# Package: flowStats (via r-universe)

November 24, 2026

**Type** Package

**Title** Statistical methods for the analysis of flow cytometry data

**Version** 4.26.0

**Author** Florian Hahne, Nishant Gopalakrishnan, Alireza Hadj Khodabakhshi, Chao-Jen Wong, Kyongryun Lee

**Maintainer** Greg Finak <greg@ozette.com>, Mike Jiang <mike@ozette.com>

**Description** Methods and functionality to analyse flow data that is beyond the basic infrastructure provided by the flowCore package.

**Suggests** xtable, testthat, openCyto, ggcyto, ggridges

**Encoding** UTF-8

**Depends** R (>= 3.0.2)

**Imports** BiocGenerics, MASS, flowCore (>= 1.99.6), flowWorkspace, ncdfFlow(>= 2.19.5), flowViz, fda (>= 2.2.6), Biobase, methods, grDevices, graphics, stats, cluster, utils, KernSmooth, lattice, ks, RColorBrewer, rrcov, corpcor, mnormt, clue

**Enhances** RBGL,graph

**License** Artistic-2.0

**Lazyload** yes

**URL** http://www.github.com/RGLab/flowStats

**BugReports** http://www.github.com/RGLab/flowStats/issues

**biocViews** ImmunoOncology, FlowCytometry, CellBasedAssays

**RoxygenNote** 7.3.0

**Config/pak/sysreqs** libgmp3-dev libjpeg-dev libpng-dev libxml2-dev libssl-dev zlib1g-dev

**Repository** Bioconductor 3.24

**Date/Publication** 2026-11-23 22:29:44 UTC

**RemoteUrl** https://github.com/bioc/flowStats

**RemoteRef** RELEASE_3_24

**RemoteSha** 68b17a57a2e1a60ded343495c344d5711a0c4d20

# Contents

flowStats-package . . . 3
addName,curv1Filter,character-method . . . 3
autoGate . . . 4
BackGating . . . 5
binByRef . . . 5
calcPBChiSquare . . . 6
calcPearsonChi . . . 7
curv1Filter-class . . . 8
curv2Filter-class . . . 10
curvPeaks . . . 12
density1d . . . 13
ellipse . . . 16
fdPar . . . 16
flowClust2Prior . . . 17
gate_singlet . . . 17
gaussNorm . . . 19
gpaSet . . . 20
idFeaturesByBackgating . . . 23
iProcrustes . . . 25
ITN . . . 27
landmarkMatrix . . . 28
lymphFilter-class . . . 29
mkPrior . . . 31
norm2Filter-class . . . 32
normalize-methods . . . 35
normQA . . . 36
overton_like . . . 37
plotBins . . . 38
plotPrior . . . 39
proBin . . . 40
quadrantGate . . . 42
rangeGate . . . 43
SimulateMixture . . . 46
spillover-flowSet . . . 47
spillover_match-flowSet . . . 49
spillover_ng-flowSet . . . 50
warpSet . . . 52

**Index** **55**

`flowStats-package` *Statistical methods for flow cytometry data analysis*

## Description

Functions, methods and classes implementing algorithmns for statistical analysis of flow cytometry data. This involves mostly data normalization and automated gating.

## Details

| | |
|---|---|
| Package: | flowStats |
| Type: | Package |
| Version: | 1.0 |
| License: | Artistic-2.0 |
| Lazyload: | yes |

## Author(s)

Florian Hahne

Maintainers: Mike Jiang <mike@ozette.ai>, Jake Wagner <jpwagner@fhcrc.org>

`addName,curv1Filter,character-method`
*These methods are copied from flowViz to eliminate its dependency on curv1Filter and curv2Filter*

## Description

These methods are copied from flowViz to eliminate its dependency on curv1Filter and curv2Filter

## Usage

```
## S4 method for signature 'curv1Filter,character'
addName(x, name, data, ...)

## S4 method for signature 'curv1Filter,logical'
addName(x, name, data, ...)

## S4 method for signature 'curv2Filter,character'
addName(x, name, data, ...)

## S4 method for signature 'curv2Filter,logical'
addName(x, name, data, ...)
```

### Arguments

| | |
|---|---|
| x | curv1Filter, curv2Filter |
| name | character or logical. Names can be generated by the filter or by the user. |
| data | flowFrame |
| ... | other arguments |

### Value

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

---

## autoGate *Automated gating of single populations in 2D*

---

### Description

This function tries to fit a single norm2Filter based on a rough preselection of the data. This function is considered internal. Please use the API provided by lymphGate.

### Usage

```
autoGate(x, ..., scale = 2.5)
```

### Arguments

| | |
|---|---|
| x | An object of class flowSet |
| ... | Named arguments or a list of the ranges used for the initial rough preselection. This gets passed on to rectangleGate, see it's documentation for details. |
| scale | The scale parameter that gets passed on to norm2Filter. |

### Details

The flowSet is first filtered using a rectangleGate and the norm2Filter is subsequently fitted to the remaining subset.

### Value

A list with items:

| | |
|---|---|
| x | The filtered flowSet. |
| n2gate | The norm2Filter object. |
| n2gateResults | The filterResult after applying the norm2Filter on the flowSet. |

### Author(s)

Florian Hahne

## See Also

lymphGate, norm2Filter

## Examples

```
library(flowCore)
data(GvHD)
flowStats:::autoGate(GvHD[10:15], "FSC-H"=c(100,500), "SSC-H"=c(0, 400))
```

---

BackGating *Sample backgating results*

---

## Description

A data frame containing the sub-populations of ITN dataset corresponding to the high-density areas on "FSC" and "SSC" channels. This dataset is yielded by backGating on channel CD3, CD8, and CD4 of the ITN sample data.

## Usage

```
data(BackGating)
```

## Source

Results from executing the following code:

library(flowCore) data(ITN)

flowStats:::backGating(ITN, xy=c("FSC", "SSC"), channels=c("CD3", "CD8", "CD4"))

---

binByRef *Bin a test data set using bins previously created by probability binning a control dataset*

---

## Description

The bins generated by probability binning a control data set can be applied to a test data set to perfom statistical comparisions by methods such as the Chi-squared test or the probability binning statistic.

## Usage

```
binByRef(binRes, data)
```

### Arguments

| | |
|---|---|
| binRes | The result generated by calling teh probBin function on a control dataset. |
| data | An object of class flowFrame |

### Value

An enviroment containing the matrices for each bin of the test data set

### Author(s)

Nishant Gopalakrishnan

### See Also

plotBins, proBin

### Examples

```
library(flowCore)
data(GvHD)
resCtrl<-proBin(GvHD[[1]],200)
resSample<-binByRef(resCtrl,GvHD[[2]])
ls(resSample)
```

---

## calcPBChiSquare *Probability binning metirc for comparing the probability binned datasets*

---

### Description

This function calculates the Probability binning metric proposed by Baggerly et al. The function utilizes the data binned using the proBin and binByRef functions.

### Usage

```
calcPBChiSquare(ctrlRes,sampRes,ctrlCount,sampCount)
```

### Arguments

| | |
|---|---|
| ctrlRes | The result generated by calling the probBin function on a control dataset. |
| sampRes | The result generated by calling the byByRef function on a test sample dataset |
| ctrlCount | The number of events in the control sample |
| sampCount | The number of events in the test sample being compared |

### Value

A list containing the statistic, p.value, observed, expected counts and the residuals

### Author(s)

Nishant Gopalakrishnan

### See Also

proBin, calcPBChiSquare

### Examples

```
library(flowCore)
data(GvHD)
# flow frame 1 is treated as  control dataset and used to generate bins
resCtrl<-proBin(GvHD[[1]][,c("FSC-H","SSC-H","Time")],200)
plotBins(resCtrl,GvHD[[1]],channels=c("FSC-H","SSC-H","Time"),title="Binned control data")
# Same bins are applied to flowFrame 16
resSample<-binByRef(resCtrl,GvHD[[16]][,c("FSC-H","SSC-H","Time")])
ctrlCount<-nrow(GvHD[[1]])
sampCount<-nrow(GvHD[[16]])
stat<-calcPBChiSquare(resCtrl,resSample,ctrlCount,sampCount)
```

---

## calcPearsonChi *Pearsons chi-square statistic for comparing the probability binned datasets*

---

### Description

This function calculates the Pearsons chi-squared statistic for comparing data binned using the proBin and binByRef functions.Internally, the function utilizes the chisq.test function.

### Usage

```
calcPearsonChi(ctrlRes,sampRes)
```

### Arguments

| | |
|---|---|
| ctrlRes | The result generated by calling the probBin function on a control dataset. |
| sampRes | The result generated by calling the byByRef function on a sample dataset |

### Value

A list containing the statistic, p.value, observed, expected counts and the residuals

### Author(s)

Nishant Gopalakrishnan

### See Also

proBin, calcPBChiSquare

## Examples

```
library(flowCore)
data(GvHD)
# flow frame 1 is treated as  control dataset and used to generate bins
resCtrl<-proBin(GvHD[[1]][,c("FSC-H","SSC-H","Time")],200)
plotBins(resCtrl,GvHD[[1]],channels=c("FSC-H","SSC-H","Time"),title="Binned control data")
# Same bins are applied to flowFrame 16
resSample<-binByRef(resCtrl,GvHD[[16]][,c("FSC-H","SSC-H","Time")])
stat<-calcPearsonChi(resCtrl,resSample)
```

---

# curv1Filter-class *Class "curv1Filter"*

---

## Description

Class and constructor for data-driven `filter` objects that selects high-density regions in one dimension.

## Usage

```
curv1Filter(x, bwFac=1.2, gridsize=rep(401, 2),
filterId="defaultCurv1Filter")
```

## Arguments

| | |
|---|---|
| `x` | Character giving the name of the measurement parameter on which the filter is supposed to work on. This can also be a list containing a single character scalar for programmatic access. |
| `filterId` | An optional parameter that sets the `filterId` slot of this filter. The object can later be identified by this name. |
| `bwFac, gridsize` | Numerics of length 1 and 2, respectively, used to set the `bwFac` and `gridsize` slots of the object. |

## Details

Areas of high local density in one dimensions are identified by detecting significant curvature regions. See *Duong, T. and Cowling, A. and Koch, I. and Wand, M.P., Computational Statistics and Data Analysis 52/9, 2008* for details. The constructor `curv1Filter` is a convenience function for object instantiation. Evaluating a `curv1Filter` results in potentially multiple sub-populations, an hence in an object of class `multipleFilterResult`. Accordingly, `curv1Filters` can be used to split flow cytometry data sets.

## Value

Returns a `curv1Filter` object for use in filtering `flowFrame`s or other flow cytometry objects.

## Extends

Class "parameterFilter", directly.

Class "concreteFilter", by class parameterFilter, distance 2.

Class "filter", by class parameterFilter, distance 3.

## Slots

**bwFac:** Object of class "numeric". The bandwidth factor used for smoothing of the density estimate.

**gridsize:** Object of class "numeric". The size of the bins used for density estimation.

**parameters:** Object of class "character", describing the parameter used to filter the flowFrame.

**filterId:** Object of class "character", referencing the filter.

## Objects from the Class

Objects can be created by calls of the form new("curvFilter", ...) or using the constructor curv1Filter. Using the constructor is the recommended way of object instantiation:

## Methods

**%in%** signature(x = "flowFrame", table = "curv1Filter"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the filter methods.

**show** signature(object = "curv1Filter"): Print information about the filter.

## Note

See the documentation in the flowViz package for plotting of curv1Filters.

## Author(s)

Florian Hahne

## See Also

curv2Filter, flowFrame, flowSet, filter for evaluation of curv1Filters and split for splitting of flow cytometry data sets based on that.

## Examples

```
library(flowStats)
library(flowCore)
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08",
package="flowCore"))

## Create directly. Most likely from a command line
curv1Filter("FSC-H", filterId="myCurv1Filter", bwFac=2)
```

```
## To facilitate programmatic construction we also have the following
c1f <- curv1Filter(filterId="myCurv1Filter", x=list("FSC-H"), bwFac=2)

## Filtering using curv1Filter
fres <- filter(dat, c1f)
fres
summary(fres)
names(fres)

## The result of curv1 filtering are multiple sub-populations
## and we can split our data set accordingly
split(dat, fres)

## We can limit the splitting to one or several sub-populations
split(dat, fres, population="rest")
split(dat, fres, population=list(keep=c("peak 2", "peak 3")))
```

---

`curv2Filter-class` *Class "curv2Filter"*

---

## Description

Class and constructor for data-driven `filter` objects that selects high-density regions in two dimensions.

## Usage

```
curv2Filter(x, y, filterId="defaultCurv2Filter", bwFac=1.2,
gridsize=rep(151, 2))
```

## Arguments

| | |
|---|---|
| `x`, `y` | Characters giving the names of the measurement parameter on which the filter is supposed to work on. `y` can be missing in which case `x` is expected to be a character vector of length 2 or a list of characters. |
| `filterId` | An optional parameter that sets the `filterId` slot of this filter. The object can later be identified by this name. |
| `bwFac`, `gridsize` | Numerics of length 1 and 2, respectively, used to set the `bwFac` and `gridsize` slots of the object. |

## Details

Areas of high local density in two dimensions are identified by detecting significant curvature regions. See *Duong, T. and Cowling, A. and Koch, I. and Wand, M.P., Computational Statistics and Data Analysis 52/9, 2008* for details. The constructor `curv2Filter` is a convenience function for

object instantiation. Evaluating a curv2Filter results in potentially multiple sub-populations, an hence in an object of class multipleFilterResult. Accordingly, curv2Filters can be used to split flow cytometry data sets.

## Value

Returns a curv2Filter object for use in filtering flowFrames or other flow cytometry objects.

## Extends

Class "parameterFilter", directly.

Class "concreteFilter", by class parameterFilter, distance 2.

Class "filter", by class parameterFilter, distance 3.

## Slots

**bwFac:** Object of class "numeric". The bandwidth factor used for smoothing of the density estimate.

**gridsize:** Object of class "numeric". The size of the bins used for density estimation.

**parameters:** Object of class "character", describing the parameters used to filter the flowFrame.

**filterId:** Object of class "character", referencing the filter.

## Objects from the Class

Objects can be created by calls of the form new("curv2Filter", ...) or using the constructor curv2Filter. The constructor is the recommended way of object instantiation:

## Methods

**%in%** signature(x = "flowFrame", table = "curv2Filter"): The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the filter methods.

**show** signature(object = "curv2Filter"): Print information about the filter.

## Note

See the documentation in the flowViz package for plotting of curv2Filters.

## Author(s)

Florian Hahne

## See Also

curv1Filter, flowFrame, flowSet, filter for evaluation of curv2Filters and split for splitting of flow cytometry data sets based on that.

## Examples

```
library(flowCore)
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08",
package="flowCore"))

## Create directly. Most likely from a command line
curv2Filter("FSC-H", "SSC-H", filterId="myCurv2Filter")

## To facilitate programmatic construction we also have the following
c2f <- curv2Filter(filterId="myCurv2Filter", x=list("FSC-H", "SSC-H"),
bwFac=2)
c2f <- curv2Filter(filterId="myCurv2Filter", x=c("FSC-H", "SSC-H"),
bwFac=2)

## Filtering using curv2Filter
fres <- filter(dat, c2f)
fres
summary(fres)
names(fres)

## The result of curv2 filtering are multiple sub-populations
## and we can split our data set accordingly
split(dat, fres)

## We can limit the splitting to one or several sub-populations
split(dat, fres, population="rest")
split(dat, fres, population=list(keep=c("area 2", "area 3")))


curv2Filter("FSC-H", "SSC-H", filterId="test filter")
```

---

# curvPeaks *Parse curv1Filter output*

---

## Description

Parse the output of curv1Filter and find modes and midpoints of the high-density regions. This function is considered to be internal.

## Usage

```
curvPeaks(x, dat, borderQuant = 0.01, n = 201, from, to, densities=NULL)
```

### Arguments

| | |
|---|---|
| x | A multipleFilterResult produced by a curv1Filter operation. |
| dat | The corresponding flowFrame. |
| borderQuant | A numeric in [0,1] giving the extreme quantiles for which high-density regions are ignored. |
| n, from, to | Arguments are passed on to density. |
| densities | The optional y values of the density estimate computed for the respective data. |

### Value

A list with items

| | |
|---|---|
| peaks | x and y locations of the modes of the regions in the density estimates. |
| regions | the left and right margins of the regions. |
| midpoints | the mean of regions. |
| regPoints | x and y locations of the outline of the significant density regions. |
| densFuns | an approximation function of the density estimate |

### Author(s)

Florian Hahne

### See Also

landmarkMatrix

### Examples

```
library(flowCore)
data(GvHD)
tmp <- filter(GvHD[[10]], curv1Filter("FSC-H"))
res <-  flowStats:::curvPeaks(tmp, exprs(GvHD[[10]])[, "FSC-H"])
```

---

## density1d *Find most likely separation between positive and negative populations in 1D*

---

### Description

The function tries to find a reasonable split point between the two hypothetical cell populations "positive" and "negative". This function is considered internal, please use the API provided by rangeGate.

## Usage

```
density1d(x, stain, alpha = "min", sd = 2, plot = FALSE, borderQuant =
0.1, absolute = TRUE, inBetween = FALSE, refLine=NULL,rare=FALSE,bwFac=1.2
,sig=NULL,peakNr=NULL, ...)
```

## Arguments

| | |
|---|---|
| x | A flowSet or flowFrame. |
| stain | A character scalar giving the flow parameter for which to compute the separation. |
| alpha | A tuning parameter that controls the location of the split point between the two populations. This has to be a numeric in the range [0,1], where values closer to 0 will shift the split point closer to the negative population and values closer to 1 will shift towards the positive population. Additionally, the value of alpha can be "min", in which case the split point will be selected as the area of lowest local density between the two populations. |
| sd | For the case where there is only a single population, the algorithm falls back to esitmating the mode of this population and a robust measure of the variance of it distribution. The sd tuning parameter controls how far away from the mode the split point is set. |
| plot | Create a plot of the results of the computation. |
| borderQuant | Usualy the instrument is set up in a way that the positive population is somewhere on the high end of the measurement range and the negative population is on the low end. This parameter allows to disregard populations with mean values in the extreme quantiles of the data range. It's value should be in the range [0,1]. |
| absolute | Logical controling whether to classify a population (positive or negative) relative to the theoretical measurment range of the instrument or the actual range of the data. This can be set to TRUE if the alignment of the measurment range is not optimal and the bulk of the data is on one end of the theoretical range. |
| inBetween | Force the algorithm to put the separator in between two peaks. If there are more than two peaks, this argument is ignored. |
| refLine | Either NULL or a numeric of lenth 1. If NULL, this parameter is ignored. When it is set to a numeric, the minor sub-population (if any) below this reference line will be igored while determining the separator between positive and negative. |
| rare | Either TRUE or FALSE, assumes that there is one major peak, and that the rare positive population is to the right of it. Uses a robust estimate of mean and variance to gate the positive cells. |
| bwFac | The bandwidth for smoothing the density estimate. User-tunable |
| sig | a value of c(NULL,"L","R"),when sig is not NULL,use the half (left or right) of signal to estimate the std and mean. |
| peakNr | when peakNr is not NULL,drop the less significant peaks by their heights |
| ... | Further arguments. |

## Details

The algorithm first tries to identify high density regions in the data. If the input is a `flowSet`, density regions will be computed on the collapsed data, hence it should have been normalized before (see `warpSet` for one possible normalization technique). The high density regions are then clasified as positive and negative populations, based on their mean value in the theoretical (or absolute if argument `absolute=TRUE`) measurement range. In case there are only two high-density regions the lower one is usually clasified as the negative populations, however the heuristics in the algorithm will force the classification towards a positive population if the mean value is already very high. The `absolute` and `borderQuant` arguments can be used to control this behaviour. The split point between populations will be drawn at the value of mimimum local density between the two populations, or, if the `alpha` argument is used, somewhere between the two populations where the value of alpha forces the point to be closer to the negative (`0 - 0.5`) or closer to the positive population (`0.5 - 1`).

If there is only a single high-density region, the algorithm will fall back to estimating the mode of the distribution (`hubers`) and a robust measure of it's variance and, in combination with the `sd` argument, set the split point somewhere in the right or left tail, depending on the classification of the region.

For more than two populations, the algorithm will still classify each population into positive and negative and compute the split point between those clusteres, similar to the two population case.

## Value

A numeric indicating the split point between positive and negative populations.

## Author(s)

Florian Hahne

## See Also

`warpSet`, `rangeGate`

## Examples

```
library(flowCore)
data(GvHD)
dat <- GvHD[pData(GvHD)$Patient==10]
dat <- transform(dat, "FL4-H"=asinh(`FL4-H`), "FL3-H"=asinh(`FL3-H`))
d <- flowStats:::density1d(dat, "FL4-H", plot=TRUE)
if(require(flowViz))
densityplot(~`FL4-H`, dat, refline=d)

## tweaking the location
flowStats:::density1d(dat, "FL4-H", plot=TRUE, alpha=0.8)

## only a single population
flowStats:::density1d(dat, "FL3-H", plot=TRUE)
flowStats:::density1d(dat, "FL3-H", plot=TRUE, sd=2)
```

`ellipse` *convert ellipse from cov/mu to points used to plot priors*

## Description

convert ellipse from cov/mu to points used to plot priors

## Usage

```
ellipse(cov, centre, level = 0.95)
```

`fdPar` *The version of fdPar from fda 2.4.0 because the new API changes the output. (specifically res$fd$coefs) and thus breaks the landmarkreg call.*

## Description

The version of fdPar from fda 2.4.0 because the new API changes the output. (specifically res$fd$coefs) and thus breaks the landmarkreg call.

## Usage

```
fdPar(fdobj = NULL, Lfdobj = NULL, lambda = 0, estimate = TRUE, penmat = NULL)
```

## Arguments

| | |
|---|---|
| `fdobj` | functional data object, functional basis object, a functional parameter object or a matrix. If it a matrix, it is replaced by fd(fdobj). If class(fdobj) == 'basisfd', it is converted to an object of class fd with a coefficient matrix consisting of a single column of zeros. |
| `Lfdobj` | either a nonnegative integer or a linear differential operator object. If NULL, Lfdobj depends on `fdobj[['basis']][['type']]`<br>• bspline `Lfdobj <- int2Lfd(max(0, norder-2))`, where `norder = norder(fdobj)`<br>• fourier Lfdobj = a harmonic acceleration operator: `Lfdobj <- vec2Lfd(c(0,(2*pi/diff(rng))^2, rng)` where rng = `fdobj[['basis']][['rangeval']]`.<br>• anything else `Lfdobj <- int2Lfd(0)` |
| `lambda` | a nonnegative real number specifying the amount of smoothing to be applied to the estimated functional parameter. |
| `estimate` | not currently used. |
| `penmat` | a roughness penalty matrix. Including this can eliminate the need to compute this matrix over and over again in some types of calculations. |

`flowClust2Prior` *Generate a prior specification based on a flowClust model This function generates a prior specification based on a flowClust fit object It can be passed to a second round of flowClust() with usePrior="yes" The prior could be estimated from a single sample, for example, and then used to speed up the convergence for other samples.*

## Description

Generate a prior specification based on a flowClust model This function generates a prior specification based on a flowClust fit object It can be passed to a second round of flowClust() with usePrior="yes" The prior could be estimated from a single sample, for example, and then used to speed up the convergence for other samples.

## Usage

```
flowClust2Prior(x, kappa, Nt = NULL, addCluster = NULL)
```

## Arguments

| | |
|---|---|
| `x` | a flowClust fit object |
| `kappa` | is the fraction of equivalent observations by which to weight this prior relative to the flowClust model. |
| `Nt` | the number of total equivalent observation |
| `addCluster` | not currently supported |

`gate_singlet` *Creates a singlet polygon gate using the prediction bands from a robust linear model*

## Description

We construct a singlet gate by applying a robust linear model with `rlm`. By default, we model the forward-scatter height (FSC-H)as a function of forward-scatter area (FSC-A). If `sidescatter` is given, forward-scatter height is as a function of `area + sidescatter + sidescatter / area`.

## Usage

```
gate_singlet(
  x,
  area = "FSC-A",
  height = "FSC-H",
  sidescatter = NULL,
  prediction_level = 0.99,
  subsample_pct = NULL,
```

```
  wider_gate = FALSE,
  filterId = "singlet",
  maxit = 5,
  ...
)

singletGate(
  x,
  area = "FSC-A",
  height = "FSC-H",
  sidescatter = NULL,
  prediction_level = 0.99,
  subsample_pct = NULL,
  wider_gate = FALSE,
  filterId = "singlet",
  maxit = 5,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a flowFrame object |
| area | character giving the channel name that records the signal intensity as peak area |
| height | character giving the channel name that records the signal intensity as peak heightchannel name of height |
| sidescatter | character giving an optional channel name for the sidescatter signal. By default, ignored. |
| prediction_level | a numeric value between 0 and 1 specifying the level to use for the prediction bands |
| subsample_pct | a numeric value between 0 and 1 indicating the percentage of observations that should be randomly selected from x to construct the gate. By default, no subsampling is performed. |
| wider_gate | logical value. If TRUE, the prediction bands used to construct the singlet gate use the robust fitted weights, which increase prediction uncertainty, especially for large FSC-A. This leads to wider gates, which are sometimes desired. |
| filterId | the name for the filter that is returned |
| maxit | the limit on the number of IWLS iterations passed to rlm |
| ... | additional arguments passed to rlm |

## Details

Because rlm relies on iteratively reweighted least squares (IRLS), the runtime to construct a singlet gate is dependent in part on the number of observations in x. To improve the runtime, we provide an option to subsample randomly a subset of x. A percentage of observations to subsample can be given in subsample_pct. By default, no subsampling is applied.

## Value

a polygonGate object with the singlet gate

## Examples

```
## Not run:
 # fr is a flowFrame
 sg <- gate_singlet(fr, area = "FSC-A", height = "FSC-H")
 sg
 # plot the gate
 xyplot(`FSC-H` ~ `FSC-A`, fr, filter = sg)

## End(Not run)
```

---

gaussNorm *Per-channel normalization based on landmark registration*

---

## Description

This funciton normalizes a set of flow cytometry data samples by identifying and aligning the high density regions (landmarks or peaks) for each channel. The data of each channel is shifted in such a way that the identified high density regions are moved to fixed locations called base landmarks.

## Usage

```
gaussNorm (flowset, channel.names, max.lms=2, base.lms=NULL,
peak.density.thr=0.05, peak.distance.thr=0.05, debug=FALSE, fname='')
```

## Arguments

| | |
|---|---|
| flowset | A flowSet. |
| channel.names | A character vector of flow parameters in flowset to be normalized. |
| max.lms | A numeric vector of the maximum number of base landmarks to be used for normalizing each channel. If it has only one value that will be used as the maximum number of base landmarks for all the channels. |
| base.lms | A list of vector for each channel that contains the base landmarks for normalizing that channel. If not specified the base landmarks are computed from the set of extracted landmarks. |
| peak.density.thr | The peaks with density value less than "peak.density.thr times maximum peak density" are discarded. |
| peak.distance.thr | The sequences of peaks that are located closer than "peak.distance.thr times range of data" are identified. Then for each sequence only one peak (the one with the highest intensity value) is used as a landmark. In other words no two landmarks are located closer than "peak.distance.thr times range of data" to each other. |

debug — Logical. Forces the function to draw before and after normalization plots for each sample. The plot of the i-th sample is stored in `paste(fname, i)` file.

fname — The pre- and post- normalization plots of the i-th sample is stored in `paste(fname, i)` file if debug is set to `TRUE`. If default value is used the plots are drawn on separate X11 windows for each sample. In this case, the function waits for a user input to draw the plots for the next sample.

## Details

Normalization is archived in three phases: (i) identifying high-density regions (landmarks) for each `flowFrame` in the `flowSet` for a single channel; (ii) computing the best matching between the landmarks and a set of fixed reference landmarks for each channel called base landmarks; (iii) manipulating the data of each channel in such a way that each landmark is moved to its matching base landmark. Please note that this normalization is on a channel-by-channel basis. Multiple channels are normalized in a loop.

## Value

A list with items `flowset`: normalized `flowSet`. `confidence`: a confidence measure of the normalization procedure.

## Author(s)

Alireza Hadj Khodabakhshi

## Examples

```
library(flowCore)
data(ITN)
dat <- transform(ITN, "CD4"=asinh(CD4), "CD3"=asinh(CD3), "CD8"=asinh(CD8))
lg <- lymphGate(dat, channels=c("CD3", "SSC"), preselection="CD4",scale=1.5)
dat <- Subset(dat, lg)
datr <- gaussNorm(dat, "CD8")$flowset
if(require(flowViz)){
  d1 <- densityplot(~CD8, dat, main="original", filter=curv1Filter("CD8"))
  d2 <- densityplot(~CD8, datr, main="normalized", filter=curv1Filter("CD8"))
  plot(d1, split=c(1,1,2,1))
  plot(d2, split=c(2,1,2,1), newpage=FALSE)
}
```

---

# gpaSet *Multi-dimensional normalization of flow cytometry data*

---

## Description

This function performs a multi-dimensional normalization of flow cytometry data (`flowSets`) using a generalized Procrustes analysis (GPA) method.

## Usage

```
gpaSet(x, params, register="backgating", bgChannels=NULL,
       bg=NULL,  rotation.only=TRUE,
       downweight.missingFeatures=FALSE, thres.sigma=2.5,
       show.workflow=FALSE,
       ask=names(dev.cur())!="pdf")
```

## Arguments

| | |
|---|---|
| x | A flowSet. |
| params | A character vector of length 2 describing the channels of interest. |
| register | A character indicating the method to be used for identifying features. Available method only includes "backgating" at the point. |
| bgChannels | A character vector indicating the channels used for backgating. If NULL, backGating will find the appropriate backgating channels. |
| bg | A data frame as the returning value of the backGating function. If not NULL, gpaSet will skip the backGating process and use the given data frame to extract potential features. |
| rotation.only | Logical for coarsing a reflection matrix to a rotation matrix. |
| downweight.missingFeatures | Logical. If TRUE, the missing features, labeled as bogus features, are down-weighted to zero. See details. |
| thres.sigma | A numerical value indicating the threshold of where to cut the tree, e.g., as resulting from diana, into several clusters. It is default to 2.5 sigma of the distribution of the heights of the cluster points. |
| show.workflow | Logical. If TRUE, the workflow of gpaSet will be displayed. |
| ask | Logical. If TRUE, the display operates in interactive mode. |

## Details

Normalization is achieved by first identifying features for each flowFrame in the flowSet for designated channels using backgating, subsequently labeling features, and finally aligning the features to a reference feature in the sense of minimizing the Frobenus norm of

$$||sFQ - \bar{F}||,$$

where $s$ is a scalar, $Q$ a rotational matrix, $F$ the matrix of features, and $\bar{F}$ the reference feature. Both $s$ and $Q$ are solved by using singular value decomposition (SVD).

Note that if feature $F_{ij}$ is missing, it is given a bogus value as $\bar{F}_{ij}$.

If downweight.missingFeatures is TRUE, the cost function becomes

$$||sW_0FQ - W_0\bar{F}||,$$

where the weighting function $W_0$ is zero if the corresponding feature is bogus.

## Value

The normalized flowSet with "GPA" attribute.

## Author(s)

C. J. Wong <cwon2@fhcrc.org>

## References

in progress

## Examples

```
library(flowCore)
## Example 1: calling up gpaSet directly
data(ITN)
data(BackGating)

tl <- transformList(colnames(ITN)[3:7], asinh, transformationId="asinh")
dat <- transform(ITN, tl)

xy = c("FSC", "SSC")
bgChannels = c("CD8", "CD4", "CD3")
## bg <- flowStats:::backGating(dat, xy=xy, channels=bgChannels)
## using pre-generated backgating results: BackGating
s <- gpaSet(dat, params=xy, bgChannels=bgChannels, bg=BackGating)

if(require(flowViz)) {
   d1 <- densityplot(~., s, channels=c("FSC", "SSC"),
                     layout=c(2,1), main="After GPA using bg")
   d2 <- xyplot(FSC ~ SSC, as(s, "flowFrame"),
                channels=c("FSC", "SSC"), main="All flowFrames")
   plot(d1)
   plot(d2)
}

## view "GPA" attribute
attr(s, "GPA")

## Not run:
library(flowCore)
## Example 2: using work flow and normalization objects
data(ITN)
ITN <- ITN[1:8, ]
wf <- workFlow(ITN)
tl <- transformList(colnames(ITN)[3:7], asinh, transformationId="asinh")
add(wf, tl)
x <- Data(wf[["asinh"]])
## normalize 'FSC' and 'SSC' channels
norm <- normalization(normFun=function(x, parameters, ...)
        gpaSet(x, parameters, ...),
        parameters = c("FSC", "SSC"),
```

```
 arguments=list(bgChannels=c("CD8", "CD3"),
                       register="backgating"),
 normalizationId="Procrustes")

add(wf, norm2, parent="asinh")
s <- Data(wf[["Procrustes"]])
if(require(flowViz)) {
   d1 <- densityplot(~., s, channels=c("FSC", "SSC"),
                     layout=c(2,1), main="After GPA using bg")
   d2 <- xyplot(FSC ~ SSC, as(s, "flowFrame"),
                channels=c("FSC", "SSC"), main="All flowFrames")
   plot(d1)
   plot(d2)
}

## End(Not run) ## end of dontrun
```

---

`idFeaturesByBackgating` *(Internal use only) Identify features of flow cytometry data using backgating*

---

## Description

Identify and labeling significant features using divisive clustering method such as `diana`.

## Usage

```
idFeaturesByBackgating(bg, nDim, thres.sigma=2.5, lambda=0.1,
                       reference.method="median",
                       plot.workflow=FALSE, ask=names(dev.cur())!="pdf")
```

## Arguments

| | |
|---|---|
| `bg` | A data frame containing subpopulations on channels of interests. Must be a returning result from `flowStats:::backGating` |
| `nDim` | An integer indicating the length of channels of interest. |
| `thres.sigma` | An numerical value indicating the threshold at which to cut tree, e.g., as resulting from 'diana', into several clusters. |
| `lambda` | A numerical value indicating the percentage of the potential features that is used as a threshold for deciding outlier clusters. The default value is 0.1. |
| `reference.method` | A character vector indicating the method for computing the reference features. If `median`, the reference feature is defined by the medain of eac cluster of features. Valid methods include `median` and `mean` only. |
| `plot.workflow` | Logical. If TURE, display the workflow of feature identification. |
| `ask` | Logical. If TRUE, the display operates in interactive mode. |

## Details

Using the resulting data frame from backGating as potential features, the algorithm follows four major steps: (i) centering the potential features, which yields the returning value TransMatrix, (ii) using diana to compute a clustering of the potential features, (iii) cutting the tree into several clusters, and (iv) accessing outliers and rendering the final registered features with labels.

In step three, the threshold for cutting the tree is computed by

$$sd * thres.sigma,$$

where $sd$ is the standard deviation of the distribution of the height between entities computed by diana.

A cluster is determined as an outlier if the number of its members is less than the median of the numbers of all clusters' members times 'lambda'.

## Value

| | |
|---|---|
| register | A list containing registered features for each sample. |

## Author(s)

Chao-Jen Wong

## See Also

diana, BackGating, gpaSet.

## Examples

```
## Not run:
library(flowCore)
data(ITN)
wf <- workFlow(ITN)
tl <- transformList(colnames(ITN)[3:7], asinh, transformationId="asinh")
dat <- trnasformList(ITN, tl)
bg <- backGating(dat, xy=c("FSC", "SSC"), channels="CD3")

## End(Not run)

data(BackGating)
results <- flowStats:::idFeaturesByBackgating(bg=BackGating, nDim=2,
                      plot.workflow=TRUE, ask=TRUE)
```

iProcrustes — *Procrustes analysis. Using singular value decomposition (SVD) to determine a linear transformation to align the points in X to the points in a reference matrix Y.*

## Description

Based on generalized Procrustes analysis, this function determines a linear transformation (rotation/reflection and scalling) of the points in matrix `x` to align them to their reference points in matrix `xbar`. The alignemnt is carried out by minimizing the distance between the points in `x` and `xbar`.

## Usage

```
iProcrustes(x, xbar, rotation.only=TRUE, scalling=TRUE, translate=FALSE)
```

## Arguments

| | |
|---|---|
| `x` | A numerical matrix to be align to points in `xbar`, the second arguement. The columns represents the coordinates of the points. The matrices `x` and `xbar` must have the same dimensions. |
| `xbar` | A numerical, reference matrix to which points in matrix `x` are to be aligned. |
| `rotation.only` | Logical. When `rotaion.only` is TRUE, it allows the function to lose reflection component of the linear transformation. Although it might not give the best-fitting aligenment, when dealing with flow cytometry data alignment, a non-reflection transformation is prefered. When `rotaion.only` is FALSE, it allows the function to retain the reflection component. |
| `scalling` | Logical. When `scalling` is FALSE, it allows the function to calculate the linear transformation without a scalling factor. That is, the returning scalling factor is set to 1. |
| `translate` | Logical. Set `translate` to FALSE when the points in matrices `x` and xbar are already centralized prior to applying this function. When `translate` is TRUE, it allows the function to translate the centroid the points in matrix `x` to that of points in `xbar`. |

## Details

Suppose the points in matrix $X$ and $\bar{X}$ are centralized (meaning their centroids are at the origin). The linear transformation of $X$ for aligning $X$ to its reference matrix $\bar{X}$., i.e., min $||sXQ - \bar{X}||_F$, is given by:

$$Q = VU^T,$$

and

$$s = trace(\bar{X}^T XQ)/trace(X^T X),$$

where V and U are the sigular value vectors of $\bar{X}^T X$ (that is, $\bar{X}^T X = U\Sigma V^T$), and $s$ is the scalling factor.

## Value

A list of the linear tranformation with items

| | |
|---|---|
| `Q` | An orthogonal, rotation/reflection matrix. |
| `scal` | A scalling factor |
| . | |
| `T` | (optional) A translation vector used to shift the centroid of the points in matrix `x` to the origin. Returned when `translate` is TRUE. |
| `T.xbar` | (optional) Centered `xbar` (that is, the centroid of the points in `xbar` is translated to the origin). Returned when `translate` is TRUE. |

Note that the return values of this function do not include the transformed matrix $scal * x * Q$ or $scal * (x - IT) * Q$, where $T$ is the translation vector and $I$ is an $n - by - 1$ vector with elements 1.

## Author(s)

C. J. Wong <cwon2@fhcrc.org>

## See Also

gpaSet

## Examples

```
## Example 1
x <- matrix(runif(20), nrow=10, ncol=2)+ 1.4
s <- matrix(c(cos(60), -sin(60), sin(60), cos(60)),
            nrow=2, ncol=2, byrow=TRUE)
xbar <- 2.2 *(x %*% s) - 0.1

lt <- iProcrustes(x, xbar, translate=TRUE) ## return linear transformation
lt

## showing result
I <- matrix(1, nrow=nrow(x), ncol=1)
tx <- x - I %*% lt$T
## get the transformed matrix xnew
xnew <- lt$scal * (tx %*% lt$Q)

if (require(lattice)) {
   xyplot(V1 ~ V2,
          do.call(make.groups, lapply(list(x=x, xbar=xbar, T.xbar=lt$T.xbar,
                  xnew=xnew),as.data.frame)),
          group=which, aspect=c(0.7), pch=c(1,3,2,4), col.symbol="black",
   main=("Align the points in x to xbar"),
          key=list(points=list(pch=c(1,3,2,4), col="black"), space="right",
                   text=list(c("x", "xbar", "T.xbar", "xnew"))))
}
```

```
## Example 2. centralized x and xbar prior to using iProcrustes
x <- matrix(runif(10), nrow=5, ncol=2)
s <- matrix(c(cos(60), -sin(60), sin(60), cos(60)),
            nrow=2, ncol=2, byrow=TRUE)
xbar <- 1.2 *(x %*% s) - 2
I <- matrix(1, nrow=nrow(x), ncol=1)
x <- x-(I %*% colMeans(x)) ## shift the centroid of points in x to the origin
xbar <- xbar - (I %*% colMeans(xbar)) ## shift centroid to the origin
lt <- iProcrustes(x, xbar, translate=FALSE) ## return linear transformation
## only return the rotation/reflection matrix and scalling factor
lt

xnew=lt$scal *(x %*% lt$Q) ## transformed matrix aligned to centralized xbar
if (require(lattice)) {
    xyplot(V1 ~ V2,
           do.call(make.groups, lapply(list(x=x,xbar=xbar,
                   xnew=xnew), as.data.frame)),
           group=which, auto.key=list(space="right"))
}
```

---

ITN *Sample flow cytometry data*

---

## Description

A flowSet cotaining data from 15 patients.

## Usage

```
data(ITN)
```

## Format

A flowSet containing 15 flowFrames. There are 3 patient groups with 5 samples each.

## Source

Immune Tolerance Network

`landmarkMatrix` *Compute and cluster high density regions in 1D*

## Description

This functions first identifies high-density regions for each `flowFrame` in a `flowSet` and subsequently tries to cluster these regions, yielding the landmarks matrix that needs to be supplied to `landmarkreg`. The function is considered to be internal.

## Usage

```
landmarkMatrix(data, fres, parm, border=0.05, peakNr=NULL, densities =
NULL, n = 201, indices=FALSE)
```

## Arguments

| | |
|---|---|
| `data` | A `flowSet`. |
| `fres` | A list of `filterResultList` objects generated by a filtering opration using a `curv1Filter`. Each list item represents the results for one of the flow parameters in `parm`. |
| `parm` | Character scalar of flow paramater to compute landmarks for. |
| `border` | A numeric in `[0,1]`. Ignore all high-density regions with mean values in the extreme percentiles of the data range. |
| `peakNr` | Force a fixed number of peaks. |
| `densities` | An optional matrix of y values of the density estimates for the `flowSet`. If this is not present, density estimates will be calculated by the function. |
| `n` | Number of bins used for the density estimation. |
| `indices` | Return matrix of population indices instead of landmark locations. These indices can be used to point into the populations identified by the curv1Filter. |

## Details

In order to normalize the data using the `landmarkreg` function in the `fda`, a set of landmarks has to be computed for each `flowFrame` in a `flowSet`. The number of lansmarks has to be the same for each frame. This function identifies high-density regions in each frame, computes a simple clustering and returns a matrix of landmark locations. Missing landmarks of individual frames are substituted by the mean landmark location of the respective cluster.

## Value

A matrix of landmark locations. Columns are landmarks and rows are `flowFrames`.

## Author(s)

Florian Hahne

## See Also

landmarkreg,warpSet

## Examples

```
library(flowCore)
data(GvHD)
tmp <- list("FSC-H"=filter(GvHD[1:3], curv1Filter("FSC-H")))
res <-  flowStats:::landmarkMatrix(GvHD[1:3], tmp, "FSC-H")
```

---

# lymphFilter-class *Automated gating of elliptical cell populations in 2D.*

---

## Description

Cell populations of roughly elliptical shape in two-dimensional projections are of huge interest in many flow cytometry applications. This function identifies a single such population, potentially from a mixture of multiple populations.

## Usage

```
lymphGate(x, channels, preselection=NULL, scale=2.5,  bwFac=1.3,
          filterId="defaultLymphGate", plot=FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | An object of class flowSet. |
| channels | A character vector of length 2 of valid flow parameters in x. |
| preselection | Either NULL, in which case this boils down to fitting a regular norm2Filter, a character scalar giving one of the flow parameters in x, or a named list of numerics specifying the initial rough preselection. The latter gets passed on to rectangleGate, see it's documentation for details. |
| scale | The scaleFactor parameter that gets passed on to norm2Filter. |
| bwFac | The bandwidth factor that gets passed on to curv1Filter. |
| filterId | A character used as filterId. |
| plot | Logical. Produce plots of filter results |
| ... | Additional arguments. |

## Details

This algorithm does not apply real mixture modelling, however it is able to identify a single elliptical cell population from a mixture of multiple such populations. The idea is to first define a rough rectangular preselection and, in a second step, fit a bivariate normal distribution to this subset only.

Depending on the value of `preselection`, the initial rough selection is either

**NULL:** No preselection at all

**character scalar** Preselection based on cells that are positive for a single marker only. This allows for back-gating, for instances by selecting `CD4+` T-cells and using this information to back-gate lymphocytes in `FSC` and `SSC`. Positive cells are identified using a `curv1Filter`.

**a named list of numerics:** Preselection by a rectangular gate. The items of the list have to be numerics of length one giving the gate boundaries in the respective dimensions.

## Value

An `ellipsoidGate` or list of `ellipsoidGate` objects

## Extends

Class `parameterFilter`, directly.

Class `concreteFilter`, by class "parameterFilter", distance 2.

Class `filter`, by class "parameterFilter", distance 3.

## Slots

See `Arguments` section for details.

`preselection`**:** Object of class `character`, the name of the flow parameter used for preselection.

`rectDef`**:** Object of class `list`, the initial rectangular selection.

`scale`**:** Object of class `numeric`.

`bwFac`**:** Object of class `numeric`.

`parameters`**:** Object of class `parameters`, the flow parameters to operate on.

`filterId`**:** Object of class `"character"`, the filter identifier.

## Objects from the Class

Objects can be created by calls of the form `new("lymphFilter", parameters, ...)` or using the constructor `lymphFilter`. The constructor is the recommended way of object instantiation.

## Author(s)

Florian Hahne

## See Also

`norm2Filter`, `curv1Filter`

## Examples

```
library(flowCore)
data(GvHD)
dat <- GvHD[pData(GvHD)$Patient==10]
dat <- transform(dat, "FL4-H"=asinh(`FL4-H`))
lg <- lymphGate(dat, channels=c("FSC-H", "SSC-H"), preselection="FL4-H",scale=1.5)

if(require(flowViz))
xyplot(`SSC-H`~`FSC-H`, dat, filter=lg)
```

---

mkPrior *Generate a flowClust prior specification*

---

## Description

Generate a flowClust prior specification from gates and data

## Usage

```
mkPrior(gate, data, nu0, Omega0, ...)

## S4 method for signature 'polygonGate,flowFrame,numeric,matrix'
mkPrior(gate, data, nu0, Omega0)

## S4 method for signature 'rectangleGate,flowFrame,numeric,matrix'
mkPrior(gate, data, nu0, Omega0)

## S4 method for signature 'rectangleGate,flowFrame,missing,missing'
mkPrior(gate, data, nu0 = NA, Omega0 = NA)

## S4 method for signature 'polygonGate,flowFrame,missing,missing'
mkPrior(gate, data, nu0 = NA, Omega0 = NA)

## S4 method for signature 'list,flowSet,missing,missing'
mkPrior(gate, data, nu0 = NA, Omega0, model.cov = "full", model.means = "full")

## S4 method for signature 'missing,flowSet,ANY,missing'
mkPrior(gate, data, nu0 = NA, Omega0, model.cov = "full", model.means = "full")

## S4 method for signature 'missing,flowFrame,missing,missing'
mkPrior(gate, data, nu0, Omega0)

## S4 method for signature 'list,flowSet,ANY,missing'
mkPrior(gate, data, nu0 = NA, Omega0, model.cov = "full", model.means = "full")
```

### Arguments

| | |
|---|---|
| gate | A list of flowCore gates. The gates should represent the SAME population gated across multiple samples. |
| data | A flowSet of the same size as the number of gates above. Each flowFrame in the flowSet should contain the events representing the population in its corresponding gate. i.e. it should be the gated data. |
| nu0 | The nu0 hyperparameter. For estimation from data, it should be nu0=NA. |
| Omega0 | The Omega0 hyperparameter. For estimation from data it can be missing. |
| ... | Not currently used. |
| model.cov, model.means | model names used for cov and means. one of c("full","DE","DU"). "full" is the default. |

### Details

Construct a prior specification. Generally not called by the user.

### Value

Return values depend on the specific method called. Not meant for user consumption.

### Author(s)

Greg Finak <gfinak@fhcrc.org>

### References

http://www.rglab.org

### Examples

```
## The function is currently defined as
```

---

## norm2Filter-class *Class "norm2Filter"*

---

### Description

Class and constructors for a filter that fits a bivariate normal distribution to a data set of paired values and selects data points according to their standard deviation from the fitted distribution.

### Usage

```
norm2Filter(x, y, method="covMcd", scale.factor=1, n=50000,
filterId="defaultNorm2Filter")
```

## Arguments

| | |
|---|---|
| x, y | Characters giving the names of the measurement parameter on which the filter is supposed to work on. y can be missing in which case x is expected to be a character vector of length 2 or a list of characters. |
| filterId | An optional parameter that sets the filterId slot of this filter. The object can later be identified by this name. |
| scale.factor, n | Numerics of length 1, used to set the scale.factor and n slots of the object. |
| method | Character in covMcd or cov.rob, used to set the method slot of the object. |

## Details

The filter fits a bivariate normal distribution to the data and selects all events within the Mahalanobis distance multiplied by the scale.factor argument. The constructor norm2Filter is a convenience function for object instantiation. Evaluating a curv2Filter results in an object of class logicalFilterResult. Accordingly, norm2Filters can be used to subset and to split flow cytometry data sets.

## Value

Returns a norm2Filter object for use in filtering flowFrames or other flow cytometry objects.

## Slots

method One of covMcd or cov.rob defining method used for computation of covariance matrix.

scale.factor Numeric vector giving factor of standard deviations used for data selection (all points within scalefac standard deviations are selected).

n Object of class "numeric", the number of events used to compute the covariance matrix of the bivariate distribution.

filterId Object of class "character" referencing the filter.

parameters Object of class "ANY" describing the parameters used to filter the flowFrame or flowSet.

## Extends

Class "parameterFilter", directly.

Class "concreteFilter", by class parameterFilter, distance 2.

Class "filter", by class parameterFilter, distance 3.

## Objects from the Class

Objects can be created by calls of the form new("norm2Filter",...) or using the constructor norm2Filter. The constructor is the recommended way.

## Methods

**%in%** `signature(x = "flowFrame", table = "norm2Filter")`: The workhorse used to evaluate the filter on data. This is usually not called directly by the user, but internally by calls to the `filter` methods.

**show** `signature(object = "norm2Filter")`: Print information about the filter.

## Note

See the documentation in the `flowViz` package for plotting of `norm2Filters`.

## Author(s)

F. Hahne

## See Also

`cov.rob`, `CovMcd`, `filter` for evaluation of `norm2Filters` and `split` and `Subset`for splitting and subsetting of flow cytometry data sets based on that.

## Examples

```
library(flowCore)
## Loading example data
dat <- read.FCS(system.file("extdata","0877408774.B08",
package="flowCore"))

## Create directly. Most likely from a command line
norm2Filter("FSC-H", "SSC-H", filterId="myCurv2Filter")

## To facilitate programmatic construction we also have the following
n2f <- norm2Filter(filterId="myNorm2Filter", x=list("FSC-H", "SSC-H"),
scale.factor=2)
n2f <- norm2Filter(filterId="myNorm2Filter", x=c("FSC-H", "SSC-H"),
scale.factor=2)

## Filtering using norm2Filter
fres <- filter(dat, n2f)
fres
summary(fres)

## The result of norm2 filtering is a logical subset
Subset(dat, fres)

## We can also split, in which case we get those events in and those
## not in the gate as separate populations
split(dat, fres)
```

`normalize-methods` *normalize a GatingSet imported with flowWorkspace, using sequential normalization on the manual gates in the GatingHierarchy.*

## Description

The method will step through the gating hierarchy in a breadth first search manner and normalize each dimension and gate not explicitly excluded in skipdims,or skipgates. The normalization approach is based on warpSet, but uses sequential normalization to alternately normalize then perform gating of the cell populations. This often helps with feature registration of popualtions lower in the gating hierarchy. FSC and SSC, as well as time are generally excluded by default. The rule of thumb, is to only normalize a channel in a gate if it is absolutely warranted.

## Usage

```
normalize(data,x,...)
```

## Arguments

| | |
|---|---|
| `data` | The `GatingSet` to be normalized. |
| `x` | `missing`. Not used in here. |
| `...` | Arguments passed to downstream functions.<br>target: The target sample to normalize the other samples in the gating set to. A `character` vector. Must match a sample name in `x`, otherwise `NULL` will use the mean (average) of the peaks identified in all samples<br>populations: A `character` vector of population names that are to be normalized.<br>dims: A `character` vector of parameter names to be normalized.<br>chunksize: For a memory-efficient implementation of normalization, set the `chunksize`,(an `integer`), which will peform normalization on chunks of the data of size `chunksize`.<br>nPeaks: A `list` of `integer` or an `integer` vector that specifies the expected number of peaks for each sample. Can be omitted to keep all peaks.<br>bwFac: The bandwidth for density estimation, a `numeric`. Affects the sensitivity for smoothing and detecting distinct peaks. |

## Details

This function implements sequential normalization using a GatingSet and a set of manual gates. For each gate in the gating hierarchy, the algorithm checks if the gate should be normalized, and which dimensions in the gate should be normalized. If normalization is warranted, this is performed prior to gating. After gating, the counts for the gate in the GatingSet are updated, and the next gate is processed. This is useful in the application of template gates to data that has staining variability in one or more channels.

## Value

Returns a `GatingSet` of normalized data.

## Author(s)

Greg Finak <greg@ozette.ai>

## See Also

See also GatingSet-class, GatingHierarchy-class, ncdfFlowSet

## Examples

```
## Not run:
#gs is a GatingSet
gs_norm <- normalize(gs
                , target = "M+T panel_903997-25.fcs"
                , populations = "cd27gate"
                , dims = "<Violet A 610/20-A>"
                , minCountThreshold = 100
                , nPeaks = list('cd27gate' = 2)
                , chunksize = 10
                , bwFac = 2
                )



#show the population statistics for before and after normalization
getPopStats(gs)
getPopStats(gs_norm)

#plot the gate to see the effects of normalization
grid.arrange(
  plotGate(gs, "cd27gate", type = "densityplot", stack = T)
  ,plotGate(gs_norm, "cd27gate", type = "densityplot", stack = T)
  )

## End(Not run)
```

---

# normQA *Normalization quality assessment*

---

## Description

Create QA plots for a flow cytometry normalization process.

## Usage

```
normQA(data, morph = c("^fsc", "^ssc"),
  channels, odat = NULL, ask = names(dev.cur()) != "pdf",
    grouping = NULL, tag.outliers = FALSE, peaksOnly = TRUE)
```

### Arguments

| | |
|---|---|
| data | a normalized flowSet. |
| morph | A character vector of channel names to use for the backgating into the morphological channels. |
| channels | The channels for which to create plots. Defaults to all normalized channels. |
| odat | The original data set, always needed if there are no warping functions available. |
| ask | Ask before creating a new plot. |
| grouping | A grouping variable in data's phenoData slot. |
| tag.outliers | Logical. Add sample name to outliers in the plots. |
| peaksOnly | Logical. Only use data when a peak was detected in a particular sample. If set to FALSE, a average peak location is estimated. |

### Details

This function assumes that the necessary information has been added as attributes to data during the normalization procedure. Depending on the available information, a set of QA plots is generated. Available plots are:

Amount of peak adjustment

Warping functions

Landmark classification confidence

Backgating of peak events in morphological channels

### Value

This function is called for its side effect of generating plots.

### Author(s)

Florian Hahne

---

## overton_like *Overton-like subtraction of densities.*

---

### Description

This function computes an Overton-like subtraction of two densities. It calculates the proportion of the reference density that is above a reference

### Usage

```
overton_like(ref, test, twosided = FALSE)
```

### Arguments

| | |
|---|---|
| ref | The reference channel specified as a vector |
| test | The test (potentially positive) channel specified as a vector |
| twosided | boolean flag testing whether the area of the density of the test curve above the reference curve will be calculated on both sides of the mode of the test curve (TRUE) or only on the positive side of the mode (FALSE, default). |

### Details

The test can be one-sided or two-sided. If one sided, it tests the region of the test density that is above the mode of the reference density. If two-sided it will look at the regions on either side of the mode of the reference density. Densities are computed on a grid of 1024, and appropriately normalized.

### Value

numeric value representing the proportion of the area of the test density above the reference density.

### Author(s)

Greg Finak

### Examples

```
A = rnorm(10000,mean=1,sd=0.5)
B = rnorm(10000,mean=2,sd=0.5)
overton_like(A,B)
```

---

## plotBins *Plots the probability bins overlaid with flowFrame data*

---

### Description

This function is useful in visualizing the differences between the binned control and sample datasets. The bins generated from the control dataset are overlaid with the sample dataset. An optional argument residuals can be used to shade each bin based on a calculated statistical measure of difference between the number of events in each bin.

### Usage

```
plotBins(binRes,data,channels,title,residuals,shadeFactor)
```

### Arguments

| | |
|---|---|
| binRes | The result generated by calling the probBin function on a control dataset. |
| data | An object of class flowFrame sample(dataset) |
| channels | The flow parameters to be plotted.In cases where more than two parameters are binned from the control set, the plotBins function plots the projections of the hyperplanes in 2 dimensions) |
| title | Optional title for the plot generated |
| residuals | A vector of length equal to the number of bins generated that can be used to shade each bin. The residuals from the calcPearsonChi function or the calcPBChiSquare function can be used to highlight the bins that are different between control and sample datasets |
| shadeFactor | Optional argument between 0 and 1 that controls the intensity of the shading of bins |

### Author(s)

Nishant Gopalakrishnan

### See Also

proBin, calcPearsonChi, calcPBChiSquare

### Examples

```
library(flowCore)
data(GvHD)
# flow frame 1 is treated as  control dataset and used to generate bins
resCtrl<-proBin(GvHD[[1]],200,channels=c("FSC-H","SSC-H"))
plotBins(resCtrl,GvHD[[1]],channels=c("FSC-H","SSC-H"),title="Binned control data")
# Same bins are applied to flowFrame 16
resSample<-binByRef(resCtrl,GvHD[[16]])
stat<-calcPearsonChi(resCtrl,resSample)
dev.new()
plotBins(resCtrl,data=GvHD[[16]],channels=c("FSC-H","SSC-H","Time"),title="Comparision 1 & 16",
residuals=stat$residuals[2,],shadeFactor=0.7)
```

---

## plotPrior *Plots a flowClust prior over some data.*

---

### Description

Plots a flowClust prior overlaid on data.

### Usage

```
plotPrior(data, prior, dims = NULL, ...)
```

### Arguments

| | |
|---|---|
| data | On object of class "flowFrame". The data to be plotted. |
| prior | An object of class "flowClustPrior", or "flowClustPriorList", returned by a call to mkPrior. |
| dims | A character vector of the dimensions to be included in the plot. The dimension names should match column names in the prior and in the flowFrame. |
| ... | Additional arguments to plotting functions, such as smooth=TRUE/FALSE |

### Details

Generates a plot of a "flowClustPrior" or "flowClustPriorList" object overlaid on some data. Plots the prior means (Mu0), prior covariance of the means (Omega0), and prior sample covariance (Lambda0).

### Value

Silently returns zero.

### Author(s)

Greg Finak <gfinak@fhcrc.org>

---

## proBin *Probability binning - a metric for evaluating multivariate differences*

---

### Description

This function divides the flowframe events into bins such that each bin contains the same number of events. The number of events falling into each bin can then be compared across the control and test samples using statistical methods such as the Chi-squared test.

### Usage

```
proBin(m, minEvents=500,channels=NULL)
```

### Arguments

| | |
|---|---|
| m | An object of class flowFrame |
| minEvents | The minEvents The minimum number of events in each bin. (i.e. the termination criterion for the probability binning algorithm) |
| channels | A character vector for the Flourescence channels on which probability binning is to be performed. Defaults is NULL, in which case, all flourescence channels are used for probability binning.( Time information, if provided in the flowFrame is discarded) |

## Details

The flowSet is first filtered using a rectangleGate and the norm2Filter is subsequently fitted to the remaining subset.

## Value

A list with items:

table — A data.frame that stores information regarding each node of the tree generated during each stage of the probability binning algorithm. Each row in the table represents a node, the first row representing the original flowFrame matrix.

The dataIndx column provides indexes for retrieving the matrices during each stage of the binning process from the enviroment data .

The parent field indicates the row number in the table that holds the parent information for the corresponding node.

The left and right columns indicates the row numbers in the table that stores information regarding the children of that particular node. The leaf nodes that hold the binned data can be identified by the nodes with the left of right values of zero( ie. no children nodes)

The visited column is used internally by the algorithm to check if a particular node has been visited during the computation process.

data — An enviroment in which the matrices generated during each stage of the probability binning process is stored. The matrices stored at the leaf nodes represent the binned events obtained after the stop criterion of minEvents has been achieved. These can be identified by the corresponding dataIndx fields provided by the rows in the table with the left or right column values of zero.

limits — A list containing the the boundaries of each hyperplane generated during probability binning

splitPars — A data.frame containing two columns splitCol - indicates the column number of the flowFrame , the split was performed.

splitMed - The median value which was used as the threshold for splitting the flowFrame

The splitCol and splitMed parameters are utilized by the plotBins and shadeBins functions in visualizing the differences between control and test sample cases.

## Author(s)

Nishant Gopalakrishnan

## See Also

plotBins, binByRef

## Examples

```
library(flowCore)
data(GvHD)
res<-proBin(GvHD[[1]],200,channels=c("FSC-H","SSC-H","FL1-H","FL4-H"))
```

quadrantGate *Automated quad gating*

## Description

This function tries to find the most likely separation of two-dimensional flow cytometry in four quadrants.

## Usage

```
quadrantGate(x, stains, alpha=c("min", "min"), sd=c(2, 2), plot=FALSE,
    filterId="defaultQuadGate", refLine.1=NULL, refLine.2=NULL
       ,rare=c(FALSE,FALSE)
       ,sig=c(NULL,NULL)
      ,...)
```

## Arguments

| | |
|---|---|
| x | A flowSet or flowFrame. |
| stains | A character vector of length two giving the two flow parameters for which the quad gate is to be computed. |
| alpha, sd | Tuning factors to control the computation of the gate boundaries. See rangeGate for details. |
| plot | Logical. Produce plots of intermediate results. |
| filterId | Character, the name assigned to the resulting filter. |
| refLine.1 | Either NULL or a numeric of lenth 1. If NULL, this parameter is ignored. When it is set to a numeric, the minor sub-population (if any) below this reference line in the first stain channel will be igored while determining the separator between positive and negative. |
| refLine.2 | Either NULL or a numeric of lenth 1. If NULL, this parameter is ignored. When it is set to a numeric, the minor sub-population (if any) below this reference line in the second stain channel will be igored while determining the separator between positive and negative. |
| rare | logical flags for two channels, Refer to density1d for more details. |
| sig | parameters for two channels. Refer to density1d for more details. |
| ... | Additional arguments |

## Details

The most likely separation between postitive and negative stains for two-dimensional data is computed based on density estimates. Essentially, the gate parameters are first fitted separately for the two parameters and later combined. See the documentation for rangeGate for details. There is a certain amount of heuristics involved in this process. The algorithm can be slightly tweaked using the alpha and sd arguments. Their values will be recycled for the two dimensions unless explicitely given as vectors of length 2.

## Value

An object of class quadGate.

## Author(s)

Florian Hahne

## See Also

quadGate, rangeGate

## Examples

```
## Not run:
library(flowCore)
data(GvHD)
dat <- GvHD[pData(GvHD)$Patient==10]
dat <- transform(dat, "FL4-H"=asinh(`FL4-H`), "FL2-H"=asinh(`FL2-H`))
qg <- quadrantGate(dat, c("FL2-H", "FL4-H"))
qg

if(require(flowViz))
xyplot(`FL2-H`~`FL4-H`, dat, filter=qg)

qg <- quadrantGate(dat, c("FL2-H", "FL4-H"), alpha=c(0.1, 0.9), plot=TRUE)
qg
split(dat, qg)

## End(Not run)
```

---

# rangeGate *Find most likely separation between positive and negative populations in 1D*

---

## Description

The function tries to find a reasonable split point between the two hypothetical cell populations "positive" and "negative".

## Usage

```
rangeGate(x, stain, alpha="min", sd=2, plot=FALSE, borderQuant=0.1,
absolute=TRUE, filterId="defaultRectangleGate", positive=TRUE,
refLine=NULL, simple = FALSE,...)

rangeFilter(stain, alpha="min", sd=2, borderQuant=0.1,
filterId="defaultRangeFilter")
```

## Arguments

| | |
|---|---|
| x | A flowSet or flowFrame. |
| stain | A character scalar giving the flow parameter for which to compute the separation. |
| alpha | A tuning parameter that controls the location of the split point between the two populations. This has to be a numeric in the range [0,1], where values closer to 0 will shift the split point closer to the negative population and values closer to 1 will shift towards the positive population. Additionally, the value of alpha can be "min", in which case the split point will be selected as the area of lowest local density between the two populations. |
| sd | For the case where there is only a single population, the algorithm falls back to esitmating the mode of this population and a robust measure of the variance of it distribution. The sd tuning parameter controls how far away from the mode the split point is set. |
| plot | Create a plot of the results of the computation. |
| borderQuant | Usualy the instrument is set up in a way that the positive population is somewhere on the high end of the measurement range and the negative population is on the low end. This parameter allows to disregard populations with mean values in the extreme quantiles of the data range. It's value should be in the range [0,1]. |
| absolute | Logical controling whether to classify a population (positive or negative) relative to the theoretical measurment range of the instrument or the actual range of the data. This can be set to TRUE if the alignment of the measurment range is not optimal and the bulk of the data is on one end of the theoretical range. |
| filterId | Character, the name assigned to the resulting filter. |
| positive | Create a range gate that includes the positive (TRUE) or the negative (FALSE) population. |
| refLine | Either NULL or a numeric of lenth 1. If NULL, this parameter is ignored. When it is set to a numeric, the minor sub-population (if any) below this reference line will be igored while determining the separator between positive and negative. |
| simple | logical scalar indicating whether to use a simple peak finding version of density1d algorithm. |
| ... | Further arguments. |

## Details

The algorithm first tries to identify high density regions in the data. If the input is a flowSet, density regions will be computed on the collapsed data, hence it should have been normalized before (see warpSet for one possible normalization technique). The high density regions are then clasified as positive and negative populations, based on their mean value in the theoretical (or absolute if argument absolute=TRUE) measurement range. In case there are only two high-density regions the lower one is usually clasified as the negative populations, however the heuristics in the algorithm will force the classification towards a positive population if the mean value is already very high. The absolute and borderQuant arguments can be used to control this behaviour. The split point

between populations will be drawn at the value of mimimum local density between the two populations, or, if the alpha argument is used, somewhere between the two populations where the value of alpha forces the point to be closer to the negative (0 - 0.5) or closer to the positive population (0.5 - 1).

If there is only a single high-density region, the algorithm will fall back to estimating the mode of the distribution (hubers) and a robust measure of it's variance and, in combination with the sd argument, set the split point somewhere in the right or left tail, depending on the classification of the region.

For more than two populations, the algorithm will still classify each population into positive and negative and compute the split point between those clusteres, similar to the two population case.

The rangeFilter class and constructor provide the means to treat rangeGate as regular flowCore filters.

## Value

A range gate, more explicitely an object of class rectangleGate.

## Methods

**%in%** signature(x = "flowFrame", table = "rangeFilter"): the work horse for doing the actual filtering. Internally, this simply calls the rangeGate function.

## Author(s)

Florian Hahne, Kyongryun Lee

## See Also

warpSet, rangeGate, rectangleGate

## Examples

```
library(flowCore)
data(GvHD)
dat <- GvHD[pData(GvHD)$Patient==10]
dat <- transform(dat, "FL4-H"=asinh(`FL4-H`), "FL3-H"=asinh(`FL3-H`))
rg <- rangeGate(dat, "FL4-H", plot=TRUE)
rg
split(dat, rg)

## Test rangeGate when settting refLine=0; it does not do anything since
## there is no sub-population below zero.
rangeGate(dat, "FL4-H", plot=FALSE, refLine=0)

rf <- rangeFilter("FL4-H")
filter(dat, rf)
```

SimulateMixture *Random Generation from a t Mixture Model with Box-Cox Transformation*

## Description

This function can be used to generate a sample from a multivariate $t$ mixture model with Box-Cox transformation.

## Usage

```
SimulateMixture(N, w, mu, sigma, nu = 4, lambda)
```

## Arguments

| | |
|---|---|
| N | The number of observations. |
| w | A vector of length $K$, containing the $K$ cluster proportions. |
| mu | A matrix of size $K \times P$, where $K$ is the number of clusters and $P$ is the dimension, containing the $K$ mean vectors. |
| sigma | An array of dimension $K \times P \times P$, containing the $K$ covariance matrices. |
| nu | The degrees of freedom used for the $t$ distribution. |
| lambda | The Box-Cox transformation parameter. If missing, the conventional $t$ distribution without transformation will be used. |

## Value

A matrix of size $N \times P$.

## Author(s)

Raphael Gottardo <<raph@stat.ubc.ca>>, Kenneth Lo <<c.lo@stat.ubc.ca>>

## See Also

flowClust

## Examples

```
### Number of components
K <- 5
### Dimension
p <- 2
### Number of observations
n <- 200
Mu <- matrix(runif(K*p, 0, 20), K, p)
Sigma <- array(0, c(K, p, p))
```

```
for (k in 1:K)
{
    Sigma[k,,][outer(1:p, 1:p, ">")] <- runif(p*(p-1)/2,-.1,.1)
    diag(Sigma[k,,]) <- runif(p,0,1)
    ### Make sigma positive definite
    Sigma[k,,] <- Sigma[k,,] %*% t(Sigma[k,,])
}

### Generate the weights
w <- rgamma(K,10,1)
w <- w/sum(w)

y <- SimulateMixture(n, w, Mu, Sigma, nu=4)
```

---

spillover-flowSet *Compute a spillover matrix from a flowSet*

---

## Description

Spillover information for a particular experiment is often obtained by running several tubes of beads or cells stained with a single color that can then be used to determine a spillover matrix for use with compensate.

When matching stain channels in x with the compensation controls, we provide a few options. If ordered, we assume the ordering of the channels in the flowSet object is the same as the ordering of the compensation-control samples. If regexpr, we use a regular expression to match the channel names with the names of each of the compensation control flowFrames (that is, sampleNames(x), which will typically be the filenames passed to read.FCS). By default, we must "guess" based on the largest statistic for the compensation control (i.e., the row).

Additionally, matching of channels to compensation control files can be accomplished using the spillover_match method, which allows the matches to be specified using a csv file. The flowSet returned by the spillover_match method should then be used as the x argument to spillover with prematched = TRUE.

## Usage

```
## S4 method for signature 'flowSet'
spillover(
  x,
  unstained = NULL,
  fsc = "FSC-A",
  ssc = "SSC-A",
  patt = NULL,
  method = "median",
  stain_match = c("intensity", "ordered", "regexpr"),
  useNormFilt = FALSE,
  prematched = FALSE,
```

```
  exact_match = FALSE
)
```

## Arguments

| | |
|---|---|
| x | A flowSet of compensation beads or cells |
| unstained | The name or index of the unstained negative control |
| fsc | The name or index of the forward scatter parameter |
| ssc | The name or index of the side scatter parameter |
| patt | An optional regular expression defining which parameters should be considered |
| method | The statistic to use for calculation. Traditionally, this has been the median so it is the default. The mean is sometimes more stable. |
| stain_match | Determines how the stain channels are matched with the compensation controls. See details. |
| useNormFilt | logical Indicating whether to apply a norm2Filter first before computing the spillover |
| prematched | a convenience argument specifying if the channels have already been matched by spillover_match. This will override the values of unstained and stain_match with unstained = "unstained" and stain_match = "regexpr". |
| exact_match | a logical specifying if we should use "regex" or "exact match" to match column names. The spillover_ng will pass exact_match and "regexpr" method will be over-ridden. |

## Details

The algorithm used is fairly simple. First, using the scatter parameters, we restrict ourselves to the most closely clustered population to reduce the amount of debris. The selected statistic is then calculated on all appropriate parameters and the unstained values swept out of the matrix. Every sample is then normalized to [0,1] with respect to the maximum value of the sample, giving the spillover in terms of a proportion of the primary channel intensity.

## Value

A matrix for each of the parameters

## Author(s)

B. Ellis, J. Wagner

## References

C. B. Bagwell & E. G. Adams (1993). Fluorescence spectral overlap compensation for any number of flow cytometry parameters. in: Annals of the New York Academy of Sciences, 677:167-184.

## See Also

compensate, spillover_match

spillover_match-flowSet *Construct a* flowSet *for use with* spillover *by matching channel names to compensation control filenames*

## Description

Spillover information for a particular experiment is often obtained by running several tubes of beads or cells stained with a single color that can then be used to determine a spillover matrix for use with compensate.

This method facilitates construction of a flowSet of compensation control flowFrames using a simple file linking filenames to channels. This resulting flowSet can then be used with spillover using the option prematched = TRUE.

Matching stain channels to compensation controls is done via a csv file (matchfile) with columns 'filename' and 'channel'. The 'channel' entries should exactly match the channel names in the FCS files. The 'filename' should be the FCS file name of each compensation control which should also be the corresponding sample name in the flowSet. There should also be one unstained control with the 'channel' entry of 'unstained'.

The method also allows for x to be missing if path is provided, pointing to a directory containing the control FCS files.

## Usage

```
## S4 method for signature 'flowSet'
spillover_match(x, fsc = "FSC-A", ssc = "SSC-A",
                                    matchfile = NULL, path)

## S4 method for signature 'missing'
spillover_match(x, fsc = "FSC-A", ssc = "SSC-A", matchfile, path)
```

## Arguments

| | |
|---|---|
| x | A flowSet of compensation beads or cells |
| fsc | The name or index of the forward scatter parameter |
| ssc | The name or index of the side scatter parameter |
| matchfile | The name or path of the csv file holding the compensation control file to channel matching information. |
| path | The name or path of the directory containing the control FCS files to be matched to channels by matchfile. |

### Value

A `flowSet` with the sample names of its `flowFrames` corresponding to the channels specified by the matchfile.

### Author(s)

B. Ellis, J. Wagner

### See Also

`compensate`, `spillover`

---

## spillover_ng-flowSet *Compute a spillover matrix from a* `flowSet`*, simplified API*

---

### Description

Spillover information for a particular experiment is often obtained by running several tubes of beads or cells stained with a single color that can then be used to determine a spillover matrix for use with `compensate`.

Matching stain channels to compensation controls is done via a matching csv file (at the path given by `matchfile`) with columns 'filename' and 'channel'. The 'channel' entries should exactly match the channel names in the FCS files. The 'filename' should be the FCS file name of each compensation control which should also be the corresponding sample name in the `flowSet`. There should also be one unstained control with the 'channel' entry of 'unstained'.

The method also allows for `x` to be missing if `path` is provided, pointing to a directory containing the control FCS files.

By default, pregating is always done on the channels using this API, and the mode of the channel is used to compute the spillover matrix. FSC and SSC channels can be provided to allow a pregating on (approximately) a population in the FSC and SSC dimensions. Also by default, a `norm2Filter` is applied before computing the spillover. These defaults can be overridden using the `pregate`, `method`, and `useNormFilt` arguments.

### Usage

```
## S4 method for signature 'flowSet'
spillover_ng(x, fsc = "FSC-A", ssc = "SSC-A",
             plot = FALSE, matchfile, path,
           useNormFilt = TRUE, patt = NULL, pregate = TRUE, method = "mode", ...)
## S4 method for signature 'missing'
spillover_ng(x, fsc = "FSC-A", ssc = "SSC-A",
             plot = FALSE, matchfile, path,
           useNormFilt = TRUE, patt = NULL, pregate = TRUE, method = "mode", ...)
```

```
## S4 method for signature 'missing'
spillover_ng(
  x,
  fsc = "FSC-A",
  ssc = "SSC-A",
  plot = FALSE,
  matchfile,
  path,
  useNormFilt = TRUE,
  patt = NULL,
  pregate = TRUE,
  method = "mode",
  ...
)
```

## Arguments

| | |
|---|---|
| x | A flowSet of compensation beads or cells |
| fsc | The name or index of the forward scatter parameter |
| ssc | The name or index of the side scatter parameter |
| plot | logical. Plots the kernel density for each channel when pregating. Displays the gate used. If pregate is set to FALSE, this argument is ignored. |
| matchfile | Name of the csv file holding the compensation control file to channel matching information. |
| path | A path to a directory containing the control files, to be used if x is not provided. |
| useNormFilt | logical Indicating whether to apply a norm2Filter first before computing the spillover |
| patt | An optional regular expression defining which parameters should be considered |
| pregate | logical Indicating whether to pregate using link{rangeGate} before computing the spillover |
| method | The statistic to use for calculation. Traditionally, this has been the median so it is the default. The mean is sometimes more stable. |
| ... | Additional arguments passed to rangeGate. |

## Details

The algorithm used is fairly simple. First, using the scatter parameters, we restrict ourselves to the most closely clustered population to reduce the amount of debris. The selected statistic is then calculated on all appropriate parameters and the unstained values swept out of the matrix. Every sample is then normalized to [0,1] with respect to the maximum value of the sample, giving the spillover in terms of a proportion of the primary channel intensity.

## Value

A matrix for each of the parameters

## Author(s)

B. Ellis

## References

C. B. Bagwell & E. G. Adams (1993). Fluorescence spectral overlap compensation for any number of flow cytometry parameters. in: Annals of the New York Academy of Sciences, 677:167-184.

## See Also

compensate, spillover

---

# warpSet *Normalization based on landmark registration*

---

## Description

This function will perform a normalization of flow cytometry data based on warping functions computed on high-density region landmarks for individual flow channels.

## Usage

```
warpSet(x, ...)

## S3 method for class 'cytoset'
warpSet(
  x,
  stains,
  grouping = NULL,
  subsample = NULL,
  peakNr = NULL,
  clipRange = 0.01,
  nbreaks = 11,
  fres,
  bwFac = 2,
  warpFuns = FALSE,
  target = NULL,
  chunksize = 10,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A flowSet. |
| ... | Further arguments that are passed on to landmarkreg. |
| stains | A character vector of flow parameters in x to be normalized. |

grouping — A character indicating one of the phenotypic variables in the phenoData slot of x used as a grouping factor. The within-group and between-group variance is computed and a warning is issued in case the latter is bigger than the former, indicating the likely removal of signal by the normalization procedure. landmarkreg.

subsample — Numeric. Reduce the number of events in each flowSet by sub sampling for all density estimation steps and the calculation of the warping functions. This can increase computation time for large data sets, however it might reduce the accuracy of the density estimates. To be used with care.

peakNr — Numeric scalar. Force a fixed number of peaks to use for the normalization.

clipRange — Only use peaks within a clipped data range. Essentially, the number indicates the percent of clipping on both sides of the data range, e.g. `min(x) - 0.01 * diff(range(x))`.

nbreaks — The number of spline sections used to approximate the data. Higher values produce more accurate results, however this comes with the cost of increaseqd computing times. For most data, the default setting is good enough.

fres — A named list of filterResultList objects. This can be used to speed up the process since the curv1Filter step can take quite some time.

bwFac — Numeric of lenght 1 used to set the bandwidth factor by curv1Filter for smoothing of the density estimate.

warpFuns — Logical indcating whether to return the normalized flowSet or a list of warping functions.

target — Character vector specifying the target sample to which other samples in the flowSet should be normalized. If NULL, then the mean of the peaks is used.

chunksize — an integer. For a memory-efficient implementation of normalization, chunksize can be set to perform normalization on chunks of the data of size chunksize

## Details

Normalization is achived by first identifying high-density regions (landmarks) for each flowFrame in the flowSet for a single channel and subsequently by computing warping functions for each flowFrame that best align these landmarks. This is based on the algorithm implemented in the landmarkreg function in the fda package. An intermediate step classifies the high-density regions, see landmarkMatrix for details.

Please note that this normalization is on a channel-by-channel basis. Multiple channels are normalized in a loop.

## Value

The normalized flowSet if warpFuns is FALSE, otherwise a list of warping functions. Additional inforamtion is attached as the warping attribute to the flowSet in form of a list.

## Note

We currently use a patched fda version.

## Author(s)

Florian Hahne

## References

J.O. Ramsay and B.W. Silverman: Applied Functional Data Analysis, Springer 2002

## See Also

curv1Filter landmarkMatrix

## Examples

```
## Not run:
library(flowCore)
data(ITN)
dat <- transform(ITN, "CD4"=asinh(CD4), "CD3"=asinh(CD3), "CD8"=asinh(CD8))
lg <- lymphGate(dat, channels=c("CD3", "SSC"), preselection="CD4",scale=1.5)
dat <- Subset(dat, lg)
datr <- warpSet(dat, "CD8", grouping="GroupID")
if(require(flowViz)){
  d1 <- densityplot(~CD8, dat, main="original", filter=curv1Filter("CD8"))
  d2 <- densityplot(~CD8, datr, main="normalized", filter=curv1Filter("CD8"))
  plot(d1, split=c(1,1,2,1))
  plot(d2, split=c(2,1,2,1), newpage=FALSE)
}

## End(Not run)
```

# Index


∗ **aplot**
plotPrior, 39
∗ **classes**
curv1Filter-class, 8
curv2Filter-class, 10
lymphFilter-class, 29
norm2Filter-class, 32
∗ **datagen**
SimulateMixture, 46
∗ **datasets**
BackGating, 5
ITN, 27
∗ **dplot**
plotPrior, 39
∗ **methods**
curv1Filter-class, 8
curv2Filter-class, 10
norm2Filter-class, 32
normalize-methods, 35
spillover-flowSet, 47
spillover_match-flowSet, 49
spillover_ng-flowSet, 50
∗ **misc**
autoGate, 4
binByRef, 5
calcPBChiSquare, 6
calcPearsonChi, 7
idFeaturesByBackgating, 23
lymphFilter-class, 29
plotBins, 38
proBin, 40
∗ **package**
flowStats-package, 3
%in%,flowFrame,curv1Filter-method *(curv1Filter-class)*, 8
%in%,flowFrame,curv2Filter-method *(curv2Filter-class)*, 10
%in%,flowFrame,lymphFilter-method *(lymphFilter-class)*, 29
%in%,flowFrame,norm2Filter-method *(norm2Filter-class)*, 32
%in%,flowFrame,rangeFilter-method *(rangeGate)*, 43

addName,curv1Filter,character-method, 3
addName,curv1Filter,logical-method *(addName,curv1Filter,character-method)*, 3
addName,curv2Filter,character-method *(addName,curv1Filter,character-method)*, 3
addName,curv2Filter,logical-method *(addName,curv1Filter,character-method)*, 3
autoGate, 4

BackGating, 5, *24*
binByRef, 5, *41*

calcPBChiSquare, 6, *7*, *39*
calcPearsonChi, 7, *39*
compensate, *47–50*, *52*
concreteFilter, *9*, *11*, *30*, *33*
cov.rob, *34*
CovMcd, *34*
curv1Filter, *11, 12*, *28–30*, *53, 54*
curv1Filter *(curv1Filter-class)*, 8
curv1Filter-class, 8
curv2Filter, *9*, *11*
curv2Filter *(curv2Filter-class)*, 10
curv2Filter-class, 10
curvPeaks, 12

density, *13*
density1d, 13, *42*
diana, *23, 24*

ellipse, 16
ellipsoidGate, *30*

fda, *28*, *53*
fdPar, 16
filter, *8–11*, *30*, *32–34*
filterResult, *4*
filterResultList, *28*
flowClust, *46*
flowClust2Prior, 17
flowFrame, *6*, *8, 9*, *11*, *13, 14*, *18*, *20, 21*, *28*, *33*, *39, 40*, *42*, *44*, *53*
flowFrames, *27*
flowSet, *4*, *9*, *11*, *14*, *19*, *21*, *27–29*, *33*, *37*, *42*, *44*, *47*, *52*
flowStats *(*flowStats-package*)*, 3
flowStats-package, 3
flowViz, *9*, *11*, *34*

gate_singlet, 17
gaussNorm, 19
gpaSet, 20, *24*, *26*

hubers, *15*, *45*

idFeatures *(*idFeaturesByBackgating*)*, 23
idFeaturesByBackgating, 23
iProcrustes, 25
ITN, 27

landmarkMatrix, *13*, 28, *53, 54*
landmarkreg, *28, 29*, *53*
logicalFilterResult, *33*
lymphFilter *(*lymphFilter-class*)*, 29
lymphFilter-class, 29
lymphGate, *4, 5*
lymphGate *(*lymphFilter-class*)*, 29

mkPrior, 31
mkPrior,list,flowSet,ANY,missing-method *(*mkPrior*)*, 31
mkPrior,list,flowSet,missing,missing-method *(*mkPrior*)*, 31
mkPrior,missing,flowFrame,missing,missing-method *(*mkPrior*)*, 31
mkPrior,missing,flowSet,ANY,missing-method *(*mkPrior*)*, 31
mkPrior,polygonGate,flowFrame,missing,missing-method *(*mkPrior*)*, 31
mkPrior,polygonGate,flowFrame,numeric,matrix-method *(*mkPrior*)*, 31
mkPrior,rectangleGate,flowFrame,missing,missing-method *(*mkPrior*)*, 31
mkPrior,rectangleGate,flowFrame,numeric,matrix-method *(*mkPrior*)*, 31
multipleFilterResult, *8*, *11*, *13*

ncdfFlowSet, *36*
norm2Filter, *4, 5*, *29, 30*, *33*, *48*, *50, 51*
norm2Filter *(*norm2Filter-class*)*, 32
norm2Filter-class, 32
normalize *(*normalize-methods*)*, 35
normalize,GatingSet,missing-method *(*normalize-methods*)*, 35
normalize,GatingSetInternal,missing-method *(*normalize-methods*)*, 35
normalize-methods, 35
normQA, 36

oneDGate *(*rangeGate*)*, 43
overton_like, 37

parameterFilter, *9*, *11*, *30*, *33*
plotBins, *6*, 38, *41*
plotPrior, 39
polygonGate, *19*
proBin, *6, 7*, *39*, 40

quadGate, *43*
quadrantGate, 42

rangeFilter *(*rangeGate*)*, 43
rangeFilter-class *(*rangeGate*)*, 43
rangeGate, *13*, *15*, *42*, 43, *43*, *45*, *51*
read.FCS, *47*
rectangleGate, *4*, *29*, *45*
rlm, *17, 18*

show,curv1Filter-method *(*curv1Filter-class*)*, 8
show,curv2Filter-method *(*curv2Filter-class*)*, 10
show,norm2Filter-method *(*norm2Filter-class*)*, 32
SimulateMixture, 46
singletGate *(*gate_singlet*)*, 17
spillover, *49, 50*, *52*
spillover *(*spillover-flowSet*)*, 47
spillover,flowSet-method *(*spillover-flowSet*)*, 47
spillover-flowSet, 47
spillover_match, *47, 48*

spillover_match
(spillover_match-flowSet), 49
spillover_match,flowSet-method
(spillover_match-flowSet), 49
spillover_match,missing-method
(spillover_match-flowSet), 49
spillover_match-flowSet, 49
spillover_ng (spillover_ng-flowSet), 50
spillover_ng,flowSet-method
(spillover_ng-flowSet), 50
spillover_ng,missing-method
(spillover_ng-flowSet), 50
spillover_ng-flowSet, 50
split, *9*, *11*, *34*
Subset, *34*
summarizeFilter,multipleFilterResult,curv1Filter-method
(curv1Filter-class), 8
summarizeFilter,multipleFilterResult,curv2Filter-method
(curv2Filter-class), 10

warpSet, *15*, *29*, *44*, *45*, 52
warpSetGS (warpSet), 52
warpSetNCDF (warpSet), 52
warpSetNCDFLowMem (warpSet), 52