# Package ‘loa’

**Type** Package  
**Title** Lattice Options and Add-Ins  
**Version** 0.2.45.4  
**Date** 2019-09-13  
**Author** Karl Ropkins

**URL** [http://loa.r-forge.r-project.org/loa.intro.html](http://loa.r-forge.r-project.org/loa.intro.html)  
**Maintainer** Karl Ropkins <karl.ropkins@gmail.com>

**Description** Various plots and functions that make use of the lattice/trellis plotting framework.  
The plots, which include loaPlot(), RgoogleMapsPlot() and trianglePlot(), use panelPal(), a function that  
extends 'lattice' and 'hexbin' package methods to automate plot subscript and panel-to-panel  
and panel-to-key synchronization/management.

**Depends** R (>= 3.5.0), lattice  
**Imports** methods, MASS, grid, png, RgoogleMaps, sp, rgdal,  
       RColorBrewer, mgcv, plyr  
**License** GPL (>= 2)  
**LazyLoad** yes  
**LazyData** yes  
**RoxygenNote** 6.1.1  
**NeedsCompilation** no  
**Repository** CRAN  
**Date/Publication** 2019-09-13 16:40:02 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>loa-package</td>
<td>2</td>
</tr>
<tr>
<td>1.1.loaPlot</td>
<td>4</td>
</tr>
<tr>
<td>1.2.loaMapPlot, and.geoplotting.tools</td>
<td>8</td>
</tr>
<tr>
<td>1.3.trianglePlot</td>
<td>14</td>
</tr>
<tr>
<td>1.4.stackPlot</td>
<td>19</td>
</tr>
</tbody>
</table>
Description

The loa package contains various plots, options and add-ins for use with the lattice package.

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>loa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>0.2.45.4</td>
</tr>
<tr>
<td>Date:</td>
<td>2019-09-13</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;= 2)</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

lattice provides an elegant and highly powerful implementation of the Trellis plotting structure described by Cleveland and colleagues. In particular the combination of `panel`...`functions, which can be layered within plots to generate novel visualisations, and simple-to-use conditioning make it a hugely effective tool when working with data.

The loa package contains a number of plot functions developed to make use of this framework. These are summarized in section 1 of this manual, and include:

1.1. loaPlot for various XYZ plots.
1.2. loaMapPlot and associated geoplotting functions.
1.3. `trianglePlot` and associated functions.
1.4. `stackPlot` and associated functions.
1.5. `loaBarPlot` and associated functions.

Other `panel...` functions and example data are summarized in sections 2 and 3, respectively:

2.1. Specialist panels, e.g. `panel.kernelDensity`.
2.2. Specialist panels for polar plotting, e.g. `panel.polarPlot`.

3.1. Example data, e.g. `lat.lon.meuse`.

While such 'stand alone' plot functions are of obvious value, the code framework is of possibly wider interest because it provides a template for the rapid third-party development of novel visualization functions and a highly flexible 'test bed' for the comparison of different data handling strategies.

Therefore, the functions in this package have been written in a relatively dissaggregated fashion so code can be easily rearranged or modified by others to quickly develop alternative plots within the `lattice` framework. Firstly, plot functions in section 1 have where possible been supplied as main plot functions and plot component functions that handle data, axes, panels, etc. Secondly, the workhorse functions, those common functions used through-out the package to simplify many routine operations have been grouped together and summarized in section 4:

4.1. `panelPal`
4.2. plot structure handlers: `formulaHandler`, etc.
4.3. Plot lim(s) and scale(s) handlers: `limsHandler, localScalesHandler`, etc.
4.4. Plot conditioning handlers: `condsPanelHandler`, etc.
4.5. Common plot argument handlers: `cexHandler, colHandler, zHandler`, etc.
4.6. Key handlers: `keyHandler`, etc.
4.7. Other panel functions: `getArgs`, etc.
4.8. List handlers: `listHandler`, etc.

And, finally, functions used for working with data post-plotting, are summarized in section 5:

5.1. Interactive functions for working with plot outputs: `getXY`, etc.

This package is very much intended to be an evolutionary exercise. I use it on a routine basis to develop plots for use elsewhere and compare data visualization methods. However, that working pattern can generate some very 'developer-centric' code. So, I would be very pleased to hear from others - what they did and did not like about the package; what they would have liked to have been different; and, perhaps most interesting for me what they are using it to do.

**Author(s)**

Karl Ropkins <k.ropkins@its.leeds.ac.uk>

**References**

Functions in loa make extensive use of code developed by others. In particular, I gratefully acknowledge the huge contributions of:


See Also

loaPlot, GoogleMap, trianglePlot

1.1.loaPlot

Description

loaPlot is a standard XYZ plotting function, where X and Y are the axes of a conventional XY plot and Z is an element (or elements if supplied in the form Z1 + Z2 + Z3...) visualized at associated XY coordinates. By default, loaPlot links Z to plot point size and color to generate a bubbleplot style output, or using modified plot calls other plot types.

Usage

loaPlot(x, data = NULL, panel = panel.loaPlot, ...
    ..., local.scales = FALSE, reset.xylims = TRUE,
    load.lists = NULL, by.group = NULL, by.zcase = NULL,
    preprocess = TRUE)

#standard panels

panel.loaPlot(..., loa.settings = FALSE)
panel.loaPlot2(..., loa.settings = FALSE)

#grids

panel.loaGrid(grid.x = NULL, grid.y = NULL,
    xlim = NULL, ylim = NULL, ...
    grid = NULL, panel.scales = NULL)

Arguments

x
A formula with the general structure z ~ x * y | cond applied like in the lattice function levelplot or a matrix. For a formula, x and y are the horizontal and vertical axes, z is any additional information to be used in point, symbol, surface or glyph generation, and cond is any additional conditioning to be applied. x and y are required elements; z and cond are typically optional. (Note: this element of the plot is handled by formulaHandler).

data
If supplied, the assumed source of elements of x, typically a data.frame.
1.1. loaPlot

panel

panel is the function to be used when generating the content of the individual panels within the \texttt{lattice} plot. By default, this is the \texttt{loa} panel function \texttt{panel.loaPlot}.

Additional arguments are passed on to related functions.
For \texttt{loaPlot} these are \texttt{colHandler}, \texttt{cexHandler} and the function set by \texttt{panel}. This mechanism provides access to most common plot parameters, e.g. \texttt{col}, \texttt{pch}, and \texttt{cex} for plot symbol color, type and size, respectively. By default, both data point color and size are \texttt{z}-scaled for \texttt{loaPlot}. If \texttt{z} is supplied, and \texttt{cex} and \texttt{col} are not set by the user in the plot command, these plot properties are managed by \texttt{cexHandler} and \texttt{colHandler}, respectively. \texttt{cexHandler} and \texttt{colHandler} arguments can be also be passed directly as part of the \texttt{loaPlot} command to fine-tune these, e.g. \texttt{cex.range} to change the \texttt{cex} range that \texttt{z} values are scaled to and \texttt{col.region} to change the color range that is applied to \texttt{z} when coloring points. See associated Help documents for further information.

local.scales

For \texttt{loaPlot} only, logical. If \texttt{TRUE}, this removes the standard \texttt{lattice} axis from the plot. It is intended to be used with \texttt{panel} functions which generate their own axes or have no axes.

reset.xylims

For \texttt{loaPlot} only, logical or character vector. If a logical, if the panel outputs are preprocessed (using \texttt{panelPal}), should the x and y limits be reset? If a character vector, one or more terms controlling post-processing plot range management: \texttt{refit.xylims}, equivalent to \texttt{reset.xylims = TRUE}; and \texttt{max.xylims}, to reset both x and y ranges to maximum.
(Note: If \texttt{xlim} or \texttt{ylim} are supplied in the plot call, these will typically override all \texttt{reset.xylims} settings.)

load.lists

For \texttt{loaPlot} only, character vector. In-development alternative to list based arguments. This option identifies plot call arguments that \texttt{loaPlot} should manage using \texttt{listLoad}. See associated help documentation for further details.

by.group, by.zcase

For \texttt{loaPlot} only. Arguments for routine by group and by zcase handling of plot inputs.
Important: These are current under review.

preprocess

For \texttt{loaPlot} only, logical, passed to \texttt{panelPal}. If \texttt{TRUE}, and used with a correctly configured \texttt{panel} function, this processes the plot input before generating the plot. This means color scales in the different plot panels and the key are automatically aligned and the associated trellis object output contains the \texttt{panel} function outputs rather than the inputs. See \texttt{panelPal} Help documents for further information.

loa.settings

For \texttt{panel...} functions only, logical, passed to \texttt{panelPal} to manage plot reworking. See associated Help documents for further information.

grid.x, grid.y, xlim, ylim, grid, panel.scales

For \texttt{panel.loaGrid} only, grid settings, typically recovered by \texttt{loaPlot}.

Details

\texttt{loaPlot} provides lattice-style conditioning/handling for a range of commonly used XYZ plotting options. It is perhaps easiest pictured as a ‘mid point’ alternative somewhere between the standard \texttt{lattice} plot functions \texttt{xyplot} and \texttt{levelplot}.
The default form of the plot uses an extension of the subscripting methods described by Deepayan Sarkar in Chapter 5 of Lattice (see sections on scatterplots and extensions). The default output is a bubble plot (see example 1 below).

Value

loaPlot returns a trellis object, much like a conventional lattice plot function.

Note

panel.loaPlot2 is an alternative versions of panel.loaPlot that is currently under revision. Please use with care.

loaPlot arguments by.group and by.zcase are currently in revision. Please use with care.

Author(s)

Karl Ropkins

References

These functions make extensive use of code developed by others.

See Also

In loa: panelPal
In other packages, see
R lattice: xyplot; and levelplot.

Examples

```r
## Example 1
## Basic usage

loaPlot(Ozone~Solar.R*Temp|Wind>8,
    data=airquality,
    col.regions="Blues")

# Notes:
# Formula structure z ~ x * y |cond like levelplot.
# Data (and groups) assignment like in standard lattice plots.
# By default z is linked to col and cex.
# Unless overridden by user inputs or group or zcase setting.
# Plot passed via ...Handler functions to provide shortcut plot
# reworking, e.g. here colHandler handles color scales
```
# using col.region to generate a color range.
# (Here, arguments like "Blues" and c("green", "red") are
# allowed and handled using functions in the RColorBrewer
# package.)

# Formula structures:
# ~ x * y like xyplot y ~ x
# ~ x * y | cond like xyplot y ~ x | cond
# z ~ x * y like xyplot y ~ x, col=f(z), cex=f(z)
# z ~ x * y | cond like xyplot y ~ x | cond, col=f(z), cex=f(z)
# z ~ x * y, groups = g like xyplot y ~ x, groups=g, cex=f(z)
# z1 + z2 ~ x * y (zcases)
# etc

## Example 2
## Basic modifications

loaPlot(Ozone~Solar.R*Temp, groups=airquality$Wind>8,
data=airquality)

# When groups are applied, by default group id is linked to col.
# The follow example illustrates three options:

loaPlot(Ozone~Solar.R*Temp, groups=airquality$Wind>8,
data=airquality,
group.args=c("pch"), pch=c(1,4),
col="blue")

# notes:
# Here, group.args is used to change the default group arguments.
# (So, pch rather than col is used to identify groups.)
# pch is then assigned by group rather than by (x,y) case or z case.
# (See panelPal Help further further details of assignments in loa.)
# col supplied by the user supercedes the default z linkage.
# (So, here cex remains z scales but col is fixed as blue.)

## Example 3
## Key handling

loaPlot(Ozone~Solar.R*Temp, data=airquality,
col.regions=c("green", "red"))

# Key settings are by the key argument (as in lattice)
# or key... arguments via keyHandler and listLoad, so e.g.:

loaPlot(Ozone~Solar.R*Temp, data=airquality,
col.regions=c("green", "red"),
key.fun = draw.loaColorKey)

# Notes:
# By default the loaPlot uses draw.loaPlotZKey to generate
# its color key unless an alternative is supplied via key.fun.  
# (Here, the draw.colorKey wrapper draw.loaColorKey is used to  
# generate a color bar similar to that in levelplot.)

## Example 4
## panels

loaPlot(Ozone~Solar.R*Temp|Wind>8, data=airquality,  
        col.regions="Reds")

# The combined use of loaPlot, panelPal and appropriately configured  
# panel functions provides automatical handling of a range of plot  
# elements, e.g.:  

loaPlot(Ozone~Solar.R*Temp|Wind>8, data=airquality,  
        col.regions="Reds", panel=panel.binPlot)

# Notes:  
# Here, the choice of default key is set by the panel... function;  
# the panel by default bins data by location and for each bin cell  
# calculates the mean Ozone concentration just like a standard  
# lattice panel would, but it also tracks these values (calculated  
# within the panels) and scales panel-to-panel and panel-to-key  
# so users do not have to do that retrospectively; and, finally,  
# it retains in-panel calculations so users can recover them.  
# (See associated helps for further details: ?panelPal about methods;  
# and ?panel.binPlot about the panel function.)

---

1.2.loaMapPlot.and.geoplotting.tools

Map plotting for lattice

Description

Plotting georeferenced data on maps using lattice, RgoogleMaps, and OpenStreetMap or GoogleMaps

Usage

loaMapPlot(x, data = NULL, panel = panel.loaPlot, map = NULL,  
           map.panel = panel.loaBGMapPlotRaster, recolor.map = FALSE,  
           show.axes = FALSE, ..., map.source = getRGMapArg,  
           lon.lat = FALSE)

RgoogleMapsPlot(x, data = NULL, ...)
#map handlers
getRGMMapArg(ylim, xlim, ..., lim.borders = 0.1)
loaMapArg(object = trellis.last.object())

#map panels
panel.loaBGMapPlotRaster(map)
panel.loaBGMapPlot(map)

#scaling and axis handlers
LatLon2MercatorXY(latitude, longitude, ...)
MercatorXY2LatLon(mx, my, ...)
axis.components.loaMap(map, xlim = NULL, ylim = NULL, ...)
xscale.components.loaMap(lim, ..., map = map)
yscale.components.loaMap(lim, ..., map = map)

#legacy functions
googleMap(...)
GoogleMap(...)
makeMapArg(...)
getMapArg(object = trellis.last.object())

Arguments

x

For MapPlots only. A formula setting the plot structure, by default \( z \sim \text{latitude} \times \text{longitude} | \text{cond} \). The axis elements latitude and longitude are required, while \( z \) and conditioning \( \text{cond} \) are optional.

data

For MapPlots only. If supplied, the assumed source of the elements of formula \( x \), typically a data.frame.

panel, map.panel

For MapPlots only. The panels to use when generating the plot data and map layers, respectively. \( \text{panel} \) is by default the standard \( \text{loa} \) scatter plot panel \( \text{panel.loaPlot} \). \( \text{map.panel} \) can be the default \( \text{panel.loaBGMapPlotRaster} \) or the alternative \( \text{loaBGMapPlot} \).

map, map.source

For MapPlots and related functions only. If supplied, \( \text{map} \) is a modified RgoogleMaps output, generated using either package and \( \text{getRSMapArg} \). If \( \text{NULL} \) (default), the MapPlot using the get function assigned using \( \text{map.source} \) to get the map.

recolor.map

For MapPlots only. If supplied, a vector of elements that R can treat as colors, used as a color scale to recolor map. This uses standard \( \text{RColorBrewer} \) functions, so can handle arguments like \( \text{recolor.map} = \text{c("white","grey")} \) for greyscale, etc. Disabled by the default, or \( \text{FALSE} \) or \( \text{NULL} \) settings.

show.axes

For MapPlots only. Logical (default \( \text{FALSE} \)) to include or exclude latitude and longitude axes and associated labelling from MapPlots.

... Additional arguments are passed on to related functions.

For MapPlots these are \( \text{getRSMapArg} \), \text{colHandler}, \text{cexHandler} and \text{xplot}.

By default both data point colour and size are \( z \)-scaled for MapPlots. So, if \( z \) is supplied, these plot properties are managed by \text{cexHandler} and \text{colHandler}. 
respectively, unless defined in the call or panel setup. `cexHandler` and `colHandler` arguments can also be passed direct as part of a `MapPlot` command to fine-tune z-scaling, e.g. `cex.range` to change the `cex` range that `z` values are scaled across and `col.region` to change the color range that is applied to `z`. See associated Help documents for further information.

Similarly, argument passing to `xyplot` in both `MapPlots` provides access to most common plot parameters, e.g. `col`, `pch`, and `cex` for plot symbol color, type and size, respectively.

`getMap` functions pass arguments to related RgoogleMaps functions.

### lon.lat
For `MapPlots` only, logical. If TRUE applies `z ~ lon * lat | cond`? This operation is handled using the `formula.type` argument in `formulaHandler`

### ylim, xlim, lim.borders, lim
The latitude and longitude plot ranges. `ylim` and `xlim` are only required by `get` functions, which use these to set the requested map size. The additional argument `lim.borders`, sets the borders to be added. The default code0.1 adds a 10% border to both axes, while two-element vectors allow provide separate control of the axes. For the axis handlers (yscale... and xscale...) the local alternative `lim` is used for both `ylim` and `xlim` in generic code.

### object
For `loaMapArg`, the `MatPlot` plot to extract a map from. (If not supplied, this is assumed to last `lattice` plot.)

### latitude, longitude
For `LatLon2MercaptorXY`, paired latitude and longitude time-series to be converted to paired Mercator X and Y time-series.

### mx, my
For `MercatorXY2LatLon`, paired Mercator X and Y time-series to be converted to paired latitude and longitude time-series.

### Details

`loaMapPlot` provides lattice-style conditioning/handling for RgoogleMaps outputs. This uses `loaPlot` and `panelPal` to manage default panel and key settings.

`RgoogleMapsPlot` is a shortcut for using `loaMapPlot` with maps produced by `RgoogleMaps`.

`getMap...` functions accept latitude and longitude ranges and related RgoogleMaps function arguments, and produces an output suitable for use as the map argument in subsequent (same area) `MatPlots` or in `panel.loaBGMPlotRaster` or `panel.loaBGMPlotMaps` panel function calls.

`LatLon2MercatorXY` converts supplied latitude and longitude time-series to Mercator X and Y time-series. `MercatorXY2LatLon` converts supplied Mercator X and Y time-series to latitude and longitude time-series. Both output results as lists containing `newX` and `newY` elements.

`panel.loaBGMPlotRaster` and `panel.loaBGMPlotMaps` are lattice panel functions that generate map layers for a lattice plot using `getMap...` function outputs.

`yscale.components.loaMap` and `xscale.components.loaMap` are y- and x-axis handlers for use with the above panels.

`axis.components.loaMap` is a wrapper that combines `yscale.components.loaMap` and `xscale.components.loaMap` and allows both axis to be set from the lattice function argument `axis` rather than each individually, via `yscale.components` and `xscale.components`. 

NOTE: The MapPlots and related panel and axis handling functions are in development functions that have been subject to changes.

googleMap and GoogleMap are RgoogleMapsPlot wrappers, included these were names of earlier versions of this function, that have been documented in previously published code.

getMapArg was the earlier name for loaMapArg.

quickMap, a crude map plot function intended to demonstrate the use of the 'handler' functions, is no longer packaged but the code is included in the examples below.

Value

MapPlots return trellis objects, much like conventional lattice plot functions.

get...MapArgs return lists for use as maps in ...MapPlot calls.

getMapArg recovers the map from an existing GoogleMap output.

panel.GoogleMapsRaster and panel.GoogleMaps generate panel outputs suitable for use in standard lattice panel functions.

yscale.components.GoogleMaps, xscale.components.GoogleMaps generate suitable latitude, longitude scales for use with map layers. axis.components.GoogleMaps is a wrapper for their routine use.

Note

Google Maps outputs are 2D projections of curve sections of the Earth’s surface. Therefore, the assignment of points within panels and the annotation of latitudes and longitudes along axis needs to be locally handled to account for this.

LatLon2MercatorXY and MercatorXY2LatLon are used to locally scale both axis and data in these plots. These apply methods and code in the "sp" package.

Important: Users wanting to add data to these plots, e.g. using update or layers in latticeExtra, should first rescale the data. Likewise, users wanting to add maps to other plots will need to rescale plotted data to use these maps. See Example 1 below.

Important: map APIs often return a map panel larger than the data (latitude, longitude) range requested. This means you may get back a map that is large than necessary. As xlim and ylim are passed to the API when they are called resetting these can produce similar effects (so you may not get exactly the map range you ask for! If you want to manually optimise an RgoogleMapsPlot map range, the best option is currently to start with:

RgoogleMapsPlot(...,size=c(640,640))

...and then change either or both size values until you generate an appropriate map size.

Author(s)

Karl Ropkins
References

This function makes extensive use of code developed by others.


Users should also always cite map sources when using outputs from these MapPlots:

For GoogleMap maps: Map layer: Map data (c) [year] Google
For OpenStreetMap maps: Map layer (c) OpenStreetMap contributors

See Also

In other packages, see

RgoogleMaps: GetMap.

sp: coordinates; proj4string; and, spTransform.

lattice: xyplot; panel.xyplot; and panel.levelplot.

Examples

```r
## Example 1
## quickMap code
## as example of third-party use of functions

quickMap <- function(lat, lon, show.data = FALSE, ...){

  #get map
  map <- getRGMapArg(lat, lon, ...)

  #scale axis for map projection
  map.axis.comps <- axis.components.loaMap(map)
  map.axis <- function(components, ...)
    axis.default(components = map.axis.comps, ...)

  #scale data for map projection
  temp <- LatLon2MercaptorXY(lat, lon)
  lat <- temp$newY
  lon <- temp$newX

  #plot data on map
  xyplot(lat~lon,
```
```r
xlim = map$xlim, ylim = map$ylim,
aspect = map$aspect,
axis = map.axis,
panel = function(...)
   {panel.loaGBMapPlotRaster(map)
   if(show.data)
      panel.xyplot(...)
   }, ...)
```

## Example 2
## Off-line GoogleMap examples

# Use a subsample of lat.lon.meuse
temp <- lat.lon.meuse[sample(1:155, 15),]
RgoogleMapsPlot(zinc~latitude*longitude, col.regions=c("grey", "darkred"),
data=temp, map=roadmap.meuse)

RgoogleMapsPlot(zinc~latitude*longitude, col.regions=c("grey", "darkred"),
panel=panel.binPlot,
data=temp, map=roadmap.meuse)

RgoogleMapsPlot(cadmium*50+copper*10+lead*2+zinc~latitude*longitude,
col.regions=c("grey", "darkred"),
key.z.main="Concentrations", panel.zcases = TRUE,
data=temp, map=roadmap.meuse)

RgoogleMapsPlot(cadmium*50+copper*10+lead*2+zinc~latitude*longitude,
col.regions=c("grey", "darkred"), panel=panel.zcasePiePlot,
data=temp, map=roadmap.meuse)

# Note 1:
# Here, the map argument is supplied so example works off-line.
# If not supplied and R is on-line, the MapPlots will get map
# from either Google or OpenStreetMap API. Repeat any of above
# without map argument when on-line. For example:
# Not run:
   RgoogleMapsPlot(zinc~latitude*longitude, col.regions=c("grey", "darkred"),
data=lat.lon.meuse)
# End(Not run)
# (The map will appear slightly different because non-default
# size and maptype settings were used to make roadmap.meuse. See
# ?roadmap.meuse for details.)

# Note 2:
# To make a map for use with panel.loaGBMapPlotRaster or panel.loaGBMapPlotRaster
# without plotting use makeRGMapArg() or makeRGMMapArg().
# To recover a map from a previously plotted loa MapPlot use loaMapArg().
```
1.3.trianglePlot

Triangle plots

Description

Triangle plot functions for Lattice.

Usage

trianglePlot(x, data = NULL, ..., ref.cols = TRUE)

#standard panels
panel.trianglePlot(x = NULL, y = NULL, a0 = NULL, b0 = NULL, c0 = NULL, ..., loa.settings = FALSE, plot = TRUE, process = TRUE)
panel.trianglePlotFrame(..., grid = NULL, axes = NULL)
panel.trianglePlotGrid(alim = NULL, blim = NULL, clim = NULL, ..., grid = TRUE, panel.scales = NULL)
panel.trianglePlotAxes(alim = NULL, blim = NULL, clim = NULL, ..., axes = TRUE, ticks=TRUE, annotation=TRUE, panel.scales = NULL)

#other panels
panel.triangleByGroupPolygon(x = NULL, y = NULL, a0 = NULL, b0 = NULL, c0 = NULL, ..., loa.settings = FALSE, plot = TRUE, process = TRUE)
panel.triangleKernelDensity(x = NULL, y = NULL, a0 = NULL, b0 = NULL, c0 = NULL, ..., loa.settings = FALSE, plot = TRUE, process = TRUE)
panel.triangleSurfaceSmooth(x = NULL, y = NULL, z = NULL, a0 = NULL, b0 = NULL, c0 = NULL, ..., loa.settings = FALSE, plot = TRUE, process = TRUE)

#data handlers
triABC2XY(a, b = NULL, c = NULL, ..., force.abc = TRUE, if.na = "remove.row", if.neg = "remove.row", verbose = FALSE)
triXY2ABC(x, y = NULL, ..., force.xy = TRUE, verbose = FALSE)
triLimsReset(ans)
triABCSSquareGrid(a, b = NULL, c = NULL, ..., n=100)
Arguments

x For trianglePlot only, a formula in structure z ~ a0 + b0 + c0 | cond. The elements a0, b0 and c0, the inputs for the three axis on the triangle plot, are required, while z and conditioning (cond) are optional. (For other functions, x may be used as the pair to y. See y below.)

data For trianglePlot only, if supplied, the assumed source of the elements of formula x, typically a data.frame.

... Additional arguments.

ref.cols Either a logical to turn off/on grid color-coding or a vector of colors to be applied to a0, b0 and c0 axes and grids. These are applied to the grid lines and axes tick and annotation components. Some users, particularly those less familiar with triangle plots, can find such color referencing helpful when analyzing such plots. By default, the colorings are quite subtle, so users can see the effect if them look for it but it does not take over the plot when it is not focused on. Finer control can be achieved using axes, ticks, grid, etc. (See below).

y, a, a0, b, b0, c, c0, z (and x in relevant functions). a/a0, b/b0 and c/c0 are the three scales of the triangle plot, and x and y are the equivalent 2-D projections. The arguments are typically options in panel functions (panel... functions), conversion functions (triABC2XY and triXY2ABC) and the scaling function triLimsReset. z is the z-case from the plot formula.

loa.settings, plot, process loaPlot arguments used to manage panelPal activity.

grid, axes, ticks, annotation User-resets for the axes, grid, tick and annotation elements of the plots. These can be NULL or FALSE to turn off, TRUE to show, a vector (in which case they are assumed to be color assignments) or a list of standard plot parameters, e.g. col, lty, lwd, etc. for color, line type and line thickness, etc. Plot parameter assignments are applied to all axes unless specific axes are identified. For example, trianglePlot calls including grid.col = 2 make all axes red, while calls including grid.a0.col = 2 only recolor the first (a0) axis.

alim, blim, clim Delimiters for a, b and c scales, equivalent to xlim and ylim in conventional plots, but less flexible. See Details below for more information.

panel.scales A local argument, typically a list, that controls the appearance of the a0/b0/c0 axes. This is roughly equivalent to the scales argument used by conventional lattice plots to handle x and y axis, but intended for non-standard scales, such as the triangle axes used here. It can be set directly or used in combination with the local scale(s) handler function localScalesHandler to override/hijack standard scales operations. (See note below).

force.abc, force.xy Logicals. If a list or data.frame is supplied to triABC2XY or triXY2ABC as a source or a/b/c or x/y respectively should appropriately named elements be used regardless of order? See Note below.

if.na Character. Handling method to be used if NA's are present. The default option 'remove.row' replaces all entries in the same row with NA's. (Note: this is
1.3.trianglePlot

different from na.omit which would remove the whole row. Here, the row is retained as NAs to maintain indices for conditioning.) Other options currently include: 'make.zero' to replace the NA with 0; and 'keep.as.is' to leave unchanged.

if.neg Character. Like if.na but for negative values: 'remove.row' to replace all entries in the same row with NAs; 'make.zero' to replace all negative values with 0; 'rescale.col' rescales any column (i.e., a, b or c) that contains a negative from zero by subtracting the minimum.

verbose Logical, default FALSE. Should a full output be returned? The alternative FALSE generates a minimal report.

ans For triLimResets only, a trellis ouput, e.g. a lattice plot, to be scaled and plotted based on the assumption that it is a trianglePlot.

n For triABCSquareGrid only, number of points to divide each axes by when generating the data grid.

Details

trianglePlot generates a triangle plot using the lattice framework.

panel.trianglePlot... functions handle the appearance of triangle plot outputs.

panel.trianglePlot, which is assigned as the default panel manages both the data layer of the plot and the plot frame (axes, grid, annotation, etc).

panel.trianglePlotAxes and panel.trianglePlotGrid generate axes and grid components of the plot, and panel.trianglePlotFrame is a wrapper for these. The data layer, which by default is panel.loaPlot, can be accessed separately using the data panel argument.

triangleKernelDensity generates a kernel density surface for supplied a0, b0 and c0 cases.

triangleSurfaceSmooth generates a smoothed surface for supplied a0, b0, c0 and z cases.

triABC2XY converts supplied (a, b, c) coordinates to an (x, y) scale suitable for use with triangle plot functions.

triXY2ABC converts supplied (x,y) coordinates from triangle plots to the associated proportional (a, b, c) scale.

There are various options for range limiting with triABC2XY, triXY2ABC, and therefore triangle plots. Firstly, limits can be set individually with alim, blim and clim, much like with xlim and ylim for conventional plots. However, they can also be set at once using lims, as in e.g. lims = c(0,1) to set all axes to full ranges, or on the basis of minimum and maximum cut-offs using abc.mins and abc.maxs, respectively.

trianglePlot uses localScalesHandler to override normal lattice handling of scales. This allows parameters for axes other than 'x' and 'y' to be passed via the scales argument for axis generation within the plot panel itself. The function does this by recovering the information for each of the local axes (here a0, b0 and c0) from scales, and passing this on to the plot as the argument panel.scales which can then be evaluated by an appropriate panel... function like panel.trianglePlotAxes. At the same time it also resets scales to stop the standard axes being generated. The intention here is two-fold. Firstly, to provide plot users with an axes control mechanism like the standard scales control of x and y that they already know. And, secondly, to provide developers with a simple framework for the quick addition of non-standard axes or scales. See localScalesHandler and panel.localScale for further details.
trianglePlot uses `getPlotArgs` to manage lattice defaults and plot developer and user resets for the different plot components (axes, ticks, grid, annotation). As with `localScalesHandler`, the intention here is to provide more routine access to higher level plot control.

**Value**

trianglePlot returns trellis objects, much like conventional lattice plot functions. `panel.trianglePlot...` functions are intended for use within a `trianglePlot` function call.

- triABC2XY returns a list containing the named components `x` and `y`, which are the 2-D (x,y) transformations of supplied (a,b,c) `trianglePlot` elements.
- triXY2ABC returns a list containing the named components `a`, `b` and `c`, which are the (a,b,c) triangle plot coordinates associated with supplied 2-D (x, y) that `trianglePlot` would generate.
- resetTriLims returns a supplied trellis object, rescaled based on the assumption that it is a triangle plot.

**Note**

**General:**

With triangle plots, the (a0, b0, c0) scales are proportional. So regardless of the absolute sizes of a coordinate set (a,b,c), values are plotted and handled as proportions, i.e. \( a/(a+b+c) \), \( b/(a+b+c) \) and \( c/(a+b+c) \), respectively. This means that absolute values of a, b and c are lost when points are plotted on these axes. So, the function `triXY2ABC` returns the relative proportions of a, b and c, not the absolute amounts, when translating a 2-D (x,y) coordinates into the associated (a, b, c) coordinates.

**Development:**

This is an in-development plot, and 'best handling' strategies have not been decided for several elements. So, future versions of these functions may differ significantly from the current version.

**In particular:**

- Current axes assignments, e.g. (a, b, c) versus (a0, b0, c0), etc., have not be finalised. So, these may change in future versions of the function.
- Currently, `trianglePlot` scale adjustment options have been limited. Options under evaluation include: (1) by `alim`, `blim`, `clim` setting, equivalent to `xlim` and `ylim`, (2) by `lims` to set all axes ranges the same, and (3) by `maxs` to setting all axes range maximums and `mins` to set all axes range minimums, etc.

These options are currently only available via the data converters.

One of the issues here is that the axes ranges are all inter-linked. The range of one axes is a function of the other two axes ranges. So, setting these can generate contradictions. For example, \( \text{lims}=c(0, 0.1) \) should in theory set all ranges to \((0, 0.1)\). But, the triangle \( a = b = c = c(0, 0.1) \) cannot exist. Therefore, the plot would attempt to recover the extended range that includes all the requested ranges \( a = c(0, 0.1), b = c(0, 0.1) \) and \( c = c(0, 0.1) \), which in this case is the full range: \( a = b = c = c(0, 1) \). Suggestions on this topic are very welcome.

**trianglePlot:**

As part of the `loa` version 0.2.19 update, `trianglePlot` was rewritten to run with the most recent version of `panelPal` function. This means all plot functions in `loa` now use the most recent version of `panelPal`. 
This update should provide improved plot handling similar to recent versions of \texttt{loaPlot} and \texttt{GoogleMap} functions which both already (from versions 0.2.0 onwards) use the latest version of \texttt{panelPal}.

\texttt{panel.trianglePlotAxes}:
Code currently in revision. Please handle with care.

\texttt{triABC2XY, triABCSquareGrid}:
Code currently in revision. Please handle with care.

\textbf{Author(s)}
Karl Ropkins

\textbf{References}
These function makes extensive use of code developed by others.
Currently, several triangle plotting methods are under evaluation for use within this package. These are:
The tri-plot method of Graham and Mudgley:
The triangle.param method of Chessel (as coded in R package ’ade4’)
And the trilinear plot of Allen as reported by Zhu:
In this version of the package tri-plot methods are used for the \texttt{triABC2XY} and \texttt{triXY2ABC} transforms and a modification \texttt{triangle.param} methods is used to calculate suitable values for \texttt{alim}, \texttt{blim} and \texttt{clim}.
As elsewhere, the use of \texttt{lattice} is also gratefully acknowledged:

\textbf{See Also}
In \texttt{loa}: For in-panel axis/scale generation, see \texttt{loaPlot, panelPal, localScalesHandler} and \texttt{panel.localScale}.
In other packages: \texttt{xyplot} in \texttt{lattice}. 
Examples

## Example 1
## Basic triangle plot usage

trianglePlot(cadmium~copper+lead+zinc|lime, 
data=lat.lon.meuse)

# Notes:
# Formula structure z ~ a0 + b0 + c0 |cond, where a0, b0 and
# c0 are the three axes of the triangle plot
# Data (and groups) assignment like in standard lattice plots.
# By default z is linked to col and cex.
# Unless overridden by user inputs or group or zcase setting.
# Plot handling is similar to loaPlot
# (So, see ?loaPlot and ?panelPal for further details.)

# Formula variations:
# basic triangle plot without z values assignment
# trianglePlot(~a0+b0+c0, ...)
# ... with z values set
# trianglePlot(z=a0+b0+c0, ...)
# ... with grouping
# trianglePlot(z=a0+b0+c0, groups=grps, ...)  

## Example 2
## Basic frame (axes, grid, tick, annotation) management

trianglePlot(~1+1+1, type="n", 
grid.alpha = 0.2, 
ticks.alpha = 0.2)  ## grid and tick alpha reset

# notes:
# Here, grid and ticks arguments are used to remove or modify these
# elements of the plot frame individually.
# Setting can be management in list form like in normal lattice or
# in a loa shorthand where e.g. the argument grid.a0.lty = 1 is equivalent
# to grid = list(a0 = list(lty = 1))
# (So, quicker if you are only changing a small number of elements.)

1.4.stackPlot

Stack plots

Description

Stack plot functions for Lattice.
Usage

```r
stackPlot(x, data = NULL, ...)
```

#standard panels

```r
panel.stackPlot(..., process=TRUE, plot=TRUE,
               loa.settings = FALSE)
```

#data handlers

##currently not exported

Arguments

- `x` For `stackPlot` only, a formula of general structure `y1 + y2 ~ x | cond`, etc. The elements `y1`, `y2`, etc are stacked on the y-axis, and plotted against `x`. Both are required.
- `data` For `stackPlot` only, if supplied, the assumed source of the elements of formula `x`, typically a `data.frame`.
- `...` Additional arguments.
- `loa.settings`, `plot`, `process` `loaPlot` arguments used to manage `panelPal` activity.

Details

- `stackPlot` generates a stack plot using the lattice framework.
- `panel.stackPlot` handles the appearance of triangle plot outputs.

Value

- `stackPlot` returns trellis objects, much like conventional `lattice` plot functions.
- `panel.stackPlot` is intended for use within a `trianglePlot` function call.

Note

Development:

This is an in-development plot, and 'best handling' strategies have not been decided for several elements. So, future versions of these functions may differ significantly from the current version.

In particular:

- `stackPlot`:
  - The `stackPlot` argument `x` may include conditioning in the form `y ~ x | cond`. However, exact handling is has not been defined, so may subject to change.
  - To generate the stacks, `stackPlot` resets y values by applying `y - min(y)` to each layer and then stacks them. It also generates a second element `y0` of associated baselines. This is then used in the form `x = c(x, rev(x)), y = c(y, rev(y0))` with `panel.polygon` to generate the stack layers.
1.5.loaBarPlot

panel.stackPlot:
Code currently in revision. Please handle with care.

Author(s)
Karl Ropkins

References
These function makes extensive use of code developed by others.
As elsewhere, the use of lattice is also gratefully acknowledged:

See Also
In loa: loaPlot and panelPal.
In other packages: xyplot and panel.polygon in lattice.

Examples

## Example 1
## Basic stack plot usage

## Not run:
stackPlot(lead~dist.m, data=lat.lon.meuse)
stackPlot(cadmium+copper+lead+zinc~dist.m, data=lat.lon.meuse)
## End(Not run)

stackPlot(cadmium*40+copper*5+lead+zinc~dist.m, data=lat.lon.meuse)

1.5.loaBarPlot loaBarPlot

Description
Bar plot variation using for Student Project.

Usage
loaBarPlot(x, y=NULL, groups=NULL, cond=NULL, data=NULL, ..., drop.nas=TRUE, stat=NULL)
Arguments

- **x**  
  Either the x case for the bar plot or a plot formula. If the x case, typically a vector of factor or grouping terms, used to assign x positions in bar plot. If a plot formula, a plot description in the format y~x|cond, where x is a factor or grouping term and y and cond are optional.

- **y**  
  (Optional) The y case for the bar plot, typically a vector of numeric terms, used with stat when calculating summary information for bar plots.

- **groups, cond**  
  (Optional) The group case for the bar plot, typically a vector of factor or grouping terms.

- **data**  
  (Optional) if supplied, the assumed source of the plot elements, x, y, groups and cond, typically a data.frame.

- **...**  
  Additional arguments, passed on to lattice function.

- **drop.nas**  
  Option to drop NAs before plotting results, default TRUE.

- **stat**  
  If supplied, the function used to summarise y data after grouping by x, groups and cond. By default, this counts x cases if y is not supplied, or calculates sum of y values if these are supplied.

Details

`loaBarPlot` summarises supplied plot data and generates a bar plot using the lattice framework.

Value

`loaBarPlot` returns trellis objects, much like conventional lattice plot functions.

Note

Development:

This is an in-development plot, and 'best handling' strategies have not been decided for several elements. So, future versions of these functions may differ significantly from the current version.

In particular:

- `loaBarPlot`:
  This is for student project, may not be staying.
  Code currently in revision. Please handle with care.

Author(s)

Karl Ropkins

References

These function makes extensive use of code developed by others.

As elsewhere, the use of lattice is also gratefully acknowledged:

2.1.specialist.panels

See Also

In loa: listUpdate and colHandler.
In other packages: barchart in lattice.

Examples

```r
## Example 1
## Basic bar plot usage

loaBarPlot(Species, Sepal.Width, data=iris, stat=mean)

# or equivalent using formula
## Not run:
loaBarPlot(Sepal.Width~Species, data=iris, stat=mean)
## End(Not run)
```

Description

Specialist panel functions for use with lattice and loa plots.

Usage

```r
panel.loaLevelPlot(x = NULL, y = NULL, z = NULL, ..., 
    loa.settings = FALSE)
panel.surfaceSmooth(x = NULL, y = NULL, z = NULL, 
    breaks = 200, x.breaks = breaks, y.breaks = breaks, 
    smooth.fun = NULL, too.far=0, ..., 
    plot = TRUE, process = TRUE, loa.settings = FALSE)
panel.kernelDensity(x, y, z = NULL, ..., n = 20, 
    local.wt = TRUE, kernel.fun = NULL, too.far = 0, 
    panel.range = TRUE, process = TRUE, plot = TRUE, 
    loa.settings = FALSE)
panel.binPlot(x = NULL, y = NULL, z = NULL, 
    breaks=20, x.breaks = breaks, y.breaks = breaks, 
    x1=NULL, x2=NULL, y1=NULL, y2=NULL, 
    statistic = mean, pad.grid = FALSE, ..., 
    plot = TRUE, process = TRUE, loa.settings = FALSE)
```
Arguments

\texttt{x, y, z} \quad \text{lattice function arguments passed down to individual panels.}

\ldots \quad \text{Additional arguments, typically passed on. See below.}

\texttt{loa.settings, process, plot}

For \texttt{panel...} functions that intended to be handled using \texttt{panelPal}, \texttt{loa.settings}, a logical indicating if the safe mode setting should be returned. \texttt{process} and \texttt{plot}, logicals, indicating if the process and plot sections of the panel function should be run. See below and \texttt{panelPal help documents} for further details.

\texttt{breaks, x.breaks, y.breaks}

\quad (For \texttt{panel.surfaceSmooth} and \texttt{panel.binPlot}) How many break points to introduce when smoothing a surface or binning data. \texttt{breaks} can be used to set the same number of breaks on both axes, while \texttt{x.breaks} and \texttt{y.breaks} can be used to set these separately.

\texttt{smooth.fun}

\quad (For \texttt{panel.surfaceSmooth}) A function that can fit a surface estimate to \texttt{(x,y,z)} data. See notes below for further details.

\texttt{too.far}

\quad (For \texttt{panel.surfaceSmooth} and \texttt{panel.kernelDensity}) The distance from original data at which to stop predicting surface values. See notes below for further details.

\texttt{n}

\quad (For \texttt{panel.kernelDensity}) the number of \texttt{x} and \texttt{y} cases to estimate when estimating density.

\texttt{local.wt}

\quad (For \texttt{panel.kernelDensity}) A logical (default \texttt{TRUE}) indicating if kernel density estimates should be weighed relative to other groups, panels, etc.

\texttt{kernel.fun}

\quad (For \texttt{panel.kernelDensity}) A function that can estimate kernel densities.

\texttt{panel.range}

\quad (For \texttt{panel.kernelDensity}) A logical (default \texttt{FALSE}) indicating if the kernel density estimation data range should be forced to the full panel range. See Below.

\texttt{x1, x2, y1, y2}

\quad (For \texttt{panel.binPlot}) Vectors giving the bin cell dimensions used when binning \texttt{x} and \texttt{y} elements. Typically ignored and calculated within the plot call.

\texttt{statistic}

\quad (For \texttt{panel.binPlot}) the function to use when calculating \texttt{z} values for each set of binned. By default, this is \texttt{mean}. So, if a \texttt{z} element is supplied in the plot call, the data is binned according to \texttt{x} and \texttt{y} values, and the mean of \texttt{z} values within each bin reported/plotted. If \texttt{z} is not supplied, \texttt{statistic} is reset to \texttt{length} to generate a frequency plot and a warning generated.

\texttt{pad.grid}

\quad For \texttt{panel.binPlot}, Logical, should empty bins be reported? If \texttt{TRUE}, they are reported as \texttt{NAs}; if \texttt{FALSE}, they are not reported.

Details

\texttt{panel.loaLevelPlot} is intended for plot data structured for use with the \texttt{lattice} function \texttt{levelplot}, e.g. regularised \texttt{(x,y,z)} or a matrix:

\texttt{loaPlot(...,panel = panel.loaLevelPlot)}

\texttt{levelplot(...)} #in lattice

Other specialist \texttt{panel...} functions can be used with the \texttt{lattice} function \texttt{xyplot}:
However, they are intended for use with loa plots that incorporate `panelPal`. This combination provides a mechanism for the routine preprocessing of panel data, the association of specialist keys, and the routine alignment of panel and legend settings in cases where values are reworked within the panel function call:

```
loaPlot(...,panel = panel.kernelDensity)
```

#etc

panel.surfaceSmooth and other similar `panel...` functions generate smoothed surfaces using supplied `(x,y,z)` data and pass this to `panel.loaLevelPlot` to plot.

By default, `panel.surfaceSmooth` uses stats function `loess` to generate a surface. Alternative smooths can be set using the `smooth.fun` argument, and the surface range can be controlled using the `too.far` argument.

`panel.kernelDensity` generates kernel density estimations based on the supplied `x` and `y` data ranges. Because it is density plot, it counts the number of `z` values. So, `z` values are ignored. It is intended to be used in the form:

```
loaPlot(~x*y,...,panel = panel.kernelDensity)
```

So, if any `z` information is supplied, users are warned that it has been ignored, e.g:

```
loaPlot(z~x*y,...,panel = panel.kernelDensity)
```

#warning generated

`panel.binPlot` bins supplied `z` data according to `x` and `y` values and associated break points (set by break arguments), and then calculates the required statistic for each of these. By default, this is `mean`, but alternative functions can be set using the `statistic` argument. It is intended to be used in form:

```
loaPlot(z~x*y,...,panel = panel.binPlot)
```

If no `z` values are supplied, as in:

```
loaPlot(~x*y,...,panel = panel.binPlot)
```

... `panel.binPlot` resets statistic to `length` (again with a warning) and gives a count of the number of elements in each bin.

**Value**

As with other `panel...` functions in this package, output are suitable for use as the `panel` argument in `loa` (and sometimes `lattice`) plot calls.

**Note**

All these `panel...` functions treat `col` and `col.regions`, etc, as discrete arguments. Typically, `col` links to lines (contour lines for surfaces, bin borders for binned data) and `col.regions` links any generates surface region.

`panel.surfaceSmooth` passes additional arguments on to the `smooth.fun` to estimate surface smooths and the `lattice` function `panel.levelplot` to generate the associated plot. If no `kernel.fun`
is supplied in the panel call, the stats function loess is used to estimate surface smooth. The
too.far argument is based on same in vis.gam function in the mgcv package.
panel.kernelDensity passes additional arguments on to the kernel.fun to estimate kernel
density and the lattice function panel.contourplot to generate the associated plot. If no
kernel.fun is supplied in the panel call, the MASS function kde2d is used to estimate kernel den-
sity.
panel.binPlot passes limited arguments on to lrect.
panel.kernelDensity and panel.binPlot are currently under review.

Author(s)
Karl Ropkins

References
These function makes extensive use of code developed by others.
(for panel.surfaceSmooth) mgcv package and too.far argument: Wood, S.N. (2004) Stable and
efficient multiple smoothing parameter estimation for generalized additive models.Journal of the

See Also
In loa: panelPal
In lattice: xyplot, levelplot, panel.contourplot, lrect

Examples

```r
## Example 1
## for data already set up for levelplot
loaPlot(volcano, panel=panel.loaLevelPlot)

## Example 2
## Surface smooth
loaPlot(copper~longitude*latitude, data= lat.lon.meuse,
panel=panel.surfaceSmooth, grid=TRUE,
too.far=0.1, col.regions=3:2)
```
Description
In development specialist panel functions for polar plotting

Usage

```r
panel.polarPlot(x = NULL, y = NULL, r = NULL, theta = NULL, ..., 
    data.panel = panel.loaPlot, loa.settings = FALSE, 
    plot = TRUE, process = TRUE)
```

#grid, axes and axes labelling

```r
panel.polarFrame(..., grid = TRUE, axes = TRUE, labels = TRUE, 
    panel.scales = NULL, grid.panel = panel.polarGrid, 
    axes.panel = panel.polarAxes, labels.panel = panel.polarLabels)
```

```r
panel.polarAxes(axes.theta = NULL, axes.r = NULL, thetalim = NULL, 
    rlim = NULL, ..., axes = NULL, panel.scales = NULL)
```

```r
panel.polarGrid(grid.theta = NULL, grid.r = NULL, 
    thetalim = NULL, rlim = NULL, ..., grid = NULL, 
    panel.scales = NULL)
```

```r
panel.polarLabels(labels.theta = NULL, labels.r = NULL, 
    thetalim = NULL, rlim = NULL, ..., labels = NULL, 
    panel.scales = NULL)
```

Arguments

- **x, y**
  The x and y coordinates of plot points.

- **r, theta**
  The equivalent polar coordinates of the plot points. If these are not supplied, 
  x and y are assumed to be polar coordinates and these are calculated by the 
  function.

- **...**
  Additional arguments, typically passed on. For panel.polarPlot these are 
  passed to the data.panel. See below.

- **data.panel**
  The panel to use to handle data once polar coordinates have been checked for or 
  generated. For panel.polarPlot, by default this is panel.loaPlot.

- **loa.settings, plot, process**
  loa panel management arguments, handled by panelPal. See associated help 
  documentation for further details.
grid, axes, labels
plot management options for the grid, axis and axis label elements of the plot. These can be logicals (TRUE to include the element or FALSE to remove it) or lists of plot parameters.

panel.scales
loa plot management argument used when generating grids, axes and labels within the plot panel.

grid.panel, axes.panel, labels.panel
Used by the panel...Frame functions to identify the panel... functions to use when generating the grid, axes and axis labelling elements of the plot.

axes.theta, axes.r, thetalim, rlim
For panel.polarAxes axes settings. axes.theta and axes.r are the theta and r coordinates of the axis reference points, tick marks, etc. thetalim and rlim are the plot/axes ranges (like xlim and ylim in standard lattice plots).

grid.theta, grid.r
Like axes.theta and axes.r but for grid.

labels.theta, labels.r
Like axes.theta and axes.r but for labels.

Details
The panel.polar... series of the functions are intended for use with loaPlot.

panel.polarPlot generates a 'bubble plot' style output on polar coordinates. It generates axes and annotation within each plot panel using the other panel functions.

panel.polarGrids, panel.polarAxes and panel.polarLabels generate plot grid, axes and axes labelling elements of the plot. panel.polarFrame provides a wrapper for these plot elements.

Users can fine-tune axes, grids and labels by supplying additional arguments in plot calls, or replace these elements with purpose written functions to more completely modify plot appearance.

Value
The panel.polar... functions are intended to be used as the panel argument in loa plot calls. So, e.g.:

```r
a <- 1:360
loaPlot(a~a*a, panel=panel.polarPlot)
```

They can also be used with relatively simple lattice plots. However, some features of loa plots managed by panelPal, e.g. default plot appearance management, automatic grouping and panel and key alignment will not be available.

Note
panel.polarPlot is in-development. Function arguments may change.

Author(s)
Karl Ropkins
References

These function makes extensive use of code developed by others.


See Also

In loa: loaplot; and panelPal.
In other packages: xyplot in lattice.

2.3.specialist.panels

Special panel functions

Description

In development specialist panel functions for generating zcase glyph structures.

Usage

panel.zcasePiePlot(..., loa.settings = FALSE)

panel.zcasePieSegmentPlot(..., zcase.rescale=TRUE, 
loa.settings = FALSE)

Arguments

... Additional arguments, typically setting the color and properties of the plotted glyphs. See below.

zcase.rescale Should the glyph element be rescaled? See below.

loa.settings loa options, to be handled by panelPal.

Details

All these panel... functions generate glyphs using z inputs and plot these at the associated (x, y) location. So, for example a called which used one of the panels and the plot formula:

z1 + z2+ z3 + z4 ~ x * y

... would plot a series of glyphs, each containing four elements that would be scaled according to z1, z2, z3 and z4, and each at the associated (x, y) location. This means there will be one discrete glyph for each row of data supplied to the plot call.

panel.zcasePiePlot generates a series of x/y referenced pie graphs. By default, pie dimensions are assigned as: Pie size (radius) proportional to sum of z cases and scaled using cexHandler (z1 + z2 + z3 + z4 for the above formula); Number of Pie segments equal to number of z cases (so, 4 for
the above formula); Pie segment width (phi proportional to the individual zcase (so, z1 / (z1 + z2 + z3 + z4) * 360 for first pie segment for the above formula).

panel.zcasePieSegmentPlot is a variation on the conventional pie plot where segment radius rather than segment width is varying by zcase.

Value

These panel... functions are intended to be used as the panel argument in loa plot calls. So, e.g.:

```r
a <- 1:10
b <- 10:1
loaPlot(a+b-a*a, panel=panel.zcasePiePlot)
loaPlot(a+b-a*a, panel=panel.zcasePieSegmentPlot)
```

Note

Functions in development. Arguments may change, e.g.:

panel.zcasePieSegmentPlot includes the argument `zcase.rescale`. This normalises data within each zcase before generating the pie segments. This might not stay.

Author(s)

Karl Ropkins

References

These function makes extensive use of code developed by others.


See Also

In loa: `loaPlot`, `panelPal`

In other packages: `xyplot` in `lattice`.

Examples

```r
## Example 1
## plotting georeferenced pie plots

# Using a subsample of lat.lon.meuse
temp <- lat.lon.meuse[sample(1:155, 15),]

## Not run:
# plot Cu/Pb/Zn pie plots at sampling locations
loaPlot(copper+lead+zinc~longitude*latitude,
        panel=panel.zcasePiePlot, data=temp)
# then rescale smaller pie segments on the fly
```
## End(Not run)

```r
loaPlot(copper*10+lead*4+zinc~longitude*latitude,
       panel=panel.zcasePiePlot, data=temp)
```

---

### 2.4.specialist.panels  Special panel functions 04

**Description**

In development specialist panel functions

**Usage**

```r
panel.compareZcases(x=x, y=y, z=NULL, ..., 
                    loa.settings = FALSE)
```

**Arguments**

- `x, y, z`  Standard plot data series, typically vectors.
- `...`      Additional arguments, typically passed on.
- `loa.settings`  loa options, to be handled by `panelPal`.

**Details**

The `panel.compareZcases` generates a simple plot which compares `z` and `y` elements.

**Value**

These `panel...` functions are intended to be used as the `panel` argument in `loa` plot calls. So, e.g.:

```r
x <- 1:10
y <- 1:10
z <- y + rnorm(10)
loaPlot(z~x+y, panel=panel.compareZcases, col.regions="Reds")
```

**Note**

These are ad hoc `panel...` functions. Not sure of their life expectancy...

**Author(s)**

Karl Ropkins
References

These function makes extensive use of code developed by others.

See Also

In loa: loaPlot, panelPal.
In other packages: xyplot in lattice.

3.1.example.data  Example data for use with loa

Description

Example data intended for use with examples in loa.

Usage

lat.lon.meuse

roadmap.meuse

Format

lat.lon.meuse is a modified form of the meuse data set taken from the sp package. Here, coordinate (x,y) elements have been transformed to latitudes and longitudes and the object class has been converted from SpatialPointsDataFrame to data.frame.

roadmap.meuse is a previously downloaded map intended for use with off-line plot examples using lat.lon.meuse.

Details

lat.lon.meuse was generated using method based on mzn object production in https://github.com/etes/Geoprocessing/blob/master/heatmap.R.

library(sp); library(gstat); library(rgdal)
data(meuse)coordinates(meuse) <- x + yproj4string(meuse) = CRS("+init=epsg:28992")meuse1 = spTransform(meuse, CRS("+init=epsg:4326"))meuse2=as.data.frame(meuse1)mzn=meuse2[,c(14,13,4)]names(mzn)<-c("Latitude","Longitude","zinc")
roadmap.meuse was generated using:
RgoogleMapsPlot(zinc~latitude*longitude, data=lat.lon.meuse, size=c(450,500), maptype="roadmap")
roadmap.meuse <- loaMapArg()

References

For meuse:
M G J Rikken and R P G Van Rijn, 1993. Soil pollution with heavy metals - an inquiry into spatial variation, cost of mapping and the risk evaluation of copper, cadmium, lead and zinc in the floodplains of the Meuse west of Stein, the Netherlands. Doctoraalveldwerkverslag, Dept. of Physical Geography, Utrecht University
Stichting voor Bodemkartering (Stiboka), 1970. Bodemkaart van Nederland : Blad 59 Peer, Blad 60 West en 60 Oost Sittard: schaal 1 : 50 000. Wageningen, Stiboka.

For sp:

Examples

```r
## data structure of lat.lon.meuse
head(lat.lon.meuse)
## Use a subsample of lat.lon.meuse
temp <- lat.lon.meuse[sample(1:155, 15),]

## various loaPlot examples
## using lat.lon.meuse
loaPlot(~longitude*latitude, data=temp)
loaPlot(cadmium~longitude*latitude, data=temp)
loaPlot(cadmium~longitude*latitude, col.regions=c("green", "red"), data=temp)
loaPlot(cadmium*50+copper*10+lead*2+zinc~longitude*latitude, panel.zcases = TRUE, col.regions=c("green", "red"), key.z.main="Concentrations", data=temp)
```
## (off line) GoogleMap example
## using lat.lon.meuse and roadmap.meuse

GoogleMap(zinc~latitude*longitude, data=temp, 
    map=roadmap.meuse, col.regions=c("grey", "darkred"))

# Note 1:
# With loaPlot and GoogleMap, note latitude, longitude axes
# assignments:
# loaPlot plots z ~ x * y | cond.
# GoogleMap plots z ~ lat * lon | cond (z ~ y * x | cond)

# Note 2:
# Here, the map argument is supplied so example works off-line.
# If not supplied and R is on-line, GoogleMap will (try to) get map
# from the Google API. Look at:
## Not run:
GoogleMap(zinc~latitude*longitude, data=lat.lon.meuse, 
    col.regions=c("grey", "darkred"))
## End(Not run)
# (The map will appear slightly different, because non-default
# size and maptype settings were used to make roadmap.meuse. See above.)

### panelPal

**Description**

`lattice` plot management using the `loa` function `panelPal`

**Usage**

```r
panelPal(ans, panel = NULL, preprocess = FALSE, 
    reset.xylims = FALSE, legend = NULL, 
    by.group = NULL, by.zcase = NULL, ...)

panelPal.old(x, y, subscripts, at, col.regions, ..., 
    panel = panel.xyplot, ignore = NULL, 
    group.fun = NULL)

loaHandler(panel = NULL, ...)
```
Arguments

ans For panelPal only, a standard trellis object, such as that generated by `lattice` function `xyplot`.

panel A panel function, e.g. `panel.xyplot`. If supplied in panelPal call, typically the one used to generate ans. If supplied in panelPal.old, the panel that is intended to used when generating a plot.

preprocess, reset.xylims, legend, by.group, by.zcase

For panelPal only, loa plot management arguments. `preprocess`: Logical, should the supplied panel function be preprocessed? `reset.xylims`: Logical, should the plot dimensions be reset if changed, e.g. by preprocessing? `legend`: the legend as with standard lattice plots, `by.group`: a vector of plot argument names to be linked to any group conditioning, `by.zcase`: a vector of plot argument names to be linked to any z case conditioning See Details below.

... Additional arguments, typically passed on.

x,y,subscripts,at,col.regions

For panelPal.old only, panel arguments passed down to individual panels.

ignore Any additional arguments that panelPal.old should ignore and pass on to panel unchecked/unmodified.

group.fun Fine control of the standard lattice plot argument `group`. It can be a vector or list containing the same number of elements as there as groups. These can be functions (or the names of functions as characters) setting individual functions for group or sets of parameters to be evaluated using the panel function. For example, the current NULL default generates a list of col and pch settings that produce a conventional grouped scatter plot output when the group argument is applied to the panel default panel.xyplot. See Details below.

Details

panelPal provides a range of generic plot management features.

Firstly, it allows plot as well as panel defaults to be managed by the panel... function. This allows the panel developer to control plot-level components of the plot output, e.g. which key to use with the plot and what default settings to apply to it. See example 1 below.

Secondly, it uses a generalised extension of the subscripting methods described by Deepayan Sarkar in Chapter 5 of Lattice (see sections on scatterplots and extensions) to automatically handle plot argument subscripting, demonstrated in example 2 below.

Thirdly, it applies an extension of the method used by the hexbin lattice panel to pass hex cell counts (calculated in panels) to the plot key and standardise the assignment of associated parameters within all panels to provide more general panel-to-panel and panel-to-scale. The method is briefly discussed in Chapter 14 of Sarkar.

This method has also been extended by isolating processing and plot components of the panel... function operation allowing results of any calculations made in-panel to be retained rather than lost when plot is generated.

Fourly, group... and zcase... arguments can used to manage plot group and zcase based plot outputs.
Some panelPal are implemented if specially structured (or loa-friendly) panel... functions are supplied. These are illustrated in the final example below.

loaHandler is a workhorse that assesses information in 'loa' friendly panel... functions. As well as checking this, loaHandler also checks the supplied panel for any default plot settings. This allows users to manage the appearance of a plot from the panel or automatically associated color keys.

Value

Both panelPal and panelPal.old are intended to be used with trellis plot outputs.

panelPal should be employed retrospectively. So, for example:

```r
p1 <- xyplot(...)  
panelPanel(p1, ...)
```

The previous version, currently retained as panelPal.old, was employed developed as a panel... function wrapper and intended to be employed within the plot call. So, for example:

```r
xyplot(..., panel = function(...) panelPal(..., panel=panel))
```

Because it is run within the plot call, and therefore within each panel called, it does not provide features that require panel-to-plot, panel-to-key or panel-to-panel communication.

loaHandler returns either a logical (FALSE if not loa 'friendly'; TRUE if loa 'friendly') or a list of default arguments to be used when plotting.

Note

The by.group and by.zcase arguments of panelPal and the group.fun argument of panelPal.old are all currently under review. Please do not use these.

Author(s)

Karl Ropkins

References

These function makes extensive use of code developed by others.

lattice:

hexbin:

panelPal.old and panelPal both apply an extension of the subscripting methods described by Deepayan Sarkar in Chapter 5 of Lattice (see sections on scatterplots and extensions) to automatically handle plot argument subscripting.

panelPal applies an extension of the method used by hex bin lattice panel to communicate hex cell counts (calculated in panels) panel-to-panel and panel-to-scale. The method is briefly discussed in Chapter 14 of Sarkar.
See Also

lattice, xyplot.

Examples

## the combination of panelPal and specially
## structured panel... functions provides
## several additional plot features:

## example 1
## plot management from the panel... functions.

# loaHandler can used to see if a panel is loa-friendly
loaHandler(panel.xyplot) #FALSE
loaHandler(panel.loaPlot) #panel defaults

# note that these include a list called
# default.settings. These are settings that are
# automatically added to the plot call.

# Here this assigns a specialist key to that
# panel. However, the same mechanism can also
# be used to turn off plot elements like the
# standard lattice axes, when using in panel
# alternatives

# first some silly data
a <- rnorm(1000)
b <- rnorm(1000)

# now compare:
# default plot
# note bubble plot style key
loaPlot(a=a*b)

# bin plot
# with classic color key
loaPlot(a=a*b, panel = panel.binPlot)

## example 2
## automatic subscripting with loa

# Other arguments are not automatically
# aligned with the main plots.
# For example, consider the data:

```r
a <- 1:10
ref <- rep(1:2, each=5)
```

# and associated lattice xyplot output:

```r
xyplot(a~a|ref, col=ref, pch=19)
```

# Here, the 'col' argument does not
# automatically track plot conditioning.

# With lattice plots you need to assign
# arguments you want to track in this
# manner using subscripts, as discussed
# in Lattice Chapter 5.

# Now compare a similar loaPlot:

```r
loaPlot(~a*a|ref, col=ref, pch=19)
```

# Here, panelPal automatically handles
# such subscripting. It extends this
# assumption to all supplied arguments.

# For example, try
```r
## Not run:
loaPlot(~a*a|ref, col=ref, pch=ref)
loaPlot(~a*a|ref, col=ref, pch=1:10)
## End(Not run)
```

---

### 4.2.plot.structure.handlers

**Handler functions for plot structure arguments.**

**Description**

Function(s) for handling (front end) plot arguments like x and strip that manage the plot structure.

**Usage**

```r
formulaHandler(x, data = NULL, groups = NULL, ..., expand.plot.args = TRUE, formula.type = "z~x*y|cond", panel.zcases = FALSE, coord.conversion = NULL, lattice.like = NULL, check.xy.dimensions = TRUE, check.coord.dimensions = TRUE, get.zcase.dimensions = TRUE, output = "extra.args")
```
4.2.plot.structure.handlers

matrixHandler(x, data = NULL, row.values=NULL, column.values=NULL, ...)  

stripHandler(..., striplab=NULL)  

getZcaseDimensions(...)  

Arguments  

x (For formulaHandler) A formula or matrix (matrixHandler) intended to be used to generate a lattice plot. See Below.  
data If supplied, the assumed source of the elements of formula x, typically a data.frame.  
groups If supplied, the grouping argument to be used with x and data.  
... Additional arguments are passed on to related functions.  
expand.plot.args For formulaHandler only, logical. Should any short elements of the plot structure be expanded?  
formula.type For formulaHandler only, character vector or function. The plot structure to be used when generating the plot, e.g. z ~ x * y | cond for loaPlot  
panel.zcases For formulaHandler only, logical. Should zcase arguments, e.g. z1 and z2 in z1 + z2 ~ x * y | cond, be treated as panel conditioning terms rather than grouping terms?  
coord.conversion For formulaHandler only, function. If supplied, the conversion to use to convert coordinate information supplied using other coordinate systems to (x,y).  
lattice.like For formulaHandler only, list. For preprocessing, a list of plot terms that can be passed directly to lattice/loa plots.  
check.xy.dimensions, check.coord.dimensions For formulaHandler only, logicals. Should the formula structure be tested before attempting to generate the plot? See Note below.  
get.zcase.dimensions For formulaHandler only, logical. Should the dimensions of any multiple zcases be calculated? See Note below.  
output For formulaHandler only, character vector. The format to return function output in.  
row.values, column.values For matrixHandler only, row and column values to be assigned to supplied matrix x.  
striplab For stripHandler only, character vector. If supplied, the label to add to the panel strip when conditioning is applied. By default, it applies the standard lattice convention, i.e., show for numerics.
Details

formulaHandler manages the formula component or x element of of loa plot functions.

For example, for loaPlot it assumes the general formula structure \( z \sim x \times y \mid \text{cond} \), and applies it in a similar fashion to the \texttt{lattice} function \texttt{levelplot}.

Within the formula part of the plot call x and y are the horizontal and vertical axes, z is any additional information to be used in point, symbol, surface or glyph generation, and cond any additional conditioning to be applied. (The coordinates, x and y, are required elements; z and cond are typically optional.)

matrixHandler converts a matrix supplied as x element of a loa plot to a formula and associated data. If row.values and column.values are supplied, these are used as \((x,y)\) values for the supplied matrix.

stripHandler manages the strip component of loa plot functions.

getZcaseDimensions tracks the dimensions of multiple z

Value

formulaHandler returns a list, containing the plot elements defined in the supplied formula.

matrixHandler returns a list containing all supplied arguments, subject to the following modifications: matrix x converted to formula \((z \sim x \times y)\); data, replaced with supplied matrix content; xlim and ylim, added is not supplied.

stripHandler returns a list containing all supplied arguments, subject to the following modifications: strip, Updated or generated if not supplied; striplab, added to strip via the strip argument var.name, if this is undeclared in call.

getZcaseDimensions returns a list containing all the supplied arguments, plus two additions arguments (if supplied in the call): zcase.zlim and z.rowsum.lim. zcase.zlim is a list of lim values, one for each zcase. z.rowsum.lim is the range of 'by-row' sums of zcases. These are calculated using any zcase information supplied in the call.

Note

These function are in development and may be subject to changes.

The current version of formulaHandler includes code from the stats function \texttt{get_all_vars}. It operates in a similar fashion to the previous version but checks zcase dimensions.

The previous version of formulaHandler was a wrapper for the \texttt{lattice} function \texttt{latticeParseFormula}. This version of formulaHandler was updated to simplify multiple z argument handling.

The latest version of formulaHandler includes more flexible formula.type handling. For example, it can now handle formulas that have more than two coordinates. As a result the check.xy.dimensions argument was replaced with a check.coord.dimensions argument. The previous argument will however remain in the function formals and function as before until all related code has been updated.

The latest version of formulaHandler uses getZcaseDimensions to calculate the dimensions of z if it is multi-part, e.g. \( z_1 + z_2 + \text{etc} \sim x \times y \) rather than \( z \sim x \times y \).

The current version of matrixHandler is based on code from levelplot.matrix in \texttt{lattice}. If used with x and data arguments it will overwrite data with the matrix content.
4.3.lims.and.scales.handlers

Author(s)
Karl Ropkins

References
This function makes extensive use of code developed by others.

See Also
In loa: loaPlot; panelPal
In other packages: levelplot in lattice.

Description
In development functions for lims and scales handling with lattice plots.

Usage

limsHandler(x=NULL, y=NULL, z=NULL, ..., lim.borders = 0.05)
localScalesHandler(scales = NULL, ..., allowed.scales = c("x", "y"),
disallowed.scales = NULL, remove.box = FALSE)
panel.localScale(x.loc, y.loc, lim, ..., panel.scale = NULL,
label.before = TRUE, x.offset = NULL, y.offset = NULL,
axis = TRUE, ticks = TRUE, annotation = TRUE)
yscale.component.log10(...)
xscale.component.log10(...)

Arguments
x, y, z x, y and/or z data series.
4.3.lims.and.scales.handlers

lim.borders numeric vector, giving the relative border to extend \( \ldots \text{lim} \) ranges by when generating axes or scales. The \texttt{lim.borders} are relative proportions. So, the default setting of 0.05 adds an extra +/- 5 are supplied the first three are treated as codex, \( y \) and \( z \) \texttt{lim.borders}, respectively. If less than three values are supplied, the three values are generated by wrapping. So, the default setting of 0.05 is equivalent to \( c(0.05, 0.05, 0.05) \).

\texttt{scales}, \texttt{panel.scale}

A list of elements like the \texttt{scales} argument used with \texttt{lattice} functions. Current default elements \texttt{draw} (= \texttt{TRUE}), \texttt{arrows} (= \texttt{FALSE}), \texttt{tick.number} (= 5), \texttt{abbreviate} (= \texttt{FALSE}), \texttt{minlength} (= 4), and \texttt{tck} (= 1).

\texttt{...}

Additional arguments.

\texttt{allowed.scales}

A character vector containing the names of the axes to be generated for as part of a local axis.

\texttt{disallowed.scales}

A character vector containing the names of any axes that are not required. Note: If found, these are removed from \texttt{scales} before evaluation.

\texttt{remove.box}

A logical, default \texttt{FALSE}. Should the box \texttt{lattice} typically places around standard plots be removed? This can be useful if you are using a \texttt{panel}...\texttt{function} to generate axes within the plot.

\texttt{x.loc, y.loc, lim}

two member vectors setting the \( x \) and \( y \) locations where the scale is to be drawn (\texttt{x.loc} ans \texttt{y.loc}), and the limits of the range to be annotated on the scale (\texttt{lim}).

Note: These are currently handled 'as is', i.e. for both locations and limit, the first element is the start point and the second is the end point, and any other elements are ignored.

\texttt{label.before, x.offset, y.offset}

Scale annotation overrides. \texttt{label.before} is a logical, which controls the position of annotation, tick marks and/or arrows, etc relative to the scale line (i.e., above/left before or below/right after). By default \texttt{panel.localScale} generates tick marks and labels at right angles to the scale line/axis. \texttt{x.offset} and \texttt{y.offset} force the offsets when adding tick marks and annotation.

\texttt{axis, ticks, annotation}

If supplied, fine controls for the appearance of the axis line, axis tick marks and axis annotation on the generated scale. These can be vectors, in which they are assumed to be color assignments, or lists of common plot parameters, such as \texttt{col}, \texttt{lty}, \texttt{lwd}, etc.

Details

\texttt{limsHandler} generates \texttt{xlim}, \texttt{ylim} and/or \texttt{zlim} ranges for axes or color scales for use in a range of plots.

\texttt{localScalesHandler} provides a relatively crude mechanism for the removal of conventional \texttt{lattice} plot axes and the generation of alternative axes using a \texttt{panel}...\texttt{function} like \texttt{panel.localScale}.

Value

\texttt{limsHandler} returns a list containing \( \ldots \text{lim} \) ranges for any of the elements codex, \( y \) and/or \( z \) supplied to it.
localScalesHandler returns a list containing the elements: scales, panel.scales and possibly par.settings. scales turns off the standard axes annotation. panel.scales is a list of named elements, one per named axis, describing the axis layout. If remove.box = TRUE, the additional argument par.settings is also supplied.

All arguments should be passed on to the associated plot.

panel.scales or axis-specific elements in panel.scales can then be evaluated by an associated panel... function run from within the lattice plot call. This would typically take the form:

panel.my.axis(panel.scale = panel.scale$axis,...)

panel.localScale is a local axis/scale plotter. It can be used in combination with localScalesHandler, and should called once for each axis that is required, or it can be used 'stand alone' panel to add a local scale to a lattice plot.

yscale.component.log10 and xscale.component.log10 are simple axis transforms for use with log to the base 10 transformed plot axes.

Note

panel.localScale is currently in revision. Scale arrows are currently not available.

Author(s)

Karl Ropkins

References

These function makes extensive use of code developed by others.


See Also

In other packages: xyplot in lattice.

Examples

## See trianglePlot Example 2 for example application
### Usage

- `condsPanelHandler(..., conds = NULL, panel = NULL, by.cond = NULL, process = TRUE, plot = TRUE)`
- `groupsPanelHandler(..., groups = NULL, panel = NULL, by.group = NULL, process = TRUE, plot = TRUE)`
- `zcasesPanelHandler(..., zcases = NULL, panel = NULL, by.zcase = NULL, process = TRUE, plot = TRUE)`
- `groupsAndZcasesPanelHandler(panel=NULL, ..., plot = TRUE, process = TRUE)`
- `groupsHandler(z = NULL, groups = NULL, ..., group.ids = NULL, handler = "zzz")`
- `zcasesHandler(z = NULL, zcases = NULL, ..., zcases.ids = NULL, handler = "zzz")`
- `groupsAndZcasesHandler(..., loa.settings = NULL)`
- `stepwiseZcasesGlyphHandler(zcases = NULL, ..., zcase.ids = NULL, panel.elements = NULL, loaGlyph = NULL)`

### Arguments

- `...`, Additional arguments. See Notes below.
- `conds`, `panel`, `by.cond`
  For all supplied additional arguments, `conds` is a vector of conditioning indices. This is typically a logical, numeric, factor or character vector which can be used to assign other elements undeclared call arguments to specific subsets. `panel` identifies the `panel...` function, and should also be supplied so `loa` can manage processing and plot activities correctly. `by.cond` identifies routine plot operations associated with the requested conditioning. This can be a list of plot arguments or `panel...` functions that should be associated with the requested conditioning. See `process` and `plot` below and associated Notes.
- `plot`, `process`, `loa.settings`
  Passed to and handled by `panelPal`. For panels that can be preprocessed, `plot` and `process` turn off or on processing and the plotting steps of the panel code. See `panelPal` Help documentation from further details.
- `groups`, `by.group`
  As `conds` and `by.cond` but for grouping.
- `zcases`, `by.zcase`
  As `conds` and `by.cond` but for `zcase` condition.
- `z`, `handler`
  The `z` data series and any associated plot arguments that need special handling.
group.ids, zcases.ids, zcase.ids
If given, vectors of the unique cases in groups and zcases, respectively.
panel.elements
If given, the names of all plot arguments that have been vectorized by panelPal.
loaGlyph
(For stepwiseZcasesGlyphHandler only), the loa glyph to drawn. See loa_glyphs
for further details.

Details
NOTE: These functions are currently in development and may be subject to changes.

condsPanelHandler is a general purpose function that can be used to routinely manage plot
conditioning within a panel... function call. It takes all undeclared arguments are supplied to it, and
subsets them by unique case in the supplied conds argument. Then it modifies each of these based
on the associated elements of by.cond and processes and/or plots the results depending on process
and plot settings.

groupsPanelHandler is similar but is intended for use with the plot call argument groups.
zcasesPanelHandler is similar but is intended for use with arguments conditioned within the z
term of the plot formula. So, for example, for unique handling of z1 and z2 cases in the plot
loaPlot(z1+z2~x*y).
groupsAndZcasesPanelHandler is a wrapper for groups and zcase that allows users to simulta-
neously and uniquely handle both types of conditioning.

stepwiseZcasesGlyphHandler is a ...Handler function for generating glyph plots based on mul-
tiple z inputs.

Value
All ..PanelHandlers functions are intended for use with panelPal. Using different combinations
of these allows plot developers a high degree of flexibility.

Note
This function is in development and may be subject to changes.

Author(s)
Karl Ropkins

References
This function makes extensive use of code developed by others.

See Also
panelPal
For information on related functions in other packages, see
lattice: xplot; panel.xplot; and panel.levelplot.
4.5.plot.argument.handlers

Common plot argument handlers

Description

Functions for use the routine handling of some common plot arguments.

Usage

```r

cexHandler(z = NULL, cex = NULL,
            cex.range = NULL, expand.outputs = TRUE,
            ref = NULL, ..., zlim = NULL)
colHandler(z = NULL, col = NULL,
            region = NULL, colorkey = FALSE, legend = NULL,
            pretty = FALSE, at = NULL, cuts = 20,
            col.regions = NULL, alpha.regions = NULL,
            expand.outputs = TRUE, ref = NULL,
            ..., zlim = NULL, output="col")
colRegionsHandler(...)
pchHandler(z = NULL, pch = NULL, pch.order = NULL,
            expand.outputs = TRUE, ref = NULL, ...
            zlim = NULL)
scalesHandler(...)
zHandler(z = NULL, expand.outputs = TRUE,
            ref = NULL, ...)
```

Arguments

- **z**
  - If supplied, a vector of values intended to used as a scale when assigning a property.
  - For `cexHandler`, the cex of e.g., points on a scatter plot. Here, size scales are managed using a reference range `cex.range`, but superseded by `cex` settings, if also supplied.
  - For `colHandler`, the color of e.g., points on a scatter plot. Here, color scales are managed using a colorkey method similar to that used by the `lattice` function `levelplot`, see below (arguments `region`, `colorkey`, `pretty`, `at`, `cuts`, `col.regions` and `alpha.regions`). If `z` is NULL or not supplied, all colors are set by `col` if supplied or as the default lattice symbol color if both `z` and `col` are not supplied.
For pchHandler, the pch of, e.g., points on a scatter plot. Here, plot symbols are managed using a reference vector pch.order, but superseded by pch settings, if also supplied.

For zHandler, any vector that should to expanded by wrapping to a given length, e.g. the length of the x (or y) data series to plotting.

cex, col, pch
For associated handlers, the parameter value(s) to be managed (i.e., cex for cexHandler, etc. Note: In all cases if these are not NULL these supersede any supplied z or ...Handler modification.

cex.range
If supplied, the range for z to be rescaled to when using this to generate a cex scale. NOTE: cex.range = FALSE disables this cex scaling and uses z values directly; cex.range = TRUE applied default scaling, equivalent to cex.range = c(0.75,3).

region, colorkey, legend, pretty, at, cuts, col.regions, alpha.regions
The colorscale settings to be used when generating a colorkey. The most useful of these are probably col.regions which can be used to reset the color scale, alpha.regions which sets the col.region alpha transparency (0 for invisible to 1 for solid) and colorkey which can be a logical (forcing the colorkey on or off) or a list of components that can be used to fine-tune the appearance of the colorkey. Note: The generation of colorscales is handled by RColorBrewer.

cexHandler manages symbol sizes. It generates a (hopefully) sensible cex scale for handling plot symbol size based on a supplied input (z).

colHandler manages colors. It works like the colorkey in levelplot in lattice, to generate a colorscale based on a supplied input (z).

colRegionsHandler is a wrapper for colHandler that can be used to with the col.regions argument.
scalesHandler is a crude method to avoid scales argument list structures.  
zHandler expands (by wrapping) or foreshortens vectors.

Value

cexHandler returns a vector, which can be used as the cex argument in many common plotting functions (e.g. plot, xyplot).

colHandler depending on output setting returns either the col vector or a list containing elements (z, col, legend, at, col.regions and alpha.regions), which can be used to create a col series scaled by z and an associated colorkey like that generated by levelplot for other lattice functions (e.g. xyplot).

colRegionsHandler returns a vector of color values suitable for use with the col.regions argument.

scalesHandler returns the supplied arguments modified as follows: all scales... arguments are converted into a single list(...); all scales.x... and scales.y... argument are converted into list(x=list(...)) and list(y=list(...)), respectively. so e.g. scales.x.rot=45 generates scales=list(x=list(rot=45)).

pchHandler returns a vector of pch values of an appropriate length, depending on expand.outputs and ref settings.

Note

cexHandler recently revised. Default cex range now smaller, in line with feedback.
scalesHandler might not be staying.

Author(s)

Karl Ropkins

References

These function makes extensive use of code developed by others.


See Also

In other packages: See xyplot in lattice.

Examples

#some trivial data
a <- 1:10
## Example 1

## Simple plot with cex handling

```r
myplot1 <- function(x, y, z = NULL, cex = NULL, 
cex.range = NULL, ...){
  #set cex
  cex <- cexHandler(z, cex, cex.range)
  #plot
  xyplot(y~x, cex = cex,...)
}
```  
myplot1(a, a, a)

# compare

```r
# Not run:
myplot1(a, a)  #like plot(x, y)
myplot1(a, a, a*100)  #as myplot1(a, a, a)
myplot1(a, b, c, 
  cex.range = c(1,5))  #cex range reset
myplot1(a, b, c, 
  cex.range = c(10,50),
  cex = 1)  #cex supersedes all else if supplied
# End(Not run)
```

## Example 2

## plot function using lists/listUpdates

```r
myplot2 <- function(x, y, z = NULL, ...){
  #my default plot
  default.args <- list(x = y~x, z = z, 
    pch = 20, cex = 4)
  #update with whatever user supplied
  plot.args <- listUpdate(default.args, list(...))
  #col Management
  plot.args$col <- do.call(colHandler, plot.args)
  do.call(xyplot, plot.args)
}
```  
# with colorkey based on z case
myplot2(a, a, a)

# compare

```r
# Not run:
myplot2(a, b, c, 
  col.regions = "Blues")  #col.regions recoloring
```
myplot2(a, b, c,
   col = "red")  ##but (again) col supersedes if supplied
## End(Not run)

# Note:
# See also example in ?listUpdate

### 4.6.key.handlers  Key handling

**Description**

Workhorse functions for routine use of keys in plots.

**Usage**

keyHandler(key = NULL, ..., output = "key")

#keys
draw.loaPlotZKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.loaColorKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.loaColorRegionsKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.zcasePlotKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.ycasePlotKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.groupPlotKey(key = NULL, draw = FALSE, vp = NULL, ...)
draw.key.log10(key = NULL, draw = FALSE, vp = NULL, ...)

**Arguments**

**key**  
The key to be used.

**...**  
Any additional arguments to be used to modify the the key before plotting.

**output**  
The format to return the function output in. This is 'key' for all routine (in plot) use.

**draw, vp**  
lattice and grid arguments using when plotting GROB objects. Generally, these can be ignored.
4.7. other.panel.functions

Details

keyHandler is a general function that routine generates defaults arguments for add a key to a plot. draw...key functions are all specialist plots keys. They are typically modifications of or variations on similar functions in lattice, e.g. draw.key and draw.colorkey.
draw.loaPlotZKey is the default 'bubble plot' key used with loaPlot.
draw.loaColorKey and draw.loaColorRegionsKey are variations on the draw.colorkey function in lattice.
draw.zcasePlotKey, draw.ycasePlotKey and draw.groupPlotKey are simple legends based on zcase, ycase and group annotation.
draw.key.log10 is a simple legend for use with log to the base 10 transformed z scale.

Value

keyHandler return a list of plot arguments to be used to generate a key.

When run within plot calls, the draw...key functions associated color keys. If they are used with loa plots and suitable panel... functions, color scales are automatically aligned.

Note

In Development: Function structures may change in future package updates.

Author(s)

Karl Ropkins

References

These functions make extensive use of code developed by others.

See Also

In other packages: See xyplot in lattice.

4.7. other.panel.functions

Other panel functions argument handlers

Description

In development panel functions for use with lattice
4.7. other.panel.functions

Usage

parHandler(scheme = NULL, ...)

#related

getArgs(source = TRUE, local.resets = TRUE,
user.resets = TRUE, is.scales.lines = FALSE,
elements = NULL, ..., defaults = list(),
defaults.only = FALSE)

getPlotArgs(defaults.as = "axis.line", source = TRUE,
local.resets = TRUE, user.resets = TRUE,
elements = NULL, ..., is.scales.lines = NULL,
defaults.only = TRUE)

isGood4LOA(arg)

Arguments

scheme The color scheme to apply. This can be a list of parameters to apply or a character vector for a pre-defined scheme. Current pre-defined schemes include 'greyscale' (for black and white figures).

source, local.resets, user.resets

When recovering plot arguments with getArgs or getPlotArgs, places to search for relevant parameters. If supplied these would typically be vectors or lists. If vectors, they are assumed to be col setting. If lists, they are assumed to be lists of named parameters for inclusion. There are two cases that need to be handled specially: (1) some sources, local.resets and/or user.resets may contain both axis-specific and general information, e.g. For a scales list, parameters to be applied just to the x axis in scales$x and parameters to be applied to all scales in scales. In such cases these need to be checked in order (see elements below.) (2) Some sources, e.g. axis scales, contain both text and line parameters, with e.g. line settings declared as col.line, etc., rather than col, etc., (which are intended for use with text.) When supplied these need to be handled correctly (see is.scales.lines below). local.resets and user.resets are intended as overrides for the code developer and user, respectively. These can be logicals as well as vectors or lists. If logicals they turn on/off the associated plot components (using isGood4LOA). The check/update order is source, then source$element, then local.reset, then local.reset$element, then user.reset, then user.reset$element. This means that the developer always has last say regarding the default appearance of a plot component and the user.
always has the very last say from the command line if the local.reset is included as a formal argument in that plot.

is.scales.lines
When recovering arguments with getArgs or getPlotArgs, should source be treated as a lattice scales list? If so, and source is checked for line parameters, line-specific terms such as col.line, etc., will be recovered as col, etc., while general terms (meant for text in scales lists) will be ignored. (Note: getPlotArgs guesses this based on defaults.as if not supplied.)

elements
When recovering arguments with getArgs or getPlotArgs, this identifies the elements in source, local.resets and user.resets that may contain case-specific information. As with lattice handling of scales axis-specific information in source$element(s) is assumed to take priority over general information in source. (Note: if elements are not declared only general/top level information in source, local.resets and user.resets is considered at present.)

... Other arguments, often ignored.

defaults, defaults.only, defaults.as
When recovering arguments with getArgs, defaults is an optional ‘fall-back’ in case nothing is recovered from source, local.resets and user.resets. defaults.only is a logical: if TRUE only parameters named in defaults are searched for, otherwise all parameters are recovered. With getPlotArgs, defaults.as selects an appropriate default. This should be a trellis parameter name, e.g. 'axis.line', 'axis.text', etc. The function uses this to identify appropriate plot parameters to search for/select, e.g. pch, col, cex, etc for 'plot.symbol', and to identify default values for each of these (if defaults.only = TRUE).

arg
For isGood4LOA a plot argument that can used to turn a plot panel or panel component on or off.

Details
getArgs returns a list of parameters/values based on lattice, developer and user settings. If multiple elements are identified as containing case-specific information, the list will contain one list of plot parameters for each named element.

getPlotArgs is a variation of getArgs intended for use with panel... and l... type lattice functions. It returns a list of plot parameters for different plot components, e.g. symbols, lines, or text.

isGood4LOA is a simple workhorse that checks if a supplied arg should be used by loa. (See value and note below.)

parHandler manages the default appearance of plots.

Value
getArgs and getPlotArgs return lists of located parameters/values. For example, the call
getPlotArgs(default.as = "axis.line")
returns a list containing the lattice defaults for an axis line (alpha, col, lty and lwd) These can then be used in combination with appropriate x and y values in llines, or panel.lines calls. The
arguments `local.resets` and `user.resets` can be added into the call to provide developer and user overrides. (See note below.)

`isGood4LOA` returns a logical (TRUE or FALSE), depending on the type of a supplied argument. This returns FALSE for NULL, for all FALSE logicals, and any arg that has previously been tagged as 'not wanted'.

`parHandler` returns a list a list suitable for use as `par.settings` with most `lattice` plots.

**Note**

`getPlotArgs` is intended as a 'workhorse' for plot developers, to recover `lattice` settings, impose their own preferences on these, and in turn to provide users with similar options to quickly override developer settings.

`isGood4LOA` only exists because I, perhaps wrongly, equate `arg = NULL` with `arg = FALSE` when that argument is a component of a plot defined in the plot formals. For example, in `trianglePlot` I want `grids = NULL` to turn off the plot grids much like `grids = FALSE`, but got fed up always writing the same everywhere. Does not mean it is right, particularly useful or even clever...

The `getPlotArgs/isGood4LOA` combination is a first attempt at providing plot developers with a simple tool to integrate plot argument management by `lattice`, the plot developer and the plot user. It is intended to be applied in the form shown in the Examples below.

Axis, tick, grid and annotation handling in `trianglePlot` is intended to illustrate this type of application.

**Author(s)**

Karl Ropkins

**References**

These function makes extensive use of code developed by others.


**See Also**

In other packages: See `xyplot` in `lattice`.

**Examples**

```r
# getPlotArgs/isGood4LOA notes

# in formals
# my.plot <- function(..., user.reset = TRUE, ...)

# in main code body
# local.resets <- [what developer wants]
# plot.arg <- getPlotArgs("[type]", source, local.reset, user.reset)

# in panel call
```
#(for panel off/on control)
#if(isGood4LOA(plot.arg)) panel...(..., plot.arg,...)

# in panel... function
# for panel component off/on control)
#if(isGood4LOA(plot.arg1)) panel...(..., plot.arg1,...)
#if(isGood4LOA(plot.arg2)) l...(..., plot.arg2,...)
# etc.

---

4.8.list.handlers  
List manipulation

Description
Workhorse functions for routine list handling in loa and elsewhere.

Usage

```r
listHandler(a, use = NULL, ignore = NULL, drop.dots=TRUE)
```

```r
listUpdate(a, b, use = NULL, ignore = NULL, use.a = use, use.b = use, ignore.a = ignore, ignore.b = ignore, drop.dots = TRUE)
```

```r
listExpand(a, ref = NULL, use = NULL, ignore = NULL, drop.dots = TRUE)
```

```r
listLoad(..., load = NULL)
```

Arguments

- **a**  
  A required list. The list to be modified.

- **b**  
  For listUpdate only, a required second list, the contents of which are used to update a with.

- **use, use.a, use.b**  
  Vectors, all defaults NULL. If supplied, a vector of the names of list entries to be used. Other entries are then discarded. use is applied to all supplied lists, while use.a, use.b, etc. can be used to subset a and b lists individually.

- **ignore, ignore.a, ignore.b**  
  Vectors, default NULL. As with use, etc, but for entries to be ignored/not passed on for modification.
ref
For listExpand only, a vector, default NULL. A reference data series, the length of which is used as the expansion length to be applied when wrapping of list entries.

drop.dots
Logical, default TRUE. If TRUE, this removes "..." entries from list names before updating.

... For listLoad only, any additional arguments.

load
For listLoad only, a character vector, default NULL. The names of any lists to be automatically generated from the additional arguments supplied as part of the command call.

Details
listHandler is a general function used by other list... functions for routine list preprocessing.
listUpdate is a list handler intended for use when managing user updates for default options (see examples).
listExpand is a list handler that expands vectors to a given reference length, intended for use for data wrapping.
listLoad is a list generator. See Note below.

Value
By default, all list... functions return results as lists.
listHandler, listUpdate and listExpand functions all return a modified (or updated) version of supplied list a.
listLoad (in-development) returns modified (or updated) version of additional arguments as a list. See Note below.

Note
listLoad is an in-development workhorse function that generates lists based on the supplied load argument.
It assumes each element of load is the name of an expected list and searches the associated additional arguments for arguments to populate it with using the rule "[load].[arg] is an element of list [load]". So, for example, for a call including the arguments load = 'key' and key.fun = draw.colorkey, it would strip out both arguments and return key = list(fun=draw.colorkey). Used in functions, it allowed list-in-list args that can be commonplace when modifying, for example, key elements of conventional lattice plots to be simplified.

Author(s)
Karl Ropkins

References
These functions make extensive use of code developed by others.
4.8.list.handlers

See Also
lattice, xyplot.

Examples

```r
## Example 1
## general

# two lists
list1 <- list(a = 1:10, b = FALSE)
list2 <- list(b = TRUE, c = "new")

# updating a with b
# keeps unchanged list1 entry, a
# updates changed list1 entry, b
# adds new (list2) entry, c
listUpdate(list1, list2)

## Example 2
## use in plot functions
## to simplify formals

## Not run:
# some data
a <- 1:10
b <- rnorm(10,5,2)

# a bad plot function
badplot <- function(x, ...){
  # setting defaults in xyplot call itself
  xyplot(x = x, pch = 20, col = "red",
         panel = function(...){
           panel.grid(-1, -1)
           panel.xyplot(...)
           panel.abline(0,1)
         }, ...)
}

badplot(a-b) #OK

# compare with
badplot(a-b, xlim = c(1,20)) #OK
badplot(a-b, col = "blue") #not OK

# because col hardcoded into badplot function
# It is duplicated in call and '...
# so user cannot update col
```

# a standard correction

stdplot <- function(x, pch = 20, col = "red", ...){

  # setting defaults in xyplot call itself
  xyplot(x = x, pch = 20, col = "red",
  panel = function(x=x, pch=pch, col=col, ...){
    panel.grid(-1, -1)
    panel.xyplot(x=x, pch=pch, col=col, ...)
    panel.abline(0,1)
  }, ...)
}

stdplot(a~b) #OK
stdplot(a~b, col = "blue",
xlim=c(1:20)) #also OK

# An alternative correction using lists and
# listUpdate that removes the need for formal
# definition of all modified plot arguments

myplot <- function(x, ...){

  # defaults I set for myplot form of xyplot
  mylist <- list(x = x, pch = 20, col = "red",
    panel = function(...){
      panel.grid(-1, -1)
      panel.xyplot(...)
      panel.abline(0,1)
    })

  # plot
  do.call(xyplot, listUpdate(mylist, list(...)))
}

myplot(a~b) #OK
myplot(a~b, col = "blue",
xlim = c(1,20)) #also OK

## End(Not run)

---

### 4.9.loa.shapes

**loa shapes**

**Description**

Simple shapes.
Usage

loaPolygon(x, y, z=NULL, rot=NULL, ..., 
polygon = NULL, loa.scale = NULL)

loaCircle(..., polygon = NULL, radius = 1)

loaPieSegment(..., polygon = NULL, start = 0, 
angle=360, radius = 1, center=TRUE)

Arguments

x, y The x and y points at which to plot the requested shape.
z If supplied a z term, most often used to set the size of the polygon.
rot The angle to rotate the polygon by before drawing it.
... Any additional arguments, usually passed on.
polygon A list with elements x and y giving the polygon/shape to be plotted.
loa.scale A list of parameters that can be used to fine-tune the polygon plotting.
radius The radius to used when drawing either circles or pie segments.
start, angle When drawing pie segments, angle the angle of of the segment and start point.
center Should the segment begin and end at the center?

Details

loaPolygon is a general function for drawing polygons. It is intended as an alternative to 1polygon, and other standard loa... shapes are typically wrappers for this function.
loaCircle draws a circle with an origin at (x, y).
loaPieSegment draws a pie segment (or slice of cake) shape. It is typically used as building block for pie plots and other similar glyph structures.

Value

All these functions generate simple shapes and are intended to be run within panel... functions as building blocks for more complex glyph type structures.

Author(s)

Karl Ropkins

References

These functions make extensive use of code developed by others.
Interactive plot functions

Description

Recovering information from existing lattice plots.

Usage

```r
getXY(n = -1, ..., unit = "native", scale.correction = NULL)
getLatLon(...,
    scale.correction = function(x) {
        temp <- MercatorXY2LatLon(x$x, x$y)
        as.list(as.data.frame(temp))
    })
```

Arguments

- `n` If positive, the maximum number of points to locate. If negative (default), unlimited.
- `unit` The unit to use when reporting located points, by default "native".
- `scale.correction` The correction to apply if the plot has locally scaled axes. See Note below.
- `...` Additional arguments, passed on to related functions.

These may be subject to revision, but are currently: `trellis.focus` for panel selection (if working with multi-panel plots) and `lpoints` to set point properties (if marking selected points). For `getLatLon`, additional arguments are also passed to `XY2LatLon` for x, y to latitude, longitude rescaling.

Details

`getXY` is an interactive function which returns the locations of points on a plot selected using the mouse (left click to select points; right click and stop to end point collection; escape to abort without returning any values).

It is a wrapper for the `grid` function `grid.locator` that behaves more like `locator`, the equivalent function intended for use with `plot` outputs.

By default `getXY` selections are not automatically marked. Adding common plot parameters to the function call overrides this behaviour, e.g. to add red symbols and lines.

```r
ans <- getXY(col = "red", pch = 4, type = "b")
```
getXY also provides a mechanism to handle data plotted on locally scaled axes. See Note below.

getLatLon is wrapper for getXY for use with loaMapPlot and other similarly georeferenced plots. See Note below.

**Value**

getXY returns the x and y coordinates of the selected points on a plot as a list containing two components, x and y.

getLatLon returns the latitude and longitude values of the selected points on a map as a list containing two components, lat and lon.

**Note**

getXY recovers the (x, y) coordinates of points selected on a previously generated plot.

Some plots, use local scaling. For example, when plotting latitude, longitude data on a map a scale correction may be used to account for the curvature of the Earth. Similarly, if different data series are plotted on primary and secondary axes in a single plot, some or all data may be normalised. In such cases scaling may be local, i.e. what you actually plot may not be exactly what the annotation says it is.

Using getXY on such plots would recover the actual (x, y) coordinates of the points selected.

However, corrections can be applied using scale.correction, if it is supplied, to convert these to the same scale as the axes annotation. The correction should be a function that can be applied directly to a standard getXY output (a list of x and y values) and rescale x and y to give their 'corrected' values.

getLatLon provides an example of the mechanism, and is for use with georeferenced plots that have been locally scaled using LatLon2MercatorXY. getLatLon uses MercatorXY2LatLon to rescale x and y values and then as... functions to convert the outputs of this step to a list format like that generated by locator, grid.locator or getXY.

**Author(s)**

Karl Ropkins

**References**

This function makes extensive use of code developed by others.


**See Also**

In other packages: See grid.locator; trellis.focus and lpoints in lattice.
5.2.plot.add.functions

General plot add... functions

Description

General functions for adding extra layers to existing plots.

Usage

add_loaPanel(lattice.plot = trellis.last.object(),
preprocess = NULL, panel = NULL,
postprocess = NULL, ...)

Arguments

lattice.plot The plot to add an extra layer or layers to. This defaults to last lattice plot produced if not set in call.
preprocess If set, a function to apply to lattice.plot before adding new panel.
panel If set, a new panel... function to add to lattice.plot after the current lattice.plot panel.
postprocess If set, a function to apply to lattice.plot after adding new panel.
... Additional arguments, passed on to related functions.

Details

add_loaPanel is a frame work function for adding additional layers to an exiting lattice plot.

Value

add_loaPanel returns the supply (or last) lattice plot with an additional plot layer added.

Note

add_loaPanel is in-development and may change significantly in future pacakge versions

Author(s)

Karl Ropkins
5.3. plot.addXYFit.functions

References
This function makes extensive use of code developed by others.

5.3. plot.addXYFit.functions

Plot addXYFit... functions

Description
Functions for adding XY fit model layers to existing plots.

Usage

#main functions

addXYLMFit(lattice.plot = trellis.last.object(),
  preprocess = addXYFit_prep,
  panel = panel.loaXYFit, ...)

addXYLOESSFit(lattice.plot = trellis.last.object(),
  preprocess = addXYFit_prep, model.method = loaXYFit_loess,
  panel = panel.loaXYFit, ...)

#others

addXYFit_prep(lattice.plot = trellis.last.object(),
  model.method = loaXYFit_lm, ...)

loaXYFit_lm(x, y, output.len = 25, formula.signif = 2,
  r2.digits = 3, group.id = NULL, ...)

loaXYFit_loess(x, y, output.len = 25, r2.digits = 3,
  group.id = NULL, ...)

panel.loaXYFit(...)

Arguments

lattice.plot The plot to add an extra layer or layers to. This defaults to last lattice plot produced if not set in call.
5.3.plot.addXYFit.functions

preprocess If set, a function to apply to lattice.plot before adding new panel.
panel If set, a new panel... function to add to lattice.plot after the current lattice.plot panel.
... Additional arguments, passed on to daughter functions.
model.method Function to use on XY data to generate fit reports.
x, y XY data from plot, typically filtered by both panel and group.
output.len Number of XY pairs to use when making fit line standard error bands.
formula.signif, r2.digits When number of significant numbers and digits to use when reporting the fit formula and r.squared, respectively.
group.id When XY data is grouped, the group identity.

Details

addXYLMFit and addXYLMFit add linear and loess model fit surfaces to exiting lattice plot.
Both functions are intended to be used in form:
[make plot, e.g. loaPlot(...) then] addXYLMFit(...)
loaXYFit_lm and loaXYFit_lm generate fit and report information for linear and loess fit surfaces.
Other functions contain code structure common to all addXYFit functions.

Value

addXYLMFit returns the supply (or last) lattice plot with an added linear fit surface layer, generated using lm().
addXYLMFit returns the supply (or last) lattice plot with an added linear fit surface layer, generated using loess().
The elements of the XYFit layers are fit the fit line, se the standard error bands, and report the fit report panels. These track properties, e.g. col, etc, of the plot they add to unless these terms are reset in the addXYFit call. They can also be removed by setting to FALSE, e.g. fit=FALSE, or updated individually using e.g. fit.col=2 to change just the fit line color to red.
The report position within the plot panel can be set using report.position which uses a relative (x=0-1, y=0-1) scale to position the report.

Note

These functions are in-development and may change significantly in future package versions.

Author(s)

Karl Ropkins
5.3.plot.addXYFit.functions

References

This function makes extensive use of code developed by others.


Examples

# add lm fit a loaPlot
addXYLMFit(loaPlot(cadmium~copper*zinc, data=lat.lon.meuse))

## Not run:
loaPlot(cadmium~copper*zinc, data=lat.lon.meuse)
addXYLOESSFit(col="red")
p <- loaPlot(cadmium~copper*zinc, data=lat.lon.meuse)
addXYLMFit(p, se.col="grey", report.position=c(0.95, 0.22))

## End(Not run)
Index

*Topic datasets
  3.1.example.data, 32

*Topic methods
  1.1.loaPlot, 4
  1.2.loaMapPlot.and.geoplotting.tools, 8
  1.3.trianglePlot, 14
  1.4.stackPlot, 19
  1.5.loaBarPlot, 21
  2.1.specialist.panels, 23
  2.2.specialist.panels, 27
  2.3.specialist.panels, 29
  2.4.specialist.panels, 31
  4.1.panel.pal, 34
  4.2.plot.structure.handlers, 38
  4.3.lims.and.scales.handlers, 41
  4.4.cond.handlers, 43
  4.5.plot.argument.handlers, 46
  4.6.key.handlers, 50
  4.7.other.panel.functions, 51
  4.8.list.handlers, 55
  4.9.loa.shapes, 58
  5.1.plot.interactives, 60
  5.2.plot.add.functions, 62
  5.3.plot.addXYFit.functions, 63

*Topic package
  loa-package, 2
  1.1.loaPlot, 4
  1.2.loaMapPlot.and.geoplotting.tools, 8
  1.3.triangle.plots (1.3.trianglePlot), 14
  1.3.trianglePlot, 14
  1.4.stack.plots (1.4.stackPlot), 19
  1.4.stackPlot, 19
  1.5.loaBarPlot, 21
  2.1.specialist.panels, 23
  2.2.specialist.panels, 27
  2.3.specialist.panels, 29
  2.4.specialist.panels, 31
  4.1.panel.pal, 34
  4.2.plot.structure.handlers, 38
  4.3.lims.and.scales.handlers, 41
  4.4.conditioning.handlers (4.4.cond.handlers), 43
  4.5.plot.argument.handlers, 46
  4.6.key.handlers, 50
  4.7.other.panel.functions, 51
  4.8.list.handlers, 55
  4.9.loa.shapes, 58
  5.1.plot.interactives, 60
  5.2.plot.add.functions, 62
  5.3.plot.addXYFit.functions, 63

add.functions (5.2.plot.add.functions), 62
add_loaPanel (5.2.plot.add.functions), 62
addXYFit.functions (5.3.plot.addXYFit.functions), 63
addXYFit_prep (5.3.plot.addXYFit.functions), 63
addXYLMFit (5.3.plot.addXYFit.functions), 63
addXYLOESSFit (5.3.plot.addXYFit.functions), 63
axis.components.loaMap (1.2.loaMapPlot.and.geoplotting.tools), 8
barchart, 23
cexHandler, 3, 5, 9, 10, 29
INDEX

- `cexHandler` (4.5.plot.argument.handlers), 46
- `colHandler`, 3, 5, 10, 23
- `colHandler` (4.5.plot.argument.handlers), 46
- `colRegionsHandler` (4.5.plot.argument.handlers), 46
- `condsPanelHandler`, 3
- `condsPanelHandler` (4.4.cond.handlers), 43
- `coordinates`, 12
- `draw.groupPlotKey` (4.6.key.handlers), 50
- `draw.key.log10` (4.6.key.handlers), 50
- `draw.loaColorKey` (4.6.key.handlers), 50
- `draw.loaColorRegionsKey` (4.6.key.handlers), 50
- `draw.loaPlotZKey` (4.6.key.handlers), 50
- `draw.ycasePlotKey` (4.6.key.handlers), 50
- `draw.zcasePlotKey` (4.6.key.handlers), 50
- `example.data` (3.1.example.data), 32
- `formulaHandler`, 3, 4, 10
- `formulaHandler` (4.2.plot.structure.handlers), 38
- `getArgs`, 3
- `getArgs` (4.7.other.panel.functions), 51
- `getLatLon` (5.1.plot.interactives), 60
- `GetMap`, 12
- `getMapArg` (1.2.loaMapPlot.and.geoplotting.tools), 8
- `getPlotArgs`, 17
- `getPlotArgs` (4.7.other.panel.functions), 51
- `getRGMapArg` (1.2.loaMapPlot.and.geoplotting.tools), 8
- `getXY`, 3
- `getXY` (5.1.plot.interactives), 60
- `getZcaseDimensions` (4.2.plot.structure.handlers), 38
- `GoogleMap`, 4, 18
- `GoogleMap` (1.2.loaMapPlot.and.geoplotting.tools), 8
- `googleMap` (1.2.loaMapPlot.and.geoplotting.tools), 8
- `grid`, 60
- `grid.locator`, 60, 61
- `groupsAndZcasesHandler` (4.4.cond.handlers), 43
- `groupsAndZcasesPanelHandler` (4.4.cond.handlers), 43
- `groupsHandler` (4.4.cond.handlers), 43
- `groupsPanelHandler` (4.4.cond.handlers), 43
- `isGood4LOA` (4.7.other.panel.functions), 51
- `keyHandler`, 3
- `keyHandler` (4.6.key.handlers), 50
- `lat.lon.meuse`, 3
- `lat.lon.meuse` (3.1.example.data), 32
- `LatLon2MercatorXY`, 61
- `LatLon2MercatorXY` (1.2.loaMapPlot.and.geoplotting.tools), 8
- `lattice`, 2–6, 12, 18, 21, 23, 24, 26, 29, 30, 32, 35, 37, 40, 41, 43, 45–48, 51, 54, 57, 60, 61
- `levelplot`, 4–6, 24, 26, 40, 41, 46–48
- `linsHandler`, 3
- `linsHandler` (4.3.lims.and.scales.handlers), 41
- `listExpand` (4.8.list.handlers), 55
- `listHandler`, 3
- `listHandler` (4.8.list.handlers), 55
- `listLoad`, 5
- `listLoad` (4.8.list.handlers), 55
- `listUpdate`, 23
- `listUpdate` (4.8.list.handlers), 55
- `loa` (loa-package), 2
- `loa-package`, 2
- `loa.glyphs`, 45
- `loa.glyphs` (4.9.loa.shapes), 58

(loa-package)
loa.shapes (4.9.loa.shapes), 58
loaBarPlot, 3
loaBarPlot (1.5.loaBarPlot), 21
loaCircle (4.9.loa.shapes), 58
loaHandler (4.1.panel.pal), 34
loaMapArg
 (1.2.loaMapPlot.and.geoplotting.tools)
 8
loaMapPlot, 2
loaMapPlot
 (1.2.loaMapPlot.and.geoplotting.tools)
 8
loaPieSegment (4.9.loa.shapes), 58
loaPlot, 2, 4, 10, 15, 18, 20, 21, 28–30, 32, 39, 41, 51
loaPlot (1.1.loaPlot), 4
loaPolygon (4.9.loa.shapes), 58
loaXYFit_lm
 (5.3.plot.addXYFit.functions), 63
loaXYFit_loess
 (5.3.plot.addXYFit.functions), 63
localScalesHandler, 3, 15, 16, 18
localScalesHandler
 (4.3.lims.and.scales.handlers), 41
locator, 60
loess, 25
lpoints, 60, 61
lrect, 26, 60
makeMapArg
 (1.2.loaMapPlot.and.geoplotting.tools), 8
matrixHandler
 (4.2.plot.structure.handlers), 38
mean, 25
MercatorXY2LatLon, 61
MercatorXY2LatLon
 (1.2.loaMapPlot.and.geoplotting.tools)
 8
na.omit, 16
panel.binPlot (2.1.specialist.panels), 23
panel.compareZcases
 (2.4.specialist.panels), 31
panel.contourplot, 26
panel.kernelDensity, 3
panel.kernelDensity
 (2.1.specialist.panels), 23
panel.levelplot, 12, 45
panel.loaBGMPlot
 (1.2.loaMapPlot.and.geoplotting.tools), 8
panel.loaBGMPlotRaster
 (1.2.loaMapPlot.and.geoplotting.tools), 8
panel.loaGrid (1.1.loaPlot), 4
panel.loaLevelPlot
 (2.1.specialist.panels), 23
panel.loaPlot, 9
panel.loaPlot (1.1.loaPlot), 4
panel.loaPlot2 (1.1.loaPlot), 4
panel.loaXYFit
 (5.3.plot.addXYFit.functions), 63
panel.localScale, 16, 18
panel.localScale
 (4.3.lims.and.scales.handlers), 41
panel.polarAxes
 (2.2.specialist.panels), 27
panel.polarFrame
 (2.2.specialist.panels), 27
panel.polarGrid
 (2.2.specialist.panels), 27
panel.polarLabels
panel.polarPlot, 3
panel.polarPlot
 (2.2.specialist.panels), 27
panel.polygon, 21
panel.stackPlot (1.4.stackPlot), 19
panel.surfaceSmooth
panel.triangleByGroupPolygon
 (1.3.trianglePlot), 14
panel.triangleKernelDensity
 (1.3.trianglePlot), 14
panel.trianglePlot (1.3.trianglePlot), 14
panel.trianglePlotAxes
(1.3.trianglePlot), 14
dpanel.trianglePlotFrame
(1.3.trianglePlot), 14
dpanel.trianglePlotGrid
(1.3.trianglePlot), 14
dpanel.triangleSurfaceSmooth
(1.3.trianglePlot), 14
dpanel.xyplot, 12, 45
dpanel.zcasePiePlot
(2.3.specialist.panels), 29
dpanel.zcasePieSegmentPlot
(2.3.specialist.panels), 29
dpanelPal, 3, 5, 6, 10, 15, 17, 18, 20, 21, 24–30, 32, 41, 44, 45
dparHandler (4.7.other.panel.functions), 51
dpchHandler
(4.5.plot.argument.handlers), 46
dplot, 48, 60
dproj4string, 12
dRColorBrewer, 9, 47
dRgoogleMapsPlot
(1.2.loaMapPlot.and.geoplotting.tools), 8
droadmap.meuse (3.1.example.data), 32
dscalesHandler
(4.5.plot.argument.handlers), 46
dscreenLatticePlot
(5.1.plot.interactives), 60
dspTransform, 12
dstackPlot, 3
dstackPlot (1.4.stackPlot), 19
dstepwiseZcasesGlyphHandler
(4.4.cond.handlers), 43
dstripHandler
(4.2.plot.structure.handlers), 38
dtrellis.focus, 60, 61
dtriABC2XY (1.3.trianglePlot), 14
dtriABCSquareGrid (1.3.trianglePlot), 14
dtrianglePlot, 3, 4, 54
dtrianglePlot (1.3.trianglePlot), 14
dtriLimsReset (1.3.trianglePlot), 14