# Package 'flowViz'

November 12, 2026

**Type** Package

**Title** Visualization for flow cytometry

**Version** 1.76.1

**Author** B. Ellis, R. Gentleman, F. Hahne, N. Le Meur, D. Sarkar, M. Jiang

**Maintainer** Mike Jiang <mike@ozette.com>

**Description** Provides visualization tools for flow cytometry data.

**Depends** R (>= 2.7.0), flowCore(>= 1.41.9), lattice

**Imports** stats4, Biobase, flowCore, graphics, grDevices, grid, KernSmooth, lattice, latticeExtra, MASS, methods, RColorBrewer, stats, utils, hexbin,IDPmisc

**Suggests** colorspace, flowStats, knitr, rmarkdown, markdown, testthat

**License** Artistic-2.0

**biocViews** ImmunoOncology, Infrastructure, FlowCytometry, CellBasedAssays, Visualization

**VignetteBuilder** knitr

**RoxygenNote** 6.1.1

**git_url** https://git.bioconductor.org/packages/flowViz

**git_branch** RELEASE_3_23

**git_last_commit** eda6fc2

**git_last_commit_date** 2026-08-25

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-11

## Contents


flowViz-package . . . . . 2
.process_flowFrame_overlay . . . . . 3
.process_overlay_flowSet . . . . . 3
addName-methods . . . . . 4
contour-methods . . . . . 5

densityplot . . . . . 6
flowPlot . . . . . 10
flowViz.par.get . . . . . 11
glines-methods . . . . . 13
glpoints-methods . . . . . 15
glpolygon-methods . . . . . 17
gpoints-methods . . . . . 19
gpolygon-methods . . . . . 21
levelplot,formula,flowSet-method . . . . . 23
plot . . . . . 25
prepanel.ecdfplot.flowset . . . . . 26
splom . . . . . 27
timeLinePlot . . . . . 29
xyplot,flowFrame,missing-method . . . . . 31

**Index** **39**

---

`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
```

---

`.process_flowFrame_overlay` *convert a single flowFrame or a list of flowFrames to a list of flowSet*

---

## Description

convert a single flowFrame or a list of flowFrames to a list of flowSet

## Usage

```
.process_flowFrame_overlay(overlay, sn)
```

## Arguments

| | |
|---|---|
| `overlay` | flowFrame or a list of flowFrame objects |
| `sn` | sample name |

---

`.process_overlay_flowSet` *extract the respective flowFrame from each flowSet based on the given sampleName*

---

## Description

extract the respective flowFrame from each flowSet based on the given sampleName

## Usage

```
.process_overlay_flowSet(overlay, nm)
```

## Arguments

| | |
|---|---|
| `overlay` | a list of flowSet |
| `nm` | sample name |

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.

## Usage

```
## S4 method for signature 'rectangleGate,character'
addName(x, name, data, gp, pos = 0.5,
  abs = FALSE, xlim, ylim, ...)
```

## Arguments

| | |
|---|---|
| x | rectangleGate, ellipsoidGate, quadGate, polygonGate or kmeansFilter |
| name | character or logical or matrix |
| data | flowFrame |
| gp | a list of graphical parameters |
| pos, abs | specifying location of the name. see 'help(xyplot)' for more details |
| xlim, ylim | limits of axis |
| ... | other arguments |

## 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

```
## S4 method for signature 'flowFrame'
contour(x, y = 1:2, nlevels = 10, bw,
  grid.size = c(65, 65), add = FALSE, xlab, ylab, xlim, ylim,
  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. |
| `xlim, ylim` | The plotting ranges. |
| `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

```
library(flowCore)
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/histogram plots for flow data*

---

### Description

For flowSets 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 implicitly condition on parameters.

## Usage

```
## S4 method for signature 'formula,flowSet'
densityplot(x, data, ...)

prepanel.densityplot.flowset.stack(x, y, frames, overlap = 0.3,
  subscripts, ..., which.channel)

panel.densityplot.flowset.stack(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,
  margin = 0.005, stats = FALSE, pos = 0.5, digits = 2,
  abs = FALSE, fitGate = TRUE, checkName = TRUE,
  plotType = "densityplot", hist.type = "density",
  breaks = "Sturges", gp, ...)

## S4 method for signature 'formula,flowFrame'
densityplot(x, data, overlay = NULL, ...)

## S4 method for signature 'formula,view'
densityplot(x, data, ...)

## S4 method for signature 'formula,ncdfFlowSet'
densityplot(x, data, ...)

## S4 method for signature 'formula,ncdfFlowList'
densityplot(x, data, ...)

## S4 method for signature 'formula,flowSet'
histogram(x, data, plotType, ...)

## S4 method for signature 'formula,flowFrame'
histogram(x, data, ...)

## S4 method for signature 'formula,ncdfFlowSet'
histogram(x, data, ...)

## S4 method for signature 'formula,ncdfFlowList'
histogram(x, data, ...)
```

## 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 flow data object that serves as a source of data, either a flowFrame or flowSet

... — More arguments, usually passed on to the underlying lattice methods.

- channels A character vector of parameters that are supposed to be plotted when the formula in x is of structure factor ~ ..
- xlab: Label for data x axis, with suitable defaults taken from the formula
- prepanel: The prepanel function. See xyplot
- panel: the panel function. See xyplot
- axis: axis function passed to lattice, default is axis.grid
- ... : other arguments passed to panel.densityplot.flowset.stack or panel.histogram.flowframe

frames — An environment containing frame-specific data.

overlap — The amount of overlap between stacked density plots. This argument is ignored for the flowFrame method.

subscripts, which.channel, channel.name, y — Internal indices necessary to map panels to parameters.

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.

channel — The name of the currently plotted flow parameter.

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).

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, fill, lwd, alpha and 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, or a factor.

refline — Logical. Add one ore more vertical reference lines to the plot. This argument is directly passed to panel.abline.

margin — Either Logical value 'FALSE' or Numeric valuein [0,1]. When 'FALSE', it doesn't do anything to the margin events. When Numeric value, it indicates margin events by horizontal bars. The value of margin is interpreted as the proportion of events on the margin over which the bars are added. E.g., a value of 0,5 means to indicate margin events if there are more than 0.5 times the total number of events. 1 means to ignore margin events completetly. For 0 bars are added even if there is only a single margin event.

`stats`, `pos`, `digits`, `abs`
: Arguments to control statistics that is associated with `filter` to be plotted. see xyplot for details.

`fitGate`
: A `logical` scalar indicating whether to display the gate as fitted 1d density gate region or simply display the gate boundaries using vertical lines. The latter would be helpful to display the gate when the gated density region is too small to see.

`checkName`
: A `logical` scalar indicating whether to validity check the channel name. Default is TRUE, which consider '(' as invalid character in channel names

`plotType`
: either 'densityplot' or 'histogram'

`hist.type`
: see 'type' argument in 'help(panel.histogram)'

`breaks`
: see 'help(hist)'

`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.

`overlay`
: see help(xyplot).

## 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).

## Examples

```
library(flowCore)
library(flowStats)
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"))

#display the gate by its boundaries with statistics
densityplot(~ `FSC-H`, GvHD[1:2], filter=curv1Filter("FSC-H"),fitGate=FALSE,stats=TRUE)

## plot a single flowFrame
densityplot(~ `SSC-H`, GvHD[[1]], margin=FALSE)

## plot histogram
histogram(~ `SSC-H`, GvHD[[1]]) #default type is 'density'
#change the type to 'count' and adjust breaks
```

```
histogram(~ `SSC-H`, GvHD[[1]], margin=FALSE, type = "count", breaks = 50)
```

---

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

```
## 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 | A filter, filterResult or filterResultList object. |
| plotParameters | A vector of charactors defining the x and y variables in terms of columns in the data. |
| logx, logy | Logical controlling wheterh the corresponding variables will be log transfromed before passing to the panel function. Default to FALSE. |
| parent | An optional argument of class filterResult that specifies a subset of the data that are inclueed in the filterResult. |
| colParent | Specifying the color for parent. See parent above. |
| colChild | Specifies the color for child. See chile above. |
| showFilter | Logical, specifying whether to show the filter. |
| gate.fill | Specifies the fill color of the gate. Default to transparent. |
| gate.border | Character or specifying the color of the gate border. Default to black. |
| xlab, ylab | Labels for data axes. |
| xlim, ylim | Numeric vectors of length 2 specifying axis limits. |
| ... | More arguments, usually passed on to the underlying lattice methods. |

## 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

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

---

# 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.

## Usage

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

flowViz.par.set(name, value, ..., theme, warn = TRUE, strict = FALSE,
  reset = 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. |
| reset | logical scalar. When TRUE, drop the entire list of old graphical parameters and reset it with the supplied one. Default is FALSE, which updates the existing parameters. |

## Details

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.

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

Deepayan Sarker, *Lattice, Multivariate Data Visualization with R*, 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)))
library(flowCore)
data(GvHD)
fs <- GvHD[1:2]

# using default ggplot2like theme
densityplot(~`FSC-H`, fs)
xyplot(`SSC-H`~`FSC-H`, fs, smooth = FALSE)

# reset it with default lattice theme
flowViz.par.set(theme =  trellis.par.get(), reset = TRUE)
densityplot(~`FSC-H`, fs)
xyplot(`SSC-H`~`FSC-H`, fs, smooth = FALSE)
```

---

# 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.

## Usage

```
## S4 method for signature 'filter,missing'
glines(x, data, verbose = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | filter or filterResult or any derived filter class |
| data | flowFrame or filterResult or character or missing or ANY |
| verbose | logical |
| ... | other arguments |

**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`.

**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`.

**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`.

**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

## Details

When plotting `flowFrames` 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` assuming that they are defined for the respective 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.

## Author(s)

F. Hahne

## See Also

`filter`, `flowFrame`, `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`.

## Usage

```
## S4 method for signature 'filter,flowFrame,missing'
glpoints(x, data, channels,
  verbose = TRUE, filterResult = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | filter, filterResult or any derived filter class |
| `data` | flowFrame or missing |
| `channels` | character or missing |
| `verbose` | logical |
| `filterResult` | filterResult class |
| `...` | other arguments |

## Details

When plotting flowFrames using the plot method provided by flowViz, the plotted parameters are recorded, which makes it possible to correctly overlay the points within filters assuming that they are defined for the respective 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 =** We can get all the information about a filter from its filterResult without the need to re-evaluate.

**"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 curv1Filters.

**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 curv2Filters.

**x = "curv1Filter", data = "flowFrame", channels =** 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.

**"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 =** see above

**"character"** see above

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

**"character"** see above

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

**"character"** see above

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

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

**"character"** see above

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

**"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.

### Usage

```
## S4 method for signature 'filter,missing'
glpolygon(x, data, verbose = TRUE,
  gpar = flowViz.par.get(), strict = TRUE, ...)
```

### Arguments

| | |
|---|---|
| x | filter or filterResult or any derived filter class |
| data | flowFrame or filterResult or character or missing or ANY |
| verbose | logical |
| gpar | a list of graphical parameters. see 'help(flowViz.par.get)' for details. |
| strict | logical |
| ... | other arguments |

### Details

When plotting flowFrames 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 assuming that they are defined for the respective 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`.

**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`.

**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

---

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.

### Usage

```
## S4 method for signature 'filter,flowFrame,missing'
gpoints(x, data, channels,
  verbose = TRUE, filterResult = NULL, ...)
```

### Arguments

| | |
|---|---|
| x | filter, filterResult or any derived filter class |
| data | flowFrame or missing |
| channels | character or missing |
| verbose | logical |
| filterResult | filterResult class |
| ... | other arguments |

### Details

When plotting flowFrames using the plot method provided by flowViz, the plotted parameters are recorded, which makes it possible to correctly overlay the points within filters assuming that they are defined for the respective 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 =** We can get all the information about a `filter` from its `filterResult` without the need to re-evaluate.

**"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 =** 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.

**"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 =** see above

**"character"** see above

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

**"character"** see above

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

**"character"** see above

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

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

**"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.

## Usage

```
## S4 method for signature 'filter,missing'
gpolygon(x, data, verbose = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | filter or filterResult or any derived filter class |
| data | flowFrame or filterResult or character or missing or ANY |
| verbose | logical |
| ... | other arguments |

## Details

When plotting flowFrames using the plot method provided by flowViz, the plotted parameters are recorded, which makes it possible to correctly overlay the outlines of filters assuming that they are defined for the respective 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 curv1Filters.

**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`.

**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`.

**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`

levelplot,formula,flowSet-method
*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

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

## 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 'flowFrame,missing'
parallelplot(x, data, reorder.by = function(x)
  var(x, na.rm = TRUE), time = "Time", exclude.time = TRUE, ...)

## S4 method for signature 'formula,flowSet'
parallelplot(x, data, time = "Time",
  exclude.time = TRUE, filter = NULL, xlab = NULL, ylab = NULL,
  ...)
```

## 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 |
| `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. |
| `n` | the number of bins on each axis to be used when evaluating the density |
| `...` | more arguments, usually passed on to the underlying lattice methods. |

`f.value`, `distribution`
: number of points used in Q-Q plot, and the reference distribution used. See `qqmath` for details.

`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`.

`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.

`filter`
: flowCore filter

## 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.

**parallelplot** `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

```
library(flowCore)
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
```

```
parallelplot(GvHD[["s6a01"]])

## Not run:

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


## End(Not run)
```

---

plot *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.

## Usage

```
## S4 method for signature 'flowFrame,missing'
plot(x, y, smooth = TRUE, ...)

## S4 method for signature 'flowFrame,character'
plot(x, y, smooth = TRUE, pch, ...)
```

## Arguments

| | |
|---|---|
| x | flowFrame |
| y | (optional) channel names |
| smooth | logical |
| ... | other arguments |
| pch | point type |

## 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.

## Author(s)

F. Hahne

## See Also

xyplot, flowFrame, densityplot

---

prepanel.ecdfplot.flowset

*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

```
prepanel.ecdfplot.flowset(x, frames, channel, f.value, ...)

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

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

## 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. |
| frames | environment containing frame-specific data |
| channel | expression involving names of columns in the data |
| f.value | determines the number of points used in the plot ecdfplot for details. |
| ... | more arguments, usually passed on to the underlying lattice methods and the panel function. |
| ref | logical; whether to add reference lines at 0 and 1 |
| 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 |

| | |
|---|---|
| data | a flowSet object that serves as a source of data |
| xlab | Labels for data axes, with suitable defaults taken from the formula |
| panel, prepanel | the panel and prepanel functions. |
| type | type of rendering; by default lines are drawn |
| as.table | logical; whether to draw panels from top left |

## 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

```
library(flowCore)
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))
```

---

# 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, ...)
```

## Arguments

| | |
|---|---|
| x | A formula describing the structure of the plot and the variables to be used in the display. |
| data | 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.

## 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

```
library(flowCore)
data(GvHD)
library(flowStats)

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` *Plot channel values against time*

---

## Description

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

## Usage

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

## S4 method for signature 'flowSet,character'
timeLinePlot(x, channel,
  type = c("stacked", "scaled", "native", "frequency"), col = NULL,
  ylab = sampleNames(x), binSize, varCut = 1, ...)

## S4 method for signature 'flowFrame,character'
timeLinePlot(x, channel, ...)

## S4 method for signature 'ANY,missing'
timeLinePlot(x, channel, ...)
```

## Arguments

| | |
|---|---|
| `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.

## 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.

## Author(s)

F. Hahne

## See Also

flowFrame, flowSet

## Examples

```
library(flowCore)
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,flowFrame,missing-method
*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

```
## 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.xyplot.flowframe.time(x, y, frame, time, xlim, ylim, ...)

panel.xyplot.flowframe.time(x, y, frame, time, type = "discrete",
  nrpoints = 0, binSize = 100, ...)

## S4 method for signature 'formula,flowFrame'
xyplot(x, data, filter = NULL,
  overlay = NULL, stats = FALSE, strip.text = NULL, ...)

prepanel.xyplot.flowframe(frame, channel.x.name, channel.y.name, x, y,
  xlim, ylim, ...)

panel.xyplot.flowframe(frame, filter = NULL, smooth = TRUE,
  margin = TRUE, outline = FALSE, channel.x.name, channel.y.name,
  pch = gp$flow.symbol$pch, alpha = gp$flow.symbol$alpha,
  cex = gp$flow.symbol$cex, col = gp$flow.symbol$col, gp, xbins = 0,
  binTrans = sqrt, stats = FALSE, pos = 0.5, digits = 2,
  abs = FALSE, overlay = NULL, checkName = TRUE, sample.ratio = 1,
  overlay.symbol = NULL, ...)

## S4 method for signature 'formula,flowSet'
xyplot(x, data, ...)

prepanel.xyplot.flowset(x, frames, channel.x.name, channel.y.name, xlim,
  ylim, ...)

panel.xyplot.flowset(x, frames, filter = NULL, channel.x, channel.y,
  overlay = NULL, stats = FALSE, ...)

## 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,
  ...)

## S4 method for signature 'formula,ncdfFlowSet'
xyplot(x, data, ...)

## S4 method for signature 'formula,ncdfFlowList'
xyplot(x, data, ...)
```

## 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, flowFrame, ncdfFlowSet, or ncdfFlowList object that serves as the source of data. |
| 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. |
| 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. |
| prepanel | The prepanel function. See xyplot. |
| panel | The panel function. See xyplot. |
| 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. |
| ... | marker.only logical specifies whether to show both channel and marker names<br>More arguments, usually passed on to the underlying lattice methods. |
| xlim, ylim | limits for data axes. If not given, they are taken from the ranges stored in flowFrame |
| 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. |
| filter | A filter, filterResult or filterResultList object or a list of such objects of the same length as the flowSet. Also a filters or A filtersList can be passed to xyplot in order to plot multiple filters/gates(with the same x,y |

parameters) on one panel to represent multiple sub-populations. The appropriate spherical 2D representation of this filter will be superimposed on the plot if `smooth=TRUE`, or the result 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).

`overlay` The extra cell events plotted on top of the current cell population. It is a `flowSet` for `panel.xyplot.flowset` function and a `flowFrame` for `xyplot(c("formula","flowFrame"))` method.

`stats, pos, digits, abs`
Arguments to control statistics that is associated with `filter` to be plotted Currently only population proportion/percentage is supported. `stats` is a `logical` scalar indicating whether to display statistics. Default is FALSE. `pos` is the `numeric` scalar (range within c(0,1)) or vector(length of 2,first is for x-axis,second for y-axis) to control the position of the statistics label. It is set as 0.5,which is the center. `digits` is an `integer` indicating the number of significant digits to be used when displaying the percentage of population statistics,Default is 2. see more details from `format` `abs` is a `logical` scalar indicating whether the `pos` is relative to the gate boundary or the entire xy-axis(absolute position). By default it is set as FALSE,which indicates the position is relative to gate.

`strip.text` A `character` that customizes the text in strip. Default is NULL, which does not display the strip box at all. It is only valid when plotting a `flowFrame`

`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. If `FALSE`, a graphical parameter `colramp` can be used to obtain a coloring of points that is indicative of their local density.

`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 convenience and much more control is available throught the `lattice`-like `par.setting` and `flowViz.par.set` customization. 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.

`xbins` The argument passed to `hexbin` ,which is the number of bins partitioning the range of xbnds. It is set as 0 by default,which plots all the events without binning. When it is larger than 0,hexbin plot engine is used for the faster plotting. Note that it is only valid when `smooth` is set as FALSE .

| | |
|---|---|
| binTrans | The argument passed to grid.hexagons ,which is a transformation function (or NULL) for the count. It is sqrt by default. |
| checkName | logical indicating whether to skip checking the bracket '(' in channel name |
| sample.ratio | numeric the ratio of sub-sampling of events to speed up plotting. |
| overlay.symbol | list of the lattice graphic parameters to format the overlay points. |
| frames | An environment containing frame-specific data. |
| 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. |
| par.settings | A list of lists of graphical parameters. See flowViz.par.set for details. |

## 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. Argument `par.settings` can be used to supply lists of graphical parameters. 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`.

## 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

```
library(flowCore)
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)

## A non-smooth version of the same data with customerized color scheme
require(IDPmisc)
colramp <- colorRampPalette(IDPcolorRamp(21))
xyplot(`FSC-H` ~ `SSC-H`, GvHD[["s5a05"]], nbin = 100,
       main="A single flowFrame", smooth=FALSE,
       colramp=colramp, pch=20, cex=0.1)

## A hexbin version of non-smooth scatter plot
xyplot(`FSC-H` ~ `SSC-H`, GvHD[["s5a05"]], xbin = 128
       ,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)

## Same bivariate scatter plot with replacing default color
require(IDPmisc)
cols <- colorRampPalette(IDPcolorRamp(21))
xyplot(`SSC-H` ~ `FSC-H` | Patient:Visit, data = GvHD, colramp=cols)

## 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), xbin = 32)

xyplot(`FSC-H` ~ Time | Patient+Visit, GvHD,
       smooth = FALSE, type = "a",
       strip = FALSE, strip.left = TRUE,
       aspect = "xy", xbin = 32)


## 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), xbin = 32)

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), xbin = 32)

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
library(flowStats)
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, xbin = 32)


## displaying filters with stats
n2gate.results <- filter(GvHD, n2gate)

xyplot(`SSC-H` ~ `FSC-H` | Visit, data=GvHD,
       subset=Patient == "6",
       filter=n2gate.results, smooth=FALSE, xbin = 32
       ,stats=TRUE
       ,abs=TRUE
       ,digits=3
       )


## displaying multiple filters in one panel with stats
recGate1<-rectangleGate("FL3-H"=c(2.3,4.1),"FL2-H"=c(6.8,9))
recGate2<-rectangleGate("FL3-H"=c(1,3),"FL2-H"=c(4,6))
filters1<-filters(list(recGate1,recGate2))
trans<-transform("FL2-H"=asinh,"FL3-H"=asinh)
trans_data<-transform(GvHD[1:2],trans)
#replicate filters object across samples
flist <- list(filters1 , filters1)
names(flist) <- sampleNames(trans_data)
xyplot(`FL2-H` ~ `FL3-H`
    ,data=trans_data
       ,filter= flist
       ,stats=TRUE
       ,margin=FALSE
       , xbin = 32
       , smooth = FALSE
       )

#display recGate2 as a overlay
overlay <- Subset(trans_data,recGate1)
xyplot(`FL2-H` ~ `FL3-H`
    ,data=trans_data
       ,filter=recGate2
       ,stats=TRUE
```

```
,margin=FALSE
, smooth = FALSE
, xbin = 32
,overlay= list(rect2 = overlay)
,par.settings = list(overlay.symbol = list(cex = 0.1))
)
```

# Index


∗ **dplot**
levelplot,formula,flowSet-method, 23
prepanel.ecdfplot.flowset, 26
splom, 27
timeLinePlot, 29
xyplot,flowFrame,missing-method, 31
∗ **methods**
addName-methods, 4
contour-methods, 5
flowPlot, 10
glines-methods, 13
glpoints-methods, 15
glpolygon-methods, 17
gpoints-methods, 19
gpolygon-methods, 21
levelplot,formula,flowSet-method, 23
plot, 25
prepanel.ecdfplot.flowset, 26
splom, 27
timeLinePlot, 29
xyplot,flowFrame,missing-method, 31
∗ **package**
flowViz-package, 2
.process_flowFrame_overlay, 3
.process_overlay_flowSet, 3

addName *(addName-methods)*, 4
addName,curv1Filter,character-method *(addName-methods)*, 4
addName,curv1Filter,logical-method *(addName-methods)*, 4
addName,curv2Filter,character-method *(addName-methods)*, 4
addName,curv2Filter,logical-method *(addName-methods)*, 4
addName,ellipsoidGate,character-method *(addName-methods)*, 4
addName,ellipsoidGate,logical-method *(addName-methods)*, 4
addName,kmeansFilter,character-method *(addName-methods)*, 4
addName,kmeansFilter,logical-method *(addName-methods)*, 4
addName,polygonGate,character-method *(addName-methods)*, 4
addName,polygonGate,logical-method *(addName-methods)*, 4
addName,quadGate,character-method *(addName-methods)*, 4
addName,quadGate,logical-method *(addName-methods)*, 4
addName,quadGate,matrix-method *(addName-methods)*, 4
addName,rectangleGate,character-method *(addName-methods)*, 4
addName,rectangleGate,logical-method *(addName-methods)*, 4
addName-methods, 4

bkde2D, *5*, *6*

contour, *5*, *6*
contour *(contour-methods)*, 5
contour,ANY-method *(contour-methods)*, 5
contour,flowFrame-method *(contour-methods)*, 5
contour,flowSet-method *(contour-methods)*, 5
contour-methods, 5
curv1Filter, *14*, *16*, *18*, *20*, *21*
curv2Filter, *14*, *16*, *18*, *20*, *22*

density, *8*
densityplot, 6, *26*

densityplot,formula,flowFrame-method
(densityplot), 6
densityplot,formula,flowSet-method
(densityplot), 6
densityplot,formula,ncdfFlowList-method
(densityplot), 6
densityplot,formula,ncdfFlowSet-method
(densityplot), 6
densityplot,formula,view-method
(densityplot), 6

ecdfplot, *26*
ecdfplot (prepanel.ecdfplot.flowset), 26
ecdfplot,formula,flowSet-method
(prepanel.ecdfplot.flowset), 26
ellipsoidGate, *22*

filter, *4*, *8*, *9*, *13–22*, *32*, *33*
filterResult, *4*, *8*, *14*, *16*, *18*, *20–22*, *32*
filterResultList, *8*, *32*
filters, *32*
filtersList, *32*
flowCore, *2*, *11*, *34*
flowFrame, *5*, *6*, *8*, *14–22*, *25*, *26*, *28–30*, *32*
flowFrames, *15*, *17*, *32*
flowPlot, 10
flowPlot,flowFrame-method (flowPlot), 10
flowSet, *5*, *6*, *8*, *29*, *30*
flowSets, *6*
flowViz (flowViz-package), 2
flowViz-package, 2
flowViz.par.get, 11
flowViz.par.set, *8*, *33–35*
flowViz.par.set (flowViz.par.get), 11
format, *33*

glines, *20*, *22*
glines (glines-methods), 13
glines,curv1Filter,ANY-method
(glines-methods), 13
glines,curv1Filter,flowFrame-method
(glines-methods), 13
glines,curv1Filter,missing-method
(glines-methods), 13
glines,curv1Filter,multipleFilterResult-method
(glines-methods), 13
glines,curv2Filter,ANY-method
(glines-methods), 13
glines,curv2Filter,flowFrame-method
(glines-methods), 13
glines,curv2Filter,multipleFilterResult-method
(glines-methods), 13
glines,ellipsoidGate,character-method
(glines-methods), 13
glines,ellipsoidGate,filterResult-method
(glines-methods), 13
glines,ellipsoidGate,flowFrame-method
(glines-methods), 13
glines,filter,missing-method
(glines-methods), 13
glines,filterResult,ANY-method
(glines-methods), 13
glines,filterResult,flowFrame-method
(glines-methods), 13
glines,kmeansFilter,ANY-method
(glines-methods), 13
glines,norm2Filter,ANY-method
(glines-methods), 13
glines,norm2Filter,flowFrame-method
(glines-methods), 13
glines,norm2Filter,flowStats-method
(glines-methods), 13
glines,norm2Filter,logicalFilterResult-method
(glines-methods), 13
glines,polygonGate,character-method
(glines-methods), 13
glines,polygonGate,filterResult-method
(glines-methods), 13
glines,polygonGate,flowFrame-method
(glines-methods), 13
glines,quadGate,character-method
(glines-methods), 13
glines,quadGate,filterResult-method
(glines-methods), 13
glines,quadGate,flowFrame-method
(glines-methods), 13
glines,rectangleGate,character-method
(glines-methods), 13
glines,rectangleGate,filterResult-method
(glines-methods), 13
glines,rectangleGate,flowFrame-method
(glines-methods), 13
glines-methods, 13
glpoints, *19*
glpoints (glpoints-methods), 15
glpoints,curv1Filter,flowFrame,character-method

(glpoints-methods), 15
glpoints,curv2Filter,flowFrame,character-method (glpoints-methods), 15
glpoints,ellipsoidGate,flowFrame,character-method (glpoints-methods), 15
glpoints,filter,flowFrame,missing-method (glpoints-methods), 15
glpoints,filter,missing,ANY-method (glpoints-methods), 15
glpoints,filterResult,flowFrame,ANY-method (glpoints-methods), 15
glpoints,filterResult,flowFrame,character-method (glpoints-methods), 15
glpoints,kmeansFilter,flowFrame,character-method (glpoints-methods), 15
glpoints,norm2Filter,flowFrame,character-method (glpoints-methods), 15
glpoints,polygonGate,flowFrame,character-method (glpoints-methods), 15
glpoints,quadGate,flowFrame,character-method (glpoints-methods), 15
glpoints,rectangleGate,flowFrame,character-method (glpoints-methods), 15
glpoints,subsetFilter,flowFrame,ANY-method (glpoints-methods), 15
glpoints-methods, 15
glpolygon, *17*
glpolygon (glpolygon-methods), 17
glpolygon,complementFilter,ANY-method (glpolygon-methods), 17
glpolygon,curv1Filter,ANY-method (glpolygon-methods), 17
glpolygon,curv1Filter,flowFrame-method (glpolygon-methods), 17
glpolygon,curv1Filter,missing-method (glpolygon-methods), 17
glpolygon,curv1Filter,multipleFilterResult-method (glpolygon-methods), 17
glpolygon,curv2Filter,ANY-method (glpolygon-methods), 17
glpolygon,curv2Filter,flowFrame-method (glpolygon-methods), 17
glpolygon,curv2Filter,multipleFilterResult-method (glpolygon-methods), 17
glpolygon,ellipsoidGate,character-method (glpolygon-methods), 17
glpolygon,ellipsoidGate,filterResult-method (glpolygon-methods), 17
glpolygon,ellipsoidGate,flowFrame-method (glpolygon-methods), 17
glpolygon,filter,missing-method (glpolygon-methods), 17
glpolygon,filterResult,ANY-method (glpolygon-methods), 17
glpolygon,filterResult,flowFrame-method (glpolygon-methods), 17
glpolygon,filterResult,missing-method (glpolygon-methods), 17
glpolygon,kmeansFilter,ANY-method (glpolygon-methods), 17
glpolygon,norm2Filter,ANY-method (glpolygon-methods), 17
glpolygon,norm2Filter,flowFrame-method (glpolygon-methods), 17
glpolygon,norm2Filter,logicalFilterResult-method (glpolygon-methods), 17
glpolygon,polygonGate,character-method (glpolygon-methods), 17
glpolygon,polygonGate,filterResult-method (glpolygon-methods), 17
glpolygon,polygonGate,flowFrame-method (glpolygon-methods), 17
glpolygon,quadGate,character-method (glpolygon-methods), 17
glpolygon,quadGate,filterResult-method (glpolygon-methods), 17
glpolygon,quadGate,flowFrame-method (glpolygon-methods), 17
glpolygon,rectangleGate,character-method (glpolygon-methods), 17
glpolygon,rectangleGate,filterResult-method (glpolygon-methods), 17
glpolygon,rectangleGate,flowFrame-method (glpolygon-methods), 17
glpolygon,subsetFilter,ANY-method (glpolygon-methods), 17
glpolygon-methods, 17
gpoints, *15*, *22*
gpoints (gpoints-methods), 19
gpoints,curv1Filter,flowFrame,character-method (gpoints-methods), 19
gpoints,curv2Filter,flowFrame,character-method (gpoints-methods), 19
gpoints,filter,flowFrame,missing-method (gpoints-methods), 19
gpoints,filter,missing,ANY-method

(gpoints-methods), 19
gpoints,filterResult,flowFrame,character-method (gpoints-methods), 19
gpoints,kmeansFilter,flowFrame,character-method (gpoints-methods), 19
gpoints,norm2Filter,flowFrame,character-method (gpoints-methods), 19
gpoints,polygonGate,flowFrame,character-method (gpoints-methods), 19
gpoints,quadGate,flowFrame,character-method (gpoints-methods), 19
gpoints,rectangleGate,flowFrame,character-method (gpoints-methods), 19
gpoints-methods, 19
gpolygon, *20*
gpolygon (gpolygon-methods), 21
gpolygon,curv1Filter,ANY-method (gpolygon-methods), 21
gpolygon,curv1Filter,flowFrame-method (gpolygon-methods), 21
gpolygon,curv1Filter,missing-method (gpolygon-methods), 21
gpolygon,curv1Filter,multipleFilterResult-method (gpolygon-methods), 21
gpolygon,curv2Filter,ANY-method (gpolygon-methods), 21
gpolygon,curv2Filter,flowFrame-method (gpolygon-methods), 21
gpolygon,curv2Filter,multipleFilterResult-method (gpolygon-methods), 21
gpolygon,ellipsoidGate,character-method (gpolygon-methods), 21
gpolygon,ellipsoidGate,filterResult-method (gpolygon-methods), 21
gpolygon,ellipsoidGate,flowFrame-method (gpolygon-methods), 21
gpolygon,filter,missing-method (gpolygon-methods), 21
gpolygon,filterResult,ANY-method (gpolygon-methods), 21
gpolygon,filterResult,flowFrame-method (gpolygon-methods), 21
gpolygon,kmeansFilter,ANY-method (gpolygon-methods), 21
gpolygon,norm2Filter,ANY-method (gpolygon-methods), 21
gpolygon,norm2Filter,flowFrame-method (gpolygon-methods), 21
gpolygon,norm2Filter,logicalFilterResult-method (gpolygon-methods), 21
gpolygon,polygonGate,character-method (gpolygon-methods), 21
gpolygon,polygonGate,filterResult-method (gpolygon-methods), 21
gpolygon,polygonGate,flowFrame-method (gpolygon-methods), 21
gpolygon,quadGate,character-method (gpolygon-methods), 21
gpolygon,quadGate,filterResult-method (gpolygon-methods), 21
gpolygon,quadGate,flowFrame-method (gpolygon-methods), 21
gpolygon,rectangleGate,character-method (gpolygon-methods), 21
gpolygon,rectangleGate,filterResult-method (gpolygon-methods), 21
gpolygon,rectangleGate,flowFrame-method (gpolygon-methods), 21
gpolygon-methods, 21
grid.hexagons, *34*

hexbin, *33*
histogram, *25*
histogram,formula,flowFrame-method (densityplot), 6
histogram,formula,flowSet-method (densityplot), 6
histogram,formula,ncdfFlowList-method (densityplot), 6
histogram,formula,ncdfFlowSet-method (densityplot), 6

kde2d, *24*
kmeansFilter, *14*, *18*, *22*

lattice-methods (levelplot,formula,flowSet-method), 23
levelplot,formula,flowSet-method, 23
lines, *13*
lpoints, *15*
lpolygon, *17*

norm2Filter, *14*, *18*, *22*

panel.abline, *8*
panel.densityplot.flowset.stack (densityplot), 6

panel.ecdfplot.flowset
(prepanel.ecdfplot.flowset), 26
panel.smoothScatter, *32*
panel.splom.flowframe (splom), 27
panel.xyplot, *32*
panel.xyplot.flowframe
(xyplot,flowFrame,missing-method), 31
panel.xyplot.flowset
(xyplot,flowFrame,missing-method), 31
parallelplot,flowFrame,missing-method
(levelplot,formula,flowSet-method), 23
parallelplot,formula,flowSet-method
(levelplot,formula,flowSet-method), 23
pdf, *24*
plot, 25
plot,flowFrame,character-method (plot), 25
plot,flowFrame,missing-method (plot), 25
points, *19*
polygon, *21*
polygonGate, *14*, *18*, *22*
prepanel.densityplot.flowset.stack
(densityplot), 6
prepanel.ecdfplot.flowset, 26
prepanel.xyplot.flowframe
(xyplot,flowFrame,missing-method), 31
prepanel.xyplot.flowset
(xyplot,flowFrame,missing-method), 31

qqmath, *24*
qqmath,formula,flowSet-method
(levelplot,formula,flowSet-method), 23
quadGate, *14*, *18*, *22*

rectangleGate, *14*, *18*, *22*

smoothScatter, *25*
splom, *25*, 27
splom,flowFrame,missing-method (splom), 27
sqrt, *34*

timeLinePlot, 29
timelineplot (timeLinePlot), 29
timeLinePlot,ANY,missing-method
(timeLinePlot), 29
timeLinePlot,flowFrame,character-method
(timeLinePlot), 29
timeLinePlot,flowSet,character-method
(timeLinePlot), 29
trellis.par.get, *11–13*
trellis.par.set, *11–13*

xyplot, *8*, *9*, *26*, *32*
xyplot
(xyplot,flowFrame,missing-method), 31
xyplot,flowFrame,missing-method, 31
xyplot,formula,flowFrame-method
(xyplot,flowFrame,missing-method), 31
xyplot,formula,flowSet-method
(xyplot,flowFrame,missing-method), 31
xyplot,formula,gateView-method
(xyplot,flowFrame,missing-method), 31
xyplot,formula,ncdfFlowList-method
(xyplot,flowFrame,missing-method), 31
xyplot,formula,ncdfFlowSet-method
(xyplot,flowFrame,missing-method), 31
xyplot,formula,view-method
(xyplot,flowFrame,missing-method), 31
xyplot,view,missing-method
(xyplot,flowFrame,missing-method), 31
xyplots, *28*