

Package ‘latticist’

January 4, 2012

Type Package

Title A graphical user interface for exploratory visualisation

Version 0.9-44

Date 2012-01-03

Author Felix Andrews <felix@nfrac.org>

Maintainer Felix Andrews <felix@nfrac.org>

Depends lattice (>= 0.16-3), latticeExtra (>= 0.5-4), vcd

Imports gWidgets (>= 0.0-30), utils

Suggests playwith, hexbin, deldir, tripack, DAAG, RGtk2,gWidgetsRGtk2, gWidgetstcltk, MASS

Description Latticist provides a graphical user interface for exploratory visualisation. It is primarily an interface to the Lattice graphics system, but also produces displays from the vcd package for categorical data. Given a multivariate dataset (either a data frame or a table),latticist attempts to produce useful displays based on the properties of the data. The displays can be customised by editing the calls used to generate them.

License GPL (>= 2)

URL <http://latticist.googlecode.com/>

Repository CRAN

Date/Publication 2012-01-04 11:47:51

R topics documented:

| | |
|------------------------------|----|
| custom.theme.black | 2 |
| cutEq | 3 |
| latticeStyleDemo | 4 |
| latticeStyleGUI | 5 |
| latticist | 7 |
| latticist.options | 9 |
| latticistCompose | 11 |

custom.theme.black *Alternative Lattice themes*

Description

Alternative Lattice themes.

Usage

```
custom.theme.black(symbol = brewer.pal(n = 8, name = "Set2"),
  fill = brewer.pal(n = 8, name = "Set2"),
  region = rev(brewer.pal(n = 9, name = "YlOrRd")),
  reference = "#444444", bg = "black", fg = "white",
  etc = TRUE)
```

Arguments

| | |
|-----------|--|
| symbol | colors for points and lines. |
| fill | colors for polygons. |
| region | color ramp for continuous regions. |
| reference | color of reference lines. |
| bg | background color. |
| fg | foreground color. |
| etc | TRUE to set extra graphical parameters designed for black backgrounds. |

Details

This is a wrapper around [custom.theme](#) with different defaults. It can be used as a Lattice themes.

The etc argument sets:

- grey strips;
- solid plot symbols;
- translucent plot symbols (alpha = 0.5);
- thick lines;
- no borders on polygons.

These settings can be modified by editing the resulting list. An easy way to do that is via [simpleTheme](#) (see examples).

Value

a list of settings suitable for passing to `trellis.par.set`.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

[custom.theme](#), [simpleTheme](#)

Examples

```
opar <- trellis.par.get()

trellis.par.set(custom.theme.black())
latticeStyleDemo()

## make changes to the theme
myTheme <- modifyList(custom.theme.black(),
                      simpleTheme(alpha.points = 0.1))
myTheme$add.line$lty <- 3
## now apply myTheme, or just:
trellis.par.set(simpleTheme(alpha.points = 0.1))
trellis.par.set(add.line = list(lty = 3))
latticeStyleDemo()

trellis.par.set(opar)
```

cutEq

Cut into equal-sized groups.

Description

Discretize using equally spaced quantiles for the breaks.

Usage

```
cutEq(x, n, type = 2, dig.lab = 4, ...)

reorderByFreq(x)
```

Arguments

| | |
|---------|--|
| x | for cutEq, a numeric vector. for reorderByFreq, a factor variable. |
| n | number of levels (groups) to cut into. |
| type | type of quantile, see quantile . |
| dig.lab | digits to use in formatting labels. |
| ... | ignored. |

Details

cutEq is similar to [cut](#) but divides into roughly equally-sized groups, rather than dividing into bins of equal width.

reorderByFreq orders levels of a factor by their frequency. It is basically `reorder(x, x, length)`, but reversed.

Value

cutEq returns an ordered factor. reorderByFreq returns a factor with the same values as its input, but with levels reordered.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

[cut](#), [cut2](#) in Hmisc package, [reorder](#)

Examples

```
summary(cut(rnorm(100), 4, dig=2))
summary(cutEq(rnorm(100), 4, dig=2))

foo <- factor(c("c", "b", "b", "a"))
levels(foo)
# "a" "b" "c"
levels(reorderByFreq(foo))
# "b" "a" "c"
```

latticeStyleDemo

Demonstrate the current Lattice style settings

Description

Plots incorporating all the major elements of Lattice style settings.

Usage

```
latticeStyleDemo(type = c("plot", "superpose", "polygons", "regions"))
```

Arguments

type which of the 4 plot types to display. If more than one are given they are shown in panels. The default is to show all 4.

Details

Note, this uses "user.text", which is not a standard element of Lattice style. However, this falls back to "add.text" – the default used by `panel.text()` – if not defined.

"user.text" is preferred to "add.text" for annotations because the latter also applies to strip text and key text.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

[show.settings.panel.usertext](#)

Examples

```
latticeStyleDemo()  
latticeStyleDemo("plot")  
latticeStyleDemo(c("superpose", "polygons"))
```

latticeStyleGUI *A plot style settings GUI*

Description

A graphical user interface to edit the Lattice theme for a given device. It can also set some simple base graphics parameters.

Usage

```
latticeStyleGUI(width = 480, height = 480, pointsize = 12,  
                target.device = dev.cur(), base.graphics = FALSE)
```

```
latticeStyleToBasePar()
```

Arguments

| | |
|--|---|
| <code>width</code> , <code>height</code> | initial size in pixels of the embedded graphic device. |
| <code>pointsize</code> | pointsize for the embedded graphic device. |
| <code>target.device</code> | the device for which to edit style settings; defaults to the current device. |
| <code>base.graphics</code> | whether to apply settings to the base graphics parameters (<code>par</code>) as well as the lattice graphics parameters. This is quite limited compared to the lattice system: not all controls in the interface will have an effect. Also, base graphics plots may or may not use the <code>par()</code> settings. |

Details

`latticeStyleGUI()` is an interface to `trellis.par.get` / `trellis.par.set`. It is not a comprehensive interface: some simplifications and omissions have been made from the full list of settings.

The graphical user interface is built on the `gWidgets` package. As such it can run under different GUI toolkits such as `gWidgetsRGtk2`, `gWidgetstcltk`, etc.

Changes take effect immediately. Load a new theme to reset. The embedded graphic device (or non-embedded, depending on the GUI toolkit) will show a preview of your settings. You can also plot on the target device while keeping the GUI open, for testing.

The GUI may be very slow to initialise. It is recommended to update to the latest version of the `gWidgets` package (and its toolkit implementations), as these include significant speed-ups.

`latticeStyleToBasePar()` attempts to apply the current lattice style settings to the base graphics system (`par` and `palette`). Only basic settings are used. See the function definition for details.

Value

`latticeStyleGUI` does not return anything, but does change graphical settings for the given device. It also creates an object `trellis.par.theme` in the global workspace containing the Lattice *theme* (i.e. list of settings). Another object `trellis.par.log` holds a subset of that: just the changes made from a built-in theme.

`latticeStyleToBasePar` returns the previous `par` settings.

Note

Plot calls can over-ride these settings, so there is no guarantee that the settings will be visible on a given plot.

Author(s)

Felix Andrews <felix@nfrac.org>

References

For an excellent introduction to and coverage of Lattice:

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

See Also

[trellis.par.get](#), `par`, [latticeStyleDemo](#)

Examples

```
if (interactive())
{
  latticeStyleGUI()
}

## a base graphics plot (from example(matplot))
```

```

sines <- outer(1:20, 1:4, function(x, y) sin(x / 20 * pi * y))
matplot(sines, pch = 1:4, type = "o")

## apply Lattice settings and re-plot
opar <- latticeStyleToBasePar()
matplot(sines, pch = 1:4, type = "o")

## apply a different Lattice theme
trellis.par.set(custom.theme.black())
latticeStyleToBasePar()
plot(Ozone ~ Wind, data = airquality, col = 1)

## reset:
par(opar)
palette("default")

```

lattice

A Lattice GUI

Description

Interactively explore a data set using [Lattice](#) displays.

Usage

```

lattice(dat,
        spec = list(),
        reorder.levels = !is.table(dat),
        ...,
        use.playwith = lattice.getOption("use.playwith"))

```

Arguments

| | |
|-----------------------------|---|
| <code>dat</code> | a data frame (with numeric and/or categorical variables), or a table. |
| <code>spec</code> | a list specifying the initial lattice display. See latticeCompose . |
| <code>reorder.levels</code> | if TRUE, change the ordering of levels of factor variables so they are in order of frequency. Note that this does not change the data, only its internal representation: see reorder . This is recommended for effective graphic displays. Factor variables of class <code>ordered</code> are not reordered. Therefore you should ensure that any variables whose levels have an inherent order are of class <code>ordered</code> : see as.ordered . Note: the result of cut is by default NOT ordered! In addition, numeric variables with discrete values in (0, 1, -1) are converted to factors. |
| <code>use.playwith</code> | whether to launch as a toolbar for playwith (requires the playwith package), or a generic gWidgets interface. |
| <code>...</code> | further arguments specific to the interface type. For the playwith interface, these are passed to playwith . For the gWidgets interface, these are: <code>width = 450</code> , <code>height = 450</code> size of the graphics device in pixels. Note, these are ignored if not using an embedded device. <code>pointsize = 12</code> text size on the graphics device. |

Details

Latticist is a graphical user interface for exploratory visualisation. It is primarily an interface to the Lattice graphics system (from the **lattice** package), but also produces displays from the **vcd** package for categorical data.

Given a multivariate dataset (either a data frame or a table), Latticist attempts to produce useful displays based on the properties of the data. The user chooses variables or expressions for the plot axes, for grouping, conditioning and subsetting. Some hypervariate displays are also available.

A minimal graphical user interface is available, built on the **gWidgets** package. This requires one of the "toolkit implementations" to be installed: **gWidgetstcltk**, **gWidgetsRGtk2** or **gWidgetsRJava**. Note that **gWidgetsRJava** is currently broken (as of version 0.0-13).

The fastest way to start, without any external system requirements, is to `install.packages("gWidgetstcltk")`. However, that does not support an embedded graphics device, so the plots will appear in a separate window.

Alternatively, Latticist can be run as a toolbar extension to **playwith**. This brings many extra features, such as dynamic zooming, identifying data points, linked brushing, etc. Note that the **playwith** package requires **RGtk2** and, therefore, the *GTK+* libraries.

To enable all types of graphic displays, install the **hexbin** and **deldir** packages.

Value

The **playwith** method invisibly returns the `playState` object representing the plot window. One can close it with `playDevOff()` or `dev.off()`.

The **gWidgets** method invisibly returns the `gwindow` object. One can close it with `dispose()`.

Author(s)

Felix Andrews <felix@nfrac.org>

References

For an excellent introduction to and coverage of Lattice:

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

The mosaic displays and extensions from **vcd** are well described in:

David Meyer, Achim Zeileis, and Kurt Hornik (2006). "The Strucplot Framework: Visualizing Multi-Way Contingency Tables with vcd". *Journal of Statistical Software*, 17(3), 1-48. <http://www.jstatsoft.org/v17/i03/>

See Also

[latticistCompose](#), [latticist.options](#), [Lattice](#), [lattice.demo](#) in the **TeachingDemos** package.

Examples

```
if (interactive()) {
  options(device.ask.default = FALSE)

  ## Not run:
  ## data frame example:
  lattice(iris)

  ## table example:
  lattice(Titanic, spec = list(groups = "Survived"))

  ## End(Not run)

  ## The GUI comes in three flavours:

  if (require("gWidgetsTcltk") &&
      isTRUE(gconfirm("Show tcl/tk-based GUI?")))
  {
    options(guiToolkit = "tcltk")
    lattice(CO2, use.playwith = FALSE)
  }

  if (require("gWidgetsRGtk2") &&
      isTRUE(gconfirm("Show RGtk2-based GUI?")))
  {
    options(guiToolkit = "RGtk2")
    lattice(CO2, use.playwith = FALSE)
  }

  if (require("playwith") &&
      isTRUE(gconfirm("Show playwith-based GUI?")))
  {
    lattice(CO2, use.playwith = TRUE)
  }
}
```

`lattice.options` *User default settings for lattice*

Description

A basic user settings facility, like `options` and `lattice.options`.

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

The available options and their current values can be seen with `str(latticist.options())`.

The options are:

`use.playwith` default for the argument to `latticist`.

`defaultPlot` see `latticistCompose`. Note, this value can be over-ridden by specifying `defaultPlot` in the `spec` argument.

`xyLineType` line type for scatterplots; one (or more) of "smooth", "r" or "a".

`add.sub`, `sub.func` set `add.sub` to FALSE to omit the subtitle. Set `sub.func` to a character value for the subtitle, or a function to generate the subtitle. It is passed arguments `spec` and `nPoints`.

`MANY`, `VERYMANY` threshold number of data points (per panel, on average), at which to change the plot type to something more appropriate for large datasets.

For `qqmath`, in the `VERYMANY` case, the `f.value` argument is used to approximate the distribution with 100 points. For `densityplot`, in the `VERYMANY` case, data points are not drawn.

`style.MANY`, `style.3panels`, **etc** these lists are combined to form a list of arguments to `simpleTheme`.

First, if the number of panels is 3 or more, the values in `style.3panels` are used. If the number of panels is 7 or more, the values in `style.7panels` are also used.

Next, if the number of data points (per panel) exceeds `MANY`, the values in `style.MANY` are used. If they also exceed `VERYMANY`, the values in `style.VERYMANY` are used *in addition* to those already set.

`max.panels` sets the maximum number of panels to show on one page.

`disc.levels`, `shingle.overlap` default number of levels to use when making shingles or factors. This can be over-ridden by specifying `nLevels` in the `spec` argument.

`shingle.overlap` sets the amount of overlap for shingles; passed to `equal.count`.

`catch.errors` in the default case (TRUE), most errors are caught and displayed to the user in a dialog box. Set to FALSE to avoid the `tryCatch` block and thus allow debugging.

See Also

`latticist`

Examples

```
str(latticist.options())
```

latticistCompose *Latticist API*

Description

Convert a simple specification list into a Lattice plot call, or the inverse operation.

Usage

```
latticistCompose(dat, spec = list(),
                 datArg = substitute(dat),
                 enclos = parent.frame())
```

Arguments

| | |
|--------|--|
| dat | a data frame (with numeric and/or categorical variables). |
| spec | a list specifying the latticist plot. See Details. |
| datArg | the symbol to use for dat in the generated call. |
| enclos | an environment to use as an enclosure around dat for evaluating expressions. |

Details

Elements of spec can include:

xvar, yvar, zvar variables (or expressions) for the x, y and z axes. If all are missing, a hypervariate plot is produced according to defaultPlot (see below). Note that x or y may be discretized by setting doXDisc or doYDisc (see below).

groups a grouping variable or color-coding covariate. This can refer to either a categorical or numeric variable. Many plot types support continuous color covariates, in which case a colorkey will be drawn. For categorical groups a standard key will be drawn.

cond, cond2 conditioning variables. These will be turned into shingles if required, with nLevels distinct levels (see below).

subset subset expression.

varSubset a character vector, the subset of variables (from the data frame) to include in hypervariate plots.

defaultPlot one of "marginal.plot", "splom" or "parallel". Specifies the type of plot to produce if xvar and yvar are missing. Note that groups is supported by all these plots, and cond is supported by "splom" and "parallel".

NOTE: when the data is a table, "parallel" does not produce a parallel plot, but rather a stacked barchart of the table. Also when the data is a table, "splom" produces a pairs layout of mosaic plots.

aspect panel aspect ratio, as a numeric value ($= y / x$), or one of the values "fill", "iso", "xy".

`doLines` whether to add lines to relevant plot types (i.e. those involving numeric variables). For `dotplot` and `cloud` this refers to droplines (type "h") (though if groups are defined in a `dotplot`, type "l" is used). In a grouped `stripplot`, group medians are joined (type "a", `fun = median`). For `qqmath` the points are joined in order (type "l"). For `xyplot` and similar, the line type depends on the nature of the data; if the data x values form a regular sequence or are few in number they are simply joined (type "l"); if duplicate x values are detected, with a reasonable number of unique values, their averages are joined (type "a"); otherwise a smoothing line is added according to `latticeist.getOption("xyLineType")`. This defaults to "smooth" (loess fit), but could reasonably be set to "r" (regression line) or "a" (joined averages).

`doHexbin` whether to use `hexbinplot` rather than `xyplot` for bivariate numeric plots. These can be faster and more effective for large datasets. Note that groups are not supported.

`doSegments`, `doAsError` if `doSegments` is TRUE when all of `xvar`, `yvar` and `zvar` are defined, a `segplot` is produced where the x values are joined to z values by horizontal segments. Alternatively, if `doAsError` is TRUE, segments are drawn from $(x - z)$ to $(x + z)$, and each x point is marked, such that z acts as an error or range about x.

`doTile` when drawing a bivariate numeric plot with a color-covariate (i.e. when `xvar`, `yvar` and groups are all numeric), this option will draw a `tileplot`, which draws a polygon enclosing each point. This may be appropriate when x and y are on the same scale.

`doXDisc`, `doYDisc` set to discretize `xvar` and/or `yvar`, if they are numeric. Either `cut` or `equal.count` is used, depending on the plot type, with `nLevels` distinct levels.

`nLevels` number of levels for discretizing `cond` and `cond2`, optionally `xvar` or `yvar`, and in some cases groups. For shingles, the amount of overlap is taken from `latticeist.getOption("shingle.overlap")`.

`x.relation`, `y.relation` defines the scales in conditioned lattice plots. Can have values in "free", "same" or "sliced".

`doSeparateStrata` when a mosaic plot is produced and `cond` or `cond2` are defined, this defines whether to separate the strata defined by conditioning variables into different panels (the default; uses `cotabplot`), or to include the conditioning variables in the one mosaic plot (`doSeparateStata = FALSE`).

Note that `xvar`, `yvar`, `zvar`, `groups`, `cond`, `cond2`, `subset` must be character strings (or NULL), and will be parsed.

Value

`latticeCompose` returns a `call`.

Author(s)

Felix Andrews <felix@nfrac.org>

References

For an excellent introduction to and coverage of Lattice:

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

See Also

[latticist](#), [Lattice](#)

Examples

```
latticistCompose(CO2)
```

```
latticistCompose(CO2, spec = list(defaultPlot = "parallel"))
```

```
latticistCompose(CO2, spec = list(xvar = "uptake"))
```

```
latticistCompose(CO2, spec = list(yvar = "uptake"))
```

```
latticistCompose(CO2, spec = list(yvar = "uptake", doYDisc = TRUE))
```

Index

- *Topic **color**
 - custom.theme.black, 2
 - latticeStyleDemo, 4
 - latticeStyleGUI, 5
- *Topic **dynamic**
 - latticist, 7
- *Topic **hplot**
 - latticist, 7
 - latticistCompose, 11
- *Topic **iplot**
 - latticist, 7
- *Topic **programming**
 - latticist.options, 9
- *Topic **utilities**
 - cutEq, 3
- as.ordered, 7
- call, 12
- custom.theme, 2, 3
- custom.theme.black, 2
- cut, 4, 7
- cutEq, 3
- equal.count, 10
- Lattice, 7, 8, 13
- lattice.options, 9
- latticeStyleDemo, 4, 6
- latticeStyleGUI, 5
- latticeStyleToBasePar
 - (latticeStyleGUI), 5
- latticist, 7, 10, 13
- latticist.getOption
 - (latticist.options), 9
- latticist.options, 8, 9
- latticistCompose, 7, 8, 10, 11
- n.level.colors (latticistCompose), 11
- options, 9
- palette, 6
- panel.usertext, 5
- par, 5, 6
- playwith, 7, 8
- quantile, 3
- reorder, 4, 7
- reorderByFreq (cutEq), 3
- show.settings, 5
- simpleColorKey (latticistCompose), 11
- simpleTheme, 2, 3, 10
- trellis.par.get, 6
- try.prepanel.loess (latticistCompose),
11