` (lines)
- `lwd=1` (lines, points)
- `font=1` (text, points)
- `fontface` (text, points)
- `fontfamily` (text, points)
- `pch=8` (lines, points)
- `adj=0` (text)
- `type="l"` (lines)
- `size=5` (rectangles, lines)
- `height=1` (rectangles)
- `lineheight=1` (text)
- `angle=0` (rectangles, but ignored)
- `density=-1` (rectangles, but ignored)

In addition, the component `border` can be included inside the `"rect"` component to control the border color of the rectangles; when specified at the top level, `border` controls the border of the entire key (see below).

`angle` and `density` are unimplemented. `size` determines the width of columns of rectangles and lines in character widths. `type` is relevant for lines; `"l"` denotes a line, `"p"` denotes a point, and `"b"` and `"o"` both denote both together. `height` gives heights of rectangles as a fraction of the default.

Other possible components of `key` are:

`reverse.rows` Logical flag, defaulting to `FALSE`. If `TRUE`, all components are reversed *after* being replicated (the details of which may depend on the value of `rep`). This is useful in certain situations, e.g., with a grouped `barchart` with `stack = TRUE` with the categorical variable on the vertical axis, where the bars in the plot will usually be ordered from bottom to top, but the corresponding legend will have the levels from top to bottom unless `reverse.rows = TRUE`. Note that in this case, unless all columns have the same number or rows, they will no longer be aligned.

`between` Numeric vector giving the amount of space (character widths) surrounding each column (split equally on both sides).

`title` String or expression giving a title for the key.

`rep` Logical flag, defaults to `TRUE`. By default, it is assumed that all columns in the key (except the `"text"`s) will have the same number of rows, and all components are replicated to be as long as the longest. This can be suppressed by specifying `rep=FALSE`, in which case the length of each column will be determined by components of that column alone.

`cex.title` Zoom factor for the title.

lines.title The amount of vertical space to be occupied by the title in lines (in multiples of itself). Defaults to 2.

padding.text The amount of space (padding) to be used above and below each row containing text, in multiples of the default, which is currently `0.2 * "lines"`. This padding is in addition to the normal height of any row that contains text, which is the minimum amount necessary to contain all the text entries.

background Background color for the legend. Defaults to the global background color.

alpha.background An alpha transparency value between 0 and 1 for the background.

border Either a color for the border, or a logical flag. In the latter case, the border color is black if `border` is `TRUE`, and no border is drawn if it is `FALSE` (the default).

transparent=FALSE Logical flag, whether legend should have a transparent background.

just A character or numeric vector of length one or two giving horizontal and vertical justification for the placement of the legend. See `grid.layout` for more precise details.

columns The number of column-blocks (drawn side by side) the legend is to be divided into.

between.columns Space between column blocks, in addition to `between`.

divide Number of point symbols to divide each line when `type` is `"b"` or `"o"` in `lines`.

legend: The legend argument can be useful if one wants to place more than one key. It also allows the use of arbitrary `"grob"`s (grid objects) as legends.

If used, `legend` must be a list, with an arbitrary number of components. Each component must be named one of `"left"`, `"right"`, `"top"`, `"bottom"`, or `"inside"`. The name `"inside"` can be repeated, but not the others. This name will be used to determine the location for that component, and is similar to the `space` component of `key`. If `key` (or `colorkey` for `levelplot` and `wireframe`) is specified, their `space` component must not conflict with the name of any component of `legend`.

Each component of `legend` must have a component called `fun`. This can be a `"grob"`, or a function (or the name of a function) that produces a `"grob"` when called. If this function expects any arguments, they must be supplied as a list in another component called `args`. For components named `"inside"`, there can be additional components called `x`, `y` and `corner`, which work in the same way as for `key`.

page: A function of one argument (page number) to be called after drawing each page. The function must be 'grid-compliant', and is called with the whole display area as the default viewport.

xlab.top, ylab.right: Labels for the x-axis on top, and y-axis on the right. Similar to `xlab` and `ylab`, but less commonly used.

main: Typically a character string or expression describing the main title to be placed on top of each page. Defaults to `NULL`.

main (as well as xlab, ylab and sub) is usually a character string or an expression that gets used as the label, but can also be a list that controls further details. Expressions are treated as specification of LaTeX-like markup as described in plotmath. The label can be a vector, in which case the components will be spaced out horizontally (or vertically for ylab). This feature can be used to provide column or row labels rather than a single axis label.

When main (etc.) is a list, the actual label should be specified as the label component (which may be unnamed if it is the first component). The label can be missing, in which case the default will be used (xlab and ylab usually have defaults, but main and sub do not). Further named arguments are passed on to textGrob; this can include arguments controlling positioning like just and rot as well as graphical parameters such as col and font (see gpar for a full list).

main, sub, xlab, ylab, xlab.top, and ylab.right can also be arbitrary "grob"s (grid graphical objects).

sub: Character string or expression (or a list or "grob") for a subtitle to be placed at the bottom of each page. See entry for main for finer control options.

par.strip.text: A list of parameters to control the appearance of strip text. Notable components are col, cex, font, and lines. The first three control graphical parameters while the last is a means of altering the height of the strips. This can be useful, for example, if the strip labels (derived from factor levels, say) are double height (i.e., contains "\n"-s) or if the default height seems too small or too large.

Additionally, the lineheight component can control the space between multiple lines. The labels can be abbreviated when shown by specifying abbreviate = TRUE, in which case the components minlength and dot (passed along to the abbreviate function) can be specified to control the details of how this is done.

layout: In general, a conditioning plot in Lattice consists of several panels arranged in a rectangular array, possibly spanning multiple pages. layout determines this arrangement.

layout is a numeric vector of length 2 or 3 giving the number of columns, rows, and pages (optional) in a multipanel display. By default, the number of columns is the number of levels of the first conditioning variable and the number of rows is the number of levels of the second conditioning variable. If there is only one conditioning variable, the default layout vector is c(0,n), where n is the number of levels of the given vector. Any time the first value in the layout vector is 0, the second value is used as the desired number of panels per page and the actual layout is computed from this, taking into account the aspect ratio of the panels and the device dimensions (via par("din")). If NA is specified for the number of rows or columns (but not both), that dimension will be filled out according to the number of panels.

The number of pages is by default set to as many as is required to plot all the panels, and so rarely needs to be specified. However, in certain situations the default calculation may be incorrect, and in that case the number of

pages needs to be specified explicitly.

skip: A logical vector (default FALSE), replicated to be as long as the number of panels (spanning all pages). For elements that are TRUE, the corresponding panel position is skipped; i.e., nothing is plotted in that position. The panel that was supposed to be drawn there is now drawn in the next available panel position, and the positions of all the subsequent panels are bumped up accordingly. This may be useful for arranging plots in an informative manner.

strip.left: strip.left can be used to draw strips on the left of each panel, which can be useful for wide short panels, as in time-series (or similar) plots. See the entry for strip for detailed usage.

xlab.default, ylab.default: Fallback default for xlab and ylab when they are not specified. If NULL, the defaults are parsed from the Trellis formula. This is rarely useful for the end-user, but can be helpful when developing new Lattice functions.

xscale.components, yscale.components: Functions that determine axis annotation for the x and y axes respectively. See documentation for xscale.components.default, the default values of these arguments, to learn more.

axis: Function responsible for drawing axis annotation. See documentation for axis.default, the default value of this argument, to learn more.

perm.cond: An integer vector, a permutation of 1:n, where n is the number of conditioning variables. By default, the order in which panels are drawn depends on the order of the conditioning variables specified in the formula. perm.cond can modify this order. If the trellis display is thought of as an n-dimensional array, then during printing, its dimensions are permuted using perm.cond as the perm argument does in aperm.

index.cond: Whereas perm.cond permutes the dimensions of the multi-dimensional array of panels, index.cond can be used to subset (or re-order) margins of that array. index.cond can be a list or a function, with behavior in each case described below.

The panel display order within each conditioning variable depends on the order of their levels. index.cond can be used to choose a 'subset' (in the R sense) of these levels, which is then used as the display order for that variable. If index.cond is a list, it has to be as long as the number of conditioning variables, and the i-th component has to be a valid indexing vector for levels(g_i), where g_i is the i-th conditioning variable in the plot (note that these levels may not contain all levels of the original variable, depending on the effects of the subset and drop.unused.levels arguments). In particular, this indexing may repeat levels, or drop some altogether. The result of this indexing determines the order of panels within that conditioning variable. To keep the order of a particular variable unchanged, the corresponding component must be set to TRUE.

Note that the components of index.cond are interpreted in the order of the conditioning variables in the original call, and is not affected by perm.cond.

Another possibility is to specify index.cond as a function. In this case, this function is called once for each panel, potentially with all arguments

that are passed to the panel function for that panel. (More specifically, if this function has a `...` argument, then all panel arguments are passed, otherwise, only named arguments that match are passed.) If there is only one conditioning variable, the levels of that variable are then sorted so that these values are in ascending order. For multiple conditioning variables, the order for each variable is determined by first taking the average over all other conditioning variables.

Although they can be supplied in high-level function calls directly, it is more typical to use `perm.cond` and `index.cond` to update an existing `"trellis"` object, thus allowing it to be displayed in a different arrangement without re-calculating the data subsets that go into each panel. In the `update.trellis` method, both can be set to `NULL`, which reverts these back to their defaults.

`par.settings`: A list that could be supplied to `trellis.par.set`. When the resulting object is plotted, these options are applied temporarily for the duration of the plotting, after which the settings revert back to what they were before. This enables the user to attach some display settings to the trellis object itself rather than change the settings globally. See also the `lattice.options` argument described above for a similar treatment of non-graphical options.

`plot.args`: A list containing possible arguments to `plot.trellis`, which will be used by the `plot` or `print` methods when drawing the object, unless overridden explicitly. This enables the user to attach such arguments to the trellis object itself. Partial matching is not performed.

Details

The high-level functions documented here, as well as other high-level Lattice functions, are generic, with the `formula` method usually doing the most substantial work. The structure of the plot that is produced is mostly controlled by the formula (implicitly in the case of the non-formula methods). For each unique combination of the levels of the conditioning variables `g1`, `g2`, `...`, a separate "packet" is produced, consisting of the points `(x,y)` for the subset of the data defined by that combination. The display can be thought of as a three-dimensional array of panels, consisting of one two-dimensional matrix per page. The dimensions of this array are determined by the `layout` argument. If there are no conditioning variables, the plot produced consists of a single packet. Each packet usually corresponds to one panel, but this is not strictly necessary (see the entry for `index.cond` above).

The coordinate system used by **lattice** by default is like a graph, with the origin at the bottom left, with axes increasing to the right and top. In particular, panels are by default drawn starting from the bottom left corner, going right and then up, unless `as.table = TRUE`, in which case panels are drawn from the top left corner, going right and then down. It is possible to set a global preference for the table-like arrangement by changing the default to `as.table=TRUE`; this can be done by setting `lattice.options(default.args = list(as.table = TRUE))`. Default values can be set in this manner for the following arguments: `as.table`, `aspect`, `between`, `page`, `main`, `sub`, `par.strip.text`, `layout`, `skip` and `strip`. Note that these global defaults are sometimes overridden by individual functions.

The order of the panels depends on the order in which the conditioning variables are specified, with `g1` varying fastest, followed by `g2`, and so on. Within a conditioning variable, the order depends

on the order of the levels (which for factors is usually in alphabetical order). Both of these orders can be modified using the `index.cond` and `perm.cond` arguments, possibly using the `update` (and other related) method(s).

Value

The high-level functions documented here, as well as other high-level Lattice functions, return an object of class `"trellis"`. The `update` method can be used to subsequently update components of the object, and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Note

Most of the arguments documented here are also applicable for the other high-level functions in the **lattice** package. These are not described in any detail elsewhere unless relevant, and this should be considered the canonical documentation for such arguments.

Any arguments passed to these functions and not recognized by them will be passed to the panel function. Most predefined panel functions have arguments that customize its output. These arguments are described only in the help pages for these panel functions, but can usually be supplied as arguments to the high-level plot.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/

See Also

`Lattice` for an overview of the package, as well as `barchart.table`, `print.trellis`, `shingle`, `banking`, `reshape`, `panel.xyplot`, `panel.bwplot`, `panel.barchart`, `panel.dotplot`, `panel.stripplot`, `panel.superpose`, `panel.loess`, `panel.average`, `strip.default`, `simpleKey` `trellis.par.set`

Examples

```
require(stats)

## Tonga Trench Earthquakes

Depth <- equal.count(quakes$depth, number=8, overlap=.1)
xyplot(lat ~ long | Depth, data = quakes)
update(trellis.last.object(),
       strip = strip.custom(strip.names = TRUE, strip.levels = TRUE),
       par.strip.text = list(cex = 0.75),
       aspect = "iso")
```

```
## Examples with data from `Visualizing Data' (Cleveland, 1993) obtained
## from http://cm.bell-labs.com/cm/ms/departments/sia/wsc/

EE <- equal.count(ethanol$E, number=9, overlap=1/4)

## Constructing panel functions on the fly; prepanel
xyplot(NOx ~ C | EE, data = ethanol,
       prepanel = function(x, y) prepanel.loess(x, y, span = 1),
       xlab = "Compression Ratio", ylab = "NOx (micrograms/J)",
       panel = function(x, y) {
           panel.grid(h = -1, v = 2)
           panel.xyplot(x, y)
           panel.loess(x, y, span=1)
       },
       aspect = "xy")

## Extended formula interface

xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
       data = iris, scales = "free", layout = c(2, 2),
       auto.key = list(x = .6, y = .7, corner = c(0, 0)))


## user defined panel functions

states <- data.frame(state.x77,
                     state.name = dimnames(state.x77)[[1]],
                     state.region = state.region)
xyplot(Murder ~ Population | state.region, data = states,
       groups = state.name,
       panel = function(x, y, subscripts, groups) {
           ltext(x = x, y = y, labels = groups[subscripts], cex=1,
                 fontfamily = "HersheySans")
       })

## Stacked bar chart

barchart(yield ~ variety | site, data = barley,
         groups = year, layout = c(1,6), stack = TRUE,
         auto.key = list(space = "right"),
         ylab = "Barley Yield (bushels/acre)",
         scales = list(x = list(rot = 45)))

bwplot(voice.part ~ height, data=singer, xlab="Height (inches)")

dotplot(variety ~ yield | year * site, data=barley)

## Grouped dot plot showing anomaly at Morris

dotplot(variety ~ yield | site, data = barley, groups = year,
        key = simpleKey(levels(barley$year), space = "right"),
        xlab = "Barley Yield (bushels/acre) ",
        aspect=0.5, layout = c(1,6), ylab=NULL)
```

```
stripplot(voice.part ~ jitter(height), data = singer, aspect = 1,
          jitter.data = TRUE, xlab = "Height (inches)")

## Interaction Plot

xyplot(decrease ~ treatment, OrchardSprays, groups = rowpos,
       type = "a",
       auto.key =
       list(space = "right", points = FALSE, lines = TRUE))

## longer version with no x-ticks

## Not run:
bwplot(decrease ~ treatment, OrchardSprays, groups = rowpos,
       panel = "panel.superpose",
       panel.groups = "panel.linejoin",
       xlab = "treatment",
       key = list(lines = Rows(trellis.par.get("superpose.line"),
                  c(1:7, 1)),
                  text = list(lab = as.character(unique(OrchardSprays$rowpos))),
                  columns = 4, title = "Row position"))

## End(Not run)
```

B_01_xyplot.ts	*Time series plotting methods*

Description

This function handles time series plotting, including cut-and-stack plots. Examples are given of
superposing, juxtaposing and styling different time series.

Usage

```
## S3 method for class 'ts'
xyplot(x, data = NULL,
       screens = if (superpose) 1 else colnames(x),
       ...,
       superpose = FALSE,
       cut = FALSE,
       type = "l",
       col = NULL,
       lty = NULL,
       lwd = NULL,
       pch = NULL,
       cex = NULL,
       fill = NULL,
```

```
        auto.key = superpose,
        panel = if (superpose) "panel.superpose"
                else "panel.superpose.plain",
        par.settings = list(),
        layout = NULL, as.table = TRUE,
        xlab = "Time", ylab = NULL,
        default.scales = list(y = list(relation =
            if (missing(cut)) "free" else "same")))
```

Arguments

x	an object of class `ts`, which may be multi-variate, i.e. have a matrix structure with multiple columns.
data	not used, and must be left as NULL.
...	additional arguments passed to `xyplot`, which may pass them on to `panel.xyplot`.
screens	factor (or coerced to factor) whose levels specify which panel each series is to be plotted in. `screens = c(1, 2, 1)` would plot series 1, 2 and 3 in panels 1, 2 and 1. May also be a named list, see Details below.
superpose	overlays all series in one panel (via `screens = 1`) and uses grouped style settings (from `trellis.par.get("superpose.line")`, etc). Note that this is just a convenience argument: its only action is to change the default values of other arguments.
cut	defines a cut-and-stack plot. `cut` can be a `list` of arguments to the function `equal.count`, i.e. `number` (number of intervals to divide into) and `overlap` (the fraction of overlap between cuts, default 0.5). If `cut` is numeric this is passed as the `number` argument.
	`cut = TRUE` tries to choose an appropriate number of cuts (up to a maximum of 6), using `banking`, and assuming a square plot region. This should have the effect of minimising wasted space when `aspect = "xy"`.
type, col, lty, lwd, pch, cex, fill	
	graphical arguments, which are processed and eventually passed to `panel.xyplot`. These arguments can also be vectors or (named) lists, see Details for more information.
auto.key	a logical, or a list describing how to draw a key. See the `auto.key` entry in `xyplot`. The default here is to draw lines, not points, and any specified style arguments should show up automatically.
panel	the panel function. It is recommended to leave this alone, but one can pass a `panel.groups` argument which is handled by `panel.superpose` for each series.
par.settings	style settings beyond the standard col, lty, lwd, etc; see `trellis.par.set` and `simpleTheme`.
layout	numeric vector of length 2 specifying number of columns and rows in the plot. The default is to fill columns with up to 6 rows.
as.table	to draw panels from top to bottom. The order is determined by the order of columns in x.

`xlab, ylab`	X axis and Y axis labels; see `xyplot`. Note in particular that `ylab` may be a character vector, in which case the labels are spaced out equally, to correspond to the panels; but *NOTE* in this case the vector should be reversed OR the argument `as.table` set to `FALSE`.
`default.scales`	`scales` specification. The default is set to have `"free"` Y axis scales unless `cut` is given. Note, users should pass the `scales` argument rather than `default.scales`.

Details

The handling of several graphical parameters is more flexible for multivariate series. These parameters can be vectors of the same length as the number of series plotted or are recycled if shorter. They can also be (partially) named list, e.g., `list(A = c(1,2), c(3,4))` in which `c(3, 4)` is the default value and `c(1, 2)` the value only for series `A`. The `screens` argument can be specified in a similar way.

Some examples are given below.

Value

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

Author(s)

Gabor Grothendieck, Achim Zeileis, Deepayan Sarkar and Felix Andrews <felix@nfrac.org>.

The first two authors developed `xyplot.ts` in their **zoo** package, including the `screens` approach. The third author developed a different `xyplot.ts` for cut-and-stack plots in the **lattice-Extra** package. The final author fused these together.

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/ (cut-and-stack plots)

See Also

`xyplot`, `panel.xyplot`, `plot.ts`, `ts`, `xyplot.zoo` in the **zoo** package.

Examples

```
xyplot(ts(c(1:10,10:1)))

### Figure 14.1 from Sarkar (2008)
xyplot(sunspot.year, aspect = "xy",
       strip = FALSE, strip.left = TRUE,
       cut = list(number = 4, overlap = 0.05))

### A multivariate example; first juxtaposed, then superposed
```

```
xyplot(EuStockMarkets, scales = list(y = "same"))
xyplot(EuStockMarkets, superpose = TRUE, aspect = "xy", lwd = 2,
    type = c("l","g"), ylim = c(0, max(EuStockMarkets)))

### Examples using screens (these two are identical)
xyplot(EuStockMarkets, screens = c(rep("Continental", 3), "UK"))
xyplot(EuStockMarkets, screens = list(FTSE = "UK", "Continental"))

### Automatic group styles
xyplot(EuStockMarkets, screens = list(FTSE = "UK", "Continental"),
    superpose = TRUE)

xyplot(EuStockMarkets, screens = list(FTSE = "UK", "Continental"),
    superpose = TRUE, xlim = extendrange(1996:1998),
    par.settings = standard.theme(color = FALSE))

### Specifying styles for series by name
xyplot(EuStockMarkets, screens = list(FTSE = "UK", "Continental"),
    col = list(DAX = "red", FTSE = "blue", "black"), auto.key = TRUE)

xyplot(EuStockMarkets, screens = list(FTSE = "UK", "Continental"),
    col = list(DAX = "red"), lty = list(SMI = 2), lwd = 1:2,
    auto.key = TRUE)

### Example with simpler data, few data points
set.seed(1)
z <- ts(cbind(a = 1:5, b = 11:15, c = 21:25) + rnorm(5))
xyplot(z, screens = 1)
xyplot(z, screens = list(a = "primary (a)", "other (b & c)"),
  type = list(a = c("p", "h"), b = c("p", "s"), "o"),
  pch = list(a = 2, c = 3), auto.key = list(type = "o"))
```

B_02_barchart.table

table methods for barchart and dotplot

Description

Contingency tables are often displayed using bar charts and dot plots. These methods operate directly on tables, bypassing the need to convert them to data frames for use with the formula interface. Matrices and arrays are also supported, by coercing them to tables.

Usage

```
## S3 method for class 'table'
barchart(x, data, groups = TRUE,
         origin = 0, stack = TRUE, ..., horizontal = TRUE)

## S3 method for class 'array'
```

```
barchart(x, data, ...)

## S3 method for class 'matrix'
barchart(x, data, ...)

## S3 method for class 'table'
dotplot(x, data, groups = TRUE, ..., horizontal = TRUE)

## S3 method for class 'array'
dotplot(x, data, ...)

## S3 method for class 'matrix'
dotplot(x, data, ...)
```

Arguments

x	A `table`, `array` or `matrix` object.
data	Should not be specified. If specified, will be ignored with a warning.
groups	A logical flag, indicating whether to use the last dimension as a grouping variable in the display.
origin, stack	
	Arguments to `panel.barchart`. The defaults for the `table` method are different.
horizontal	Logical flag, indicating whether the plot should be horizontal (with the categorical variable on the y-axis) or vertical.
...	Other arguments, passed to the underlying `formula` method.

Details

The first dimension is used as the variable on the categorical axis. The last dimension is optionally used as a grouping variable (to produce stacked barcharts by default). All other dimensions are used as conditioning variables. The order of these variables cannot be altered (except by permuting the original argument beforehand using `t` or `aperm`). For more flexibility, use the formula method after converting the table to a data frame using the relevant `as.data.frame` method.

Value

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

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`barchart`, `t`, `aperm`, `table`, `panel.barchart`, `Lattice`

Examples

```
barchart(Titanic, scales = list(x = "free"),
         auto.key = list(title = "Survived"))
```

B_03_histogram   *Histograms and Kernel Density Plots*

Description

Draw Histograms and Kernel Density Plots, possibly conditioned on other variables.

Usage

```
histogram(x, data, ...)
densityplot(x, data, ...)
## S3 method for class 'formula'
histogram(x,
          data,
          allow.multiple, outer = TRUE,
          auto.key = FALSE,
          aspect = "fill",
          panel = lattice.getOption("panel.histogram"),
          prepanel, scales, strip, groups,
          xlab, xlim, ylab, ylim,
          type = c("percent", "count", "density"),
          nint = if (is.factor(x)) nlevels(x)
          else round(log2(length(x)) + 1),
          endpoints = extend.limits(range(as.numeric(x),
                        finite = TRUE), prop = 0.04),
          breaks,
          equal.widths = TRUE,
          drop.unused.levels =
              lattice.getOption("drop.unused.levels"),
          ...,
          lattice.options = NULL,
          default.scales = list(),
          default.prepanel =
              lattice.getOption("prepanel.default.histogram"),
          subscripts,
          subset)

## S3 method for class 'numeric'
histogram(x, data = NULL, xlab, ...)
## S3 method for class 'factor'
histogram(x, data = NULL, xlab, ...)

## S3 method for class 'formula'
```

```
densityplot(x,
            data,
            allow.multiple = is.null(groups) || outer,
            outer = !is.null(groups),
            auto.key = FALSE,
            aspect = "fill",
            panel = lattice.getOption("panel.densityplot"),
            prepanel, scales, strip, groups, weights,
            xlab, xlim, ylab, ylim,
            bw, adjust, kernel, window, width, give.Rkern,
            n = 50, from, to, cut, na.rm,
            drop.unused.levels =
                lattice.getOption("drop.unused.levels"),
            ...,
            lattice.options = NULL,
            default.scales = list(),
            default.prepanel =
                lattice.getOption("prepanel.default.densityplot"),
            subscripts,
            subset)
## S3 method for class 'numeric'
densityplot(x, data = NULL, xlab, ...)

do.breaks(endpoints, nint)
```

Arguments

x	The object on which method dispatch is carried out.
	For the `formula` method, x can be a formula of the form `~ x	g1 * g2 * ...`, indicating that histograms or kernel density estimates of the x variable should be produced conditioned on the levels of the (optional) variables g1, g2, .... x should be numeric (or possibly a factor in the case of `histogram`), and each of g1, g2, ... should be either factors or shingles.
	As a special case, the right hand side of the formula can contain more than one term separated by '+' signs (e.g., `~ x1 + x2 | g1 * g2`). What happens in this case is described in the documentation for [xyplot](). Note that in either form, all the terms in the formula must have the same length after evaluation.
	For the `numeric` and `factor` methods, x is the variable whose histogram or Kernel density estimate is drawn. Conditioning is not allowed in these cases.
data	For the `formula` method, an optional data source (usually a data frame) in which variables are to be evaluated (see [xyplot]() for details). `data` should not be specified for the other methods, and is ignored with a warning if it is.
type	A character string indicating the type of histogram that is to be drawn. `"percent"` and `"count"` give relative frequency and frequency histograms respectively, and can be misleading when breakpoints are not equally spaced. `"density"` produces a density histogram.
	`type` defaults to `"density"` when the breakpoints are unequally spaced or `breaks = NULL`, and to `"percent"` otherwise.

nint   An integer specifying the number of histogram bins, applicable only when `breaks`
                   is unspecified or `NULL` in the call. Ignored when the variable being plotted is a
                   factor.

endpoints   A numeric vector of length 2 indicating the range of x-values that is to be cov-
                   ered by the histogram. This applies only when `breaks` is unspecified and the
                   variable being plotted is not a factor. In `do.breaks`, this specifies the interval
                   that is to be divided up.

breaks   Usually a numeric vector of length (number of bins + 1) defining the breakpoints
                   of the bins. Note that when breakpoints are not equally spaced, the only value
                   of `type` that makes sense is density. When unspecified, the default is to use

                            breaks = seq_len(1 + nlevels(x)) - 0.5

                   when `x` is a factor, and

                            breaks = do.breaks(endpoints, nint)

                   otherwise. Breakpoints calculated in such a manner are used in all panels.

                   Other values of `breaks` are possible, in which case they affect the display in
                   each panel differently. A special value of `breaks` is `NULL`, in which case
                   the number of bins is determined by `nint` and then breakpoints are chosen
                   according to the value of `equal.widths`. Other valid values of `breaks` are
                   those of the `breaks` argument in [hist]. This allows specification of `breaks`
                   as an integer giving the number of bins (similar to `nint`), as a character string
                   denoting a method, or as a function.

equal.widths   A logical flag, relevant only when `breaks=NULL`. If `TRUE`, equally spaced
                   bins will be selected, otherwise, approximately equal area bins will be selected
                   (typically producing unequally spaced breakpoints).

n   Integer, giving the number of points at which the kernel density is to be evalu-
                   ated. Passed on as an argument to [density].

panel   A function, called once for each panel, that uses the packet (subset of panel
                   variables) corresponding to the panel to create a display. The default panel
                   functions [panel.histogram] and [panel.densityplot] are documented
                   separately, and have arguments that can be used to customize its output in var-
                   ious ways. Such arguments can usually be directly supplied to the high-level
                   function.

allow.multiple, outer
                   See [xyplot].

auto.key   See [xyplot].

aspect   See [xyplot].

prepanel   See [xyplot].

scales   See [xyplot].

strip   See [xyplot].

groups   See [xyplot]. Note that the default panel function for `histogram` does not
                   support grouped displays, whereas the one for `densityplot` does.

xlab, ylab   See [xyplot](.).

xlim, ylim   See [xyplot](.).

drop.unused.levels

               See [xyplot](.).

lattice.options

               See [xyplot](.).

default.scales

               See [xyplot](.).

subscripts   See [xyplot](.).

subset   See [xyplot](.).

default.prepanel

               Fallback prepanel function. See [xyplot](.).

weights   numeric vector of weights for the density calculations, evaluated in the non-standard manner used for groups and terms in the formula, if any. If this is specified, it is subsetted using subscripts inside the panel function to match it to the corresponding x values.

               At the time of writing, weights do not work in conjunction with an extended formula specification (this is not too hard to fix, so just bug the maintainer if you need this feature).

bw, adjust, width

               Arguments controlling bandwidth. Passed on as arguments to [density](.).

kernel, window

               The choice of kernel. Passed on as arguments to [density](.).

give.Rkern   Logical flag, passed on as argument to [density](.). This argument is made available only for ease of implementation, and will produce an error if TRUE.

from, to, cut

               Controls range over which density is evaluated. Passed on as arguments to [density](.).

na.rm   Logical flag specifying whether NA values should be ignored. Passed on as argument to [density](.), but unlike in density, the default is TRUE.

...   Further arguments. See corresponding entry in [xyplot](.) for non-trivial details.

Details

histogram draws Conditional Histograms, and densityplot draws Conditional Kernel Density Plots. The default panel function uses the [density](.) function to compute the density estimate, and all arguments accepted by density can be specified in the call to densityplot to control the output. See documentation of density for details. Note that the default value of the argument n of density is changed to 50.

These and all other high level Trellis functions have several arguments in common. These are extensively documented only in the help page for xyplot, which should be consulted to learn more detailed usage.

do.breaks is an utility function that calculates breakpoints given an interval and the number of pieces to break it into.

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

The form of the arguments accepted by the default panel function panel.histogram is different from that in S-PLUS. Whereas S-PLUS calculates the heights inside histogram and passes only the breakpoints and the heights to the panel function, **lattice** simply passes along the original variable x along with the breakpoints. This approach is more flexible; see the example below with an estimated density superimposed over the histogram.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/

See Also

xyplot, panel.histogram, density, panel.densityplot, panel.mathdensity, Lattice

Examples

```
require(stats)
histogram( ~ height | voice.part, data = singer, nint = 17,
          endpoints = c(59.5, 76.5), layout = c(2,4), aspect = 1,
          xlab = "Height (inches)")

histogram( ~ height | voice.part, data = singer,
          xlab = "Height (inches)", type = "density",
          panel = function(x, ...) {
              panel.histogram(x, ...)
              panel.mathdensity(dmath = dnorm, col = "black",
                                args = list(mean=mean(x),sd=sd(x)))
          } )

densityplot( ~ height | voice.part, data = singer, layout = c(2, 4),
            xlab = "Height (inches)", bw = 5)
```

`B_04_qqmath`	*Q-Q Plot with Theoretical Distribution*

Description

Draw quantile-Quantile plots of a sample against a theoretical distribution, possibly conditioned on other variables.

Usage

```
qqmath(x, data, ...)

## S3 method for class 'formula'
qqmath(x,
       data,
       allow.multiple = is.null(groups) || outer,
       outer = !is.null(groups),
       distribution = qnorm,
       f.value = NULL,
       auto.key = FALSE,
       aspect = "fill",
       panel = lattice.getOption("panel.qqmath"),
       prepanel = NULL,
       scales, strip, groups,
       xlab, xlim, ylab, ylim,
       drop.unused.levels = lattice.getOption("drop.unused.levels"),
       ...,
       lattice.options = NULL,
       default.scales = list(),
       default.prepanel = lattice.getOption("prepanel.default.qqmath"),
       subscripts,
       subset)
## S3 method for class 'numeric'
qqmath(x, data = NULL, ylab, ...)
```

Arguments

x	The object on which method dispatch is carried out.
	For the `"formula"` method, x should be a formula of the form `~ x	g1 * g2 * ...`, where x should be a numeric variable. For the `"numeric"` method, x should be a numeric vector.
data	For the `formula` method, an optional data source (usually a data frame) in which variables are to be evaluated (see [xyplot](#) for details). data should not be specified for the other methods, and is ignored with a warning if it is.
distribution	A quantile function that takes a vector of probabilities as argument and produces the corresponding quantiles from a theoretical distribution. Possible values are

qnorm, qunif, etc. Distributions with other required arguments need to be provided as user-defined functions (see example with qt).

f.value   An optional numeric vector of probabilities, quantiles corresponding to which should be plotted. This can also be a function of a single integer (representing sample size) that returns such a numeric vector. A typical value for this argument is the function ppoints, which is also the S-PLUS default. If specified, the probabilities generated by this function is used for the plotted quantiles, through the quantile function for the sample, and the function specified as the distribution argument for the theoretical distribution.

                 f.value defaults to NULL, which has the effect of using ppoints for the quantiles of the theoretical distribution, but the exact data values for the sample. This is similar to what happens for qqnorm, but different from the S-PLUS default of f.value=ppoints.

                 For large x, this argument can be used to restrict the number of points plotted. See also the tails.n argument in panel.qqmath.

panel   A function, called once for each panel, that uses the packet (subset of panel variables) corresponding to the panel to create a display. The default panel function panel.qqmath is documented separately, and has arguments that can be used to customize its output in various ways. Such arguments can usually be directly supplied to the high-level function.

allow.multiple, outer
                 See xyplot.

auto.key   See xyplot.

aspect   See xyplot.

prepanel   See xyplot.

scales   See xyplot.

strip   See xyplot.

groups   See xyplot.

xlab, ylab   See xyplot.

xlim, ylim   See xyplot.

drop.unused.levels
                 See xyplot.

lattice.options
                 See xyplot.

default.scales
                 See xyplot.

subscripts   See xyplot.

subset   See xyplot.

default.prepanel
                 Fallback prepanel function. See xyplot.

...   Further arguments. See corresponding entry in xyplot for non-trivial details.

Details

qqmath produces Q-Q plots of the given sample against a theoretical distribution. The default behaviour of qqmath is different from the corresponding S-PLUS function, but is similar to qqnorm. See the entry for f.value for specifics.

The implementation details are also different from S-PLUS. In particular, all the important calculations are done by the panel (and prepanel function) and not qqmath itself. In fact, both the arguments distribution and f.value are passed unchanged to the panel and prepanel function. This allows, among other things, display of grouped Q-Q plots, which are often useful. See the help page for panel.qqmath for further details.

This and all other high level Trellis functions have several arguments in common. These are extensively documented only in the help page for xyplot, which should be consulted to learn more detailed usage.

Value

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

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

xyplot, panel.qqmath, panel.qqmathline, prepanel.qqmathline, Lattice, quantile

Examples

```
qqmath(~ rnorm(100), distribution = function(p) qt(p, df = 10))
qqmath(~ height | voice.part, aspect = "xy", data = singer,
       prepanel = prepanel.qqmathline,
       panel = function(x, ...) {
           panel.qqmathline(x, ...)
           panel.qqmath(x, ...)
       })
vp.comb <-
    factor(sapply(strsplit(as.character(singer$voice.part), split = " "),
                  "[", 1),
           levels = c("Bass", "Tenor", "Alto", "Soprano"))
vp.group <-
    factor(sapply(strsplit(as.character(singer$voice.part), split = " "),
                  "[", 2))
qqmath(~ height | vp.comb, data = singer,
       groups = vp.group, auto.key = list(space = "right"),
       aspect = "xy",
       prepanel = prepanel.qqmathline,
       panel = function(x, ...) {
           panel.qqmathline(x, ...)
           panel.qqmath(x, ...)
```

```
      })
```

Description

Quantile-Quantile plots for comparing two Distributions

Usage

```
qq(x, data, ...)

## S3 method for class 'formula'
qq(x, data, aspect = "fill",
   panel = lattice.getOption("panel.qq"),
   prepanel, scales, strip,
   groups, xlab, xlim, ylab, ylim, f.value = NULL,
   drop.unused.levels = lattice.getOption("drop.unused.levels"),
   ...,
   lattice.options = NULL,
   qtype = 7,
   default.scales = list(),
   default.prepanel = lattice.getOption("prepanel.default.qq"),
   subscripts,
   subset)
```

Arguments

x	The object on which method dispatch is carried out.
	For the `"formula"` method, x should be a formula of the form `y ~ x | g1 * g2 * ...`, where x should be a numeric variable, and y a factor, shingle, character, or numeric variable, with the restriction that there must be exactly two levels of y, which divide the values of x into two groups. Quantiles for these groups will be plotted against each other along the two axes.
data	For the `formula` method, an optional data source (usually a data frame) in which variables are to be evaluated (see [xyplot](#) for details).
f.value	An optional numeric vector of probabilities, quantiles corresponding to which should be plotted. This can also be a function of a single integer (representing sample size) that returns such a numeric vector. A typical value for this argument is the function ppoints, which is also the S-PLUS default. If specified, the probabilities generated by this function is used for the plotted quantiles, through the quantile function.
	f.value defaults to NULL, which is equivalent to
	`f.value = function(n) ppoints(n, a = 1)`

This has the effect of including the minimum and maximum data values in the computed quantiles. This is similar to what happens for qqplot but different from the default behaviour of qq in S-PLUS.

For large x, this argument can be used to restrict the number of quantiles plotted.

panel   A function, called once for each panel, that uses the packet (subset of panel variables) corresponding to the panel to create a display. The default panel function panel.qq is documented separately, and has arguments that can be used to customize its output in various ways. Such arguments can usually be directly supplied to the high-level function.

qtype   The type argument for quantile.

aspect   See xyplot.

prepanel   See xyplot.

scales   See xyplot.

strip   See xyplot.

groups   See xyplot.

xlab, ylab   See xyplot.

xlim, ylim   See xyplot.
drop.unused.levels
             See xyplot.
lattice.options
             See xyplot.
default.scales
             See xyplot.

subscripts   See xyplot.

subset   See xyplot.
default.prepanel
             Fallback prepanel function. See xyplot.

...   Further arguments. See corresponding entry in xyplot for non-trivial details.

Details

qq produces Q-Q plots of two samples. The default behaviour of qq is different from the corresponding S-PLUS function. See the entry for f.value for specifics.

This and all other high level Trellis functions have several arguments in common. These are extensively documented only in the help page for xyplot, which should be consulted to learn more detailed usage.

Value

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

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

[xyplot](), [panel.qq](), [qqmath](), [Lattice]()

Examples

```
qq(voice.part ~ height, aspect = 1, data = singer,
   subset = (voice.part == "Bass 2" | voice.part == "Tenor 1"))
```

B_06_levelplot   *Level plots and contour plots*

Description

Draw Level Plots and Contour plots.

Usage

```
levelplot(x, data, ...)
contourplot(x, data, ...)

## S3 method for class 'formula'
levelplot(x,
          data,
          allow.multiple = is.null(groups) || outer,
          outer = TRUE,
          aspect = "fill",
          panel = lattice.getOption("panel.levelplot"),
          prepanel = NULL,
          scales = list(),
          strip = TRUE,
          groups = NULL,
          xlab,
          xlim,
          ylab,
          ylim,
          at,
          cuts = 15,
          pretty = FALSE,
          region = TRUE,
          drop.unused.levels =
              lattice.getOption("drop.unused.levels"),
          ...,
          lattice.options = NULL,
```

```
             default.scales = list(),
             default.prepanel =
                 lattice.getOption("prepanel.default.levelplot"),
             colorkey = region,
             col.regions,
             alpha.regions,
             subset = TRUE)

## S3 method for class 'formula'
contourplot(x,
             data,
             panel = lattice.getOption("panel.contourplot"),
             default.prepanel =
                 lattice.getOption("prepanel.default.contourplot"),
             cuts = 7,
             labels = TRUE,
             contour = TRUE,
             pretty = TRUE,
             region = FALSE,
             ...)

## S3 method for class 'table'
levelplot(x, data = NULL, aspect = "iso", ..., xlim, ylim)

## S3 method for class 'table'
contourplot(x, data = NULL, aspect = "iso", ..., xlim, ylim)

## S3 method for class 'matrix'
levelplot(x, data = NULL, aspect = "iso",
             ..., xlim, ylim,
             row.values = seq_len(nrow(x)),
             column.values = seq_len(ncol(x)))

## S3 method for class 'matrix'
contourplot(x, data = NULL, aspect = "iso",
             ..., xlim, ylim,
             row.values = seq_len(nrow(x)),
             column.values = seq_len(ncol(x)))


## S3 method for class 'array'
levelplot(x, data = NULL, ...)

## S3 method for class 'array'
contourplot(x, data = NULL, ...)
```

Arguments

x for the `formula` method, a formula of the form `z ~ x * y | g1 * g2 * ...`, where `z` is a numeric response, and `x`, `y` are numeric values evaluated on a rectangular grid. `g1, g2, ...` are optional conditional variables, and must be either factors or shingles if present.

Calculations are based on the assumption that all x and y values are evaluated on a grid (defined by their unique values). The function will not return an error if this is not true, but the display might not be meaningful. However, the x and y values need not be equally spaced.

Both `levelplot` and `wireframe` have methods for `matrix`, `array`, and `table` objects, in which case `x` provides the z vector described above, while its rows and columns are interpreted as the x and y vectors respectively. This is similar to the form used in `filled.contour` and `image`. For higher-dimensional arrays and tables, further dimensions are used as conditioning variables. Note that the dimnames may be duplicated; this is handled by calling [make.unique](#) to make the names unique (although the original labels are used for the x- and y-axes).

data For the `formula` methods, an optional data frame in which variables in the formula (as well as `groups` and `subset`, if any) are to be evaluated. Usually ignored with a warning in other cases.

row.values, column.values

Optional vectors of values that define the grid when x is a matrix. `row.values` and `column.values` must have the same lengths as `nrow(x)` and `ncol(x)` respectively. By default, row and column numbers.

panel panel function used to create the display, as described in [xyplot](#)

aspect For the `matrix` methods, the default aspect ratio is chosen to make each cell square. The usual default is `aspect="fill"`, as described in [xyplot](#).

at numeric vector giving breakpoints along the range of z. Contours (if any) will be drawn at these heights, and the regions in between would be colored using `col.regions`. In the latter case, values outside the range of `at` will not be drawn at all. This serves as a way to limit the range of the data shown, similar to what a `zlim` argument might have been used for. However, this also means that when supplying `at` explicitly, one has to be careful to include values outside the range of z to ensure that all the data are shown.

col.regions color vector to be used if regions is TRUE. The general idea is that this should be a color vector of moderately large length (longer than the number of regions. By default this is 100). It is expected that this vector would be gradually varying in color (so that nearby colors would be similar). When the colors are actually chosen, they are chosen to be equally spaced along this vector. When there are more regions than colors in `col.regions`, the colors are recycled. The actual color assignment is performed by [level.colors](#), which is documented separately.

alpha.regions

numeric, specifying alpha transparency (works only on some devices)

colorkey logical specifying whether a color key is to be drawn alongside the plot, or a list describing the color key. The list may contain the following components:

> space: location of the colorkey, can be one of "left", "right", "top"
> and "bottom". Defaults to "right".
>
> x, y: location, currently unused
>
> col: A color ramp specification, as in the col.regions argument in level.colors
>
> at: numeric vector specifying where the colors change. must be of length 1
> more than the col vector.
>
> labels: a character vector for labelling the at values, or more commonly, a
> list describing characteristics of the labels. This list may include compo-
> nents labels, at, cex, col, rot, font, fontface and fontfamily.
>
> tick.number: approximate number of ticks.
>
> corner: interacts with x, y; unimplemented
>
> width: width of the key
>
> height: length of key w.r.t side of plot.
>
> raster: Logical flag indicating whether the colorkey should be rendered as a
> raster image using grid.raster. See also panel.levelplot.raster.
>
> interpolate: Logical flag, passed to rasterGrob when raster=TRUE.

contour   logical, whether to draw contour lines.

cuts   number of levels the range of z would be divided into

labels   typically a logical indicating whether contour lines should be labelled, but other
                   possibilities for more sophisticated control exists. Details are documented in the
                   help page for panel.levelplot, to which this argument is passed on un-
                   changed. That help page also documents the label.style argument, which
                   affects how the labels are rendered.

pretty   logical, whether to use pretty cut locations and labels

region   logical, whether regions between contour lines should be filled

allow.multiple, outer, prepanel, scales, strip, groups, xlab, xlim, ylab, ylim, dro
                   these arguments are described in the help page for xyplot.

default.prepanel
                   Fallback prepanel function. See xyplot.

...   other arguments. Some are processed by levelplot or contourplot, and
                   those unrecognized are passed on to the panel function.

Details

These and all other high level Trellis functions have several arguments in common. These are
extensively documented only in the help page for xyplot, which should be consulted to learn
more detailed usage.

Other useful arguments are mentioned in the help page for the default panel function panel.levelplot
(these are formally arguments to the panel function, but can be specified in the high level calls di-
rectly).

Value

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

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. `http://lmdvr.r-forge.r-project.org/`

See Also

`xyplot`, `Lattice`, `panel.levelplot`

Examples

```
x <- seq(pi/4, 5 * pi, length.out = 100)
y <- seq(pi/4, 5 * pi, length.out = 100)
r <- as.vector(sqrt(outer(x^2, y^2, "+")))
grid <- expand.grid(x=x, y=y)
grid$z <- cos(r^2) * exp(-r/(pi^3))
levelplot(z~x*y, grid, cuts = 50, scales=list(log="e"), xlab="",
          ylab="", main="Weird Function", sub="with log scales",
          colorkey = FALSE, region = TRUE)


#S-PLUS example
require(stats)
attach(environmental)
ozo.m <- loess((ozone^(1/3)) ~ wind * temperature * radiation,
       parametric = c("radiation", "wind"), span = 1, degree = 2)
w.marginal <- seq(min(wind), max(wind), length.out = 50)
t.marginal <- seq(min(temperature), max(temperature), length.out = 50)
r.marginal <- seq(min(radiation), max(radiation), length.out = 4)
wtr.marginal <- list(wind = w.marginal, temperature = t.marginal,
        radiation = r.marginal)
grid <- expand.grid(wtr.marginal)
grid[, "fit"] <- c(predict(ozo.m, grid))
contourplot(fit ~ wind * temperature | radiation, data = grid,
            cuts = 10, region = TRUE,
            xlab = "Wind Speed (mph)",
            ylab = "Temperature (F)",
            main = "Cube Root Ozone (cube root ppb)")
detach()
```

B_07_cloud *3d Scatter Plot and Wireframe Surface Plot*

Description

Generic functions to draw 3d scatter plots and surfaces. The `"formula"` methods do most of the actual work.

Usage

```
cloud(x, data, ...)
wireframe(x, data, ...)

## S3 method for class 'formula'
cloud(x,
      data,
      allow.multiple = is.null(groups) || outer,
      outer = FALSE,
      auto.key = FALSE,
      aspect = c(1,1),
      panel.aspect = 1,
      panel = lattice.getOption("panel.cloud"),
      prepanel = NULL,
      scales = list(),
      strip = TRUE,
      groups = NULL,
      xlab,
      ylab,
      zlab,
      xlim = if (is.factor(x)) levels(x) else range(x, finite = TRUE),
      ylim = if (is.factor(y)) levels(y) else range(y, finite = TRUE),
      zlim = if (is.factor(z)) levels(z) else range(z, finite = TRUE),
      at,
      drape = FALSE,
      pretty = FALSE,
      drop.unused.levels,
      ...,
      lattice.options = NULL,
      default.scales =
      list(distance = c(1, 1, 1),
           arrows = TRUE,
           axs = axs.default),
      default.prepanel = lattice.getOption("prepanel.default.cloud"),
      colorkey,
      col.regions,
      alpha.regions,
      cuts = 70,
      subset = TRUE,
      axs.default = "r")

## S3 method for class 'formula'
wireframe(x,
          data,
          panel = lattice.getOption("panel.wireframe"),
          default.prepanel = lattice.getOption("prepanel.default.wireframe"),
          ...)
```

```
## S3 method for class 'matrix'
cloud(x, data = NULL, type = "h",
      zlab = deparse(substitute(x)), aspect, ...,
      xlim, ylim, row.values, column.values)

## S3 method for class 'table'
cloud(x, data = NULL, groups = FALSE,
      zlab = deparse(substitute(x)),
      type = "h", ...)

## S3 method for class 'matrix'
wireframe(x, data = NULL,
          zlab = deparse(substitute(x)), aspect, ...,
          xlim, ylim, row.values, column.values)
```

Arguments

x
: The object on which method dispatch is carried out.

  For the `"formula"` methods, a formula of the form `z ~ x * y | g1 *
  g2 * ...`, where `z` is a numeric response, and `x`, `y` are numeric values. `g1`,
  `g2`, `...`, if present, are conditioning variables used for conditioning, and must
  be either factors or shingles. In the case of `wireframe`, calculations are based
  on the assumption that the `x` and `y` values are evaluated on a rectangular grid
  defined by their unique values. The grid points need not be equally spaced.

  For `wireframe`, `x`, `y` and `z` may also be matrices (of the same dimension),
  in which case they are taken to represent a 3-D surface parametrized on a 2-D
  grid (e.g., a sphere). Conditioning is not possible with this feature. See details
  below.

  Missing values are allowed, either as `NA` values in the `z` vector, or missing rows
  in the data frame (note however that in that case the X and Y grids will be
  determined only by the available values). For a grouped display (producing
  multiple surfaces), missing rows are not allowed, but `NA`-s in `z` are.

  Both `wireframe` and `cloud` have methods for `matrix` objects, in which
  case `x` provides the `z` vector described above, while its rows and columns are
  interpreted as the `x` and `y` vectors respectively. This is similar to the form used
  in `persp`.

data
: for the `"formula"` methods, an optional data frame in which variables in the
  formula (as well as `groups` and `subset`, if any) are to be evaluated. `data`
  should not be specified except when using the `"formula"` method.

row.values, column.values
: Optional vectors of values that define the grid when `x` is a matrix. `row.values`
  and `column.values` must have the same lengths as `nrow(x)` and `ncol(x)`
  respectively. By default, row and column numbers.

allow.multiple, outer, auto.key, prepanel, strip, groups, xlab, xlim, ylab, ylim, c
: These arguments are documented in the help page for [xyplot](). For the `cloud.table`
  method, `groups` must be a logical indicating whether the last dimension should
  be used as a grouping variable as opposed to a conditioning variable. This is only
  relevant if the table has more than 2 dimensions.

type   type of display in `cloud` (see `panel.3dscatter` for details). Defaults to
                 `"h"` for the `matrix` method.

`aspect, panel.aspect`

                 Unlike other high level functions, `aspect` is taken to be a numeric vector of
                 length 2, giving the relative aspects of the y-size/x-size and z-size/x-size of the
                 enclosing cube. The usual role of the `aspect` argument in determining the as-
                 pect ratio of the panel (see `xyplot` for details) is played by `panel.aspect`,
                 except that it can only be a numeric value.

                 For the `matrix` methods, the default y/x aspect is `ncol(x) / nrow(x)`
                 and the z/x aspect is the smaller of the y/x aspect and 1.

panel   panel function used to create the display. See `panel.cloud` for (non-trivial)
                 details.

`default.prepanel`

                 Fallback prepanel function. See `xyplot`.

scales   a list describing the scales. As with other high level functions (see `xyplot` for
                 details), this list can contain parameters in name=value form. It can also contain
                 components with the special names x, y and z, which can be similar lists with
                 axis-specific values overriding the ones specified in `scales`.

                 The most common use for this argument is to set `arrows=FALSE`, which
                 causes tick marks and labels to be used instead of arrows being drawn (the de-
                 fault). Both can be suppressed by `draw=FALSE`. Another special component
                 is `distance`, which specifies the relative distance of the axis label from the
                 bounding box. If specified as a component of `scales` (as opposed to one of
                 `scales$z` etc), this can be (and is recycled if not) a vector of length 3, speci-
                 fying distances for the x, y and z labels respectively.

                 Other components that work in the `scales` argument of `xyplot` etc. should
                 also work here (as long as they make sense), including explicit specification of
                 tick mark locations and labels. (Not everything is implemented yet, but if you
                 find something that should work but does not, feel free to bug the maintainer.)

                 Note, however, that for these functions `scales` cannot contain information
                 that is specific to particular panels. If you really need that, consider using the
                 `scales.3d` argument of `panel.cloud`.

axs.default   Unlike 2-D display functions, `cloud` does not expand the bounding box to
                 slightly beyound the range of the data, even though it should. This is primarily
                 because this is the natural behaviour in `wireframe`, which uses the same code.
                 `axs.default` is intended to provide a different default for `cloud`. However,
                 this feature has not yet been implemented.

zlab   Specifies a label describing the z variable in ways similar to `xlab` and `ylab`
                 (i.e. "grob", character string, expression or list) in other high level functions.
                 Additionally, if `zlab` (and `xlab` and `ylab`) is a list, it can contain a component
                 called `rot`, controlling the rotation for the label

zlim   limits for the z-axis. Similar to `xlim` and `ylim` in other high level functions

drape   logical, whether the wireframe is to be draped in color. If `TRUE`, the height of
                 a facet is used to determine its color in a manner similar to the coloring scheme
                 used in `levelplot`. Otherwise, the background color is used to color the
                 facets. This argument is ignored if `shade = TRUE` (see `panel.3dwire`).

`at, col.regions, alpha.regions`

these arguments are analogous to those in [levelplot](). if `drape=TRUE`, `at` gives the vector of cutpoints where the colors change, and `col.regions` the vector of colors to be used in that case. `alpha.regions` determines the alpha-transparency on supporting devices. These are passed down to the panel function, and also used in the colorkey if appropriate. The default for `col.regions` and `alpha.regions` is derived from the Trellis setting `"regions"`

`cuts`   if `at` is unspecified, the approximate number of cutpoints if `drape=TRUE`

`pretty`   whether automatic choice of cutpoints should be prettfied

`colorkey`   logical indicating whether a color key should be drawn alongside, or a list describing such a key. See [levelplot]() for details.

`...`   Any number of other arguments can be specified, and are passed to the panel function. In particular, the arguments `distance`, `perspective`, `screen` and `R.mat` are very important in determining the 3-D display. The argument `shade` can be useful for `wireframe` calls, and controls shading of the rendered surface. These arguments are described in detail in the help page for [panel.cloud]().

Additionally, an argument called `zoom` may be specified, which should be a numeric scalar to be interpreted as a scale factor by which the projection is magnified. This can be useful to get the variable names into the plot. This argument is actually only used by the default prepanel function.

Details

These functions produce three dimensional plots in each panel (as long as the default panel functions are used). The orientation is obtained as follows: the data are scaled to fall within a bounding box that is contained in the [-0.5, 0.5] cube (even smaller for non-default values of `aspect`). The viewing direction is given by a sequence of rotations specified by the `screen` argument, starting from the positive Z-axis. The viewing point (camera) is located at a distance of `1/distance` from the origin. If `perspective=FALSE`, `distance` is set to 0 (i.e., the viewing point is at an infinite distance).

`cloud` draws a 3-D Scatter Plot, while `wireframe` draws a 3-D surface (usually evaluated on a grid). Multiple surfaces can be drawn by `wireframe` using the `groups` argument (although this is of limited use because the display is incorrect when the surfaces intersect). Specifying `groups` with `cloud` results in a `panel.superpose`-like effect (via [panel.3dscatter]()).

`wireframe` can optionally render the surface as being illuminated by a light source (no shadows though). Details can be found in the help page for [panel.3dwire](). Note that although arguments controlling these are actually arguments for the panel function, they can be supplied to `cloud` and `wireframe` directly.

For single panel plots, `wireframe` can also plot parametrized 3-D surfaces (i.e., functions of the form f(u,v) = (x(u,v), y(u,v), z(u,v)), where values of (u,v) lie on a rectangle. The simplest example of this sort of surface is a sphere parametrized by latitude and longitude. This can be achieved by calling `wireframe` with a formula x of the form `z~x*y`, where x, y and z are all matrices of the same dimension, representing the values of x(u,v), y(u,v) and z(u,v) evaluated on a discrete rectangular grid (the actual values of (u,v) are irrelevant).

When this feature is used, the heights used to calculate `drape` colors or shading colors are no longer the z values, but the distances of (x,y,z) from the origin.

Note that this feature does not work with `groups`, `subscripts`, `subset`, etc. Conditioning variables are also not supported in this case.

The algorithm for identifying which edges of the bounding box are 'behind' the points doesn't work in some extreme situations. Also, `panel.cloud` tries to figure out the optimal location of the arrows and axis labels automatically, but can fail on occasion (especially when the view is from 'below' the data). This can be manually controlled by the `scpos` argument in `panel.cloud`.

These and all other high level Trellis functions have several other arguments in common. These are extensively documented only in the help page for `xyplot`, which should be consulted to learn more detailed usage.

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

There is a known problem with grouped `wireframe` displays when the (x, y) coordinates represented in the data do not represent the full evaluation grid. The problem occurs whether the grouping is specified through the `groups` argument or through the formula interface, and currently causes memory access violations. Depending on the circumstances, this is manifested either as a meaningless plot or a crash. To work around the problem, it should be enough to have a row in the data frame for each grid point, with an `NA` response (`z`) in rows that were previously missing.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/

See Also

`Lattice` for an overview of the package, as well as `xyplot`, `levelplot`, `panel.cloud`.

For interaction, see `panel.identify.cloud`.

Examples

```
## volcano  ## 87 x 61 matrix
wireframe(volcano, shade = TRUE,
          aspect = c(61/87, 0.4),
          light.source = c(10,0,10))

g <- expand.grid(x = 1:10, y = 5:15, gr = 1:2)
g$z <- log((g$x^g$g + g$y^2) * g$gr)
wireframe(z ~ x * y, data = g, groups = gr,
          scales = list(arrows = FALSE),
```

```
            drape = TRUE, colorkey = TRUE,
            screen = list(z = 30, x = -60))

cloud(Sepal.Length ~ Petal.Length * Petal.Width | Species, data = iris,
      screen = list(x = -90, y = 70), distance = .4, zoom = .6)

## cloud.table

cloud(prop.table(Titanic, margin = 1:3),
      type = c("p", "h"), strip = strip.custom(strip.names = TRUE),
      scales = list(arrows = FALSE, distance = 2), panel.aspect = 0.7,
      zlab = "Proportion")[, 1]

## transparent axes

par.set <-
    list(axis.line = list(col = "transparent"),
         clip = list(panel = "off"))
print(cloud(Sepal.Length ~ Petal.Length * Petal.Width,
            data = iris, cex = .8,
            groups = Species,
            main = "Stereo",
            screen = list(z = 20, x = -70, y = 3),
            par.settings = par.set,
            scales = list(col = "black")),
      split = c(1,1,2,1), more = TRUE)
print(cloud(Sepal.Length ~ Petal.Length * Petal.Width,
            data = iris, cex = .8,
            groups = Species,
            main = "Stereo",
            screen = list(z = 20, x = -70, y = 0),
            par.settings = par.set,
            scales = list(col = "black")),
      split = c(2,1,2,1))
```

 B_08_splom   *Scatter Plot Matrices*

Description

Draw Conditional Scatter Plot Matrices and Parallel Coordinate Plots

Usage

```
splom(x, data, ...)
parallel(x, data, ...)

## S3 method for class 'formula'
splom(x,
```

```
       data,
       auto.key = FALSE,
       aspect = 1,
       between = list(x = 0.5, y = 0.5),
       panel = lattice.getOption("panel.splom"),
       prepanel,
       scales,
       strip,
       groups,
       xlab,
       xlim,
       ylab = NULL,
       ylim,
       superpanel = lattice.getOption("panel.pairs"),
       pscales = 5,
       varnames = NULL,
       drop.unused.levels,
       ...,
       lattice.options = NULL,
       default.scales,
       default.prepanel = lattice.getOption("prepanel.default.splom"),
       subset = TRUE)
## S3 method for class 'formula'
parallel(x,
         data,
         auto.key = FALSE,
         aspect = "fill",
         between = list(x = 0.5, y = 0.5),
         panel = lattice.getOption("panel.parallel"),
         prepanel,
         scales,
         strip,
         groups,
         xlab = NULL,
         xlim,
         ylab = NULL,
         ylim,
         varnames = NULL,
         horizontal.axis = TRUE,
         drop.unused.levels,
         ...,
         lattice.options = NULL,
         default.scales,
         default.prepanel = lattice.getOption("prepanel.default.parallel"),
         subset = TRUE)

## S3 method for class 'data.frame'
splom(x, data = NULL, ..., groups = NULL, subset = TRUE)
```

```
## S3 method for class 'matrix'
splom(x, data = NULL, ..., groups = NULL, subset = TRUE)

## S3 method for class 'matrix'
parallel(x, data = NULL, ..., groups = NULL, subset = TRUE)
## S3 method for class 'data.frame'
parallel(x, data = NULL, ..., groups = NULL, subset = TRUE)
```

Arguments

x	The object on which method dispatch is carried out.
	For the "formula" method, a formula describing the structure of the plot, which should be of the form ~ x	g1 * g2 * ..., where x is a data frame or matrix. Each of g1,g2,... must be either factors or shingles. The conditioning variables g1, g2, ... may be omitted.
	For the data.frame methods, a data frame.
data	For the formula methods, an optional data frame in which variables in the formula (as well as groups and subset, if any) are to be evaluated.
aspect	aspect ratio of each panel (and subpanel), square by default for splom.
between	to avoid confusion between panels and subpanels, the default is to show the panels of a splom plot with space between them.
panel	Usual interpretation for parallel, namely the function that creates the display within each panel.
	For splom, the terminology is slightly complicated. The role played by the panel function in most other high-level functions is played here by the superpanel function, which is responsible for the display for each conditional data subset. panel is simply an argument to the default superpanel function panel.pairs, and is passed on to it unchanged. It is used there to create each pairwise display. See [panel.pairs](#) for more useful options.
superpanel	function that sets up the splom display, by default as a scatterplot matrix.
pscales	a numeric value or a list, meant to be a less functional substitute for the scales argument in xyplot etc. This argument is passed to the superpanel function, and is handled by the default superpanel function panel.pairs. The help page for the latter documents this argument in more detail.
varnames	A character or expression vector or giving names to be used for the variables in x. By default, the column names of x.
horizontal.axis	
	logical indicating whether the parallel axes should be laid out horizontally (TRUE) or vertically (FALSE).
auto.key, prepanel, scales, strip, groups, xlab, xlim, ylab, ylim, drop.unused.leve	
	See [xyplot](#)
default.prepanel	
	Fallback prepanel function. See [xyplot](#).
...	Further arguments. See corresponding entry in [xyplot](#) for non-trivial details.

Details

splom produces Scatter Plot Matrices. The role usually played by panel is taken over by superpanel, which takes a data frame subset and is responsible for plotting it. It is called with the coordinate system set up to have both x- and y-limits from 0.5 to ncol(z) + 0.5. The only built-in option currently available is panel.pairs, which calls a further panel function for each pair (i, j) of variables in z inside a rectangle of unit width and height centered at c(i, j) (see panel.pairs for details).

Many of the finer customizations usually done via arguments to high level function like xyplot are instead done by panel.pairs for splom. These include control of axis limits, tick locations and prepanel calcultions. If you are trying to fine-tune your splom plot, definitely look at the panel.pairs help page. The scales argument is usually not very useful in splom, and trying to change it may have undesired effects.

parallel draws Parallel Coordinate Plots. (Difficult to describe, see example.)

These and all other high level Trellis functions have several arguments in common. These are extensively documented only in the help page for xyplot, which should be consulted to learn more detailed usage.

Value

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

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

xyplot, Lattice, panel.pairs, panel.parallel.

Examples

```
super.sym <- trellis.par.get("superpose.symbol")
splom(~iris[1:4], groups = Species, data = iris,
      panel = panel.superpose,
      key = list(title = "Three Varieties of Iris",
                 columns = 3,
                 points = list(pch = super.sym$pch[1:3],
                 col = super.sym$col[1:3]),
                 text = list(c("Setosa", "Versicolor", "Virginica")))))
splom(~iris[1:3]|Species, data = iris,
      layout=c(2,2), pscales = 0,
      varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"),
      page = function(...) {
          ltext(x = seq(.6, .8, length.out = 4),
                y = seq(.9, .6, length.out = 4),
                labels = c("Three", "Varieties", "of", "Iris"),
                cex = 2)
```

```
      })
parallel(~iris[1:4] | Species, iris)
parallel(~iris[1:4], iris, groups = Species,
         horizontal.axis = FALSE, scales = list(x = list(rot = 90)))
```

B_09_tmd   *Tukey Mean-Difference Plot*

Description

tmd Creates Tukey Mean-Difference Plots from a trellis object returned by xyplot, qq or qqmath.
The prepanel and panel functions are used as appropriate. The formula method for tmd is pro-
vided for convenience, and simply calls tmd on the object created by calling xyplot on that
formula.

Usage

```
tmd(object, ...)

## S3 method for class 'trellis'
tmd(object,
    xlab = "mean",
    ylab = "difference",
    panel,
    prepanel,
    ...)

prepanel.tmd.qqmath(x,
            f.value = NULL,
            distribution = qnorm,
            qtype = 7,
            groups = NULL,
            subscripts, ...)
panel.tmd.qqmath(x,
            f.value = NULL,
            distribution = qnorm,
            qtype = 7,
            groups = NULL,
            subscripts, ...,
            identifier = "tmd")
panel.tmd.default(x, y, groups = NULL, ...,
                identifier = "tmd")
prepanel.tmd.default(x, y, ...)
```

Arguments

object   An object of class "trellis" returned by xyplot, qq or qqmath.

xlab	x label
ylab	y label
panel	panel function to be used. See details below.
prepanel	prepanel function. See details below.
f.value, distribution, qtype	
	see panel.qqmath.
groups, subscripts	
	see xyplot.
x, y	data as passed to panel functions in original call.
...	other arguments
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Details

The Tukey Mean-difference plot is produced by modifying the (x,y) values of each panel as follows: the new coordinates are given by x=(x+y)/2 and y=y-x, which are then plotted. The default panel function(s) add a reference line at y=0 as well.

tmd acts on the a "trellis" object, not on the actual plot this object would have produced. As such, it only uses the arguments supplied to the panel function in the original call, and completely ignores what the original panel function might have done with this data. tmd uses these panel arguments to set up its own scales (using its prepanel argument) and display (using panel). It is thus important to provide suitable prepanel and panel functions to tmd depending on the original call.

Such functions currently exist for xyplot, qq (the ones with default in their name) and qqmath, as listed in the usage section above. These assume the default displays for the corresponding high-level call. If unspecified, the prepanel and panel arguments default to suitable choices.

tmd uses the update method for "trellis" objects, which processes all extra arguments supplied to tmd.

Value

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

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

qq, qqmath, xyplot, Lattice

Examples

```
tmd(qqmath(~height | voice.part, data = singer))
```

B_10_rfs   *Residual and Fit Spread Plots*

Description

Plots fitted values and residuals (via qqmath) on a common scale for any object that has methods for fitted values and residuals.

Usage

```
rfs(model, layout=c(2, 1), xlab="f-value", ylab=NULL,
    distribution = qunif,
    panel, prepanel, strip, ...)
```

Arguments

model   a fitted model object with methods `fitted.values` and `residuals`. Can be the value returned by `oneway`

layout   default layout is c(2,1)

xlab   defaults to `"f.value"`

distribution   the distribution function to be used for qqmath

ylab, panel, prepanel, strip
               See [xyplot](#)

...   other arguments, passed on to [qqmath](#).

Value

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

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[oneway](#), [qqmath](#), [xyplot](#), [Lattice](#)

Examples

```
rfs(oneway(height ~ voice.part, data = singer, spread = 1), aspect = 1)
```

`B_11_oneway` *Fit One-way Model*

Description

Fits a One-way model to univariate data grouped by a factor, the result often being displayed using `rfs`

Usage

```
oneway(formula, data, location=mean, spread=function(x) sqrt(var(x)))
```

Arguments

`formula`	formula of the form `y ~ x` where `y` is the numeric response and `x` is the grouping factor
`data`	data frame in which the model is to be evaluated
`location`	function or numeric giving the location statistic to be used for centering the observations, e.g. `median`, 0 (to avoid centering).
`spread`	function or numeric giving the spread statistic to be used for scaling the observations, e.g. `sd`, 1 (to avoid scaling).

Value

A list with components

`location`	vector of locations for each group.
`spread`	vector of spreads for each group.
`fitted.values`	
	vector of locations for each observation.
`residuals`	residuals (`y - fitted.values`).
`scaled.residuals`	
	residuals scaled by `spread` for their group

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`rfs`, `Lattice`

C_01_trellis.device

Initializing Trellis Displays

Description

Initialization of a display device with appropriate graphical parameters.

Usage

```
trellis.device(device = getOption("device"),
               color = !(dev.name == "postscript"),
               theme = lattice.getOption("default.theme"),
               new = TRUE,
               retain = FALSE,
               ...)

standard.theme(name, color)
canonical.theme(name, color)
col.whitebg()
```

Arguments

device	function (or the name of one as a character string) that starts a device. Admissible values depend on the platform and how R was compiled (see `Devices`), but usually `"pdf"`, `"postscript"`, `"png"`, `"jpeg"` and at least one of `"X11"`, `"windows"` and `"quartz"` will be available.
color	logical, whether the initial settings should be color or black and white. Defaults to `FALSE` for postscript devices, `TRUE` otherwise. Note that this only applies to the initial choice of colors, which can be overridden using `theme` or subsequent calls to `trellis.par.set` (and by arguments supplied directly in high level calls for some settings).
theme	list of components that changes the settings of the device opened, or, a function that when called produces such a list. The function name can be supplied as a quoted string. These settings are only used to modify the default settings (determined by other arguments), and need not contain all possible parameters.
	A possible use of this argument is to change the default settings by specifying `lattice.options(default.theme = "col.whitebg")`. For back-compatibility, this is initially (when lattice is loaded) set to `options(lattice.theme)`.
	If `theme` is a function, it will not be supplied any arguments, however, it is guaranteed that a device will already be open when it is called, so one may use `.Device` inside the function to ascertain what device has been opened.
new	logical flag indicating whether a new device should be started. If `FALSE`, the options for the current device are changed to the defaults determined by the other arguments.

retain	logical. If TRUE and a setting for this device already exists, then that is used instead of the defaults for this device. By default, pre-existing settings are over-written (and lost).
name	name of the device for which the setting is required, as returned by .Device
...	additional parameters to be passed to the device function, most commonly file for non-screen devices, as well as height, width, etc. See the help file for individual devices for admissible arguments.

Details

Trellis Graphics functions obtain the default values of various graphical parameters (colors, line types, fonts, etc.) from a customizable "settings" list. This functionality is analogous to par for standard R graphics and, together with lattice.options, mostly supplants it (par settings are mostly ignored by Lattice). Unlike par, Trellis settings can be controlled separately for each different device type (but not concurrently for different instances of the same device). standard.theme and col.whitebg produce predefined settings (a.k.a. themes), while trellis.device provides a high level interface to control which "theme" will be in effect when a new device is opened. trellis.device is called automatically when a "trellis" object is plotted, and the defaults can be used to provide sufficient control, so in a properly configured system it is rarely necessary for the user to call trellis.device explicitly.

The standard.theme function is intended to provide device specific settings (e.g. light colors on a grey background for screen devices, dark colors or black and white for print devices) which were used as defaults prior to R 2.3.0. However, these defaults are not always appropriate, due to the variety of platforms and hardware settings on which R is used, as well as the fact that a plot created on a particular device may be subsequently used in many different ways. For this reason, a "safe" default is used for all devices from R 2.3.0 onwards. The old behaviour can be reinstated by setting standard.theme as the default theme argument, e.g. by putting options(lattice.theme = "standard.theme") in a startup script (see the entry for theme above for details).

Value

standard.theme returns a list of components defining graphical parameter settings for Lattice displays. It is used internally in trellis.device, and can also be used as the theme argument to trellis.par.set, or even as theme in trellis.device to use the defaults for another device. canonical.theme is an alias for standard.theme.

col.whitebg returns a similar (but smaller) list that is suitable as the theme argument to trellis.device and trellis.par.set. It contains settings values which provide colors suitable for plotting on a white background. Note that the name col.whitebg is somewhat of a misnomer, since it actually sets the background to transparent rather than white.

Note

Earlier versions of trellis.device had a bg argument to set the background color, but this is no longer supported. If supplied, the bg argument will be passed on to the device function; however, this will have no effect on the Trellis settings. It is rarely meaningful to change the background alone; if you feel the need to change the background, consider using the theme argument instead.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer. http://lmdvr.r-forge.r-project.org/

See Also

Lattice for an overview of the lattice package.

Devices for valid choices of device on your platform.

trellis.par.get and trellis.par.set can be used to query and modify the settings *after* a device has been initialized. The par.settings argument to high level functions, described in xyplot, can be used to attach transient settings to a "trellis" object.

C_02_trellis.par.get

Graphical Parameters for Trellis Displays

Description

Functions used to query, display and modify graphical parameters for fine control of Trellis displays. Modifications are made to the settings for the currently active device only.

Usage

```
trellis.par.set(name, value, ..., theme, warn = TRUE, strict = FALSE)
trellis.par.get(name = NULL)
show.settings(x = NULL)
```

Arguments

name	A character string giving the name of a component. If unspecified in trellis.par.get(), the return value is a named list containing all the current settings (this can be used to get the valid values for name).
value	a list giving the desired value of the component. Components that are already defined as part of the current settings but are not mentioned in value will remain unchanged.
theme	a list decribing how to change the settings, similar to what is returned by trellis.par.get(). This is purely for convenience, allowing multiple calls to trellis.par.set to be condensed into one. The name of each component must be a valid name as described above, with the corresponding value a valid value as described above.
	As in trellis.device, theme can also be a function that produces such a list when called. The function name can be supplied as a quoted string.

`...`	Multiple settings can be specified in `name = value` form. Equivalent to calling with `theme = list(...)`
`warn`	A logical flag, indicating whether a warning should be issued when `trellis.par.get` is called when no graphics device is open.
`strict`	Usually a logical flag, indicating whether the `value` should be interpreted strictly. Usually, assignment of value to the corresponding named component is fuzzy in the sense that sub-components that are absent from `value` but not currently `NULL` are retained. By specifying `strict = TRUE`, such values will be removed.
	An even stricter interpretation is allowed by specifying `strict` as a numeric value larger than `1`. In that case, top-level components not specified in the call will also be removed. This is primarily for internal use.
`x`	optional list of components that change the settings (any valid value of `theme`). These are used to modify the current settings (obtained by `trellis.par.get`) before they are displayed.

Details

The various graphical parameters (color, line type, background etc) that control the look and feel of Trellis displays are highly customizable. Also, R can produce graphics on a number of devices, and it is expected that a different set of parameters would be more suited to different devices. These parameters are stored internally in a variable named `lattice.theme`, which is a list whose components define settings for particular devices. The components are idenified by the name of the device they represent (as obtained by `.Device`), and are created as and when new devices are opened for the first time using `trellis.device` (or Lattice plots are drawn on a device for the first time in that session).

The initial settings for each device defaults to values appropriate for that device. In practice, this boils down to three distinct settings, one for screen devices like `x11` and `windows`, one for black and white plots (mostly useful for `postscript`) and one for color printers (color `postcript`, `pdf`).

Once a device is open, its settings can be modified. When another instance of the same device is opened later using `trellis.device`, the settings for that device are reset to its defaults, unless otherwise specified in the call to `trellis.device`. But settings for different devices are treated separately, i.e., opening a postscript device will not alter the x11 settings, which will remain in effect whenever an x11 device is active.

The functions `trellis.par.*` are meant to be interfaces to the global settings. They always apply on the settings for the currently ACTIVE device.

`trellis.par.get`, called without any arguments, returns the full list of settings for the active device. With the `name` argument present, it returns that component only. `trellis.par.get` sets the value of the `name` component of the current active device settings to `value`.

`trellis.par.get` is usually used inside trellis functions to get graphical parameters before plotting. Modifications by users via `trellis.par.set` is traditionally done as follows:

```
add.line <- trellis.par.get("add.line")
add.line$col <- "red"
trellis.par.set("add.line", add.line)
```

More convenient (but not S compatible) ways to do this are

```
trellis.par.set(list(add.line = list(col = "red")))
```

and

```
trellis.par.set(add.line = list(col = "red"))
```

The actual list of the components in `trellis.settings` has not been finalized, so I'm not attempting to list them here. The current value can be obtained by `print(trellis.par.get())`. Most names should be self-explanatory.

`show.settings` provides a graphical display summarizing some of the values in the current settings.

Value

`trellis.par.get` returns a list giving parameters for that component. If `name` is missing, it returns the full list.

Note

In some ways, `trellis.par.get` and `trellis.par.set` together are a replacement for the `par` function used in traditional R graphics. In particular, changing `par` settings has little (if any) effect on lattice output. Since lattice plots are implemented using Grid graphics, its parameter system *does* have an effect unless overridden by a suitable lattice parameter setting. Such parameters can be specified as part of a lattice theme by including them in the `grid.pars` component (see `gpar` for a list of valid parameter names).

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`trellis.device`, `Lattice`, `gpar`

Examples

```
show.settings()
```

C_03_simpleTheme   *Function to generate a simple theme*

Description

Simple interface to generate a list appropriate as a theme, typically used as the `par.settings` argument in a high level call

Usage

```
simpleTheme(col, alpha,
            cex, pch, lty, lwd, font, fill, border,
            col.points, col.line,
            alpha.points, alpha.line)
```

Arguments

`col, col.points, col.line`

> A color specification. `col` is used for components `"plot.symbol"`, `"plot.line"`, `"plot.polygon"`, `"superpose.symbol"`, `"superpose.line"`, and `"superpose.polygon"`. `col.points` overrides `col`, but is used only for `"plot.symbol"` and `"superpose.symbol"`. Similarly, `col.line` overrides `col` for `"plot.line"` and `"superpose.line"`. The arguments can be vectors, but only the first component is used for scalar targets (i.e., the ones without `"superpose"` in their name).

`alpha, alpha.points, alpha.line`

> A numeric alpha transparency specification. The same rules as `col`, etc., apply.

`cex, pch, font`

> Parameters for points. Applicable for components `plot.symbol` (for which only the first component is used) and `superpose.symbol` (for which the arguments can be vectors).

`lty, lwd` Parameters for lines. Applicable for components `plot.line` (for which only the first component is used) and `superpose.line` (for which the arguments can be vectors).

`fill` fill color, applicable for components `plot.symbol`, `plot.polygon`, `superpose.symbol`, and `superpose.polygon`.

`border` border color, applicable for components `plot.polygon` and `superpose.polygon`.

Details

The appearance of a lattice display depends partly on the "theme" active when the display is plotted (see `trellis.device` for details). This theme is used to obtain defaults for various graphical parameters, and in particular, the `auto.key` argument works on the premise that the same source is used for both the actual graphical encoding and the legend. The easiest way to specify custom settings for a particular display is to use the `par.settings` argument, which is usually tedious to construct as it is a nested list. The `simpleTheme` function can be used in such situations as a wrapper that generates a suitable list given parameters in simple `name=value` form, with the nesting made implicit. This is less flexible, but straightforward and sufficient in most situations.

Value

A list that would work as the `theme` argument to `trellis.device` and `trellis.par.set`, or as the `par.settings` argument to any high level lattice function such as `xyplot`.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`, based on a suggestion from John Maindonald.

See Also

trellis.device, xyplot, Lattice

Examples

```
str(simpleTheme(pch = 16))

dotplot(variety ~ yield | site, data = barley, groups = year,
        auto.key = list(space = "right"),
        par.settings = simpleTheme(pch = 16),
        xlab = "Barley Yield (bushels/acre) ",
        aspect=0.5, layout = c(1,6))
```

C_04_lattice.options

Low-level Options Controlling Behaviour of Lattice

Description

Functions to handle settings used by lattice. Their main purpose is to make code maintainance easier, and users normally should not need to use these functions. However, fine control at this level maybe useful in certain cases.

Usage

```
lattice.options(...)
lattice.getOption(name)
```

Arguments

name	character giving the name of a setting
...	new options can be defined, or existing ones modified, using one or more arguments of the form name = value or by passing a list of such tagged values. Existing values can be retrieved by supplying the names (as character strings) of the components as unnamed arguments.

Details

These functions are modeled on options and getOption, and behave similarly for the most part. Some of the available components are documented here, but not all. The purpose of the ones not documented are either fairly obvious, or not of interest to the end-user.

panel.error A function, or NULL. If the former, every call to the panel function will be wrapped inside tryCatch with the specified function as an error handler. The default is to use the panel.error function. This prevents the plot from failing due to errors in a single panel, and leaving the grid operations in an unmanageable state. If set to NULL, errors in panel functions will not be caught using tryCatch.

`save.object` Logical flag indicating whether a `"trellis"` object should be saved when plotted for subsequent retrieval and further manipulation. Defaults to `TRUE`.

`layout.widths`, `layout.heights` Controls details of the default space allocation in the grid layout created in the course of plotting a `"trellis"` object. Not of interest to the end-user, who should instead use the similiarly named component in the graphical settings, modifiable using `trellis.par.set`.

`drop.unused.levels`

`legend.bbox`

Value

`lattice.getOption` returns the value of a single component, whereas `lattice.options` always returns a list with one or more named components. When changing the values of components, the old values of the modified components are returned by `lattice.options`. If called without any arguments, the full list is returned.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`options`, `trellis.device`, `trellis.par.get`, `Lattice`

Examples

```
names(lattice.options())
str(lattice.getOption("layout.widths"))
```

`C_05_print.trellis` *Plot and Summarize Trellis Objects*

Description

The `print` and `plot` methods produce a graph from a `"trellis"` object. The `print` method is necessary for automatic plotting. `plot` method is essentially an alias, provided for convenience. The `summary` method gives a textual summary of the object. `dim` and `dimnames` describe the cross-tabulation induced by conditioning. `panel.error` is the default handler used when an error occurs while executing the panel function.

Usage

```
## S3 method for class 'trellis'
plot(x, position, split,
     more = FALSE, newpage = TRUE,
     packet.panel = packet.panel.default,
     draw.in = NULL,
```

```
        panel.height = lattice.getOption("layout.heights")$panel,
        panel.width = lattice.getOption("layout.widths")$panel,
        save.object = lattice.getOption("save.object"),
        panel.error = lattice.getOption("panel.error"),
        prefix,
        ...)
## S3 method for class 'trellis'
print(x, ...)

## S3 method for class 'trellis'
summary(object, ...)

## S3 method for class 'trellis'
dim(x)
## S3 method for class 'trellis'
dimnames(x)

panel.error(e)
```

Arguments

x, object   an object of class "trellis"

position   a vector of 4 numbers, typically c(xmin, ymin, xmax, ymax) that give the lower-
            left and upper-right corners of a rectangle in which the Trellis plot of x is to be
            positioned. The coordinate system for this rectangle is [0-1] in both the x and y
            directions.

split   a vector of 4 integers, c(x,y,nx,ny) , that says to position the current plot at the
            x,y position in a regular array of nx by ny plots. (Note: this has origin at top
            left)

more   A logical specifying whether more plots will follow on this page.

newpage   A logical specifying whether the plot should be on a new page. This option is
            specific to lattice, and is useful for including lattice plots in an arbitrary grid
            viewport (see the details section).

packet.panel a function that determines which packet (data subset) is plotted in which panel.
            Panels are always drawn in an order such that columns vary the fastest, then rows
            and then pages. This function determines, given the column, row and page and
            other relevant information, the packet (if any) which should be used in that panel.
            By default, the association is determnined by matching panel order with packet
            order, which is determined by varying the first conditioning variable the fastest,
            then the second, and so on. This association rule is encoded in the default,
            namely the function packet.panel.default, whose help page details the
            arguments supplied to whichever function is specified as the packet.panel
            argument.

draw.in   An optional (grid) viewport (used as the name argument in downViewport)
            in which the plot is to be drawn. If specified, the newpage argument is ignored.
            This feature is not well-tested.

panel.width, panel.height

        lists with 2 components, that should be valid `x` and `units` arguments to `unit()` (the `data` argument cannot be specified currently, but can be considered for addition if needed). The resulting `unit` object will be the width/height of each panel in the Lattice plot. These arguments can be used to explicitly control the dimensions of the panel, rather than letting them expand to maximize available space. Vector widths are allowed, and can specify unequal lengths across rows or columns.

        Note that this option should not be used in conjunction with non-default values of the `aspect` argument in the original high level call (no error will be produced, but the resulting behaviour is undefined).

save.object   logical, specifying whether the object being printed is to be saved. The last object thus saved can be subsequently retrieved. This is an experimental feature that should allow access to a panel's data after the plot is done, making it possible to enhance the plot after the fact. This also allows the user to invoke the `update` method on the current plot, even if it was not assigned to a variable explicitly. For more details, see `trellis.focus`.

panel.error   a function, or a character string naming a function, that is to be executed when an error occurs during the execution of the panel function. The error is caught (using `tryCatch`) and supplied as the only argument to `panel.error`. The default behaviour (implemented as the `panel.error` function) is to print the corresponding error message in the panel and continue. To stop execution on error, use `panel.error = stop`.

        Normal error recovery and debugging tools are unhelpful when `tryCatch` is used. `tryCatch` can be completely bypassed by setting `panel.error` to NULL.

prefix   A character string acting as a prefix identifying the plot of a `"trellis"` object, primarily used in constructing viewport and grob names, to distinguish similar viewports if a page contains multiple plots. The default is based on the serial number of the current plot on the current page (specifically, `"plot_01"`, `"plot_02"`, etc.). If supplied explicitly, this must be a valid R symbol name (briefly, it must start with a letter or a period followed by a letter) and must not contain the grid path separator (currently `"::"`).

e   an error condition caught by `tryCatch`

...   extra arguments, ignored by the `print` method. All arguments to the `plot` method are passed on to the `print` method.

Details

This is the default print method for objects of class `"trellis"`, produced by calls to functions like `xyplot`, `bwplot` etc. It is usually called automatically when a trellis object is produced. It can also be called explicitly to control plot positioning by means of the arguments `split` and `position`.

When `newpage = FALSE`, the current grid viewport is treated as the plotting area, making it possible to embed a Lattice plot inside an arbitrary grid viewport. The `draw.in` argument provides an alternative mechanism that may be simpler to use.

The print method uses the information in x (the object to be printed) to produce a display using the Grid graphics engine. At the heart of the plot is a grid layout, of which the entries of most interest to the user are the ones containing the display panels.

Unlike in older versions of Lattice (and Grid), the grid display tree is retained after the plot is produced, making it possible to access individual viewport locations and make additions to the plot. For more details and a lattice level interface to these viewports, see `trellis.focus`.

Note

Unlike S-PLUS, trying to position a multipage display (using position and/or split) will mess things up.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`Lattice`, `unit`, `update.trellis`, `trellis.focus`, `packet.panel.default`

Examples

```
p11 <- histogram( ~ height | voice.part, data = singer, xlab="Height")
p12 <- densityplot( ~ height | voice.part, data = singer, xlab = "Height")
p2 <- histogram( ~ height, data = singer, xlab = "Height")


## simple positioning by split
print(p11, split=c(1,1,1,2), more=TRUE)
print(p2, split=c(1,2,1,2))

## Combining split and position:
print(p11, position = c(0,0,.75,.75), split=c(1,1,1,2), more=TRUE)
print(p12, position = c(0,0,.75,.75), split=c(1,2,1,2), more=TRUE)
print(p2, position = c(.5,.75,1,1), more=FALSE)

## Using seekViewport

## repeat same plot, with different polynomial fits in each panel
xyplot(Armed.Forces ~ Year, longley, index.cond = list(rep(1, 6)),
       layout = c(3, 2),
       panel = function(x, y, ...)
       {
           panel.xyplot(x, y, ...)
           fm <- lm(y ~ poly(x, panel.number()))
           llines(x, predict(fm))
       })

## Not run:
grid::seekViewport(trellis.vpname("panel", 1, 1))
cat("Click somewhere inside the first panel:\n")
```

```
ltext(grid::grid.locator(), lab = "linear")

## End(Not run)

grid::seekViewport(trellis.vpname("panel", 1, 1))
grid::grid.text("linear")

grid::seekViewport(trellis.vpname("panel", 2, 1))
grid::grid.text("quadratic")

grid::seekViewport(trellis.vpname("panel", 3, 1))
grid::grid.text("cubic")

grid::seekViewport(trellis.vpname("panel", 1, 2))
grid::grid.text("degree 4")

grid::seekViewport(trellis.vpname("panel", 2, 2))
grid::grid.text("degree 5")

grid::seekViewport(trellis.vpname("panel", 3, 2))
grid::grid.text("degree 6")
```

```
C_06_update.trellis
```
Retrieve and Update Trellis Object

Description

Update method for objects of class `"trellis"`, and a way to retrieve the last printed trellis object (that was saved).

Usage

```
## S3 method for class 'trellis'
update(object,
       panel,
       aspect,
       as.table,
       between,
       key,
       auto.key,
       legend,
       layout,
       main,
       page,
       par.strip.text,
       prepanel,
       scales,
       skip,
```

```
        strip,
        strip.left,
        sub,
        xlab,
        ylab,
        xlab.top,
        ylab.right,
        xlim,
        ylim,
        xscale.components,
        yscale.components,
        axis,
        par.settings,
        plot.args,
        lattice.options,
        index.cond,
        perm.cond,
        ...)

## S3 method for class 'trellis'
t(x)

## S3 method for class 'trellis'
x[i, j, ..., drop = FALSE]

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

Arguments

`object, x`	The object to be updated, of class `"trellis"`.
`i, j`	indices to be used. Names are not currently allowed.
`drop`	logical, whether dimensions with only one level are to be dropped. Currently ignored, behaves as if it were `FALSE`.
`panel, aspect, as.table, between, key, auto.key, legend, layout, main, page, par.st`	arguments that will be used to update `object`. See details below.
`prefix`	A character string acting as a prefix identifying the plot of a `"trellis"` object. Only relevant when a particular page is occupied by more than one plot. Defaults to the value appropriate for the last `"trellis"` object printed. See `trellis.focus`.

Details

All high level lattice functions such as `xyplot` produce an object of (S3) class `"trellis"`, which is usually displayed by its `print` method. However, the object itself can be manipulated and modified to a large extent using the `update` method, and then re-displayed as needed.

Most arguments to high level functions can also be supplied to the `update` method as well, with some exceptions. Generally speaking, anything that would needs to change the data within each

panel is a no-no (this includes the `formula`, `data`, `groups`, `subscripts` and `subset`). Everything else is technically game, though might not be implemented yet. If you find something missing that you wish to have, feel free to make a request.

Not all arguments accepted by a Lattice function are processed by `update`, but the ones listed above should work. The purpose of these arguments are described in the help page for `xyplot`. Any other argument is added to the list of arguments to be passed to the `panel` function. Because of their somewhat special nature, updates to objects produced by `cloud` and `wireframe` do not work very well yet.

The `"["` method is a convenient shortcut for updating `index.cond`. The `t` method is a convenient shortcut for updating `perm.cond` in the special (but frequent) case where there are exactly two conditioning variables, when it has the effect of switching ('transposing') their order.

The print method for `"trellis"` objects optionally saves the object after printing it. If this feature is enabled, `trellis.last.object` can retrieve it. By default, the last object plotted is retrieved, but if multiple objects are plotted on the current page, then others can be retrieved using the appropriate `prefix` argument. If `trellis.last.object` is called with arguments, these are used to update the retrieved object before returning it.

Value

An object of class `trellis`, by default plotted by `print.trellis`. `trellis.last.object` returns `NULL` is no saved object is available.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

`trellis.object`, `Lattice`, `xyplot`

Examples

```
spots <- by(sunspots, gl(235, 12, labels = 1749:1983), mean)
old.options <- lattice.options(save.object = TRUE)
xyplot(spots ~ 1749:1983, xlab = "", type = "l",
       scales = list(x = list(alternating = 2)),
       main = "Average Yearly Sunspots")
update(trellis.last.object(), aspect = "xy")
trellis.last.object(xlab = "Year")
lattice.options(old.options)
```

`C_07_shingles`   *shingles*

Description

Functions to handle shingles

Usage

```
shingle(x, intervals=sort(unique(x)))
equal.count(x, ...)
as.shingle(x)
is.shingle(x)

## S3 method for class 'shingle'
plot(x, panel, xlab, ylab, ...)

## S3 method for class 'shingle'
print(x, showValues = TRUE, ...)

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

## S3 method for class 'shingleLevel'
print(x, ...)

## S3 method for class 'shingle'
summary(object, showValues = FALSE, ...)



## S3 method for class 'shingle'
x[subset, drop = FALSE]
as.factorOrShingle(x, subset, drop)
```

Arguments

x	numeric variable or R object, shingle in `plot.shingle` and `x[]`. An object (list of intervals) of class "shingleLevel" in `print.shingleLevel`
object	shingle object to be summarized
showValues	logical, whether to print the numeric part. If FALSE, only the intervals are printed
intervals	numeric vector or matrix with 2 columns
subset	logical vector
drop	whether redundant shingle levels are to be dropped
panel, xlab, ylab	
	standard Trellis arguments (see [xyplot](#))
...	other arguments, passed down as appropriate. For example, extra arguments to `equal.count` are passed on to `co.intervals`. graphical parameters can be passed as arguments to the `plot` method.

Details

A shingle is a data structure used in Trellis, and is a generalization of factors to 'continuous' variables. It consists of a numeric vector along with some possibly overlapping intervals. These inter-

vals are the 'levels' of the shingle. The `levels` and `nlevels` functions, usually applicable to factors, also work on shingles. The implementation of shingles is slightly different from S.

There are print methods for shingles, as well as for printing the result of `levels()` applied to a shingle. For use in labelling, the `as.character` method can be used to convert levels of a shingle to character strings.

`equal.count` converts `x` to a shingle using the equal count algorithm. This is essentially a wrapper around `co.intervals`. All arguments are passed to `co.intervals`.

`shingle` creates a shingle using the given `intervals`. If `intervals` is a vector, these are used to form 0 length intervals.

`as.shingle` returns `shingle(x)` if `x` is not a shingle.

`is.shingle` tests whether `x` is a shingle.

`plot.shingle` displays the ranges of shingles via rectangles. `print.shingle` and `summary.shingle` describe the shingle object.

Value

`x$intervals` for `levels.shingle(x)`, logical for `is.shingle`, an object of class `"trellis"` for `plot` (printed by default by `print.trellis`), and an object of class `"shingle"` for the others.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[xyplot](), [co.intervals](), [Lattice]()

Examples

```
z <- equal.count(rnorm(50))
plot(z)
print(z)
print(levels(z))
```

  `D_draw.colorkey`   *Produce a Colorkey for levelplot*

Description

Produces (and possibly draws) a Grid frame grob which is a colorkey that can be placed in other Grid plots. Used in levelplot

Usage

```
draw.colorkey(key, draw=FALSE, vp=NULL)
```

Arguments

key
: A list determining the key. See documentation for `levelplot`, in particular the section describing the `colorkey` argument, for details.

draw
: logical, whether the grob is to be drawn.

vp
: viewport

Value

A Grid frame object (that inherits from `"grob"`)

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

xyplot

D_draw.key | *Produce a Legend or Key*

Description

Produces (and possibly draws) a Grid frame grob which is a legend (aka key) that can be placed in other Grid plots.

Usage

```
draw.key(key, draw=FALSE, vp=NULL, ...)
```

Arguments

key
: A list determining the key. See documentation for `xyplot`, in particular the section describing the `key` argument, for details.

draw
: logical, whether the grob is to be drawn.

vp
: viewport

...
: ignored

Value

A Grid frame object (that inherits from 'grob').

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

[xyplot](#)

`D_level.colors`	*A function to compute false colors representing a numeric or categorical variable*

Description

Calculates false colors from a numeric variable (including factors, using their numeric codes) given a color scheme and breakpoints.

Usage

```
level.colors(x, at, col.regions, colors = TRUE, ...)
```

Arguments

x	A numeric or [factor](#) variable.
at	A numeric variable of breakpoints defining intervals along the range of x.
col.regions	A specification of the colors to be assigned to each interval defined by at. This could be either a vector of colors, or a function that produces a vector of colors when called with a single argument giving the number of colors. See details below.
colors	logical indicating whether colors should be computed and returned. If FALSE, only the indices representing which interval (among those defined by at) each value in x falls into is returned.
...	Extra arguments, ignored.

Details

If at has length n, then it defines n-1 intervals. Values of x outside the range of at are not assigned to an interval, and the return value is NA for such values.

Colors are chosen by assigning a color to each of the n-1 intervals. If col.regions is a palette function (such as [topo.colors](#), or the result of calling [colorRampPalette](#)), it is called with n-1 as an argument to obtain the colors. Otherwise, if there are exactly n-1 colors in col.regions, these get assigned to the intervals. If there are fewer than n-1 colors, col.regions gets recycled. If there are more, a more or less equally spaced (along the length of col.regions) subset is chosen.

Value

A vector of the same length as x. Depending on the colors argument, this could be either a vector of colors (in a form usable by R), or a vector of integer indices representing which interval the values of x fall in.

Author(s)

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

See Also

`levelplot`, `colorRampPalette`.

Examples

```
depth.col <-
    with(quakes,
         level.colors(depth, at = do.breaks(range(depth), 30),
                      col.regions = terrain.colors))


xyplot(lat ~ long | equal.count(stations), quakes,
       strip = strip.custom(var.name = "Stations"),
       colours = depth.col,
       panel = function(x, y, colours, subscripts, ...) {
           panel.xyplot(x, y, pch = 21, col = "transparent",
                        fill = colours[subscripts], ...)
       })
```

`D_make.groups`   *Grouped data from multiple vectors*

Description

Combines two or more vectors, possibly of different lengths, producing a data frame with a second
column indicating which of these vectors that row came from. This is mostly useful for getting data
into a form suitable for use in high level Lattice functions.

Usage

```
make.groups(...)
```

Arguments

`...`   one or more vectors of the same type (coercion is attempted if not), or one or
                     more data frames with similar columns, with possibly differing number of rows.

Value

When all the input arguments are vectors, a data frame with two columns

data   all the vectors supplied, concatenated

which   factor indicating which vector the corresponding data value came from

When all the input arguments are data frames, the result of rbind applied to them, along with an additional which column as described above.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

Lattice

Examples

```
sim.dat <-
    make.groups(uniform = runif(200),
                exponential = rexp(175),
                lognormal = rlnorm(150),
                normal = rnorm(125))
qqmath( ~ data | which, sim.dat, scales = list(y = "free"))
```

D_simpleKey   *Function to generate a simple key*

Description

Simple interface to generate a list appropriate for draw.key

Usage

```
simpleKey(text, points = TRUE,
          rectangles = FALSE,
          lines = FALSE,
          col, cex, alpha, font,
          fontface, fontfamily,
          lineheight, ...)
```

Arguments

`text`	character or expression vector, to be used as labels for levels of the grouping variable
`points`	logical
`rectangles`	logical
`lines`	logical

`col, cex, alpha, font, fontface, fontfamily, lineheight`
　　　　　Used as top-level components of the list produced, to be used for the text labels. Defaults to the values in `trellis.par.get("add.text")`

`...`	further arguments added to the list, eventually passed to `draw.key`

Details

A lattice plot can include a legend (key) if an appropriate list is specified as the `key` argument to a high level Lattice function such as `xyplot`. This key can be very flexible, but that flexibility comes at a cost: this list needs to be fairly complicated even in simple situations. `simpleKey` is designed as a useful shortcut in the common case of a key drawn in conjunction with a grouping variable, using the default graphical settings.

The `simpleKey` function produces a suitable `key` argument using a simpler interface. The resulting list will use the `text` argument as a text component, along with at most one set each of points, rectangles, and lines. The number of entries (rows) in the key will be the length of the `text` component. The graphical parameters for the additional components will be derived from the default graphical settings (wherein lies the simplification, as otherwise these would have to be provided explicitly).

Calling `simpleKey` directly is usually unnecessary. It is most commonly invoked (during the plotting of the "trellis" object) when the `auto.key` argument is supplied in a high-level plot with a `groups` argument. In that case, the `text` argument of `simpleKey` defaults to `levels(groups)`, and the defaults for the other arguments depend on the relevant high-level function. Note that these defaults can be overridden by supplying `auto.key` as a list containing the replacement values.

Value

A list that would work as the `key` argument to [xyplot](), etc.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[Lattice](), [draw.key](), [trellis.par.get](), and [xyplot](), specifically the entry for `auto.key`.

`D_strip.default`   *Default Trellis Strip Function*

Description

`strip.default` is the function that draws the strips by default in Trellis plots. Users can write their own strip functions, but most commonly this involves calling `strip.default` with a slightly different arguments. `strip.custom` provides a convenient way to obtain new strip functions that differ from `strip.default` only in the default values of certain arguments.

Usage

```
strip.default(which.given,
              which.panel,
              var.name,
              factor.levels,
              shingle.intervals,
              strip.names = c(FALSE, TRUE),
              strip.levels = c(TRUE, FALSE),
              sep = " : ",
              style = 1,
              horizontal = TRUE,
              bg = trellis.par.get("strip.background")$col[which.given],
              fg = trellis.par.get("strip.shingle")$col[which.given],
              par.strip.text = trellis.par.get("add.text"))
strip.custom(...)
```

Arguments

which.given   integer index specifying which of the conditioning variables this strip corre-
                sponds to.

which.panel   vector of integers as long as the number of conditioning variables. The contents
                are indices specifying the current levels of each of the conditioning variables
                (thus, this would be unique for each distinct packet). This is identical to the
                return value of `which.packet`, which is a more accurate name.

var.name   vector of character strings or expressions as long as the number of conditioning
                variables. The contents are interpreted as names for the conditioning variables.
                Whether they are shown on the strip depends on the values of `strip.names`
                and `style` (see below). By default, the names are shown for shingles, but not
                for factors.

factor.levels

                vector of character strings or expressions giving the levels of the conditioning
                variable currently being drawn. For more than one conditioning variable, this
                will vary with `which.given`. Whether these levels are shown on the strip de-
                pends on the values of `strip.levels` and `style` (see below). `factor.levels`
                may be specified for both factors and shingles (despite the name), but by default

they are shown only for factors. If shown, the labels may optionally be abbreviated by specifying suitable components in `par.strip.text` (see [xyplot](#))

shingle.intervals

if the current strip corresponds to a shingle, this should be a 2-column matrix giving the levels of the shingle. (of the form that would be produced by **printing** `levels(shingle)`). Otherwise, it should be `NULL`

strip.names   a logical vector of length 2, indicating whether or not the name of the conditioning variable that corresponds to the strip being drawn is to be written on the strip. The two components give the values for factors and shingles respectively. This argument is ignored for a factor when `style` is not one of 1 and 3.

strip.levels   a logical vector of length 2, indicating whether or not the level of the conditioning variable that corresponds to the strip being drawn is to be written on the strip. The two components give the values for factors and shingles respectively.

sep   character or expression, serving as a separator if the name and level are both to be shown.

style   integer, with values 1, 2, 3, 4 and 5 currently supported, controlling how the current level of a factor is encoded. Ignored for shingles (actually, when `shingle.intervals` is non-null.

The best way to find out what effect the value of `style` has is to try them out. Here is a short description: for a style value of 1, the strip is colored in the background color with the strip text (as determined by other arguments) centered on it. A value of 3 is the same, except that a part of the strip is colored in the foreground color, indicating the current level of the factor. For styles 2 and 4, the part corresponding to the current level remains colored in the foreground color, however, for style = 2, the remaining part is not colored at all, whereas for 4, it is colored with the background color. For both these, the names of all the levels of the factor are placed on the strip from left to right. Styles 5 and 6 produce the same effect (they are subtly different in S, this implementation corresponds to 5), they are similar to style 1, except that the strip text is not centered, it is instead positioned according to the current level.

Note that unlike S-PLUS, the default value of `style` is 1. `strip.names` and `strip.levels` have no effect if `style` is not 1 or 3.

horizontal   logical, specifying whether the labels etc should be horizontal. `horizontal=FALSE` is useful for strips on the left of panels using `strip.left=TRUE`

par.strip.text

list with parameters controlling the text on each strip, with components `col`, `cex`, `font`, etc.

bg   strip background color.

fg   strip foreground color.

...   arguments to be passed on to `strip.default`, overriding whatever value it would have normally assumed

Details

default strip function for trellis functions. Useful mostly because of the `style` argument — non-default styles are often more informative, especially when the names of the levels of the factor `x` are

small. Traditional use is as `strip = function(...) strip.default(style=2,...)`, though this can be simplified by the use of `strip.custom`.

Value

`strip.default` is called for its side-effect, which is to draw a strip appropriate for multi-panel Trellis conditioning plots. `strip.custom` returns a function that is similar to `strip.default`, but with different defaults for the arguments specified in the call.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[xyplot](), [Lattice]()

Examples

```
## Traditional use
xyplot(Petal.Length ~ Petal.Width | Species, iris,
       strip = function(..., style) strip.default(..., style = 4))

## equivalent call using strip.custom
xyplot(Petal.Length ~ Petal.Width | Species, iris,
       strip = strip.custom(style = 4))

xyplot(Petal.Length ~ Petal.Width | Species, iris,
       strip = FALSE,
       strip.left = strip.custom(style = 4, horizontal = FALSE))
```

D_trellis.object   *A Trellis Plot Object*

Description

This class of objects is returned by high level lattice functions, and is usually plotted by default by its [print]() method.

Details

A trellis object, as returned by high level lattice functions like [xyplot](), is a list with the `"class"` attribute set to `"trellis"`. Many of the components of this list are simply the arguments to the high level function that produced the object. Among them are: `as.table`, `layout`, `page`, `panel`, `prepanel`, `main`, `sub`, `par.strip.text`, `strip`, `skip`, `xlab` `ylab`, `par.settings`, `lattice.options` and `plot.args`. Some other typical components are:

`formula` the Trellis formula used in the call

`index.cond` list with index for each of the conditioning variables

`perm.cond` permutation of the order of the conditioning variables

`aspect.fill` logical, whether `aspect` is `"fill"`

`aspect.ratio` numeric, aspect ratio to be used if `aspect.fill` is `FALSE`

`call` call that generated the object.

`condlevels` list with levels of the conditioning variables

`legend` list describing the legend(s) to be drawn

`panel.args` a list as long as the number of panels, each element being a list itself, containing the arguments in named form to be passed to the panel function in that panel.

`panel.args.common` a list containing the arguments common to all the panel functions in `name=value` form

`x.scales` list describing x-scale, can consist of several other lists, paralleling panel.args, if x-relation is not `"same"`

`y.scales` list describing y-scale, similar to `x.scales`

`x.between` numeric vector of interpanel x-space

`y.between` numeric vector of interpanel y-space

`x.limits` numeric vector of length 2 or list, giving x-axis limits

`y.limits` similar to `x.limits`

`packet.sizes` array recording the number of observations in each packet

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

[Lattice](#), [xyplot](#), [print.trellis](#)

`E_interaction`	*Functions to Interact with Lattice Plots*

Description

The classic Trellis paradigm is to plot the whole object at once, without the possibility of interacting with it afterwards. However, by keeping track of the grid viewports where the panels and strips are drawn, it is possible to go back to them afterwards and enhance them one panel at a time. These functions provide convenient interfaces to help in this. Note that these are still experimental and the exact details may change in future.

Usage

```
panel.identify(x, y = NULL,
                subscripts = seq_along(x),
                labels = subscripts,
                n = length(x), offset = 0.5,
                threshold = 18, ## in points, roughly 0.25 inches
                panel.args = trellis.panelArgs(),
                ...)
panel.identify.qqmath(x, distribution, groups, subscripts, labels,
                       panel.args = trellis.panelArgs(),
                       ...)
panel.identify.cloud(x, y, z, subscripts,
                      perspective, distance,
                      xlim, ylim, zlim,
                      screen, R.mat, aspect, scales.3d,
                      ...,
                      panel.3d.identify,
                      n = length(subscripts),
                      offset = 0.5,
                      threshold = 18,
                      labels = subscripts,
                      panel.args = trellis.panelArgs())
panel.link.splom(threshold = 18, verbose = getOption("verbose"), ...)
panel.brush.splom(threshold = 18, verbose = getOption("verbose"), ...)


trellis.vpname(name = c("position", "split", "split.location", "toplevel",
                        "figure", "panel", "strip", "strip.left",
                        "legend", "main", "sub", "xlab", "ylab",
                        "xlab.top", "ylab.right", "page"),
               column, row,
               side = c("left", "top", "right", "bottom", "inside"),
               clip.off = FALSE, prefix)
trellis.grobname(name,
                 type = c("", "panel", "strip", "strip.left"),
                 group = 0,
                 column = lattice.getStatus("current.focus.column",
                                             prefix = prefix),
                 row = lattice.getStatus("current.focus.row",
                                          prefix = prefix),
                 prefix = lattice.getStatus("current.prefix"))
trellis.focus(name, column, row, side, clip.off,
              highlight = interactive(), ..., prefix,
              guess = TRUE, verbose = getOption("verbose"))
trellis.switchFocus(name, side, clip.off, highlight, ..., prefix)
trellis.unfocus()
trellis.panelArgs(x, packet.number)
```

Arguments

x, y, z	variables defining the contents of the panel. In the case of trellis.panelArgs, a "trellis" object.
n	the number of points to identify by default (overridden by a right click)
subscripts	an optional vector of integer indices associated with each point. See details below.
labels	an optional vector of labels associated with each point. Defaults to subscripts

distribution, groups

typical panel arguments of [panel.qqmath](). These will usually be obtained from panel.args

offset	the labels are printed either below, above, to the left or to the right of the identified point, depending on the relative location of the mouse click. The offset specifies (in "char" units) how far from the identified point the labels should be printed.
threshold	threshold in grid's "points" units. Points further than these from the mouse click position are not considered
panel.args	list that contains components names x (and usually y), to be used if x is missing. Typically, when called after trellis.focus, this would appropriately be the arguments passed to that panel.

perspective, distance, xlim, ylim, zlim, screen, R.mat, aspect, scales.3d

arguments as passed to [panel.cloud](). These are required to recompute the relevant three-dimensional projections in panel.identify.cloud.

panel.3d.identify

the function that is responsible for the actual interaction once the data rescaling and rotation computations have been done. By default, an internal function similar to panel.identify is used.

name	A character string indicating which viewport or grob we are looking for. Although these do not necessarily provide access to all viewports and grobs created by a lattice plot, they cover most of the ones that end-users may find interesting.
	trellis.vpname and trellis.focus deal with viewport names only, and only accept the values explicitly listed above. trellis.grobname is meant to create names for grobs, and can currently accept any value.
	If name, as well as column and row is missing in a call to trellis.focus, the user can click inside a panel (or an associated strip) to focus on that panel. Note however that this assumes equal width and height for each panel, and may not work when this is not true.
	When name is "panel", "strip", or "strip.left", column and row must also be specified. When name is "legend", side must also be specified.
column, row	integers, indicating position of the panel or strip that should be assigned focus in the Trellis layout. Rows are usually calculated from the bottom up, unless the plot was created with as.table=TRUE
guess	logical. If TRUE, and the display has only one panel, that panel will be automatically selected by a call to trellis.focus.

side
character string, relevant only for legends (i.e., when `name="legend"`), indicating their position. Partial specification is allowed, as long as it is unambiguous.

clip.off
logical, whether clipping should be off, relevant when `name` is `"panel"` or `"strip"`. This is necessary if axes are to be drawn outside the panel or strip. Note that setting `clip.off=FALSE` does not necessarily mean that clipping is on; that is determined by conditions in effect during printing.

type
A character string specifying whether the grob is specific to a particular panel or strip.

When `type` is `"panel"`, `"strip"`, or `"strip.left"`, information about the panel is added to the grob name.

group
An integer specifying whether the grob is specific to a particular group within the plot.

When `group` is greater than zero, information about the group is added to the grob name.

prefix
A character string acting as a prefix identifying the plot of a `"trellis"` object, primarily used to distinguish otherwise equivalent viewports in different plots. This only becomes relevant when a particular page is occupied by more than one plot. Defaults to the value appropriate for the last `"trellis"` object printed, as determined by the `prefix` argument in `print.trellis`.

Users should not usually need to supply a value for this argument except to interact with an existing plot other than the one plotted last.

For `switchFocus`, ignored except when it does not match the prefix of the currently active plot, in which case an error occurs.

highlight
logical, whether the viewport being assigned focus should be highlighted. For `trellis.focus`, the default is `TRUE` in interactive mode, and `trellis.switchFocus` by default preserves the setting currently active.

packet.number
integer, which panel to get data from. See `packet.number` for details on how this is calculated

verbose
whether details will be printed

...
For `panel.identify.qqmath`, extra parameters are passed on to `panel.identify`. For `panel.identify`, extra arguments are treated as graphical parameters and are used for labelling. For `trellis.focus` and `trellis.switchFocus`, these are used (in combination with `lattice.options`) for highlighting the chosen viewport if so requested. Graphical parameters can be supplied for `panel.link.splom`.

Details

`panel.identify` is similar to `identify`. When called, it waits for the user to identify points (in the panel being drawn) via mouse clicks. Clicks other than left-clicks terminate the procedure. Although it is possible to call it as part of the panel function, it is more typical to use it to identify points after plotting the whole object, in which case a call to `trellis.focus` first is necessary.

panel.link.splom is meant for use with splom, and requires a panel to be chosen using trellis.focus before it is called. Clicking on a point causes that and the corresponding proections in other pairwise scatter plots to be highlighted. panel.brush.splom is a (misnamed) alias for panel.link.splom, retained for back-compatibility.

panel.identify.qqmath is a specialized wrapper meant for use with the display produced by qqmath. panel.identify.qqmath is a specialized wrapper meant for use with the display produced by cloud. It would be unusual to call them except in a context where default panel function arguments are available through trellis.panelArgs (see below).

One way in which panel.identify etc. are different from identify is in how it uses the subscripts argument. In general, when one identifies points in a panel, one wants to identify the origin in the data frame used to produce the plot, and not within that particular panel. This information is available to the panel function, but only in certain situations. One way to ensure that subscripts is available is to specify subscripts = TRUE in the high level call such as xyplot. If subscripts is not explicitly specified in the call to panel.identify, but is available in panel.args, then those values will be used. Otherwise, they default to seq_along(x). In either case, the final return value will be the subscripts that were marked.

The process of printing (plotting) a Trellis object builds up a grid layout with named viewports which can then be accessed to modify the plot further. While full flexibility can only be obtained by using grid functions directly, a few lattice functions are available for the more common tasks.

trellis.focus can be used to move to a particular panel or strip, identified by its position in the array of panels. It can also be used to focus on the viewport corresponding to one of the labels or a legend, though such usage would be less useful. The exact viewport is determined by the name along with the other arguments, not all of which are relevant for all names. Note that when more than one object is plotted on a page, trellis.focus will always go to the plot that was created last. For more flexibility, use grid functions directly (see note below).

After a successful call to trellis.focus, the desired viewport (typically panel or strip area) will be made the 'current' viewport (plotting area), which can then be enhanced by calls to standard lattice panel functions as well as grid functions.

It is quite common to have the layout of panels chosen when a "trellis" object is drawn, and not before then. Information on the layout (specifically, how many rows and columns, and which packet belongs in which position in this layout) is retained for the last "trellis" object plotted, and is available through trellis.currentLayout.

trellis.unfocus unsets the focus, and makes the top level viewport the current viewport.

trellis.switchFocus is a convenience function to switch from one viewport to another, while preserving the current row and column. Although the rows and columns only make sense for panels and strips, they would be preserved even when the user switches to some other viewport (where row/column is irrelevant) and then switches back.

Once a panel or strip is in focus, trellis.panelArgs can be used to retrieve the arguments that were available to the panel function at that position. In this case, it can be called without arguments as

```
trellis.panelArgs()
```

This usage is also allowed when a "trellis" object is being printed, e.g. inside the panel functions or the axis function (but not inside the prepanel function). trellis.panelArgs can

also retrieve the panel arguments from any `"trellis"` object. Note that for this usage, one needs to specify the `packet.number` (as described under the `panel` entry in `xyplot`) and not the position in the layout, because a layout determines the panel only **after** the object has been printed.

It is usually not necessary to call `trellis.vpname` and `trellis.grobname` directly. However, they can be useful in generating appropriate names in a portable way when using grid functions to interact with the plots directly, as described in the note below.

Value

`panel.identify` returns an integer vector containing the subscripts of the identified points (see details above). The equivalent of `identify` with `pos=TRUE` is not yet implemented, but can be considered for addition if requested.

`trellis.panelArgs` returns a named list of arguments that were available to the panel function for the chosen panel.

`trellis.vpname` and `trellis.grobname` return character strings.

`trellis.focus` has a meaningful return value only if it has been used to focus on a panel interactively, in which case the return value is a list with components `col` and `row` giving the column and row positions respectively of the chosen panel, unless the choice was cancelled (by a right click), in which case the return value is `NULL`. If click was outside a panel, both `col` and `row` are set to 0.

Note

The viewports created by lattice are accessible to the user through `trellis.focus` as described above. Functions from the grid package can also be used directly. For example, `current.vpTree` can be used to inspect the current viewport tree and `seekViewport` or `downViewport` can be used to navigate to these viewports. For such usage, `trellis.vpname` and `trellis.grobname` provides a portable way to access the appropriate viewports and grobs by name.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>. Felix Andrews provided initial implementations of `panel.identify.qqmath` and support for focusing on panels interctively.

See Also

`identify`, `Lattice`, `print.trellis`, `trellis.currentLayout`, `current.vpTree`, `viewports`

Examples

```
## Not run:
xyplot(1:10 ~ 1:10)
trellis.focus("panel", 1, 1)
panel.identify()

## End(Not run)
```

```
xyplot(Petal.Length ~ Sepal.Length | Species, iris, layout = c(2, 2))
Sys.sleep(1)

trellis.focus("panel", 1, 1)
do.call("panel.lmline", trellis.panelArgs())
Sys.sleep(0.5)
trellis.unfocus()

trellis.focus("panel", 2, 1)
do.call("panel.lmline", trellis.panelArgs())
Sys.sleep(0.5)
trellis.unfocus()

trellis.focus("panel", 1, 2)
do.call("panel.lmline", trellis.panelArgs())
Sys.sleep(0.5)
trellis.unfocus()


## choosing loess smoothing parameter

p <- xyplot(dist ~ speed, cars)

panel.loessresid <-
    function(x = panel.args$x,
             y = panel.args$y,
             span,
             panel.args = trellis.panelArgs())
{
    fm <- loess(y ~ x, span = span)
    xgrid <- do.breaks(current.panel.limits()$xlim, 50)
    ygrid <- predict(fm, newdata = data.frame(x = xgrid))
    panel.lines(xgrid, ygrid)
    pred <- predict(fm)
    ## center residuals so that they fall inside panel
    resids <- y - pred + mean(y)
    fm.resid <- loess.smooth(x, resids, span = span)
    ##panel.points(x, resids, col = 1, pch = 4)
    panel.lines(fm.resid, col = 1)
}


spans <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)
update(p, index.cond = list(rep(1, length(spans))))
panel.locs <- trellis.currentLayout()

i <- 1

for (row in 1:nrow(panel.locs))
    for (column in 1:ncol(panel.locs))
    if (panel.locs[row, column] > 0)
{
    trellis.focus("panel", row = row, column = column,
```

```
                      highlight = FALSE)
    panel.loessresid(span = spans[i])
    grid::grid.text(paste("span = ", spans[i]),
                     x = 0.25,
                     y = 0.75,
                     default.units = "npc")
    trellis.unfocus()
    i <- i + 1
}
```

F_1_panel.barchart *Default Panel Function for barchart*

Description

Default panel function for `barchart`.

Usage

```
panel.barchart(x, y, box.ratio = 1, box.width,
                horizontal = TRUE,
                origin = NULL, reference = TRUE,
                stack = FALSE,
                groups = NULL,
                col = if (is.null(groups)) plot.polygon$col
                        else superpose.polygon$col,
                border = if (is.null(groups)) plot.polygon$border
                            else superpose.polygon$border,
                lty = if (is.null(groups)) plot.polygon$lty
                        else superpose.polygon$lty,
                lwd = if (is.null(groups)) plot.polygon$lwd
                        else superpose.polygon$lwd,
                ..., identifier = "barchart")
```

Arguments

x	Extent of Bars. By default, bars start at left of panel, unless `origin` is specified, in which case they start there
y	Horizontal location of bars, possibly factor
box.ratio	ratio of bar width to inter-bar space
box.width	thickness of bars in absolute units; overrides `box.ratio`. Useful for specifying thickness when the categorical variable is not a factor, as use of `box.ratio` alone cannot achieve a thickness greater than 1.
horizontal	logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of `bwplot` for a fuller explanation.

origin   the origin for the bars. For grouped displays with `stack = TRUE`, this argu-
                ment is ignored and the origin set to 0. Otherwise, defaults to `NULL`, in which
                case bars start at the left (or bottom) end of a panel. This choice is somewhat
                unfortuntate, as it can be misleading, but is the default for historical reasons. For
                tabular (or similar) data, `origin = 0` is usually more appropriate; if not, one
                should reconsider the use of a bar chart in the first place (dot plots are often a
                good alternative).

reference   logical, whether a reference line is to be drawn at the origin

stack   logical, relevant when groups is non-null. If `FALSE` (the default), bars for dif-
                ferent values of the grouping variable are drawn side by side, otherwise they are
                stacked.

groups   optional grouping variable

col, border, lty, lwd
                Graphical parameters for the bars. By default, the trellis parameter `plot.polygon`
                is used if there is no grouping variable, otherwise `superpose.polygon` is
                used. `col` gives the fill color, `border` the border color, and `lty` and `lwd` the
                line type and width of the borders.

...   extra arguments will be accepted but ignored

identifier   A character string that is prepended to the names of grobs that are created by
                this panel function.

Details

A barchart is drawn in the panel. Note that most arguments controlling the display can be supplied
to the high-level `barchart` call directly.

Author(s)

Deepayan Sarkar <`Deepayan.Sarkar@R-project.org`>

See Also

barchart

Examples

```
barchart(yield ~ variety | site, data = barley,
         groups = year, layout = c(1,6), origin = 0,
         ylab = "Barley Yield (bushels/acre)",
         scales = list(x = list(abbreviate = TRUE,
                                minlength = 5)))
```

F_1_panel.bwplot   *Default Panel Function for bwplot*

Description

This is the default panel function for `bwplot`.

Usage

```
panel.bwplot(x, y, box.ratio = 1,
              box.width = box.ratio / (1 + box.ratio),
              horizontal = TRUE,
              pch, col, alpha, cex,
              font, fontfamily, fontface,
              fill, varwidth = FALSE,
              notch = FALSE, notch.frac = 0.5,
              ...,
              levels.fos,
              stats = boxplot.stats,
              coef = 1.5,
              do.out = TRUE,
              identifier = "bwplot")
```

Arguments

x, y	numeric vector or factor. Boxplots drawn for each unique value of y (x) if `horizontal` is TRUE (FALSE)
box.ratio	ratio of box thickness to inter box space
box.width	thickness of box in absolute units; overrides `box.ratio`. Useful for specifying thickness when the categorical variable is not a factor, as use of `box.ratio` alone cannot achieve a thickness greater than 1.
horizontal	logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of bwplot for a fuller explanation.
pch, col, alpha, cex, font, fontfamily, fontface	
	graphical parameters controlling the dot. `pch="|"` is treated specially, by replacing the dot with a line (similar to boxplot)
fill	color to fill the boxplot
varwidth	logical. If TRUE, widths of boxplots are proportional to the number of points used in creating it.
notch	if `notch` is TRUE, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap this is 'strong evidence' that the two medians differ (Chambers et al., 1983, p. 62). See boxplot.stats for the calculations used.

`notch.frac`	numeric in (0,1). When `notch=TRUE`, the fraction of the box width that the notches should use.
`stats`	a function, defaulting to [`boxplot.stats`](#), that accepts a numeric vector and returns a list similar to the return value of `boxplot.stats`. The function must accept arguments `coef` and `do.out` even if they do not use them (a `...` argument is good enough). This function is used to determine the box and whisker plot.
`coef`, `do.out`	passed to `stats`
`levels.fos`	numeric values corresponding to positions of the factor or shingle variable. For internal use.
`...`	further arguments, ignored.
`identifier`	A character string that is prepended to the names of grobs that are created by this panel function.

Details

Creates Box and Whisker plot of x for every level of y (or the other way round if `horizontal=FALSE`). By default, the actual boxplot statistics are calculated using `boxplot.stats`. Note that most arguments controlling the display can be supplied to the high-level `bwplot` call directly.

Although the graphical parameters for the dot representing the median can be controlled by optional arguments, many others can not. These parameters are obtained from the relevant settings parameters (`"box.rectangle"` for the box, `"box.umbrella"` for the whiskers and `"plot.symbol"` for the outliers).

Author(s)

Deepayan Sarkar <`Deepayan.Sarkar@R-project.org`>

See Also

[`bwplot`](#), [`boxplot.stats`](#)

Examples

```
bwplot(voice.part ~ height, data = singer,
       xlab = "Height (inches)",
       panel = function(...) {
           panel.grid(v = -1, h = 0)
           panel.bwplot(...)
       },
       par.settings = list(plot.symbol = list(pch = 4)))

bwplot(voice.part ~ height, data = singer,
       xlab = "Height (inches)",
       notch = TRUE, pch = "|")
```

F_1_panel.cloud   *Default Panel Function for cloud*

Description

These are default panel functions controlling `cloud` and `wireframe` displays.

Usage

```
panel.cloud(x, y, subscripts, z,
            groups = NULL,
            perspective = TRUE,
            distance = if (perspective) 0.2 else 0,
            xlim, ylim, zlim,
            panel.3d.cloud = "panel.3dscatter",
            panel.3d.wireframe = "panel.3dwire",
            screen = list(z = 40, x = -60),
            R.mat = diag(4), aspect = c(1, 1),
            par.box = NULL,
            xlab, ylab, zlab,
            xlab.default, ylab.default, zlab.default,
            scales.3d,
            proportion = 0.6,
            wireframe = FALSE,
            scpos,
            ...,
            at,
            identifier = "cloud")
panel.wireframe(...)
panel.3dscatter(x, y, z, rot.mat, distance,
                groups, type = "p",
                xlim, ylim, zlim,
                xlim.scaled, ylim.scaled, zlim.scaled,
                zero.scaled,
                col, col.point, col.line,
                lty, lwd, cex, pch,
                cross, ..., .scale = FALSE, subscripts,
                identifier = "3dscatter")
panel.3dwire(x, y, z, rot.mat = diag(4), distance,
             shade = FALSE,
             shade.colors.palette = trellis.par.get("shade.colors")$palette,
             light.source = c(0, 0, 1000),
             xlim, ylim, zlim,
             xlim.scaled,
             ylim.scaled,
             zlim.scaled,
             col = if (shade) "transparent" else "black",
```

```
                        lty = 1, lwd = 1,
                        alpha,
                        col.groups = superpose.polygon$col,
                        polynum = 100,
                        ...,
                        .scale = FALSE,
                        drape = FALSE,
                        at,
                        col.regions = regions$col,
                        alpha.regions = regions$alpha,
                        identifier = "3dwire")
```

Arguments

x, y, z
: numeric (or possibly factors) vectors representing the data to be displayed. The interpretation depends on the context. For `panel.cloud` these are essentially the same as the data passed to the high level plot (except if `formula` was a matrix, the appropriate x and y vectors are generated). By the time they are passed to `panel.3dscatter` and `panel.3dwire`, they have been appropriately subsetted (using `subscripts`) and scaled (to lie inside a bounding box, usually the [-0.5, 0.5] cube).

    Further, for `panel.3dwire`, x and y are shorter than z and represent the sorted locations defining a rectangular grid. Also in this case, z may be a matrix if the display is grouped, with each column representing one surface.

    In `panel.cloud` (called from `wireframe`) and `panel.3dwire`, x, y and z could also be matrices (of the same dimension) when they represent a 3-D surface parametrized on a 2-D grid.

subscripts
: index specifying which points to draw. The same x, y and z values (representing the whole data) are passed to `panel.cloud` for each panel. `subscripts` specifies the subset of rows to be used for the particular panel.

groups
: specification of a grouping variable, passed down from the high level functions.

perspective
: logical, whether to plot a perspective view. Setting this to FALSE is equivalent to setting `distance` to 0

distance
: numeric, between 0 and 1, controls amount of perspective. The distance of the viewing point from the origin (in the transformed coordinate system) is 1 / `distance`. This is described in a little more detail in the documentation for cloud

screen
: A list determining the sequence of rotations to be applied to the data before being plotted. The initial position starts with the viewing point along the positive z-axis, and the x and y axes in the usual position. Each component of the list should be named one of "x", "y" or "z" (repititions are allowed), with their values indicating the amount of rotation about that axis in degrees.

R.mat
: initial rotation matrix in homogeneous coordinates, to be applied to the data before `screen` rotates the view further.

par.box
: graphical parameters for box, namely, col, lty and lwd. By default obtained from the parameter `box.3d`.

xlim, ylim, zlim

> limits for the respective axes. As with other lattice functions, these could each be a numeric 2-vector or a character vector indicating levels of a factor.

panel.3d.cloud, panel.3d.wireframe

> functions that draw the data-driven part of the plot (as opposed to the bounding box and scales) in `cloud` and `wireframe`. This function is called after the 'back' of the bounding box is drawn, but before the 'front' is drawn.
>
> Any user-defined custom display would probably want to change these functions. The intention is to pass as much information to this function as might be useful (not all of which are used by the defaults). In particular, these functions can expect arguments called `xlim`, `ylim`, `zlim` which give the bounding box ranges in the original data scale and `xlim.scaled`, `ylim.scaled`, `zlim.scaled` which give the bounding box ranges in the transformed scale. More arguments can be considered on request.

aspect   aspect as in `cloud`

xlab, ylab, zlab

> Labels, have to be lists. Typically the user will not manipulate these, but instead control this via arguments to `cloud` directly.

xlab.default   for internal use

ylab.default   for internal use

zlab.default   for internal use

scales.3d   list defining the scales

proportion   numeric scalar, gives the length of arrows as a proportion of the sides

scpos   A list with three components x, y and z (each a scalar integer), describing which of the 12 sides of the cube the scales should be drawn. The defaults should be OK. Valid values are x: 1, 3, 9, 11; y: 8, 5, 7, 6 and z: 4, 2, 10, 12. (See comments in the source code of `panel.cloud` to see the details of this enumeration.)

wireframe   logical, indicating whether this is a wireframe plot

drape   logical, whether the facets will be colored by height, in a manner similar to `levelplot`. This is ignored if `shade=TRUE`.

at, col.regions, alpha.regions

> deals with specification of colors when `drape = TRUE` in [wireframe](). `at` can be a numeric vector, `col.regions` a vector of colors, and `alpha.regions` a numeric scalar controlling transparency. The resulting behaviour is similar to [levelplot](), `at` giving the breakpoints along the z-axis where colors change, and the other two determining the colors of the facets that fall in between.

rot.mat   4x4 transformation matrix in homogeneous coordinates. This gives the rotation matrix combining the `screen` and `R.mat` arguments to [panel.cloud]()

type   Character vector, specifying type of cloud plot. Can include one or more of `"p"`, `"l"`, `"h"` or `"b"`. `"p"` and `"l"` mean 'points' and 'lines' respectively, and `"b"` means 'both'. `"h"` stands for 'histogram', and causes a line to be drawn from each point to the X-Y plane (i.e., the plane representing $z = 0$), or the lower (or upper) bounding box face, whichever is closer.

xlim.scaled, ylim.scaled, zlim.scaled

                 axis limits (after being scaled to the bounding box)

zero.scaled   z-axis location (after being scaled to the bounding box) of the X-Y plane in the original data scale, to which lines will be dropped (if within range) from each point when `type = "h"`

cross   logical, defaults to `TRUE` if `pch = "+"`. `panel.3dscatter` can represent each point by a 3d 'cross' of sorts (it's much easier to understand looking at an example than from a description). This is different from the usual `pch` argument, and reflects the depth of the points and the orientation of the axes. This argument indicates whether this feature will be used.

                 This is useful for two reasons. It can be set to `FALSE` to use `"+"` as the plotting character in the regular sense. It can also be used to force this feature in grouped displays.

shade   logical, indicating whether the surface is to be colored using an illumination model with a single light source

shade.colors.palette

                 a function (or the name of one) that is supposed to calculate the color of a facet when shading is being used. Three pieces of information are available to the function: first, the cosine of the angle between the incident light ray and the normal to the surface (representing foreshortening); second, the cosine of half the angle between the reflected ray and the viewing direction (useful for non-Lambertian surfaces); and third, the scaled (average) height of that particular facet with respect to the total plot z-axis limits.

                 All three numbers should be between 0 and 1. The `shade.colors.palette` function should return a valid color. The default function is obtained from the trellis settings.

light.source a 3-vector representing (in cartesian coordinates) the light source. This is relative to the viewing point being (0, 0, 1/distance) (along the positive z-axis), keeping in mind that all observations are bounded within the [-0.5, 0.5] cube

polynum   quadrilateral faces are drawn in batches of `polynum` at a time. Drawing too few at a time increases the total number of calls to the underlying `grid.polygon` function, which affects speed. Trying to draw too many at once may be unnecessarily memory intensive. This argument controls the trade-off.

col.groups   colors for different groups

col, col.point, col.line, lty, lwd, cex, pch, alpha

                 Graphical parameters. Some other arguments (such as `lex` for line width) may also be passed through the `...` argument.

...   other parameters, passed down when appropriate

.scale   Logical flag, indicating whether x, y, and z should be assumed to be in the original data scale and hence scaled before being plotted. x, y, and z are usually already scaled. However, setting `.scale=TRUE` may be helpful for calls to `panel.3dscatter` and `panel.3dwire` in user-supplied panel functions.

identifier   A character string that is prepended to the names of grobs that are created by this panel function.

Details

These functions together are responsible for the content drawn inside each panel in `cloud` and `wireframe`. `panel.wireframe` is a wrapper to `panel.cloud`, which does the actual work.

`panel.cloud` is responsible for drawing the content that does not depend on the data, namely, the bounding box, the arrows/scales, etc. At some point, depending on whether `wireframe` is TRUE, it calls either `panel.3d.wireframe` or `panel.3d.cloud`, which draws the data-driven part of the plot.

The arguments accepted by these two functions are different, since they have essentially different purposes. For cloud, the data is unstructured, and x, y and z are all passed to the `panel.3d.cloud` function. For wireframe, on the other hand, x and y are increasing vectors with unique values, defining a rectangular grid. z must be a matrix with `length(x) * length(y)` rows, and as many columns as the number of groups.

`panel.3dscatter` is the default `panel.3d.cloud` function. It has a `type` argument similar to `panel.xyplot`, and supports grouped displays. It tries to honour depth ordering, i.e., points and lines closer to the camera are drawn later, overplotting more distant ones. (Of course there is no absolute ordering for line segments, so an ad hoc ordering is used. There is no hidden point removal.)

`panel.3dwire` is the default `panel.3d.wireframe` function. It calculates polygons corresponding to the facets one by one, but waits till it has collected information about `polynum` facets, and draws them all at once. This avoids the overhead of drawing `grid.polygon` repeatedly, speeding up the rendering considerably. If `shade = TRUE`, these attempt to color the surface as being illuminated from a light source at `light.source`. `palette.shade` is a simple function that provides the deafult shading colors

Multiple surfaces are drawn if `groups` is non-null in the call to `wireframe`, however, the algorithm is not sophisticated enough to render intersecting surfaces correctly.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

`cloud`, `utilities.3d`

F_1_panel.densityplot

Default Panel Function for densityplot

Description

This is the default panel function for `densityplot`.

Usage

```
panel.densityplot(x, darg, plot.points = "jitter", ref = FALSE,
                  groups = NULL, weights = NULL,
                  jitter.amount, type, ...,
                  identifier = "density")
```

Arguments

x	data points for which density is to be estimated
darg	list of arguments to be passed to the density function. Typically, this should be a list with zero or more of the following components: bw, adjust, kernel, window, width, give.Rkern, n, from, to, cut, na.rm (see density for details)
plot.points	logical specifying whether or not the data points should be plotted along with the estimated density. Alternatively, a character string specifying how the points should be plotted. Meaningful values are "rug", in which case panel.rug is used to plot a 'rug', and "jitter", in which case the points are jittered vertically to better distinguish overlapping points.
ref	logical, whether to draw x-axis
groups	an optional grouping variable. If present, panel.superpose will be used instead to display each subgroup
weights	numeric vector of weights for the density calculations. If this is specified, the ... part must also include a subscripts argument that matches the weights to x.
jitter.amount	
	when plot.points="jitter", the value to use as the amount argument to jitter.
type	type argument used to plot points, if requested. This is not expected to be useful, it is available mostly to protect a type argument, if specified, from affecting the density curve.
...	extra graphical parameters. Note that additional arguments to panel.rug cannot be passed on through panel.densityplot.
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

densityplot, jitter

F_1_panel.dotplot *Default Panel Function for dotplot*

Description

Default panel function for dotplot.

Usage

```
panel.dotplot(x, y, horizontal = TRUE,
              pch, col, lty, lwd,
              col.line, levels.fos,
              groups = NULL,
              ...,
              identifier = "dotplot")
```

Arguments

x,y	variables to be plotted in the panel. Typically y is the 'factor'
horizontal	logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of bwplot for a fuller explanation.
pch, col, lty, lwd, col.line	graphical parameters
levels.fos	locations where reference lines will be drawn
groups	grouping variable (affects graphical parameters)
...	extra parameters, passed to panel.xyplot which is responsible for drawing the foreground points (panel.dotplot only draws the background reference lines).
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Details

Creates (possibly grouped) Dotplot of x against y or vice versa

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

dotplot

```
F_1_panel.histogram
```
Default Panel Function for histogram

Description

This is the default panel function for `histogram`.

Usage

```
panel.histogram(x,
                breaks,
                equal.widths = TRUE,
                type = "density",
                nint = round(log2(length(x)) + 1),
                alpha, col, border, lty, lwd,
                ...,
                identifier = "histogram")
```

Arguments

`x`	The data points for which the histogram is to be drawn
`breaks`	The breakpoints for the histogram
`equal.widths`	logical used when `breaks==NULL`
`type`	Type of histogram, possible values being `"percent"`, `"density"` and `"count"`
`nint`	Number of bins for the histogram
`alpha, col, border, lty, lwd`	
	graphical parameters for bars; defaults are obtained from the `plot.polygon` settings.
`...`	other arguments, passed to `hist` when deemed appropriate
`identifier`	A character string that is prepended to the names of grobs that are created by this panel function.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`histogram`

```
F_1_panel.levelplot
```
Panel Functions for levelplot and contourplot

Description

These are the default panel functions for [levelplot](#) and contourplot. Also documented is an alternative raster-based panel function for use with levelplot.

Usage

```
panel.levelplot(x, y, z,
                subscripts,
                at = pretty(z),
                shrink,
                labels,
                label.style = c("mixed", "flat", "align"),
                contour = FALSE,
                region = TRUE,
                col = add.line$col,
                lty = add.line$lty,
                lwd = add.line$lwd,
                border = "transparent",
                ...,
                col.regions = regions$col,
                alpha.regions = regions$alpha,
                identifier = "levelplot")
panel.contourplot(...)

panel.levelplot.raster(x, y, z,
                    subscripts,
                    at = pretty(z),
                    ...,
                    col.regions = regions$col,
                    alpha.regions = regions$alpha,
                    interpolate = FALSE,
                    identifier = "levelplot")
```

Arguments

x, y, z	Variables defining the plot.
subscripts	Integer vector indicating what subset of x, y and z to draw.
at	Numeric vector giving breakpoints along the range of z. See [levelplot](#) for details.

shrink   Either a numeric vector of length 2 (meant to work as both x and y components),
                or a list with components x and y which are numeric vectors of length 2. This
                allows the rectangles to be scaled proportional to the z-value. The specification
                can be made separately for widths (x) and heights (y). The elements of the length
                2 numeric vector gives the minimum and maximum proportion of shrinkage
                (corresponding to min and max of z).

labels   Either a logical scalar indicating whether the labels are to be drawn, or a char-
                acter or expression vector giving the labels associated with the `at` values. Al-
                ternatively, `labels` can be a list with the following components:

                `labels`: a character or expression vector giving the labels. This can be omit-
                        ted, in which case the defaults will be used.
                `col, cex, alpha`: graphical parameters for label texts
                `fontfamily, fontface, font`: font used for the labels

label.style   Controls how label positions and rotation are determined. A value of `"flat"`
                causes the label to be positioned where the contour is flattest, and the label is
                not rotated. A value of `"align"` causes the label to be drawn as far from the
                boundaries as possible, and the label is rotated to align with the contour at that
                point. The default is to mix these approaches, preferring the flattest location
                unless it is too close to the boundaries.

contour   A logical flag, specifying whether contour lines should be drawn.

region   A logical flag, specifying whether inter-contour regions should be filled with
                appropriately colored rectangles.

col, lty, lwd
                graphical parameters for contour lines

border   border color for rectangles used when `region=TRUE`.

...   Extra parameters.

col.regions   A vector of colors, or a function to produce a vecor of colors, to be used if
                `region=TRUE`. Each interval defined by `at` is assigned a color, so the number
                of colors actually used is one less than the length of `at`. See `level.colors`
                for details on how the color assignment is done.

alpha.regions
                numeric scalar controlling transparency of facets

interpolate   logical, passed to `grid.raster`.

identifier   A character string that is prepended to the names of grobs that are created by
                this panel function.

Details

The same panel function is used for both `levelplot` and `contourplot` (which differ only in
default values of some arguments). `panel.contourplot` is a simple wrapper to `panel.levelplot`.

When `contour=TRUE`, the `contourLines` function is used to calculate the contour lines.

`panel.levelplot.raster` is an alternative panel function that uses the raster drawing abil-
ities in R 2.11.0 and higher (through `grid.raster`). It has fewer options (e.g., can only render
data on an equispaced grid), but can be more efficient. When using `panel.levelplot.raster`,
it may be desirable to render the color key in the same way. This is possible, but must be done sep-
arately; see `levelplot` for details.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

levelplot, level.colors, contourLines

Examples

```
require(grid)

levelplot(rnorm(10) ~ 1:10 + sort(runif(10)), panel = panel.levelplot)

suppressWarnings(plot(levelplot(rnorm(10) ~ 1:10 + sort(runif(10)),
                                panel = panel.levelplot.raster,
                                interpolate = TRUE)))

levelplot(volcano, panel = panel.levelplot.raster)

levelplot(volcano, panel = panel.levelplot.raster,
          col.regions = topo.colors, cuts = 30, interpolate = TRUE)
```

F_1_panel.pairs   *Default Superpanel Function for splom*

Description

This is the default superpanel function for splom.

Usage

```
panel.pairs(z,
            panel = lattice.getOption("panel.splom"),
            lower.panel = panel,
            upper.panel = panel,
            diag.panel = "diag.panel.splom",
            as.matrix = FALSE,
            groups = NULL,
            panel.subscripts,
            subscripts,
            pscales = 5,
            prepanel.limits = scale.limits,
            varnames = colnames(z),
            varname.col, varname.cex, varname.font,
            varname.fontfamily, varname.fontface,
            axis.text.col, axis.text.cex, axis.text.font,
```

```
                axis.text.fontfamily, axis.text.fontface,
                axis.text.lineheight,
                axis.line.col, axis.line.lty, axis.line.lwd,
                axis.line.alpha, axis.line.tck,
                ...)
diag.panel.splom(x = NULL,
                varname = NULL, limits, at = NULL, labels = NULL,
                draw = TRUE, tick.number = 5,
                varname.col, varname.cex,
                varname.lineheight, varname.font,
                varname.fontfamily, varname.fontface,
                axis.text.col, axis.text.alpha,
                axis.text.cex, axis.text.font,
                axis.text.fontfamily, axis.text.fontface,
                axis.text.lineheight,
                axis.line.col, axis.line.alpha,
                axis.line.lty, axis.line.lwd,
                axis.line.tck,
                ...)
```

Arguments

z   The data frame used for the plot.

panel, lower.panel, upper.panel
             The panel function used to display each pair of variables. If specified, `lower.panel`
             and `upper.panel` are used for panels below and above the diagonal respec-
             tively.

             In addition to extra arguments not recognized by `panel.pairs`, the list of
             arguments passed to the panel function also includes arguments named `i` and
             `j`, with values indicating the row and column of the scatterplot matrix being
             plotted.

diag.panel   The panel function used for the diagonals. See arguments to `diag.panel.splom`
             to know what arguments this function is passed when called.

as.matrix   logical. If `TRUE`, the layout of the panels will have origin on the top left instead
             of bottom left (similar to `pairs`). This is in essence the same functionality as
             provided by `as.table` for the panel layout

groups   Grouping variable, if any

panel.subscripts
             logical specifying whether the panel function accepts an argument named `subscripts`.

subscripts   The indices of the rows of `z` that are to be displayed in this (super)panel.

pscales   Controls axis labels, passed down from `splom`. If `pscales` is a single number,
             it indicates the approximate number of equally-spaced ticks that should appear
             on each axis. If `pscales` is a list, it should have one component for each
             column in `z`, each of which itself a list with the following valid components:

             `at`: a numeric vector specifying tick locations

             `labels`: character vector labels to go with at

    `limits`: numeric 2-vector specifying axis limits (should be made more flexible at some point to handle factors)

    These are specifications on a per-variable basis, and used on all four sides in the diagonal cells used for labelling. Factor variables are labelled with the factor names. Use `pscales=0` to supress the axes entirely.

`prepanel.limits`

    A function to calculate suitable axis limits given a single argument `x` containing a data vector. The return value of the function should be similar to the `xlim` or `ylim` argument documented in xyplot; that is, it should be a numeric or DateTime vector of length 2 defining a range, or a character vector representing levels of a factor.

    Most high-level lattice plots (such as `xyplot`) use the `prepanel` function for deciding on axis limits from data. This function serves a similar function by calculating the per-variable limits. These limits can be overridden by the corresponding `limits` component in the `pscales` list.

`x`
    data vector corresponding to that row / column (which will be the same for diagonal 'panels').

`varname`
    (scalar) character string or expression that is to be written centred within the panel

`limits`
    numeric of length 2, or, vector of characters, specifying the scale for that panel (used to calculate tick locations when missing)

`at`
    locations of tick marks

`labels`
    optional labels for tick marks

`draw`
    A logical flag specifying whether to draw the tick marks and labels. If `FALSE`, variable names are shown but axis annotation is omitted.

`tick.number`   A Numeric scalar giving the suggested number of tick marks.

`varnames`
    A character or expression vector or giving names to be used for the variables in `x`. By default, the column names of `x`.

`varname.col`   Color for the variable name in each diagonal panel. See gpar for details on this and the other graphical parameters listed below.

`varname.cex`   Size multiplier for the variable name in each diagonal panel.

`varname.lineheight`
    Line height for the variable name in each diagonal panel.

`varname.font, varname.fontfamily, varname.fontface`
    Font specification for the variable name in each diagonal panel.

`axis.text.col`
    Color for axis label text.

`axis.text.cex`
    Size multiplier for axis label text.

`axis.text.font, axis.text.fontfamily, axis.text.fontface`
    Font specification for axis label text.

`axis.text.lineheight`
    Line height for axis label text.

```
axis.text.alpha
```
               Alpha-transparency for axis label text.

```
axis.line.col
```
               Color for the axes.

```
axis.line.lty
```
               Line type for the axes.

```
axis.line.lwd
```
               Line width for the axes.

```
axis.line.alpha
```
               Alpha-transparency for the axes.

```
axis.line.tck
```
               A numeric multiplier for the length of tick marks in diagonal panels.

```
...
```
               Further arguments, passed on to `panel`, `lower.panel`, `upper.panel`, and `diag.panel` from `panel.pairs`. Currently ignored by `diag.panel.splom`.

Details

`panel.pairs` is the function that is actually used as the panel function in a `"trellis"` object produced by `splom`.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

splom

Examples

```
Cmat <- outer(1:6,1:6,
              function(i,j) rainbow(11, start=.12, end=.5)[i+j-1])

splom(~diag(6), as.matrix = TRUE,
      panel = function(x, y, i, j, ...) {
          panel.fill(Cmat[i,j])
          panel.text(.5,.5, paste("(",i,",",j,")",sep=""))
      })
```

`F_1_panel.parallel` *Default Panel Function for parallel*

Description

This is the default panel function for `parallel`.

Usage

```
panel.parallel(x, y, z, subscripts,
               groups = NULL,
               col, lwd, lty, alpha,
               common.scale = FALSE,
               lower,
               upper,
               ...,
               horizontal.axis = TRUE,
               identifier = "parallel")
```

Arguments

x, y   dummy variables, ignored.

z   The data frame used for the plot. Each column will be coerced to numeric before being plotted, and an error will be issued if this fails.

subscripts   The indices of the rows of z that are to be displyed in this panel.

groups   An optional grouping variable. If specified, different groups are distinguished by use of different graphical parameters (i.e., rows of z in the same group share parameters).

col, lwd, lty, alpha
                graphical parameters (defaults to the settings for `superpose.line`). If `groups` is non-null, these parameters used one for each group. Otherwise, they are recycled and used to distinguish between rows of the data frame z.

common.scale   logical, whether a common scale should be used columns of z. Defaults to `FALSE`, in which case the horizontal range for each column is different (as determined by `lower` and `upper`).

lower, upper   numeric vectors replicated to be as long as the number of columns in z. Determines the lower and upper bounds to be used for scaling the corresponding columns of z after coercing them to numeric. Defaults to the minimum and maximum of each column. Alternatively, these could be functions (to be applied on each column) that return a scalar.

...   other arguments (ignored)

horizontal.axis
                logical indicating whether the parallel axes should be laid out horizontally (`TRUE`) or vertically (`FALSE`).

identifier   A character string that is prepended to the names of grobs that are created by this panel function.

Details

Produces parallel coordinate plots, which are easier to understand from an example than through a
verbal description. See example for parallel

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

References

Inselberg, Alfred (2009) *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*, Springer. ISBN: 978-0-387-21507-5.

Inselberg, A. (1985) "The Plane with Parallel Coordinates", *The Visual Computer*.

See Also

parallel

F_1_panel.qqmath   *Default Panel Function for qqmath*

Description

This is the default panel function for qqmath.

Usage

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

Arguments

x   vector (typically numeric, coerced if not) of data values to be used in the panel.

f.value, distribution
                       Defines how quantiles are calculated. See qqmath for details.

qtype   The type argument to be used in quantile

groups   An optional grouping variable. Within each panel, one Q-Q plot is produced
                       for every level of this grouping variable, differentiated by different graphical
                       parameters.

...   Further arguments, often graphical parameters, eventually passed on to panel.xyplot.
                       Arguments grid and abline of panel.xyplot may be particularly useful.

tails.n	number of data points to represent exactly on each tail of the distribution. This reproduces the effect of f.value = NULL for the extreme data values, while approximating the remaining data. It has no effect if f.value = NULL. If tails.n is given, qtype is forced to be 1.
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Details

Creates a Q-Q plot of the data and the theoretical distribution given by distribution. Note that most of the arguments controlling the display can be supplied directly to the high-level qqmath call.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[qqmath](#)

Examples

```
set.seed(0)
xx <- rt(10000, df = 10)
qqmath(~ xx, pch = "+", distribution = qnorm,
       grid = TRUE, abline = c(0, 1),
       xlab.top = c("raw", "ppoints(100)", "tails.n = 50"),
       panel = function(..., f.value) {
           switch(panel.number(),
                   panel.qqmath(..., f.value = NULL),
                   panel.qqmath(..., f.value = ppoints(100)),
                   panel.qqmath(..., f.value = ppoints(100), tails.n = 50))
       }, layout = c(3, 1))[c(1,1,1)]
```

F_1_panel.stripplot

Default Panel Function for stripplot

Description

This is the default panel function for stripplot. Also see panel.superpose

Usage

```
panel.stripplot(x, y, jitter.data = FALSE,
                factor = 0.5, amount = NULL,
                horizontal = TRUE, groups = NULL,
                ...,
                identifier = "stripplot")
```

Arguments

`x, y`	coordinates of points to be plotted
`jitter.data`	whether points should be jittered to avoid overplotting. The actual jittering is performed inside `panel.xyplot`, using its `jitter.x` or `jitter.y` argument (depending on the value of `horizontal`).
`factor, amount`	
	amount of jittering, see `jitter`
`horizontal`	logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of `bwplot` for a fuller explanation.
`groups`	optional grouping variable
`...`	additional arguments, passed on to `panel.xyplot`
`identifier`	A character string that is prepended to the names of grobs that are created by this panel function.

Details

Creates stripplot (one dimensional scatterplot) of `x` for each level of `y` (or vice versa, depending on the value of `horizontal`)

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`stripplot`, `jitter`

F_1_panel.xyplot   *Default Panel Function for xyplot*

Description

This is the default panel function for `xyplot`. Also see `panel.superpose`. The default panel functions for `splom` and `qq` are essentially the same function.

Usage

```
panel.xyplot(x, y, type = "p",
             groups = NULL,
             pch, col, col.line, col.symbol,
             font, fontfamily, fontface,
             lty, cex, fill, lwd,
             horizontal = FALSE, ...,
             grid = FALSE, abline = NULL,
```

```
                jitter.x = FALSE, jitter.y = FALSE,
                factor = 0.5, amount = NULL,
                identifier = "xyplot")
panel.splom(..., identifier = "splom")
panel.qq(..., identifier = "qq")
```

Arguments

x,y   variables to be plotted in the scatterplot

type   character vector consisting of one or more of the following: `"p"`, `"l"`, `"h"`,
                 `"b"`, `"o"`, `"s"`, `"S"`, `"r"`, `"a"`, `"g"`, `"smooth"`. If type has more than one
                 element, an attempt is made to combine the effect of each of the components.

                 The behaviour if any of the first six are included in type is similar to the effect
                 of type in plot (type `"b"` is actually the same as `"o"`). `"r"` adds a regres-
                 sion line (same as panel.lmline, except for default graphical parameters),
                 and `"smooth"` adds a lowess fit (same as panel.loess). `"g"` adds a refer-
                 ence grid using panel.grid in the background. `"a"` has the effect of calling
                 panel.average, which can be useful for creating interaction plots. The ef-
                 fect of several of these specifications depend on the value of horizontal.

                 Type `"s"` (and `"S"`) sorts the values along one of the axes (depending on
                 horizontal); this is unlike the behavior in plot. For the latter behavior,
                 use type = `"s"` with panel = panel.points.

                 See example(xyplot) and demo(lattice) for examples.

groups   an optional grouping variable. If present, panel.superpose will be used
                 instead to display each subgroup

col, col.line, col.symbol
                 default colours are obtained from plot.symbol and plot.line using trellis.par.get.

font, fontface, fontfamily
                 font used when pch is a character

pch, lty, cex, lwd, fill
                 other graphical parameters. fill serves the purpose of bg in points for
                 certain values of pch

horizontal   A logical flag controlling the orientation for certain type's, e.g., `"h"`, `"s"`, ans
                 `"S"`.

...   Extra arguments, if any, for panel.xyplot. In most cases panel.xyplot
                 ignores these. For types "r" and "smooth", these are passed on to panel.lmline
                 and panel.loess respectively.

grid   A logical flag, character string, or list specifying whether and how a background
                 grid should be drawn. This provides the same functionality as type="g", but
                 is the preferred alternative as the effect type="g" is conceptually different
                 from that of other type values (which are all data-dependent). Using the grid
                 argument also allows more flexibility.

                 Most generally, grid can be a list of arguments to be supplied to panel.grid,
                 which is called with those arguments. Three shortcuts are available:

                 TRUE: roughly equivalent to list(h = -1, v = -1)
                 `"h"`: roughly equivalent to list(h = -1, v = 0)

> "v": roughly equivalent to `list(h = 0, v = -1)`
>
> No grid is drawn if `grid = FALSE`.

abline   A numeric vector or list, specifying arguments arguments for `panel.abline`,
                which is called with those arguments. If specified as a (possibly named) numeric
                vector, `abline` is coerced to a list. This allows arguments of the form `abline`
                `= c(0, 1)`, which adds the diagonal line, or `abline = c(h = 0, v =`
                `0)`, which adds the x- and y-axes to the plot. Use the list form for finer control;
                e.g., `abline = list(h = 0, v = 0, col = "grey")`.

                For more flexibility, use `panel.abline` directly.

jitter.x, jitter.y
                logical, whether the data should be jittered before being plotted.

factor, amount
                controls amount of jittering.

identifier   A character string that is prepended to the names of grobs that are created by
                this panel function.

Details

Creates scatterplot of `x` and `y`, with various modifications possible via the type argument. `panel.qq`
draws a 45 degree line before calling `panel.xyplot`.

Note that most of the arguments controlling the display can be supplied directly to the high-level
(e.g. `xyplot`) call.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`panel.superpose`, `xyplot`, `splom`

Examples

```
types.plain <- c("p", "l", "o", "r", "g", "s", "S", "h", "a", "smooth")
types.horiz <- c("s", "S", "h", "a", "smooth")
horiz <- rep(c(FALSE, TRUE), c(length(types.plain), length(types.horiz)))

types <- c(types.plain, types.horiz)

x <- sample(seq(-10, 10, length.out = 15), 30, TRUE)
y <- x + 0.25 * (x + 1)^2 + rnorm(length(x), sd = 5)

xyplot(y ~ x | gl(1, length(types)),
       xlab = "type",
       ylab = list(c("horizontal=TRUE", "horizontal=FALSE"), y = c(1/6, 4/6)),
       as.table = TRUE, layout = c(5, 3),
       between = list(y = c(0, 1)),
```

```
        strip = function(...) {
            panel.fill(trellis.par.get("strip.background")$col[1])
            type <- types[panel.number()]
            grid::grid.text(label = sprintf('"%s"', type),
                            x = 0.5, y = 0.5)
            grid::grid.rect()
        },
        scales = list(alternating = c(0, 2), tck = c(0, 0.7), draw = FALSE),
        par.settings =
        list(layout.widths = list(strip.left = c(1, 0, 0, 0, 0))),
        panel = function(...) {
            type <- types[panel.number()]
            horizontal <- horiz[panel.number()]
            panel.xyplot(...,
                         type = type,
                         horizontal = horizontal)
        })[rep(1, length(types))]
```

F_2_llines   *Replacements of traditional graphics functions*

Description

These functions are intended to replace common low level traditional graphics functions, primarily
for use in panel functions. The originals can not be used (at least not easily) because lattice panel
functions need to use grid graphics. Low level drawing functions in grid can be used directly as
well, and is often more flexible. These functions are provided for convenience and portability.

Usage

```
lplot.xy(xy, type, pch, lty, col, cex, lwd,
         font, fontfamily, fontface,
         col.line, col.symbol, alpha, fill,
         origin = 0, ..., identifier, panel)

llines(x, ...)
lpoints(x, ...)
ltext(x, ...)

## Default S3 method:
llines(x, y = NULL, type = "l",
       col, alpha, lty, lwd, ..., identifier, panel)
## Default S3 method:
lpoints(x, y = NULL, type = "p", col, pch, alpha, fill,
        font, fontfamily, fontface, cex, ..., identifier, panel)
## Default S3 method:
ltext(x, y = NULL, labels = seq_along(x),
```

```
        col, alpha, cex, srt = 0,
        lineheight, font, fontfamily, fontface,
        adj = c(0.5, 0.5), pos = NULL, offset = 0.5, ..., identifier, panel)

  lsegments(x0, y0, x1, y1, x2, y2,
            col, alpha, lty, lwd,
            font, fontface, ..., identifier, panel)
lrect(xleft, ybottom, xright, ytop,
      x = (xleft + xright) / 2,
      y = (ybottom + ytop) / 2,
      width = xright - xleft,
      height = ytop - ybottom,
      col = "transparent",
      border = "black",
      lty = 1, lwd = 1, alpha = 1,
      just = "center",
      hjust = NULL, vjust = NULL,
      ..., identifier, panel)
larrows(x0 = NULL, y0 = NULL, x1, y1, x2 = NULL, y2 = NULL,
        angle = 30, code = 2, length = 0.25, unit = "inches",
        ends = switch(code, "first", "last", "both"),
        type = "open",
        col = add.line$col,
        alpha = add.line$alpha,
        lty = add.line$lty,
        lwd = add.line$lwd,
        fill = NULL, ..., identifier, panel)
lpolygon(x, y = NULL,
         border = "black", col = "transparent", fill = NULL,
         font, fontface, ..., identifier, panel)

panel.lines(...)
panel.points(...)
panel.segments(...)
panel.text(...)
panel.rect(...)
panel.arrows(...)
panel.polygon(...)
```

Arguments

  x, y, x0, y0, x1, y1, x2, y2, xy
               locations. x2 and y2 are available for for S compatibility.

length, unit determines extent of arrow head. length specifies the length in terms of unit,
               which can be any valid grid unit as long as it doesn't need a data argument.
               unit defaults to inches, which is the only option in the base version of the
               function, arrows.

angle, code, type, labels, srt, adj, pos, offset
               arguments controlling behaviour. See respective base functions for details. For

	`larrows` and `panel.larrows`, `type` is either `"open"` or `"closed"`, indicating the type of arrowhead.
ends	serves the same function as `code`, using descriptive names rather than integer codes. If specified, this overrides `code`

`col, alpha, lty, lwd, fill, pch, cex, lineheight, font, fontfamily, fontface, col.l`

graphical parameters. `fill` applies to points when `pch` is in `21:25` and specifies the fill color, similar to the `bg` argument in the base graphics function `points`. For devices that support alpha-transparency, a numeric argument `alpha` between 0 and 1 can controls transparency. Be careful with this, since for devices that do not support alpha-transparency, nothing will be drawn at all if this is set to anything other than 0.

`fill`, `font` and `fontface` are included in `lpolygon` and `lsegments` only to ensure that they are not passed down (as `gpar` does not like them).

origin	for `type="h"` or `type="H"`, the value to which lines drop down.

`xleft, ybottom, xright, ytop`

    see `rect`

`width, height, just, hjust, vjust`

    finer control over rectangles, see `grid.rect`

...	extra arguments, passed on to lower level functions as appropriate.
identifier	A character string that is prepended to the name of the grob that is created.
panel	A logical value indicating whether the grob is specific to a particular panel.

Details

These functions are meant to be grid replacements of the corresponding base R graphics functions, to allow existing Trellis code to be used with minimal modification. The functions `panel.*` are essentially identical to the `l*` versions, are recommended for use in new code (as opposed to ported code) as they have more readable names.

See the documentation of the base functions for usage. Not all arguments are always supported. All these correspond to the default methods only.

Note

There is a new `type="H"` option wherever appropriate, which is similar to `type="h"`, but with horizontal lines.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

`points`, `lines`, `rect`, `text`, `segments`, `arrows`, `Lattice`

```
F_2_panel.functions
```
Useful Panel Function Components

Description

These are predefined panel functions available in lattice for use in constructing new panel functions (often on-the-fly).

Usage

```
panel.abline(a = NULL, b = 0,
             h = NULL, v = NULL,
             reg = NULL, coef = NULL,
             col, col.line, lty, lwd, alpha, type,
             ...,
             reference = FALSE,
             identifier = "abline")
panel.refline(...)

panel.curve(expr, from, to, n = 101,
            curve.type = "l",
            col, lty, lwd, type,
            ...,
            identifier = "curve")
panel.rug(x = NULL, y = NULL,
          regular = TRUE,
          start = if (regular) 0 else 0.97,
          end = if (regular) 0.03 else 1,
          x.units = rep("npc", 2),
          y.units = rep("npc", 2),
          col, col.line, lty, lwd, alpha,
          ...,
          identifier = "rug")
panel.average(x, y, fun = mean, horizontal = TRUE,
              lwd, lty, col, col.line, type,
              ...,
              identifier = "linejoin")
panel.linejoin(x, y, fun = mean, horizontal = TRUE,
               lwd, lty, col, col.line, type,
               ...,
               identifier = "linejoin")
```

```
panel.fill(col, border, ..., identifier = "fill")
panel.grid(h=3, v=3, col, col.line, lty, lwd, x, y, ..., identifier = "grid")
panel.lmline(x, y, ..., identifier = "lmline")
panel.loess(x, y, span = 2/3, degree = 1,
            family = c("symmetric", "gaussian"),
            evaluation = 50,
            lwd, lty, col, col.line, type,
            horizontal = FALSE,
            ..., identifier = "loess")
panel.mathdensity(dmath = dnorm, args = list(mean=0, sd=1),
                  n = 50, col, col.line, lwd, lty, type,
                  ..., identifier = "mathdensity")
```

Arguments

x, y	variables defining the contents of the panel. In `panel.grid` these are optional and are used only to choose an appropriate method of `pretty`.
a, b	Coefficients of the line to be added by `panel.abline`. `a` can be a vector of length 2, representing the coefficients of the line to be added, in which case `b` should be missing. `a` can also be an appropriate 'regression' object, i.e., an object which has a `coef` method that returns a length 2 numeric vector. The corresponding line will be plotted. The `reg` argument will override `a` if specified.
coef	Coefficients of the line to be added as a length 2 vector
reg	A regression object. The corresponding fitted line will be drawn
h, v	For `panel.abline`, these are numeric vectors giving locations respectively of horizontal and vertical lines to be added to the plot, in native coordinates. For `panel.grid`, these usually specify the number of horizontal and vertical reference lines to be added to the plot. Alternatively, they can be negative numbers. `h=-1` and `v=-1` are intended to make the grids aligned with the axis labels. This doesn't always work; all that actually happens is that the locations are chosen using `pretty`, which is also how the label positions are chosen in the most common cases (but not for factor variables, for instance). `h` and `v` can be negative numbers other than −1, in which case −h and −v (as appropriate) is supplied as the `n` argument to `pretty`.
reference	logical indicating whether the default graphical parameters for `panel.abline` should be taken from the "reference.line" parameter settings. The default is to take them from the "add.line" settings. The `panel.refline` function is a wrapper around `panel.abline` that calls it with `reference=TRUE`.
expr	expression as a function of x or a function to plot as a curve
n	the number of points to use for drawing the curve
regular	logical indicating whether the 'rug' is to be drawn on the regular side (left / bottom) or not (right / top)
start, end	endpoints of rug segments, in normalized parent coordinates (between 0 and 1). Defaults depend on value of regular, and cover 3% of the panel width and height

x.units, y.units

     character vector, replicated to be of length two. Specifies the (grid) units associated with `start` and `end` above. `x.units` and `y.units` are for the rug on the x-axis and y-axis respectively (and thus are associated with `start` and `end` values on the y and x scales respectively).

from, to   optional lower and upper x-limits of curve. If missing, limits of current panel are used

curve.type   type of curve (`"p"` for points, etc), passed to llines

col, col.line, lty, lwd, alpha, border

     graphical parameters

type   Usually ignored by the panel functions documented here; the argument is present only to make sure an explicitly specified `type` argument (perhaps meant for another function) doesn't affect the display.

span, degree, family, evaluation

     arguments to `loess.smooth`, for which `panel.loess` is essentially a wrapper.

fun   the function that will be applied to the subset of x(y) determined by the unique values of y(x)

horizontal   logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. Interpretation of other arguments change accordingly. See documentation of bwplot for a fuller explanation.

dmath   A vectorized function that produces density values given a numeric vector named x, e.g., `dnorm`

args   list giving additional arguments to be passed to dmath

...   graphical parameters can be supplied. see function definition for details. Color can usually be specified by `col`, `col.line` and `col.symbol`, the last two overriding the first for lines and points respectively.

identifier   A character string that is prepended to the names of grobs that are created by this panel function.

Details

  `panel.abline` adds a line of the form `y=a+bx` or vertical and/or horizontal lines. Graphical parameters are obtained from the "add.line" settings by default. `panel.refline` is similar, but uses the "reference.line" settings for the defaults.

  `panel.grid` draws a reference grid.

  `panel.curve` adds a curve, similar to what `curve` does with `add = TRUE`. Graphical parameters for the line are obtained from the `add.line` setting.

  `panel.average` treats one of x and y as a factor (according to the value of `horizontal`), calculates `fun` applied to the subsets of the other variable determined by each unique value of the factor, and joins them by a line. Can be used in conjunction with `panel.xyplot` and more commonly with panel.superpose to produce interaction plots. See xyplot documentation for an example. `panel.linejoin` is an alias for `panel.average` retained for back-compatibility and may go away in future.

`panel.mathdensity` plots a (usually theoretical) probability density function. This can be useful in conjunction with `histogram` and `densityplot` to visually estimate goodness of fit (note, however, that `qqmath` is more suitable for this).

`panel.rug` adds a *rug* representation of the (marginal) data to the panel, much like `rug`.

`panel.lmline(x, y)` is equivalent to `panel.abline(lm(y~x))`.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

`loess.smooth`, `panel.axis`, `panel.identify identify`, `trellis.par.set`

F_2_panel.qqmathline

Useful panel function with qqmath

Description

Useful panel function with qqmath. Draws a line passing through the points (usually) determined by the .25 and .75 quantiles of the sample and the theoretical distribution.

Usage

```
panel.qqmathline(x, y = x,
                 distribution = qnorm,
                 probs = c(0.25, 0.75),
                 qtype = 7,
                 groups = NULL,
                 ...,
                 identifier = "qqmathline")
```

Arguments

x	The original sample, possibly reduced to a fewer number of quantiles, as determined by the `f.value` argument to `qqmath`
y	an alias for x for backwards compatibility
distribution	quantile function for reference theoretical distribution.
probs	numeric vector of length two, representing probabilities. Corresponding quantile pairs define the line drawn.
qtype	the `type` of quantile computation used in `quantile`
groups	optional grouping variable. If non-null, a line will be drawn for each group.
...	other arguments.
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

[`prepanel.qqmathline`](), [`qqmath`](), [`quantile`]()

`F_2_panel.smoothScatter`

Lattice panel function analogous to smoothScatter

Description

This function allows the user to place `smoothScatter` plots in lattice graphics.

Usage

```
panel.smoothScatter(x, y = NULL,
                    nbin = 64, cuts = 255,
                    bandwidth,
                    colramp,
                    nrpoints = 100,
                    transformation = function(x) x^0.25,
                    pch = ".",
                    cex = 1, col="black",
                    range.x,
                    ...,
                    raster = FALSE,
                    subscripts,
                    identifier = "smoothScatter")
```

Arguments

x	Numeric vector containing x-values or n by 2 matrix containing x and y values.
y	Numeric vector containing y-values (optional). The length of x must be the same as that of y.
nbin	Numeric vector of length 1 (for both directions) or 2 (for x and y separately) containing the number of equally spaced grid points for the density estimation.
cuts	number of cuts defining the color gradient
bandwidth	Numeric vector: the smoothing bandwidth. If missing, these functions come up with a more or less useful guess. This parameter then gets passed on to the function [`bkde2D`]().
colramp	Function accepting an integer n as an argument and returning n colors.

nrpoints	Numeric vector of length 1 giving number of points to be superimposed on the density image. The first nrpoints points from those areas of lowest regional densities will be plotted. Adding points to the plot allows for the identification of outliers. If all points are to be plotted, choose nrpoints = Inf.
transformation	
	Function that maps the density scale to the color scale.
pch, cex	graphical parameters for the nrpoints "outlying" points shown in the display
range.x	see bkde2D for details.
col	points color parameter
...	Further arguments that are passed on to panel.levelplot.
raster	logical; if TRUE, panel.levelplot.raster is used, making potentially smaller output files.
subscripts	ignored, but necessary for handling of ...in certain situations. Likely to be removed in future.
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Details

This replicates the display part of the smoothScatter function by replacing standard graphics calls by grid-compatible ones.

Value

The function is called for its side effects, namely the production of the appropriate plots on a graphics device.

Author(s)

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

Examples

```
ddf <- as.data.frame(matrix(rnorm(40000), ncol = 4) + 3 * rnorm(10000))
ddf[, c(2,4)] <- (-ddf[, c(2,4)])
xyplot(V1 ~ V2 + V3, ddf, outer = TRUE,
       panel = panel.smoothScatter, aspect = "iso")
splom(ddf, panel = panel.smoothScatter, nbin = 64, raster = TRUE)
```

`F_2_panel.superpose`

Panel Function for Display Marked by groups

Description

These are panel functions for Trellis displays useful when a grouping variable is specified for use within panels. The x (and y where appropriate) variables are plotted with different graphical parameters for each distinct value of the grouping variable.

Usage

```
panel.superpose(x, y = NULL, subscripts, groups,
                panel.groups = "panel.xyplot",
                ...,
                col, col.line, col.symbol,
                pch, cex, fill, font,
                fontface, fontfamily,
                lty, lwd, alpha,
                type = "p", grid = FALSE,
                distribute.type = FALSE)
panel.superpose.2(..., distribute.type = TRUE)

panel.superpose.plain(...,
                      col, col.line, col.symbol,
                      pch, cex, fill, font,
                      fontface, fontfamily,
                      lty, lwd, alpha)
```

Arguments

x,y	Coordinates of the points to be displayed. Usually numeric.
panel.groups	The panel function to be used for each subgroup of points. Defaults to panel.xyplot. To be able to distinguish between different levels of the originating group inside panel.groups, it will be supplied two special arguments called group.number and group.value which will hold the numeric code and factor level corresponding to the current level of groups. No special care needs to be taken when writing a panel.groups function if this feature is not used.
subscripts	An integer vector of subscripts giving indices of the x and y values in the original data source. See the corresponding entry in xyplot for details.
groups	A grouping variable. Different graphical parameters will be used to plot the subsets of observations given by each distinct value of groups. The default graphical parameters are obtained from the "superpose.symbol" and "superpose.line" settings using trellis.par.get wherever appropriate.

type   Usually a character vector specifying how each group should be drawn. For-
            mally, it is passed on to the panel.groups function, which must know what
            to do with it. By default, panel.groups is panel.xyplot, whose help
            page describes the admissible values.

            The functions panel.superpose and panel.superpose.2 differ only
            in the default value of distribute.type, which controls the way the type
            argument is interpreted. If distribute.type = FALSE, then the interpre-
            tation is the same as for panel.xyplot for each of the unique groups. In
            other words, if type is a vector, all the individual components are honoured
            concurrently. If distribute.type = TRUE, type is replicated to be as
            long as the number of unique values in groups, and one component used for
            the points corresponding to the each different group. Even in this case, it is
            possible to request multiple types per group, specifying type as a list, each
            component being the desired type vector for the corresponding group.

            If distribute.type = FALSE, any occurrence of "g" in type causes a
            grid to be drawn, and all such occurrences are removed before type is passed
            on to panel.groups.

grid   Logical flag specifying whether a background reference grid should be drawn.
            See panel.xyplot for details.

col   A vector color specification. See Details.

col.line   A vector color specification. See Details.

col.symbol A vector color specification. See Details.

pch   A vector plotting character specification. See Details.

cex   A vector size factor specification. See Details.

fill   A vector fill color specification. See Details.

font, fontface, fontfamily
            A vector color specification. See Details.

lty   A vector color specification. See Details.

lwd   A vector color specification. See Details.

alpha   A vector alpha-transparency specification. See Details.

...   Extra arguments. Passed down to panel.superpose from panel.superpose.2,
            and to panel.groups from panel.superpose.

distribute.type
            logical controlling interpretation of the type argument.

Details

panel.superpose divides up the x (and optionally y) variable(s) by the unique values of
groups[subscripts], and plots each subset with different graphical parameters. The graphi-
cal parameters (col.symbol, pch, etc.) are usually supplied as suitable atomic vectors, but can
also be lists. When panel.groups is called for the i-th level of groups, the corresponding
element of each graphical parameter is passed to it. In the list form, the individual components can
themselves be vectors.

The actual plot for each subgroup is created by the panel.groups function. With the default
panel.groups, the col argument is overridden by col.line and col.symbol for lines and

points respectively, which default to the `"superpose.line"` and `"superpose.symbol"` settings. However, `col` will still be supplied as an argument to `panel.groups` functions that make use of it, with a default of `"black"`. The defaults of other graphical parameters are also taken from the `"superpose.line"` and `"superpose.symbol"` settings as appropriate. The `alpha` parameter takes it default from the `"superpose.line"` setting.

`panel.superpose` and `panel.superpose.2` differ essentially in how `type` is interpreted by default. The default behaviour in `panel.superpose` is the opposite of that in S, which is the same as that of `panel.superpose.2`.

`panel.superpose.plain` is the same as `panel.superpose`, except that the default settings for the style arguments are the same for all groups and are taken from the default plot style. It is used in `xyplot.ts`.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org> (`panel.superpose.2` originally contributed by Neil Klepeis)

See Also

Different functions when used as `panel.groups` gives different types of plots, for example `panel.xyplot`, `panel.dotplot` and `panel.linejoin` (This can be used to produce interaction plots).

See `Lattice` for an overview of the package.

F_2_panel.violin   *Panel Function to create Violin Plots*

Description

This is a panel function that can create a violin plot. It is typically used in a high-level call to `bwplot`.

Usage

```
panel.violin(x, y, box.ratio = 1, box.width,
             horizontal = TRUE,
             alpha, border, lty, lwd, col,
             varwidth = FALSE,
             bw, adjust, kernel, window,
             width, n = 50, from, to, cut,
             na.rm, ...,
             identifier = "violin")
```

Arguments

x, y	numeric vector or factor. Violin plots are drawn for each unique value of y (x) if horizontal is TRUE (FALSE)
box.ratio	ratio of the thickness of each violin and inter violin space
box.width	thickness of the violins in absolute units; overrides box.ratio. Useful for specifying thickness when the categorical variable is not a factor, as use of box.ratio alone cannot achieve a thickness greater than 1.
horizontal	logical. If FALSE, the plot is 'transposed' in the sense that the behaviours of x and y are switched. x is now the 'factor'. See documentation of bwplot for a fuller explanation.
alpha, border, lty, lwd, col	
	graphical parameters controlling the violin. Defaults are taken from the "plot.polygon" settings.
varwidth	logical. If FALSE, the densities are scaled separately for each group, so that the maximum value of the density reaches the limit of the allocated space for each violin (as determined by box.ratio). If TRUE, densities across violins will have comparable scale.
bw, adjust, kernel, window, width, n, from, to, cut, na.rm	
	arguments to density, passed on as appropriate
...	arguments passed on to density.
identifier	A character string that is prepended to the names of grobs that are created by this panel function.

Details

Creates Violin plot of x for every level of y. Note that most arguments controlling the display can be supplied to the high-level (typically bwplot) call directly.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

bwplot, density

Examples

```
bwplot(voice.part ~ height, singer,
       panel = function(..., box.ratio) {
           panel.violin(..., col = "transparent",
                        varwidth = FALSE, box.ratio = box.ratio)
           panel.bwplot(..., fill = NULL, box.ratio = .1)
       } )
```

```
F_3_prepanel.default
```
                                    Default Prepanel Functions

---

Description

These prepanel functions are used as fallback defaults in various high level plot functions in Lattice.
These are rarely useful to normal users but may be helpful in developing new displays.

Usage

```
prepanel.default.bwplot(x, y, horizontal, nlevels, origin, stack, ...)
prepanel.default.histogram(x, breaks, equal.widths, type, nint, ...)
prepanel.default.qq(x, y, ...)
prepanel.default.xyplot(x, y, type, subscripts, groups, ...)
prepanel.default.cloud(perspective, distance,
                       xlim, ylim, zlim,
                       screen = list(z = 40, x = -60),
                       R.mat = diag(4),
                       aspect = c(1, 1), panel.aspect = 1,
                       ..., zoom = 0.8)
prepanel.default.levelplot(x, y, subscripts, ...)
prepanel.default.qqmath(x, f.value, distribution, qtype,
                        groups, subscripts, ..., tails.n = 0)
prepanel.default.densityplot(x, darg, groups, weights, subscripts, ...)
prepanel.default.parallel(x, y, z, ..., horizontal.axis)
prepanel.default.splom(z, ...)
```

Arguments

x, y   x and y values, numeric or factor

horizontal   logical, applicable when one of the variables is to be treated as categorical (factor
                or shingle).

horizontal.axis

                logical indicating whether the parallel axes should be laid out horizontally (TRUE)
                or vertically (FALSE).

nlevels   number of levels of such a categorical variable.

origin, stack

                for barcharts or the type="h" plot type

breaks, equal.widths, type, nint

                details of histogram calculations. type has a different meaning in prepanel.default.xyplot
                (see panel.xyplot)

groups, subscripts

See [xyplot](). Whenever appropriate, calculations are done separately for each group and then combined.

weights   numeric vector of weights for the density calculations. If this is specified, it is subsetted by subscripts to match it to x.

perspective, distance, xlim, ylim, zlim, screen, R.mat, aspect, panel.aspect, zoom

see [panel.cloud]()

f.value, distribution, tails.n

see panel.qqmath

darg   list of arguments passed to [density]()

z   see [panel.parallel]() and [panel.pairs]()

qtype   type of [quantile]()

...   other arguments, usually ignored

Value

A list with components xlim, ylim, dx and dy, and possibly xat and yat, the first two being used to calculate panel axes limits, the last two for banking computations. The form of these components are described in the help page for [xyplot]().

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[xyplot](), [banking](), [Lattice](). See documentation of corresponding panel functions for more details about the arguments.

F_3_prepanel.functions
Useful Prepanel Function for Lattice

Description

These are predefined prepanel functions available in Lattice.

Usage

```
prepanel.lmline(x, y, ...)
prepanel.loess(x, y, span, degree, family, evaluation, ...)
prepanel.qqmathline(x, y = x, distribution = qnorm,
                    probs = c(0.25, 0.75), qtype = 7,
                    groups, subscripts,
                    ...)
```

Arguments

x, y   x and y values, numeric or factor

distribution   quantile function for theoretical distribution. This is automatically passed in
                when this is used as a prepanel function in qqmath.

qtype   type of quantile

probs   numeric vector of length two, representing probabilities. If used with aspect="xy",
                the aspect ratio will be chosen to make the line passing through the correspond-
                ing quantile pairs as close to 45 degrees as possible.

span, degree, family, evaluation
                arguments controlling the underlying loess smooth

groups, subscripts
                See xyplot. Whenever appropriate, calculations are done separately for each
                group and then combined.

...   other arguments

Value

usually a list with components xlim, ylim, dx and dy, the first two being used to calculate panel
axes limits, the last two for banking computations. The form of these components are described
under xyplot. There are also several prepanel functions that serve as the default for high level
functions, see prepanel.default.xyplot

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

trellis.par.get, xyplot, banking, Lattice. See loess.smooth for further options
to prepanel.loess

G_axis.default   *Default axis annotation utilities*

Description

Lattice funtions provide control over how the plot axes are annotated through a common interface.
There are two levels of control. The xscale.components and yscale.components ar-
guments can be functions that determine tick mark locations and labels given a packet. For more
direct control, the axis argument can be a function that actually draws the axes. The functions
documented here are the defaults for these arguments. They can additonally be used as components
of user written replacements.

Usage

```
xscale.components.default(lim,
                          packet.number = 0,
                          packet.list = NULL,
                          top = TRUE,
                          ...)
yscale.components.default(lim,
                          packet.number = 0,
                          packet.list = NULL,
                          right = TRUE,
                          ...)
axis.default(side = c("top", "bottom", "left", "right"),
             scales, components, as.table,
             labels = c("default", "yes", "no"),
             ticks = c("default", "yes", "no"),
             ..., prefix)
```

Arguments

`lim`	the range of the data in that packet (data subset corresponding to a combination of levels of the conditioning variable). The range is not necessarily numeric; e.g. for factors, they could be character vectors representing levels, and for the various date-time representations, they could be vectors of length 2 with the corresponding class.
`packet.number`	which packet (counted according to the packet order, described in [`print.trellis`](#)) is being processed. In cases where all panels have the same limits, this function is called only once (rather than once for each packet), in which case this argument will have the value `0`.
`packet.list`	list, as long as the number of packets, giving all the actual packets. Specifically, each component is the list of arguments given to the panel function when and if that packet is drawn in a panel. (This has not yet been implemented.)
`top, right`	the value of the `top` and `right` components of the result, as appropriate. See below for interpretation.
`side`	on which side the axis is to be drawn. The usual partial matching rules apply.
`scales`	the appropriate component of the `scales` argument supplied to the high level function, suitably standardized.
`components`	list, similar to those produced by `xscale.components.default` and `yscale.components.de`
`as.table`	the `as.table` argument in the high level function.
`labels`	whether labels are to be drawn. By default, the rules determined by `scales` are used.
`ticks`	whether labels are to be drawn. By default, the rules determined by `scales` are used.
`...`	many other arguments may be supplied, and are passed on to other internal functions.

prefix   A character string identifying the plot being drawn (see `print.trellis`).
                Used to retrieve location of current panel in the overall layout, so that axes can
                be drawn appropriately.

Details

These functions are part of a new API introduced in lattice 0.14 to provide the user more control
over how axis annotation is done. While the API has been designed in anticipation of use that was
previously unsupported, the implementation has initially focused on reproducing existing capabil-
ities, rather than test new features. At the time of writing, several features are unimplemented. If
you require them, please contact the maintainer.

Value

`xscale.components.default` and `yscale.components.default` return a list of the
form suitable as the `components` argument of `axis.default`. Valid components in the return
value of `xscale.components.default` are:

num.limit   A numeric limit for the box.

bottom   A list with two elements, `ticks` and `labels`. `ticks` must be a list with
                components `at` and `tck` which give the location and lengths of tick marks. `tck`
                can be a vector, and will be recycled to be as long as `at`. `labels` must be a list
                with components `at`, `labels`, and `check.overlap`. `at` and `labels` give
                the location and labels of the tick labels; this is usually the same as the location
                of the ticks, but is not required to be so. `check.overlap` is a logical flag
                indicating whether overlapping of labels should be avoided by omitting some of
                the labels while rendering.

top   This can be a logical flag; if `TRUE`, `top` is treated as being the same as `bottom`;
                if `FALSE`, axis annotation for the top axis is omitted. Alternatively, `top` can be
                a list like `bottom`.

Valid components in the return value of `yscale.components.default` are `left` and `right`.
Their interpretations are analogous to (respectively) the `bottom` and `top` components described
above.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`Lattice`, `xyplot`, `print.trellis`

Examples

```
str(xscale.components.default(c(0, 1)))

set.seed(36872)
rln <- rlnorm(100)
```

```
densityplot(rln,
            scales = list(x = list(log = 2), alternating = 3),
            xlab = "Simulated lognormal variates",
            xscale.components = function(...) {
                ans <- xscale.components.default(...)
                ans$top <- ans$bottom
                ans$bottom$labels$labels <- parse(text = ans$bottom$labels$labels)
                ans$top$labels$labels <-
                    if (require(MASS))
                        fractions(2^(ans$top$labels$at))
                    else
                        2^(ans$top$labels$at)
                ans
            })


## Direct use of axis to show two temperature scales (Celcius and
## Fahrenheit).  This does not work for multi-row plots, and doesn't
## do automatic allocation of space


F2C <- function(f) 5 * (f - 32) / 9
C2F <- function(c) 32 + 9 * c / 5

axis.CF <-
    function(side, ...)
{
    ylim <- current.panel.limits()$ylim
    switch(side,
           left = {
               prettyF <- pretty(ylim)
               labF <- parse(text = sprintf("%s ~ degree * F", prettyF))
               panel.axis(side = side, outside = TRUE,
                          at = prettyF, labels = labF)
           },
           right = {
               prettyC <- pretty(F2C(ylim))
               labC <- parse(text = sprintf("%s ~ degree * C", prettyC))
               panel.axis(side = side, outside = TRUE,
                          at = C2F(prettyC), labels = labC)
           },
           axis.default(side = side, ...))
}

xyplot(nhtemp ~ time(nhtemp), aspect = "xy", type = "o",
       scales = list(y = list(alternating = 3)),
       axis = axis.CF, xlab = "Year", ylab = "Temperature",
       main = "Yearly temperature in New Haven, CT")

## version using yscale.components

yscale.components.CF <-
```

```
    function(...)
{
    ans <- yscale.components.default(...)
    ans$right <- ans$left
    ans$left$labels$labels <-
        parse(text = sprintf("%s ~ degree * F", ans$left$labels$at))
    prettyC <- pretty(F2C(ans$num.limit))
    ans$right$ticks$at <- C2F(prettyC)
    ans$right$labels$at <- C2F(prettyC)
    ans$right$labels$labels <-
        parse(text = sprintf("%s ~ degree * C", prettyC))
    ans
}


xyplot(nhtemp ~ time(nhtemp), aspect = "xy", type = "o",
       scales = list(y = list(alternating = 3)),
       yscale.components = yscale.components.CF,
       xlab = "Year", ylab = "Temperature",
       main = "Yearly temperature in New Haven, CT")
```

G_banking   *Banking*

Description

Calculates banking slope

Usage

```
banking(dx, dy)
```

Arguments

dx, dy   vector of consecutive x, y differences.

Details

banking is the banking function used when aspect = "xy" in high level Trellis functions. It
is usually not very meaningful except with xyplot. It considers the absolute slopes (based on dx
and dy) and returns a value which when adjusted by the panel scale limits will make the median of
the above absolute slopes correspond to a 45 degree line.

This function was inspired by the discussion of banking in the documentation for Trellis Graphics
available at Bell Labs' website (see Lattice), but is most likely identical to an algorithm de-
scribed by Cleveland et al (see below). It is not clear (to the author) whether this is the algorithm
used in S-PLUS. Alternative banking rules, implemented as a similar function, can be used as a
drop-in replacement by suitably modifying lattice.options("banking").

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

References

Cleveland, William S. and McGill, Marilyn E. and McGill, Robert (1988) "The Shape Parameter of a Two-variable Graph", *Journal of the American Statistical Association*, **83**, 289–300.

See Also

`Lattice`, `xyplot`

Examples

```
## with and without banking

plot <- xyplot(sunspot.year ~ 1700:1988, xlab = "", type = "l",
               scales = list(x = list(alternating = 2)),
               main = "Yearly Sunspots")
print(plot, position = c(0, .3, 1, .9), more = TRUE)
print(update(plot, aspect = "xy", main = "", xlab = "Year"),
      position = c(0, 0, 1, .3))

## cut-and-stack plot (see also xyplot.ts)

xyplot(sunspot.year ~ time(sunspot.year) | equal.count(time(sunspot.year)),
       xlab = "", type = "l", aspect = "xy", strip = FALSE,
       scales = list(x = list(alternating = 2, relation = "sliced")),
       as.table = TRUE, main = "Yearly Sunspots")
```

```
G_latticeParseFormula
```
Parse Trellis formula

Description

this function is used by high level Lattice functions like `xyplot` to parse the formula argument and evaluate various components of the data.

Usage

```
latticeParseFormula(model, data, dimension = 2,
                    subset = TRUE, groups = NULL,
                    multiple, outer,
                    subscripts,
                    drop)
```

Arguments

`model`	the model/formula to be parsed. This can be in either of two possible forms, one for 2d and one for 3d formulas, determined by the `dimension` argument. The 2d formulas are of the form `y ~ x| g1 * ...  *gn`, and the 3d formulas are of the form `z ~ x * y	g1 * ...* gn`. In the first form, `y` may be omitted. The conditioning variables `g1, ...,gn` can be omitted in either case.
`data`	the environment/dataset where the variables in the formula are evaluated.
`dimension`	dimension of the model, see above
`subset`	index for choosing a subset of the data frame
`groups`	the grouping variable, if present
`multiple, outer`	
	logicals, determining how a '+' in the y and x components of the formula are processed. See [xyplot](#) for details
`subscripts`	logical, whether subscripts are to be calculated
`drop`	logical or list, similar to the `drop.unused.levels` argument in [xyplot](#), indicating whether unused levels of conditioning factors and data variables that are factors are to be dropped.

Value

returns a list with several components, including `left, right, left.name, right.name, condition` for 2-D, and `left, right.x, right.y, left.name, right.x.name, right.y.name, condition` for 3-D. Other possible components are groups, subscr

Author(s)

Saikat DebRoy, Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

See Also

[xyplot](#), [Lattice](#)

`G_packet.panel.default`

Associating Packets with Panels

Description

When a `"trellis"` object is plotted, panels are always drawn in an order such that columns vary the fastest, then rows and then pages. An optional function can be specified that determines, given the column, row and page and other relevant information, the packet (if any) which should be used in that panel. The function documented here implements the default behaviour, which is to match panel order with packet order, determined by varying the first conditioning variable the fastest, then the second, and so on. This matching is performed after any reordering and/or permutation of the conditioning variables.

Usage

```
packet.panel.default(layout, condlevels, page, row, column,
                     skip, all.pages.skip = TRUE)
```

Arguments

layout   the `layout` argument in high level functions, suitably standardized.

condlevels   a list of levels of conditioning variables, after relevant permutations and/or re-ordering of levels

page, row, column
    the location of the panel in the coordinate system of pages, rows and columns.

skip   the `skip` argument in high level functions

all.pages.skip
    whether `skip` should be replicated over all pages. If `FALSE`, `skip` will be replicated to be only as long as the number of positions on a page, and that template will be used for all pages.

Value

A suitable combination of levels of the conditioning variables in the form of a numeric vector as long as the number of conditioning variables, with each element an integer indexing the levels of the corresponding variable. Specifically, if the return value is `p`, then the `i`-th conditioning variable will have level `condlevels[[i]][p[i]]`.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`Lattice`, `xyplot`

Examples

```
packet.panel.page <- function(n)
{
    ## returns a function that when used as the 'packet.panel'
    ## argument in print.trellis plots page number 'n' only
    function(layout, page, ...) {
        stopifnot(layout[3] == 1)
        packet.panel.default(layout = layout, page = n, ...)
    }
}

data(mtcars)
HP <- equal.count(mtcars$hp, 6)
```

```
p <-
    xyplot(mpg ~ disp | HP * factor(cyl),
           mtcars, layout = c(0, 6, 1))

print(p, packet.panel = packet.panel.page(1))
print(p, packet.panel = packet.panel.page(2))
```

G_panel.axis   *Panel Function for Drawing Axis Ticks and Labels*

Description

panel.axis is the function used by lattice to draw axes. It is typically not used by users, except those wishing to create advanced annotation. Keep in mind issues of clipping when trying to use it as part of the panel function. current.panel.limits can be used to retrieve a panel's x and y limits.

Usage

```
panel.axis(side = c("bottom", "left", "top", "right"),
           at,
           labels = TRUE,
           draw.labels = TRUE,
           check.overlap = FALSE,
           outside = FALSE,
           ticks = TRUE,
           half = !outside,
           which.half,
           tck = as.numeric(ticks),
           rot = if (is.logical(labels)) 0 else c(90, 0),
           text.col, text.alpha, text.cex, text.font,
           text.fontfamily, text.fontface, text.lineheight,
           line.col, line.lty, line.lwd, line.alpha)

current.panel.limits(unit = "native")
```

Arguments

side	character string indicating which side axes are to be drawn on. Partial specification is allowed.
at	location of labels
labels	the labels to go along with at. The labels can be a character vector or a vector of expressions. Alternatively, this can be a logical. If TRUE, the labels are derived from at, otherwise, labels are empty.
draw.labels	logical indicating whether labels are to be drawn

check.overlap	
	logical, whether to check for overlapping of labels. This also has the effect of removing `at` values that are 'too close' to the limits.
outside	logical, whether to the labels draw outside the panel or inside. Note that `outside=TRUE` will only have a visible effect if clipping is disabled for the viewport (panel).
ticks	logical, whether to draw the tickmarks
half	logical, whether only half of scales will be drawn for each side
which.half	character string, one of `"lower"` and `"upper"`. Indicates which half is to be used for tick locations if `half=TRUE`. Defaults to whatever is suitable for splom
tck	numeric scalar, multiplier for tick length. Can be negative.
rot	rotation angles for labels in degrees. Can be a vector of length 2 for x- and y-axes respectively
text.col	Color for the axis label text. See gpar for more details on this and the other graphical parameters listed below.
text.alpha	Alpha-transparency value for the axis label text.
text.cex	Size multiplier for the axis label text.
text.font, text.fontfamily, text.fontface	
	Font for the axis label text.
text.lineheight	
	Line height for the axis label text.
line.col	for the axis label text.
line.lty	for the axis.
line.lwd	for the axis.
line.alpha	for the axis.
unit	which grid unit the values should be in

Details

`panel.axis` can draw axis tick marks inside or outside a panel (more precisely, a grid viewport). It honours the (native) axis scales. Used in panel.pairs for splom, as well as for all the usual axis drawing by the print method for `"trellis"` objects. It can also be used to enhance plots 'after the fact' by adding axes.

Value

`current.panel.limits` returns a list with components `xlim` and `ylim`, which are both numeric vectors of length 2, giving the scales of the current panel (viewport). The values correspond to the unit system specified by unit, by default `"native"`.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

Lattice, xyplot, trellis.focus, unit

G_panel.number   *Accessing Auxiliary Information During Plotting*

Description

Control over lattice plots are provided through a collection of user specifiable functions that perform various tasks during the plotting. Not all information is available to all functions. The functions documented here attempt to provide a consistent interface to access relevant information from within these user specified functions, namely those specified as the `panel`, `strip` and `axis` functions.

Usage

```
current.row(prefix)
current.column(prefix)
panel.number(prefix)
packet.number(prefix)
which.packet(prefix)

trellis.currentLayout(which = c("packet", "panel"), prefix)
```

Arguments

which   whether return value (a matrix) should contain panel numbers or packet numbers, which are usually, but not necessarily, the same (see below for details).

prefix   A character string acting as a prefix identifying the plot of a `"trellis"` object. Only relevant when a particular page is occupied by more than one plot. Defaults to the value appropriate for the last `"trellis"` object printed. See `trellis.focus`.

Value

`trellis.currentLayout` returns a matrix with as many rows and columns as in the layout of panels in the current plot. Entries in the matrix are integer indices indicating which packet (or panel; see below) occupies that position, with 0 indicating the absence of a panel. `current.row` and `current.column` return integer indices specifying which row and column in the layout are currently active. `panel.number` returns an integer counting which panel is being drawn (starting from 1 for the first panel, a.k.a. the panel order). `packet.number` gives the packet number according to the packet order, which is determined by varying the first conditioning variable the fastest, then the second, and so on. `which.packet` returns the combination of levels of the conditioning variables in the form of a numeric vector as long as the number of conditioning variables, with each element an integer indexing the levels of the corresponding variable.

Note

The availability of these functions make redundant some features available in earlier versions of lattice, namely optional arguments called `panel.number` and `packet.number` that were made available to `panel` and `strip`. If you have written such functions, it should be enough to replace instances of `panel.number` and `packet.number` by the corresponding function calls. You should also remove `panel.number` and `packet.number` from the argument list of your function to avoid a warning.

If these accessor functions are not enough for your needs, feel free to contact the maintainer and ask for more.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[Lattice](), [xyplot]()

`G_Rows`	*Extract rows from a list*

Description

Convenience function to extract subset of a list. Usually used in creating keys.

Usage

```
Rows(x, which)
```

Arguments

`x`	list with each member a vector of the same length
`which`	index for members of `x`

Value

A list similar to `x`, with each `x[[i]]` replaced by `x[[i]][which]`

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

[xyplot](), [Lattice]()

G_utilities.3d   *Utility functions for 3-D plots*

Description

These are (related to) the default panel functions for `cloud` and `wireframe`.

Usage

```
ltransform3dMatrix(screen, R.mat)
ltransform3dto3d(x, R.mat, dist)
```

Arguments

x   x can be a numeric matrix with 3 rows for `ltransform3dto3d`

screen   list, as described in `panel.cloud`

R.mat   4x4 transformation matrix in homogeneous coordinates

dist   controls transformation to account for perspective viewing

Details

`ltransform3dMatrix` and `ltransform3dto3d` are utility functions to help in computation of projections. These functions are used inside the panel functions for `cloud` and `wireframe`. They may be useful in user-defined panel functions as well.

The first function takes a list of the form of the `screen` argument in `cloud` and `wireframe` and a `R.mat`, a 4x4 transformation matrix in homogeneous coordinates, to return a new 4x4 transformation matrix that is the result of applying `R.mat` followed by the rotations in `screen`. The second function applies a 4x4 transformation matrix in homogeneous coordinates to a 3xn matrix representing points in 3-D space, and optionally does some perspective computations. (There has been no testing with non-trivial transformation matrices, and my knowledge of the homogeneous coordinate system is very limited, so there may be bugs here.)

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

See Also

`cloud`, `panel.cloud`

H_barley *Yield data from a Minnesota barley trial*

Description

Total yield in bushels per acre for 10 varieties at 6 sites in each of two years.

Usage

```
barley
```

Format

A data frame with 120 observations on the following 4 variables.

yield Yield (averaged across three blocks) in bushels/acre.

variety Factor with levels `"Svansota"`, `"No.  462"`, `"Manchuria"`, `"No.  475"`, `"Velvet"`, `"Peatland"`, `"Glabron"`, `"No.  457"`, `"Wisconsin No.  38"`, `"Trebi"`.

year Factor with levels `1932`, `1931`

site Factor with 6 levels: `"Grand Rapids"`, `"Duluth"`, `"University Farm"`, `"Morris"`, `"Crookston"`, `"Waseca"`

Details

These data are yields in bushels per acre, of 10 varieties of barley grown in 1/40 acre plots at University Farm, St. Paul, and at the five branch experiment stations located at Waseca, Morris, Crookston, Grand Rapids, and Duluth (all in Minnesota). The varieties were grown in three randomized blocks at each of the six stations during 1931 and 1932, different land being used each year of the test.

Immer et al. (1934) present the data for each Year*Site*Variety*Block. The data here is the average yield across the three blocks.

Immer et al. (1934) refer (once) to the experiment as being conducted in 1930 and 1931, then later refer to it (repeatedly) as being conducted in 1931 and 1932. Later authors have continued the confusion.

Cleveland (1993) suggests that the data for the Morris site may have had the years switched.

Author(s)

Documentation contributed by Kevin Wright.

Source

Immer, R. F., H. K. Hayes, and LeRoy Powers. (1934). Statistical Determination of Barley Varietal Adaptation. *Journal of the American Society of Agronomy*, **26**, 403–419.

References

Cleveland, William S. (1993) *Visualizing Data*. Hobart Press, Summit, New Jersey.

Fisher, R. A. (1971) *The Design of Experiments*. Hafner, New York, 9th edition.

See Also

immer in the MASS package for data from the same experiment (expressed as total yield for 3 blocks) for a subset of varieties.

Examples

```
# Graphic suggesting the Morris data switched the years 1931 and 1932
# Figure 1.1 from Cleveland
dotplot(variety ~ yield | site, data = barley, groups = year,
        key = simpleKey(levels(barley$year), space = "right"),
        xlab = "Barley Yield (bushels/acre) ",
        aspect=0.5, layout = c(1,6), ylab=NULL)
```

H_environmental   *Atmospheric environmental conditions in New York City*

Description

Daily measurements of ozone concentration, wind speed, temperature and solar radiation in New York City from May to September of 1973.

Usage

```
environmental
```

Format

A data frame with 111 observations on the following 4 variables.

ozone Average ozone concentration (of hourly measurements) of in parts per billion.

radiation Solar radiation (from 08:00 to 12:00) in langleys.

temperature Maximum daily emperature in degrees Fahrenheit.

wind Average wind speed (at 07:00 and 10:00) in miles per hour.

Author(s)

Documentation contributed by Kevin Wright.

Source

Bruntz, S. M., W. S. Cleveland, B. Kleiner, and J. L. Warner. (1974). The Dependence of Ambient Ozone on Solar Radiation, Wind, Temperature, and Mixing Height. In *Symposium on Atmospheric Diffusion and Air Pollution*, pages 125–128. American Meterological Society, Boston.

References

Cleveland, William S. (1993) *Visualizing Data*. Hobart Press, Summit, New Jersey.

Examples

```
# Scatter plot matrix with loess lines
splom(~environmental,
  panel=function(x,y){
    panel.xyplot(x,y)
    panel.loess(x,y)
  }
)


# Conditioned plot similar to figure 5.3 from Cleveland
attach(environmental)
Temperature <- equal.count(temperature, 4, 1/2)
Wind <- equal.count(wind, 4, 1/2)
xyplot((ozone^(1/3)) ~ radiation | Temperature * Wind,
  aspect=1,
prepanel = function(x, y)
prepanel.loess(x, y, span = 1),
panel = function(x, y){
panel.grid(h = 2, v = 2)
panel.xyplot(x, y, cex = .5)
panel.loess(x, y, span = 1)
},
xlab = "Solar radiation (langleys)",
ylab = "Ozone (cube root ppb)")
detach()

# Similar display using the coplot function
with(environmental,{
  coplot((ozone^.33) ~ radiation | temperature * wind,
  number=c(4,4),
  panel = function(x, y, ...) panel.smooth(x, y, span = .8, ...),
  xlab="Solar radiation (langleys)",
  ylab="Ozone (cube root ppb)")
})
```

H_ethanol	*Engine exhaust fumes from burning ethanol*

Description

Ethanol fuel was burned in a single-cylinder engine. For various settings of the engine compression and equivalence ratio, the emissions of nitrogen oxides were recorded.

Usage

```
ethanol
```

Format

A data frame with 88 observations on the following 3 variables.

NOx Concentration of nitrogen oxides (NO and NO2) in micrograms/J.

C Compression ratio of the engine.

E Equivalence ratio–a measure of the richness of the air and ethanol fuel mixture.

Author(s)

Documentation contributed by Kevin Wright.

Source

Brinkman, N.D. (1981) Ethanol Fuel—A Single-Cylinder Engine Study of Efficiency and Exhaust Emissions. *SAE transactions*, **90**, 1410–1424.

References

Cleveland, William S. (1993) *Visualizing Data*. Hobart Press, Summit, New Jersey.

Examples

```
## Constructing panel functions on the fly
EE <- equal.count(ethanol$E, number=9, overlap=1/4)
xyplot(NOx ~ C | EE, data = ethanol,
       prepanel = function(x, y) prepanel.loess(x, y, span = 1),
       xlab = "Compression ratio", ylab = "NOx (micrograms/J)",
       panel = function(x, y) {
           panel.grid(h=-1, v= 2)
           panel.xyplot(x, y)
           panel.loess(x,y, span=1)
       },
       aspect = "xy")

# Wireframe loess surface fit.  See Figure 4.61 from Cleveland.
require(stats)
with(ethanol, {
  eth.lo <- loess(NOx ~ C * E, span = 1/3, parametric = "C",
                  drop.square = "C", family="symmetric")
  eth.marginal <- list(C = seq(min(C), max(C), length.out = 25),
                       E = seq(min(E), max(E), length.out = 25))
  eth.grid <- expand.grid(eth.marginal)
  eth.fit <- predict(eth.lo, eth.grid)
  wireframe(eth.fit ~ eth.grid$C * eth.grid$E,
            shade=TRUE,
    screen = list(z = 40, x = -60, y=0),
    distance = .1,
```

```
    xlab = "C", ylab = "E", zlab = "NOx")
})
```

H_melanoma	*Melanoma skin cancer incidence*

Description

These data from the Connecticut Tumor Registry present age-adjusted numbers of melanoma skin-cancer incidences per 100,000 people in Connectict for the years from 1936 to 1972.

Usage

```
melanoma
```

Format

A data frame with 37 observations on the following 2 variables.

year Years 1936 to 1972.

incidence Rate of melanoma cancer per 100,000 population.

Note

This dataset is not related to the melanoma dataset in the **boot** package with the same name.

The S-PLUS 6.2 help for the melanoma data says that the incidence rate is per *million*, but this is not consistent with data found at the National Cancer Institute (http://www.nci.nih.gov).

Author(s)

Documentation contributed by Kevin Wright.

Source

Houghton, A., E. W. Munster, and M. V. Viola. (1978). Increased Incidence of Malignant Melanoma After Peaks of Sunspot Activity. *The Lancet*, **8**, 759–760.

References

Cleveland, William S. (1993) *Visualizing Data*. Hobart Press, Summit, New Jersey.

Examples

```
# Time-series plot.  Figure 3.64 from Cleveland.
xyplot(incidence ~ year,
  data = melanoma,
aspect = "xy",
panel = function(x, y)
panel.xyplot(x, y, type="o", pch = 16),
ylim = c(0, 6),
xlab = "Year",
ylab = "Incidence")
```

H_singer	*Heights of New York Choral Society singers*

Description

Heights in inches of the singers in the New York Choral Society in 1979. The data are grouped according to voice part. The vocal range for each voice part increases in pitch according to the following order: Bass 2, Bass 1, Tenor 2, Tenor 1, Alto 2, Alto 1, Soprano 2, Soprano 1.

Usage

```
singer
```

Format

A data frame with 235 observations on the following 2 variables.

height Height in inches of the singers.

voice.part (Unordered) factor with levels `"Bass 2"`, `"Bass 1"`, `"Tenor 2"`, `"Tenor 1"`, `"Alto 2"`, `"Alto 1"`, `"Soprano 2"`, `"Soprano 1"`.

Author(s)

Documentation contributed by Kevin Wright.

Source

Chambers, J.M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. (1983). *Graphical Methods for Data Analysis*. Chapman and Hall, New York.

References

Cleveland, William S. (1993) *Visualizing Data*. Hobart Press, Summit, New Jersey.

Examples

```
# Separate histogram for each voice part (Figure 1.2 from Cleveland)
histogram(~ height | voice.part,
          data = singer,
          aspect=1,
          layout = c(2, 4),
          nint=15,
          xlab = "Height (inches)")

# Quantile-Quantile plot (Figure 2.11 from Cleveland)
qqmath(~ height | voice.part,
       data=singer,
       aspect=1,
       layout=c(2,4),
       prepanel = prepanel.qqmathline,
       panel = function(x, ...) {
         panel.grid()
         panel.qqmathline(x, ...)
         panel.qqmath(x, ...)
       },
       xlab = "Unit Normal Quantile",
       ylab="Height (inches)")
```

I_lset   *Interface to modify Trellis Settings - Deprecated*

Description

A (hopefully) simpler alternative to `trellis.par.get/set`. This is deprecated, and the same functionality is now available with `trellis.par.set`

Usage

```
lset(theme = col.whitebg())
```

Arguments

theme   a list decribing how to change the settings of the current active device. Valid
                components are those in the list returned by `trellis.par.get()`. Each
                component must itself be a list, with one or more of the appropriate compo-
                nents (need not have all components). Changes are made to the settings for the
                currently active device only.

Author(s)

Deepayan Sarkar `<Deepayan.Sarkar@R-project.org>`

Index