flowViz

November 11, 2009

R topics documented:

`addName-methods` *Add gate names to a flowViz plot.*

Description

These methods add gate names to a `flowViz` plot, either derived from the population identifiers or as provided by the user. These methods are ment for internal use and are usually not called directly by the user.

Value

The methods are called for their side effects. No value is returned.

Methods

x = "curv1Filter", name = "character" User-provided names.

x = "curv1Filter", name = "logical" Get names from the `filter` or `filterResult` object

x = "curv2Filter", name = "character" see above

x = "curv2Filter", name = "logical" see above

x = "ellipsoidGate", name = "character" see above

x = "ellipsoidGate", name = "logical" see above

x = "kmeansFilter", name = "character" see above

x = "kmeansFilter", name = "logical" see above

x = "polygonGate", name = "character" see above

x = "polygonGate", name = "logical" see above

x = "quadGate", name = "character" see above

x = "quadGate", name = "logical" see above

x = "quadGate", name = "matrix" see above

x = "rectangleGate", name = "character" see above

x = "rectangleGate", name = "logical" see above

Author(s)

F. Hahne

`contour-methods` *Contour plots for flow data*

Description

Basic contour plots for both `flowFrame`s and `flowSet`s. The densities for the contours are estimated using the fast kernel density estimation algorithm `bkde2D`.

Usage

```
## method for 'flowFrame' objects
## S4 method for signature 'flowFrame':
contour(
    x,
    y=1:2,
    nlevels=10,
    bw,
    grid.size=c(65,65),
    add=FALSE,
    xlab,
    ylab,
    lwd=1,
    lty=1,
    col=par("fg"),
```

```
    fill="transparent",
    ...)

## method for 'flowSet' objects
## S4 method for signature 'flowSet':
contour(
    x,
    y=1:2,
    add=FALSE,
    xlab,
    ylab,
    lwd=1,
    lty=1,
    col=par("fg"),
    fill="transparent",
    ...)
```

Arguments

`x`	An object of class `flowFrame` or `flowSet`.
`y`	Numeric or character vector of length 2 indicating the channels to plot.
`nlevels`	The approximate number of contour line levels, see `contour` for details.
`bw`	The bandwidth factor used for the kernel density estimation, see `bkde2D` for details.
`grid.size`	The grid size used for the kernel density estimation, see `bkde2D` for details.
`add`	Logical, indicating whether contour lines should be superimposed on an existing plot.
`xlab, ylab`	The axis annotation.
`lwd, lty, col, fill`	The usual plotting parameters, i.e. the line width, line type, line color and fill color. When using a fill color you should consider alpha blending to improve the results.
`...`	Parameters that are passed on to the plotting functions.

Methods

x = "flowFrame" A regular contour plot of the flow data in the frame. It can be added on top of an existing plot using the `add` argument.

x = "flowSet" Overlay of contours of densities for each individual frame in the set. You should consider using differnt colors and alpha blending to improve the result. This is only useful for a very limited number of frames in a set (~5), for larger sets you should consider a panelled lattice-type plot. Not that `bw`, `gridSize` and `nlevels` are passed on via the ... argument.

Author(s)

F. Hahne

See Also

`bkde2D`, `contour`, `flowFrame`, `flowSet`

Examples

```
data(GvHD)

## simple contour plot
contour(GvHD[[1]])

## overlay with existing plot
plot(GvHD[[1]], c("FSC-H", "SSC-H"))
contour(GvHD[[1]], add=TRUE, col="lightgray", lty=3)

## colored contours
contour(GvHD[[1]], fill="red")
cols <- rainbow(3, alpha=0.1)
contour(GvHD[[1]], fill=cols, col=cols)

## overlay of multiple flowFrames in a flowSet
contour(GvHD[1:3], col=cols, fill=cols)
```

`densityplot` *One-dimensional density plots for flow data*

Description

So far, density plots are only implemented for `flowSet`. The idea is to horizontally stack plots of density estimates for all frames in the `flowSet` for one or several flow parameters. In the latter case, each parameter will be plotted in a separate panel, i.e., we implicitely condition on parameters.

Usage

```
## method for 'flowSet' objects
## S4 method for signature 'formula, flowSet':
densityplot(
    x,
    data,
    xlab,
    as.table=TRUE,
    overlap=0.3,
    prepanel=prepanel.densityplot.flowset,
    panel = panel.densityplot.flowset,
    filter=NULL,
    scales=list(y=list(draw=F)),
    channels,
    ...)

prepanel.densityplot.flowset(
    x,
    y,
    darg=list(n=50, na.rm=TRUE),
    frames,
```

```
          overlap=0.3,
          subscripts,
          ...,
          which.channel)

panel.densityplot.flowset(
          x,
          y,
          darg=list(n=50, na.rm=TRUE),
          frames,
          channel,
          overlap = 0.3,
          channel.name,
          filter=NULL,
          fill=superpose.polygon$col,
          lty=superpose.polygon$lty,
          lwd=superpose.polygon$lwd,
          alpha=superpose.polygon$alpha,
          col=superpose.polygon$border,
          groups=NULL,
          refline=NULL,
          gpar,
          ...)


## methods for various workflow objects
## S4 method for signature 'formula, view':
densityplot(
          x,
          data,
          ...)

## S4 method for signature 'view, missing':
densityplot(
          x,
          data,
          channels,
          ...)
```

Arguments

`x`	A formula describing the structure of the plot and the variables to be used in the display. The structure of the formula is `factor ~ parameter`, where `factor` can be any of the phenotypic factors in the `phenoData` slot or an appropriate factor object and `parameter` is a flow parameter. Panels for multiple parameters are drawn if the formula structure is similar to `factor ~ parameter1 + parameter2`, and `factor` can be missing, in which case the sample names are used as y-variable. To facilitate programatic access, the formula can be of special structure `factor ~ .`, in which case the optional `channel` argument is considered for parameter selection. For the workflow methods, `x` can also be one of the several workflow objects.

`data`	A `flowSet` object that serves as a source of data, either a `flowFrame` or `flowSet`, or one of the several workflow objects.
`xlab`	Label for data x axis, with suitable defaults taken from the formula
`as.table, scales, darg`	These arguments are passed unchanged to the corresponding methods in lattice, and are listed here only because they provide different defaults. See documentation for the original methods for details. `darg` gets passed on to `density`.
`overlap`	The amount of overlap between stacked density plots
`prepanel`	The prepanel function. See `xyplot`
`panel`	the panel function. See `xyplot`
`filter`	A `filter`, `filterResult` or `filterResultList` object or a list of such objects of the same length as the `flowSet`. If applicable, the gate region will be superiposed on the density curves using color shading. The software will figure out whether the `filter` needs to be evaluated in order to be plotted (in which case providing a `filterResult` can speed things up considerably).
`channels`	A character vector of parameters that are supposed to be plotted when the formula in `x` is of structure `factor ~ .`.
`subscripts, which.channel, channel.name, y`	Internal indices necessary to map panels to parameters.
`frames`	An environment containing frame-specific data.
`channel`	The name of the currently plotted flow parameter.
`col, fill, lty, lwd, alpha`	Graphical parameters. These mostly exist for conveniance and much more control is available throught the `lattice`-like `par.setting` and `flowViz.par.set` customization. The relevant parameter category for density plots is `gate.density` with available parameters `col, \code{fill}, \code{lwd} \code{alpha} and \code{lty}`. See `flowViz.par.set` for details.
`groups`	Use identical colors for grouping. The value of the argument is expected to be a phenotypic variable in the `flowSet`.
`refline`	Add one ore more vertical reference lines to the plot. This argument is directly passed to `panel.abline`.
`gpar`	A list of graphical parameters that are passed down to the low level panel functions. This is for internal use only. The public user interface to set graphical parameters is either `par.settings` for customization of a single call or `flowViz.par.set` for customization of session-wide defaults.
`...`	More arguments, usually passed on to the underlying lattice methods.

Details

Not all standard lattice arguments will have the intended effect, but many should. For a fuller description of possible arguments and their effects, consult documentation on lattice (Trellis docs would also work for the fundamentals).

Methods

densityplot `signature(x = "formula", data = "flowSet")`: Creates density plots for one or several channels, with samples stacked according to a `phenoData` variable. Colors are used to indicate common values of this covariate across panels. Filters can be added as the optional `filter` arguments. See `xyplot` for details.

densityplot `signature(x = "formula", data = "view")`: A method to create density plots for workspace `view` objects. This still allows for some level of customization, but most defaults will be set depending on the input object.

densityplot `signature(x = "view", data = "missing")`: The default method for view objects. All defaults will be set.

Examples

```
data(GvHD)
GvHD <- GvHD[pData(GvHD)$Patient %in% 6:7]

densityplot(~ `FSC-H`, GvHD)

densityplot(~ `FSC-H` + `SSC-H`, GvHD)

densityplot(~ ., GvHD[1:3])

## include a filter
densityplot(~ `FSC-H`, GvHD, filter=curv1Filter("FSC-H"))
```

`ecdfplot` *Method implementing Lattice ECDF plots for flow data*

Description

This function creates Trellis displays of Empirical Cumulative Distribution Functions from flow cytometry data using a formula interface.

Usage

```
## methods for 'flowSet' objects

## S4 method for signature 'formula, flowSet':
ecdfplot(x, data, xlab, ylab,
         f.value = function(n) ppoints(ceiling(sqrt(n))),
         type = "l", as.table = TRUE,
         ...)

prepanel.ecdfplot.flowset(x, frames, channel,
                          f.value, ...)
panel.ecdfplot.flowset(x, frames, channel,
                       f.value, ref = TRUE,
                       groups = NULL, subscripts,
                       col, col.points, pch, cex,
                       alpha, col.line, lty, lwd, ...)
```

Arguments

`x`	a formula describing the structure of the plot and the variables to be used in the display. For the prepanel and panel functions, a vector of names for the flow frames to be used in the panel.
`data`	a `flowSet` object that serves as a source of data
`xlab, ylab`	Labels for data axes, with suitable defaults taken from the formula
`f.value`	determines the number of points used in the plot `ecdfplot` for details.
`type`	type of rendering; by default lines are drawn
`as.table`	logical; whether to draw panels from top left
`ref`	logical; whether to add reference lines at 0 and 1
`frames`	environment containing frame-specific data
`channel`	expression involving names of columns in the data
`groups, subscripts`	grouping variable, if specified, and subscripts indexing which frames are being used in the panel. See `xyplot` for details.
`col, col.points, pch, cex, alpha, col.line, lty, lwd`	vector of graphical parameters that are replicated for each group
`...`	more arguments, usually passed on to the underlying lattice methods and the panel function.

Methods

ecdfplot `signature(x = "formula", data = "flowSet")`: plote empirical CDF for a given channel, with one or more samples per panel

See Also

Not all standard lattice arguments will have the intended effect, but many should. For a fuller description of possible arguments and their effects, consult documentation on lattice.

Examples

```
data(GvHD)

ecdfplot(~ `FSC-H` | Patient, GvHD, f.value = ppoints(100))

ecdfplot(~ asinh(`FSC-H`) | Patient, GvHD,
         strip = strip.custom(strip.names = TRUE),
         ref = FALSE)

ecdfplot(~ asinh(`FSC-H`) | Patient, GvHD, groups = Visit,
         strip = strip.custom(strip.names = TRUE),
         ref = FALSE, auto.key = list(columns = 4))
```

`flowPlot` *Standard Plots for Flow Cytometry Data*

Description

A method that makes standard plots from a `flowFrame`. The user may also provide various `filter` or `filterResult` arguments to customize the plot.

Usage

```
flowPlot(x, ...)

## S4 method for signature 'flowFrame':
flowPlot(x, child, filter = NULL,
         plotParameters = c("FSC-H","SSC-H"),
         logx = FALSE, logy = FALSE,
         parent, colParent="Grey", colChild="Blue",
         showFilter = TRUE, gate.fill = "transparent",
         gate.border = "black", xlab, ylab, xlim, ylim,
         ...)
```

Arguments

`x`	An object of class `flowFrame` that contains the data to be plotted.
`child`	An optional argument of class `filterResult` that specifies a subset of the data that are included in the `filterResult`
`filter`	
`plotParameters`	
`logx`	
`logy`	
`parent`	
`colParent`	
`colChild`	
`showFilter`	
`gate.fill`	
`gate.border`	
`xlab`	
`ylab`	
`xlim`	
`ylim`	
`...`	

Details

The plot that is most commonly used in flow cytometry data analysis is usuall called a "dot plot". In common statistical language, we would call this a scatter plot. The basic idea is a 2-dimensional plot that shows the location of every cell in regard to the measurements made on it, for example, forward scatter vs side scatter. Most applications will, in addition to the data, want to show information about one or more filters (gates). Since there can be a very large number of cells in a sample, it is common to show a smoothed version of the data that doesn't involve registering every point on the graph.

Author(s)

P. Haaland

See Also

`flowCore`

Examples

```
data(GvHD)
flowPlot(GvHD[["s5a01"]])
flowPlot(transform("SSC-H"=asinh,"FSC-H"=asinh) %on% GvHD[["s5a01"]])
```

`flowViz-package` *Visualization for flow cytometry*

Description

Functions and methods to visualize flow cytometry data. This package heavily depends on the flowCore package.

Details

Package:	flowViz
Type:	Package
Version:	0.2.1
Date:	2006-11-16
License:	Artistic

Traditionally, large parts of the analysis process of flow cytometry data has been mostly qualitative. To this end, dedicated visualization techniques have been used for both quality control and infernece of the data. This package provides a number of different visualization tools for flow data.

Author(s)

Maintainer: Florian Hahne <f.hahne@dkfz.de> Authors: T. Duong, B. Ellis, R. Gentleman, F. Hahne, N. Le Meur, D. Sarkar, M. Tang

See Also

`flowCore`

Examples

```
## examples go here
```

`flowViz.par.get` *Query and set session-wide graphical parameter defaults.*

Description

`flowViz.par.get` is the equivalent to `trellis.par.get`. It queries the session wide defaults for all `lattice` and `flowViz` graphical parameters.

`flowViz.par.set` is the equivalent to `trellis.par.set`. It sets the same set of graphical parameters, either in the `flowViz` package or directly in `lattice`.

Usage

```
flowViz.par.get(name = NULL)

flowViz.par.set(name, value, ..., theme, warn = TRUE, strict = FALSE)
```

Arguments

`name`	The name of a parameter category to set.
`value`	A named list of values to set for category `name` or a list of such lists if `name` is missing.
`...`	Further arguments that get passed on.
`theme`	The theme to set. See `trellis.par.set` for details.
`warn`	This gets passed on directly to `trellis.par.set`.
`strict`	This gets passed on directly to `trellis.par.set`.

Details

Getting and setting graphical parameters in `flowViz` follows exactly the mechanism of the `lattice` package. For all purpose and intentions, `flowViz.par.get` and `flowViz.par.get` can be viewed as wrappers around their `lattice` counterparts `trellis.par.get` and `trellis.par.set` and you should consult their documentation for further details.

We introduce four new categories of graphical parameters that are relevant for `flowViz` plots:

gate Controls the appearance of gate boundaries in `xyplots` (if `smooth=TRUE`) or of the points within a gate region (`smooth=FALSE`). Available parameters are `col`, `cex,pch,alpha,lwd,lty` and `fill`.

gate.density Controls the appearance of gate boundaries in `densityplots`. Available parameters are `col`, `alpha,lwd,lty` and `fill`.

flow.symbol Controls the appearance of 'regular' points in a `flowViz` plot. Available parameters are `col`, `cex`,`pch`,`alpha` and `fill`.

gate.text Controls the appearance of the text used for gate names. Available parameters are `col`, `cex`,`font`,`alpha` and `lineheight`.

Value

`flowViz.par.get` returns a list of graphical parameter defaults, if `name` is not empty, only for this particular category. For an empty `name` argument, the function returns all parameter defaults, including the ones specified in the lattice package.

`flowViz.par.set` is called for its side-effects of setting default parameters.

Note

Because parameter settings in `lattice` are device-dependent, `flowViz.par.get` will open a (default) device none is open at the time of the query.

Author(s)

F. Hahne

References

Lattice, Multivariate Data Visualization with R.

Deepayan Sarkar

Springer, New York, 2008

See Also

`trellis.par.get` and `trellis.par.set`

Examples

```
## Return all available parameters, including lattice ones
flowViz.par.get()

## Set the font for gate names
flowViz.par.set("gate.text", list(font=2))

## Query only the gate.text category
flowViz.par.get("gate.text")

## Set a lattice parameter
plot.symbol <- trellis.par.get("plot.symbol")
flowViz.par.set("plot.symbol", list(col="red"))
trellis.par.get("plot.symbol")

## undo all settings
flowViz.par.set(list(plot.symbol=plot.symbol, gate.text=list(font=1)))
```

`hexbin` *Fast hexagon binning.*

Description

Yet another hexagon binning routine - this one adapted from the scagnostics package. This is very preliminary and may not be retained.

Usage

```
hexbin(x, y, bins = 50)
```

Arguments

`x`	x values
`y`	y values
`bins`	The approximate number of bins, in the x direction.

Details

This implementation maps x and y to the unit square and does the binning in that region.

Value

A list, with componentes `counts`, `xbin` and `ybin`.

References

The scagnostics package.

Examples

```
x = runif(100)
y = runif(100)
hb = hexbin(x,y)
```

`lattice-methods` *Methods implementing Lattice displays for flow data*

Description

Various methods implementing multipanel visualizations for flow data using infrastructure provided in the lattice package. The original generics for these methods are defined in lattice, and these S4 methods (mostly) dispatch on a formula and the `data` argument which must be of class `flowSet` or `flowFrame`. The formula has to be fairly basic: conditioning can be done using phenodata variables and channel names (the `colnames` slot) can be used as panel variables. See examples below for sample usage.

Usage

```
## methods for 'flowSet' objects
## S4 method for signature 'formula, flowSet':
qqmath(
    x,
    data,
    xlab,
    ylab,
    f.value = function(n) ppoints(ceiling(sqrt(n))),
    distribution = qnorm,
    ...)

## S4 method for signature 'formula, flowSet':
levelplot(
    x,
    data,
    xlab,
    ylab,
    as.table = TRUE,
    contour = TRUE,
    labels = FALSE,
    n = 50,
    ...)

## methods for 'flowFrame' objects
## S4 method for signature 'flowFrame, missing':
parallel(
    x,
    data,
    reorder.by = function(x) var(x, na.rm = TRUE),
    time = "Time",
    exclude.time = TRUE,
    ...)
```

Arguments

`x`	a formula describing the structure of the plot and the variables to be used in the display.
`data`	a `flowSet` object that serves as a source of data.
`xlab, ylab`	Labels for data axes, with suitable defaults taken from the formula
`f.value, distribution`	number of points used in Q-Q plot, and the reference distribution used. See `qqmath` for details.
`n`	the number of bins on each axis to be used when evaluating the density
`as.table, contour, labels`	These arguments are passed unchanged to the corresponding methods in lattice, and are listed here only because they provide different defaults. See documentation for the original methods for details.
`time`	A character string giving the name of the column recording time.

`exclude.time`	logical, specifying whether to exclude the time variable from a scatter plot matrix or parallel coordinates plot. It is rarely meaningful not to do so.
`reorder.by`	a function, which is applied to each column. The columns are ordered by the results. Reordering can be suppressed by setting this to `NULL`.
`...`	more arguments, usually passed on to the underlying lattice methods.

Details

Not all standard lattice arguments will have the intended effect, but many should. For a fuller description of possible arguments and their effects, consult documentation on lattice (Trellis docs would also work for the fundamentals).

Methods

qqmath `signature(x = "formula", data = "flowSet")`: creates theoretical quantile plots of a given channel, with one or more samples per panel

levelplot `signature(x = "formula", data = "flowSet")`: similar to the `xyplot` method, but plots estimated density (using `kde2d`) with a common z-scale and an optional color key.

parallel `signature(x = "flowFrame", data = "missing")`: draws a parallel coordinates plot of all channels (excluding time, by default) of a `flowFrame` object. This is rarely useful without transparency, but that is currently only possible with the `pdf` device (and perhaps the aqua device as well).

Examples

```
data(GvHD)

qqmath( ~ `FSC-H` | factor(Patient), GvHD,
       grid = TRUE, type = "l",
       f.value = ppoints(100))

## contourplot of bivariate density:

require(colorspace)
YlOrBr <- c("#FFFFD4", "#FED98E", "#FE9929", "#D95F0E", "#993404")
colori <- colorRampPalette(YlOrBr)
levelplot(asinh(`SSC-H`) ~ asinh(`FSC-H`) | Visit + Patient, GvHD, n = 20,
          col.regions = colori(50), main = "Contour Plot")


## parallel coordinate plots

parallel(GvHD[["s6a01"]])

## Not run:

## try with PDF device
parallel(GvHD[["s7a01"]], alpha = 0.01)

## End(Not run)
```

`glines-methods` *Drawing filter boundaries*

Description

These methods extend the basic graphics `lines` methods for drawing of `filter` boundaries. They allow for multiple dispatch, since not all `filter` types need to be evaluated for plotting, but this decision should be made internally.

Details

When plotting `flowFrame`s using the `plot` or `xyplot` methods provided by `flowViz`, the plotted parameters are recorded, which makes it possible to correctly overlay the outlines of `filter`s assuming that they are defined for the repective parameters. Warnings and error will be cast for the cases where the parameters are non-distinct or ambigious.

The flow parameters plotted can be passed on to any of the methods through the optional `channels` argument, which always gets precedence over automatically detected parameters.

The methods support all plotting parameters that are available for the `base lines` functions.

Methods

x = "filter", data = "missing" General method for all objects inheriting from `filter`. This is used as the default when no more explicit method is found. It tries to find the plotted parameters from the internal `flowViz.state` environment. This only works if the flow data has been plotted using the `plot` or `xyplot` methods provided by this `flowViz` package.

x = "filterResult", data = "ANY" General method for all `filterResult` object. This basically extracts the `filter` from the `filterResult` and dispatches on that.

x = "filterResult", data = "flowFrame" For some `filter` types we need the raw data to re-evaluate the filter.

x = "curv1Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv1Filter`s.

x = "curv1Filter", data = "flowFrame" see above

x = "curv1Filter", data = "missing" see above

x = "curv1Filter", data = "multipleFilterResult" see above

x = "curv2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv2Filter`s.

x = "curv2Filter", data = "flowFrame" see above

x = "curv2Filter", data = "multipleFilterResult" see above

x = "kmeansFilter", data = "ANY" We don't know how to plot outlines of a `kmeansFilter`, hence we warn.

x = "norm2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `norm2Filter`s.

x = "norm2Filter", data = "flowFrame" see above

x = "norm2Filter", data = "logicalFilterResult" see above

x = "polygonGate", data = "character" We can plot a `polygonGate` directly from the gate definition.

x = "polygonGate", data = "filterResult" see above

x = "polygonGate", data = "flowFrame" see above

x = "quadGate", data = "character" We can plot a `quadGate` directly from the gate definition.

x = "quadGate", data = "filterResult" see above

x = "quadGate", data = "flowFrame" see above

x = "rectangleGate", data = "character" We can plot a `rectangleGate` directly from the gate definition.

x = "rectangleGate", data = "filterResult" see above

x = "rectangleGate", data = "flowFrame" see above

x = "ellipsoidGate", data = "character" We can plot a `rectangleGate` directly from the gate definition.

x = "ellipsoidGate", data = "filterResult" see above

x = "ellipsoidGate", data = "flowFrame" see above

Author(s)

F. Hahne

See Also

`filter`, `flowFrame`, `glines`, `gpoints`

`glpoints-methods` *Adding points within a gate to a plot*

Description

These methods extend the lattice `lpoints` methods for drawing of points contained within a `filter`. They allow for multiple dispatch, since not all `filter` types need to be evaluated for plotting, but this decision should be made internally. In any case, we need the raw data in the form of a `flowFrame`.

Details

When plotting `flowFrame`s using the `plot` method provided by `flowViz`, the plotted parameters are recorded, which makes it possible to correctly overlay the points within `filter`s assuming that they are defined for the repective parameters. Warnings and error will be cast for the cases where the parameters are non-distinct or ambigious. These methods are meant to be used within lattice panel functions and are probably not of much use outside of those.

Methods

x = "filter", data = "flowFrame", channels = "missing" General method for all objects inheriting from `filter`. This is used as the default when no more explicit method is found. It tries to find the plotted parameters from the internal `flowViz.state` environment. This only works if the flow data has been plotted using the `plot` methods provided by this `flowViz` package.

x = "filter", data = "missing", channels = "ANY" This gives a useful error message when we don't get what we need.

x = "filterResult", data = "flowFrame", channels = "character" We can get all the information about a `filter` from its `filterResult` without the need to re-evaluate.

x = "curv1Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv1Filter`s.

x = "curv1Filter", data = "flowFrame" see above

x = "curv1Filter", data = "missing" see above

x = "curv1Filter", data = "multipleFilterResult" see above

x = "curv2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv2Filter`s.

x = "curv1Filter", data = "flowFrame", channels = "character" We evaluate the `filter` on the `flowFrame` and plot the subset of selected points. By default, every subpopulation (if there are any) is colored differently.

x = "curv2Filter", data = "flowFrame", channels = "character" see above

x = "kmeansFilter", data = "flowFrame", channels = "character" see above

x = "norm2Filter", data = "flowFrame", channels = "character" see above

x = "polygonGate", data = "flowFrame", channels = "character" see above

x = "quadGate", data = "flowFrame", channels = "character" see above

x = "rectangleGate", data = "flowFrame", channels = "character" see above

x = "ellipsoidGate", data = "flowFrame", channels = "character" see above

Author(s)

F. Hahne

See Also

`filter`, `flowFrame`, `glpolygon`

`glpolygon-methods` *Drawing filter regions*

Description

These methods extend the lattice `lpolygon` methods for drawing of `filter` regions. They allow for multiple dispatch, since not all `filter` types need to be evaluated for plotting, but this decision should be made internally.

Details

When plotting `flowFrame`s using the any of the lattice-type `plot` method provided by `flowViz`, the plotted parameters are recorded, which makes it possible to correctly overlay the outlines of `filter`s assuming that they are defined for the repective parameters. Warnings and error will be cast for the cases where the parameters are non-distinct or ambigious. These methods are meant to be used within lattice panel functions and are probably not of much use outside of those.

Value

The methods will return the outlines of the gate region as polygon vertices.

Methods

x = "filter", data = "missing" General method for all objects inheriting from `filter`. This is used as the default when no more explicit method is found. It tries to find the plotted parameters from the internal `flowViz.state` environment. This only works if the flow data has been plotted using the `plot` methods provided by this `flowViz` package.

x = "filterResult", data = "missing" General method for all `filterResult` object. This basically extracts the `filter` from the `filterResult` and dispatches on that.

x = "filterResult", data = "flowFrame" For some `filter` types we need the raw data to re-evaluate the filter.

x = "curv1Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv1Filter`s.

x = "curv1Filter", data = "flowFrame" see above

x = "curv1Filter", data = "missing" see above

x = "curv1Filter", data = "multipleFilterResult" see above

x = "curv2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv2Filter`s.

x = "curv2Filter", data = "flowFrame" see above

x = "curv2Filter", data = "multipleFilterResult" see above

x = "kmeansFilter", data = "ANY" We don't know how to plot regions of a `kmeansFilter`, hence we warn.

x = "norm2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `norm2Filter`s.

x = "norm2Filter", data = "flowFrame" see above

x = "norm2Filter", data = "logicalFilterResult" see above

x = "polygonGate", data = "character" We can plot a `polygonGate` directly from the gate definition.

x = "polygonGate", data = "filterResult" see above

x = "polygonGate", data = "flowFrame" see above

x = "quadGate", data = "character" We can plot a `quadGate` directly from the gate definition.

x = "quadGate", data = "filterResult" see above

x = "quadGate", data = "flowFrame" see above

x = "rectangleGate", data = "character" We can plot a `rectangleGate` directly from the gate definition.

x = "rectangleGate", data = "filterResult" see above

x = "rectangleGate", data = "flowFrame" see above

x = "ellipsoidGate", data = "character" We can plot a `rectangleGate` directly from the gate definition.

x = "ellipsoidGate", data = "filterResult" see above

x = "ellipsoidGate", data = "flowFrame" see above

Author(s)

F. Hahne

See Also

`filter`, `flowFrame`, `glpoints`

`plot-methods` *Very basic plotting of flowFrames*

Description

A basic method to plot `flowFrame` objects. Depending on the number of dimensions, different types of plots are generates. See below for details.

Details

Basic plots for `flowFrame` objects. If the object has only a single parameter this produces a `histogram`. For exactly two parameters we plot a bivariate density map (see `smoothScatter`) and for more than two parameters we produce a simple `splom` plot. To select specific parameters from a `flowFrame` for plotting, either subset the object or specify the parameters as a character vector in the second argument to `plot`. The `smooth` parameters lets you toggle between density-type `smoothScatter` plots and regular scatter- or pairs plots. For far more sophisticated plotting of flow cytometry data, see the lattice-style plot methods provided by this package.

Methods

x = "flowFrame", y = "ANY" We decide on the number of parameters in the `flowFrame`which plot type to use.

x = "flowFrame", y = "missing" see above

x = "flowFrame", y = "character" The parameters to plot are given as a second argument in the form of a character vector.

Author(s)

F. Hahne

See Also

`xyplot`, `flowFrame`, `densityplot`

`gpoints-methods` *Adding points within a gate to a plot*

Description

These methods extend the basic graphics `points` methods for drawing of points contained within a `filter`. They allow for multiple dispatch, since not all `filter` types need to be evaluated for plotting, but this decision should be made internally. In any case, we need the raw data in the form of a `flowFrame`.

Details

When plotting `flowFrame`s using the `plot` method provided by `flowViz`, the plotted parameters are recorded, which makes it possible to correctly overlay the points within `filter`s assuming that they are defined for the repective parameters. Warnings and error will be cast for the cases where the parameters are non-distinct or ambigious.

Methods

x = "filter", data = "flowFrame", channels = "missing" General method for all objects inheriting from `filter`. This is used as the default when no more explicit method is found. It tries to find the plotted parameters from the internal `flowViz.state` environment. This only works if the flow data has been plotted using the `plot` methods provided by this `flowViz` package.

x = "filter", data = "missing", channels = "ANY" This gives a useful error message when we don't get what we need.

x = "filterResult", data = "flowFrame", channels = "character" We can get all the information about a `filter` from its `filterResult` without the need to re-evaluate.

x = "curv1Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv1Filter`s.

x = "curv1Filter", data = "flowFrame" see above

x = "curv1Filter", data = "missing" see above

x = "curv1Filter", data = "multipleFilterResult" see above

x = "curv2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv2Filter`s.

x = "curv1Filter", data = "flowFrame", channels = "character" We evaluate the `filter` on the `flowFrame` and plot the subset of selected points. By default, every subpopulation (if there are any) is colored differently.

x = "curv2Filter", data = "flowFrame", channels = "character" see above

x = "kmeansFilter", data = "flowFrame", channels = "character" see above

x = "norm2Filter", data = "flowFrame", channels = "character" see above

x = "polygonGate", data = "flowFrame", channels = "character" see above

x = "quadGate", data = "flowFrame", channels = "character" see above

x = "rectangleGate", data = "flowFrame", channels = "character" see above

Author(s)

F. Hahne

See Also

`filter`, `flowFrame`, `glines`, `gpolygon`

`gpolygon-methods` *Drawing filter regions*

Description

These methods extend the basic graphics `polygon` methods for drawing of `filter` regions. They allow for multiple dispatch, since not all `filter` types need to be evaluated for plotting, but this decision should be made internally.

Details

When plotting `flowFrame`s using the `plot` method provided by `flowViz`, the plotted parameters are recorded, which makes it possible to correctly overlay the outlines of `filter`s assuming that they are defined for the repective parameters. Warnings and error will be cast for the cases where the parameters are non-distinct or ambigious.

The flow parameters plotted can be passed on to any of the methods through the optional `channels` argument, which always gets precedence over automatically detected parameters.

The methods support all plotting parameters that are available for the `base` `polygon` functions.

Methods

x = "filter", data = "missing" General method for all objects inheriting from `filter`. This is used as the default when no more explicit method is found. It tries to find the plotted parameters from the internal `flowViz.state` environment. This only works if the flow data has been plotted using the `plot` methods provided by this `flowViz` package.

x = "filterResult", data = "ANY" General method for all `filterResult` object. This basically extracts the `filter` from the `filterResult` and dispatches on that.

x = "filterResult", data = "flowFrame" For some `filter` types we need the raw data to re-evaluate the filter.

x = "curv1Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv1Filter`s.

x = "curv1Filter", data = "flowFrame" see above

x = "curv1Filter", data = "missing" see above

x = "curv1Filter", data = "multipleFilterResult" see above

x = "curv2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `curv2Filter`s.

x = "curv2Filter", data = "flowFrame" see above

x = "curv2Filter", data = "multipleFilterResult" see above

x = "kmeansFilter", data = "ANY" We don't know how to plot regions of a `kmeansFilter`, hence we warn.

x = "norm2Filter", data = "ANY" We either need a `filterResult` or the raw data as a `flowFrame` for `norm2Filter`s.

x = "norm2Filter", data = "flowFrame" see above

x = "norm2Filter", data = "logicalFilterResult" see above

x = "polygonGate", data = "character" We can plot a `polygonGate` directly from the gate definition.

x = "polygonGate", data = "filterResult" see above

x = "polygonGate", data = "flowFrame" see above

x = "quadGate", data = "character" We can plot a `quadGate` directly from the gate definition.

x = "quadGate", data = "filterResult" see above

x = "quadGate", data = "flowFrame" see above

x = "rectangleGate", data = "character" We can plot a `rectangleGate` directly from the gate definition.

x = "rectangleGate", data = "filterResult" see above

x = "rectangleGate", data = "flowFrame" see above

x = "ellipsoidGate", data = "character" We can plot a `ellipsoidGate` directly from the gate definition.

x = "ellipsoidGate", data = "filterResult" see above

x = "ellipsoidGate", data = "flowFrame" see above

Author(s)

F. Hahne

See Also

`filter`, `flowFrame`, `glines`, `gpoints`

`splom` *Method implementing Lattice scatter plot matrices for flow data.*

Description

This function create Trellis scatter plots matrices (splom) from flow cytometry data.

Usage

```
## S4 method for signature 'flowFrame, missing':
splom(
    x,
    data,
    pscales,
    time,
    exclude.time=TRUE,
    names=FALSE,
    ...)

panel.splom.flowframe(
    x,
    frame,
    ...)
```

Arguments

`x`	A formula describing the structure of the plot and the variables to be used in the display.
`data, frame`	A `flowFrame` object that serves as the source of data.
`pscales`	This arguments is passed unchanged to the corresponding methods in lattice, and is listed here only because it provides a different default. See documentation for the original methods for details.
`time`	A character string giving the name of the data column recording time. If not provided, we try to guess from the available parameters.
`exclude.time`	Logical, specifying whether to exclude the time variable from a scatter plot matrix. Defaults to `TRUE`.
`names`	Logical specifying wether gate names should be added to the plot. Currently, this feature is not supported for splom plots.
`...`	More arguments, usually passed on to the underlying lattice methods.

Details

The function draws a scatter plot matrix of the data for each flow parameter in a `flowFrame`. For the most, one can think about this as a rectangular arrangement of separate `xyplots`, and most of that functionality is also available here. To be more precise, the function repeatedly calls `panel.xyplot.flowframe` to do the actual plotting. Please see its documentation for details.

Methods

splom `signature(x = "flowFrame", data = "missing")`: Creates a scatter plot matrix for a whole `flowFrame`. The actual plotting is done by `panel.xyplot.flowframe`, so most additional arguments from the regular `xyplot` are also valid here.

Author(s)

F. Hahne, D. Sarkar

See Also

Not all standard lattice arguments will have the intended effect, but many should. For a fuller description of possible arguments and their effects, consult documentation on lattice.

Examples

```
data(GvHD)

tf <- transformList(colnames(GvHD)[3:7], asinh)
dat <- tf %on% GvHD[[3]]

## scatter plot matrix of individual flowFrames
lattice.options(panel.error=NULL)
splom(dat)

splom(dat[,1:3], smooth = FALSE)

## displaying filters
```

```
rg <- rectangleGate("FSC-H"=c(200,400), "SSC-H"=c(300,700),
"FL1-H"=c(2,4), "FL2-A"=c(4,7))
splom(dat, filter=rg)

splom(dat, filter=rectangleGate("FSC-H"=c(400,800)))

splom(dat[,1:4], smooth = FALSE, filter=norm2Filter("FSC-H", "SSC-H", scale=1.5))
```

`timeLinePlot-methods`

Plot channel values against time

Description

Plots values of one parameter for each flowFrame in a flowSet against time.

Details

Plotting flow cytometry data against the time domain can help to identify problems with the fluidics or drifts in the instrument setting during measurement runs.

This function creates plots for all flowFrames in a flowSet for a given parameter against time. A barplot legend indicates the deviation from the median for each sample. There is also a flowFrame method, which will create a plot for a single flowFrame only.

In addition, the function computes a quality score for each frame, which essentially is the sum of the positive distances of each bin mean from a frame-specific confidence interval, divided by the number of bins. Values larger than zero indicate a problem.

Value

A numeric vector of quality scores.

Methods

General usage:

`timeLinePlot(x, channel, type=c("stacked", "scaled", "native"), col, ylab=names(x), binSize, varCut=1, ...)`

x An object of class `flowFrame` or `flowSet` containing the data to be plotted.

channel The parameter for which the data is to be plotted

type One in 'stacked', 'scaled' or 'native'. 'stacked' will plot the measurements for the frames on top of each other. 'scaled' will align the median values around zero and 'native' will plot the values in the original dimensions of the measurement range.

col Optional color parameter.

ylab The axis annotation to add on the y-axis for stacked plots.

binSize The number of events per bin. If not set, a reasonable default is computed.

varCut The cutoff in the adjusted variance to which the quality score is computed. Basically, all values that are outside of the confidence interval defined by `\[my - signma * varCut, my + sigma * varCut\]` will contribute to a positive quality score value.

... Further arguments that are passed on to the base plotting functions.

x = "ANY", channel = "missing" This casts a useful error message about missing inputs.

x = "flowFrame", channel = "character" For a `flowFrame` we only plot a single parameter against time.

x = "flowSet", channel = "character" For a `flowSet`, we plot the lines for a single parameter against time for each `flowFrame` in the set.

Author(s)

F. Hahne

See Also

`flowFrame`, `flowSet`

Examples

```
data(GvHD)
opar <- par(ask=TRUE)

res <- timeLinePlot(GvHD[[1]], "SSC-H")
res

res <- timeLinePlot(GvHD, "SSC-H")

res <- timeLinePlot(GvHD, "SSC-H", type="scaled", varCut=4)

res <- timeLinePlot(GvHD[1:4], "SSC-H", type="native", binSize=50)

par(opar)
```

`xyplot` *Methods implementing Lattice xyplots for flow data.*

Description

These functions create Trellis scatter plots (a.k.a. dot plots in the Flow Cytometry community) from flow cytometry data.

Usage

```
## Method for 'flowFrame' objects without a formula.
## This creates plots of all flow parameters agains
## time.
## S4 method for signature 'flowFrame, missing':
xyplot(
     x,
     data,
     time,
```

```
        xlab,
        ylab="",
        layout,
        prepanel=prepanel.xyplot.flowframe.time,
        panel=panel.xyplot.flowframe.time,
        type="discrete",
        ...)

## prepanel function for time line plots of flowFrames
prepanel.xyplot.flowframe.time(
        x,
        y,
        frame,
        time,
        ...)

## panel function for time line plots of flowFrames
panel.xyplot.flowframe.time(
        x,
        y,
        frame,
        time,
        type="discrete",
        nrpoints=0,
        binSize=100,
        ...)


## method for formulae with 'flowFrame' objects
## S4 method for signature 'formula, flowFrame':
xyplot(
        x,
        data,
        smooth=TRUE,
        prepanel=prepanel.xyplot.flowframe,
        panel=panel.xyplot.flowframe,
        ...)

## prepanel function for generic xyplots of flowFrames
prepanel.xyplot.flowframe(
        frame,
        channel.x.name,
        channel.y.name,
        ...)


## panel function for generic xyplots of flowFrames
panel.xyplot.flowframe(
        x,
        y,
        frame,
        filter=NULL,
```

```
        smooth=TRUE,
        margin=TRUE,
        outline=FALSE,
        channel.x.name,
        channel.y.name,
        pch=gpar$flow.symbol$pch,
        alpha=gpar$flow.symbol$alpha,
        cex=gpar$flow.symbol$cex,
        col=gpar$flow.symbol$col,
        gp,
        ...)


## method for 'flowSet' objects
## S4 method for signature 'formula, flowSet':
xyplot(
    x,
    data,
    xlab,
    ylab,
    as.table=TRUE,
    prepanel=prepanel.xyplot.flowset,
    panel=panel.xyplot.flowset,
    pch = ".",
    smooth = TRUE,
    filter = NULL,
    par.settings=NULL,
    ...)

## prepanel function for generic xyplots of flowSets
prepanel.xyplot.flowset(
    x,
    frames,
    channel.x.name,
    channel.y.name,
    ...)

## panel function for generic xyplots of flowSets
panel.xyplot.flowset(
    x,
    frames,
    filter=NULL,
    channel.x,
    channel.y,
    ...)

## method for various workflow objects
## S4 method for signature 'formula, view':
xyplot(
    x,
    data,
    ...)
```

```
## S4 method for signature 'view, missing':
xyplot(
    x,
    data,
    ...)

## S4 method for signature 'formula, gateView':
xyplot(
    x,
    data,
    filter=NULL,
    par.settings,
    ...)
```

Arguments

`x`	A formula describing the structure of the plot and the variables to be used in the display. In the `prepanel` and `panel` functions, also the names of `flowFrames` or any of the annotation data columns in the `phenoData` slot.
`data, y, frame`	a `flowSet` or `flowFrame` object that serves as the source of data. For the workflow methods, this can also be various `view` or `actionItem` objects.
`time`	A character string giving the name of the data column recording time. If not provided, we try to guess from the available parameters.
`xlab, ylab`	Labels for data axes, with suitable defaults taken from the formula.
`as.table, layout`	These arguments are passed unchanged to the corresponding methods in lattice, and are listed here only because they provide different defaults. See documentation for the original methods for details.
`type`	type of rendering; see `panel.xyplot` for details. For the basic `flowFrame` method without a detailed formula, the addtional type `discrete` is available, which plots a smoothed average of the flow cytometry values against time.
`nrpoints`	The number of points plotted on the smoothed plot in sparse regions. This is only listed here because we use a different default. See `panel.smoothScatter` for details.
`binSize`	The size of a bin (i.e., the number of events within a bin) used for the smoothed average timeline plots.
`channel.x.name, channel.y.name`	Character strings giving corresponding names used to match filter parameters if applicable.
`smooth`	Logical. If `TRUE`, `panel.smoothScatter` is used to display a partially smoothed version of the data. Otherwise, events are plotted individually, as in a standard scatter plot.
`filter`	A `filter`, `filterResult` or `filterResultList` object or a list of such objects of the same length as the `flowSet`. The appropriate spherical 2D representation of this filter will be superimposed on the plot if `smooth=TRUE`, or the resukt of the filtering operation will be indicated by grouping if `smooth=FALSE`

	. The software will figure out whether the `filter` needs to be evaluated in order to be plotted (in which case providing a `filterResult` can speed things up considerably).
`margin`	Logical indicating whether to truncate the density estimation on the margins of the measurement range and plot margin events as lines if `smooth=TRUE`. To avoid visual artifacts it is highly recommended to set this option to `TRUE`.
`outline`	Logical, specifying whether to add the boundaries of a gate to the plot when `smooth=FALSE` in addition to the grouping. Defaults to `FALSE`.
`pch, cex, col, alpha`	Graphical parameters used when `smooth=FALSE`. These mostly exist for conveniance and much more control is available throught the `lattice`-like `par.setting` and `flowViz.par.set` customization. See `flowViz.par.set` for details.
`par.settings`	A list of lists of graphical parameters. See `flowViz.par.set` for details.
`gp`	A list of graphical parameters that are passed down to the low level panel functions. This is for internal use only. The public user interface to set graphical parameters is either `par.settings` for customization of a single call or `flowViz.par.set` for customization of session-wide defaults.
`prepanel`	The prepanel function. See `xyplot`.
`panel`	The panel function. See `xyplot`.
`channel.x, channel.y`	Expressions defining the x and y variables in terms of columns in the data. Can involve functions or multiple columns from the data, however this usage is discouraged.
`frames`	An environment containing frame-specific data.
`...`	More arguments, usually passed on to the underlying lattice methods.

Details

The implementation of `xyplot` in `flowViz` is very close to the original `lattice` version. Concepts like conditioning and the use of panels apply directly to the flow cytometry data. The single fundamental difference is that conditioning variables are not evaluated in the context of the raw data, but rather in the `phenoData` slot environment (only for the `flowSet` methods. Thus, we can directly condition on pheotypic variables like sample groups, patients or treatments.

In the formula interface, the primary and secondary variables (separated by the tilde) have to be valid parameter names. Please note that frequently used variants like `FSC-H` and `SSC-H` are not syntactically correct R symbols, and need to be wrapped in ` `. E.g., `` `FSC-H` ``. For `flowSets`, the use of a conditioning variable is optional. We implicitely condition on `flowFrames` and the default is to arrange panels by sample names.

Methods

xyplot `signature(x = "flowFrame", data = "missing")`: Creates diagnostic time series plots of flow parameter values against time. These plots are useful to detect quality issues in the raw data. If not provided explicitely via the `tine` argument, the time parameter will be automatically detected. The additional arguments `xlab`, `ylab`, `nrpoints`, and `layout` are only listed because `flowViz` provides different defaults. Internally, they are directly passed on to the underlying `lattice` functions. Argument `type` can be a combination of any of the types allowed in `lattice xyplots`, or `discrete`, in which case a smoothed average of the parameter against time is plotted. `binSize` controls the binning that is used for the smoothing procedure.

xyplot `signature(x = "formula", data = "flowFrame")`: Creates scatter plots (a.k.a. dot plots) of a pair of FCM channels. Depending on the setting of the `smooth` argument, the data will be rendered as a partially smoothed density estimate (`smooth=TRUE`, the default) or as a regular scatter plot with separate points for individual events. The formula interface allows for fairly general plotting, however there are certain limitations on the use of expressions as part of the formulae. Unless you are sure about what you are doing, you should transform the raw data in a separate step using one of the tools in the `flowCore` package rather than inline using the formula interface. The method allows to superimpose gating results though the `filter` argument. If `smooth=TRUE`, we try to add spherical 2D representations of the gates if applicable. For `smooth=FALSE`, gates are indicated by a grouping mechanism using different point shapes or colors (unless `outline` is also `TRUE`, in which case the gate outlines are superimposed in addition to the grouping). Argument `margins` controls how events on the margins of the measurement range are treated. The default (`TRUE`) is to discard them from any density estimation and later add them as separate glyphs. See `flowViz.par.set` for details on controlling graphical parameters in these plots.

xyplot `signature(x = "formula", data = "flowSet")`: Scatter plots from a `flowSet` object. We allow for conditioning on variables in the `phenoData` slot of the `flowSet`. All additional arguments that apply to the `flowFrame` method are also valid for `flowSets`.

xyplot `signature(x = "formula", data = "view")`: Scatter plots from a `view` object. Depending on the particulars of the view, the method tries to come up with reasonable defaults. Full customization is also available though all the before mentioned arguments.

xyplot `signature(x = "view", data = "missing")`: Scatter plots from a `view` object. This is the fallback method in the case of absolutely no customization. It serves as the default plot method for all `views`.

xyplot `signature(x = "formula", data = "gateView")`: Scatter plots from a `gateView` object. This allows for customization of the `gateView` plots but still provides reasonable defaults, e.g., for added gates.

Author(s)

F. Hahne, D. Sarkar

See Also

Not all standard lattice arguments will have the intended effect, but many should. For a fuller description of possible arguments and their effects, consult documentation on lattice.

Examples

```
data(GvHD)
GvHD <- GvHD[pData(GvHD)$Patient %in% 5:6]

## a bivariate scatterplot
## by default ('smooth=TRUE') panel.smoothScatter is used
xyplot(`FSC-H` ~ `SSC-H`, GvHD[["s5a05"]], nbin = 100,
main="A single flowFrame")

## A non-smooth version of the same data
xyplot(`FSC-H` ~ `SSC-H`, GvHD[["s5a05"]], nbin = 100,
main="A single flowFrame", smooth=FALSE)

## Visual artifacts created by the pileup of margin events
```

```
xyplot(`FSC-H` ~ `SSC-H`, GvHD[["s5a05"]], nbin = 100,
main="A single flowFrame", margin=FALSE)

## simple bivariate scatter plot (a.k.a. dot plot)
## for the whole flowSet, conditioning on Patient and
## Visit
xyplot(`SSC-H` ~ `FSC-H` | Patient:Visit, data = GvHD)

## several examples with time on the X axis
## first for a flowFrame
xyplot(GvHD[[1]])

## and for flowSets
xyplot(`FSC-H` ~ Time | Visit, GvHD,
       smooth = FALSE, type = "l",
       subset = (Patient == 5))

xyplot(`FSC-H` ~ Time | Patient+Visit, GvHD,
       smooth = FALSE, type = "a",
       strip = FALSE, strip.left = TRUE,
       aspect = "xy")

## combine plots for two channels
ssc.time <-

    xyplot(`SSC-H` ~ Time | factor(Patient):factor(Visit), GvHD,
           smooth = FALSE, type = "a",
           strip = FALSE,
           strip.left = strip.custom(horizontal = TRUE),
           par.strip.text = list(lines = 3),
           between = list(y = rep(c(0, 0.5), c(6, 1))),
           scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
           layout = c(1, 14))

fsc.time <-
    xyplot(`FSC-H` ~ Time | factor(Patient):factor(Visit), GvHD,
           smooth = FALSE, type = "a",
           strip = FALSE,
           strip.left = strip.custom(horizontal = TRUE),
           par.strip.text = list(lines = 3),
           between = list(y = rep(c(0, 0.5), c(6, 1))),
           scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
           layout = c(1, 14))

plot(fsc.time, split = c(1, 1, 2, 1))
plot(ssc.time, split = c(2, 1, 2, 1), newpage = FALSE)

## saving plots as variables allows more manipulation
plot(update(fsc.time[8:14], layout = c(1, 7)),
     split = c(1, 1, 1, 2))

plot(update(ssc.time[8:14], layout = c(1, 7)),
     split = c(1, 2, 1, 2), newpage = FALSE)

## displaying filters
n2gate <- norm2Filter("SSC-H", "FSC-H")
```

```
xyplot(`SSC-H` ~ `FSC-H` | Patient:Visit, data = GvHD,
       filter=n2gate, subset=Patient==5)

xyplot(`SSC-H` ~ `FSC-H` | Patient:Visit,
       data=transform("SSC-H"=asinh,"FSC-H"=asinh) %on% GvHD,
       smooth=FALSE, filter=n2gate, subset=Patient==5)

n2gate.results <- filter(GvHD, n2gate)

xyplot(`SSC-H` ~ `FSC-H` | Visit, data=GvHD,
       subset=Patient == "6",
       filter=n2gate.results, smooth=FALSE)
```

Index