Package ‘openCyto’

February 25, 2025

Type Package

Title Hierarchical Gating Pipeline for flow cytometry data

Version 2.18.0

Date 2012-06-11

Author Mike Jiang, John Ramey, Greg Finak, Raphael Gottardo

Maintainer Mike Jiang <mike@ozette.com>

Description This package is designed to facilitate the automated gating methods in sequential way to mimic the manual gating strategy.

License AGPL-3.0-only

LazyLoad yes

Imports methods,Biobase,BiocGenerics,flowCore(>= 1.99.17),flowViz,ncdfFlow(>= 2.11.34),flowWorkspace(>= 3.99.1),flowClust(>= 3.11.4),RBGL,graph,data.table,RColorBrewer

Suggests flowWorkspaceData, knitr, rmarkdown, markdown, testthat, utils, tools, parallel, ggcyto, CytoML, flowStats(>= 4.5.2), MASS

biocViews ImmunoOncology, FlowCytometry, DataImport, Preprocessing, DataRepresentation

Encoding UTF-8

VignetteBuilder knitr

LinkingTo cpp11, BH

RoxygenNote 7.2.1

git_url https://git.bioconductor.org/packages/openCyto

git_branch RELEASE_3_20

git_last_commit dc6d560

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-02-24

Contents

`as.data.table` *convert a gatingTemplate object to a data.table*

Description

It is the inverse function of gatingTemplate constructor.

Usage

```
## S3 method for class 'gatingTemplate'
as.data.table(x, keep.rownames = FALSE)
```

Arguments

`x`	gatingTemplate object
`keep.rownames`	not used

Value

a data.table

`boolMethod-class` *A class to represent a boolean gating method.*

Description

It extends `refGate` class.

CytoExploreR_exports *CytoExploreR exports*

Description

Exported wrappers of internal functions for use by CytoExploreR

Usage

```
CytoExploreR_.argDeparser(args, split = TRUE)

CytoExploreR_.preprocess_csv(dt, strict = TRUE)
```

dims,gtMethod-method *get gating method dimensions*

Description

get gating method dimensions

Usage

```
## S4 method for signature 'gtMethod'
dims(x)
```

Arguments

x	gtMethod

dummyMethod-class *A class to represent a dummy gating method that does nothing but serves as reference to be refered by other population*

Description

It is generated automatically by the csv template preprocessing to handle the gating function that returns multiple gates.

`fast_rlm` *robust linear model using an M estimator*

Description

rewritten in c++, till eval stats::lm.wfit r function in underlying cpp11 code It is internally used for singletGate, thus its output format may not be generic enough for common model fitting . e.g. it doesn't take formula as input

Usage

```
fast_rlm(x, y, maxit = 20)
```

Arguments

`x`	matrix with first column as weight (default can be 1s), the rest columns are predict variable
`y`	numeric vector as response
`maxit`	maximum iterations

`fcEllipsoidGate` *constuctor for* `fcEllipsoidGate`

Description

constuctor for `fcEllipsoidGate`

Usage

```
fcEllipsoidGate(x, priors, posts)
```

Arguments

`x`	a `ellipsoidGate` object
`priors`	a `list` storing priors
`posts`	a `list` storing posteriors

`fcEllipsoidGate-class` *a concrete class that reprents the ellipsoidGate generated by flowClust*

Description

It stores priors and posteriors as well as the actual ellipsoidGate.

`fcFilter-class`	*a virtual class that represents the gating result generated by flowClust gating function*

Description

Bascially it extends flowCore 'filter classes to have extra slot to store priors and posteriors

`fcFilterList`	*constuctor for* `fcFilterList`

Description

constuctor for `fcFilterList`

Usage

```
fcFilterList(x)
```

Arguments

`x`	`list` of `fcFilter` (i.e. `fcPolygonGate` or `fcRectangleGate`)

`fcFilterList-class`	*a class that extends* `filterList` *class.*

Description

Each filter in the filterList must extends the `fcFilter` class

`fcPolygonGate`	*constuctor for* `fcPolygonGate`

Description

constuctor for `fcPolygonGate`

Usage

```
fcPolygonGate(x, priors, posts)
```

Arguments

`x`	a `polygonGate` object
`priors`	a `list` storing priors
`posts`	a `list` storing posteriors

`fcPolygonGate-class` *a concrete class that reprents the polygonGate generated by flowClust*

Description

It stores priors and posteriors as well as the actual polygonGate.

`fcRectangleGate` *constuctor for* `fcRectangleGate`

Description

constuctor for `fcRectangleGate`

Usage

```
fcRectangleGate(x, priors, posts)
```

Arguments

x	a `rectangleGate` object
priors	a `list` storing priors
posts	a `list` storing posteriors

`fcRectangleGate-class` *a concrete class that reprents the rectangleGate generated by flowClust*

Description

It stores priors and posteriors as well as the actual rectangleGate.

`fcTree` *constructor of* `fcTree`

Description

It adds an extra node data slot "fList"(which is a `filterList` object) to the `gatingTemplate`

Usage

```
fcTree(gt)
```

Arguments

gt	a `gatingTemplate` object

fcTree-class *A class to represent a flowClust tree.*

Description

It is a graphNEL used as a container to store priors and posteriors for each flowClust gate that can be visualized for the purpose of fine-tunning parameters for flowClust algorithm

gate_flowclust_1d *Applies flowClust to 1 feature to determine a cutpoint between the minimum cluster and all other clusters.*

Description

We cluster the observations in `fr` into `K` clusters.

Usage

```
gate_flowclust_1d(
  fr,
  params,
  filterId = "",
  K = NULL,
  trans = 0,
  min.count = -1,
  max.count = -1,
  nstart = 1,
  prior = NULL,
  criterion = c("BIC", "ICL"),
  cutpoint_method = c("boundary", "min_density", "quantile", "posterior_mean",
    "prior_density"),
  neg_cluster = 1,
  cutpoint_min = NULL,
  cutpoint_max = NULL,
  min = NULL,
  max = NULL,
  quantile = 0.99,
  quantile_interval = c(0, 10),
  plot = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

`fr`	a `flowFrame` object
`params`	`character` channel to be gated on
`filterId`	A `character` string that identifies the filter created.

K — the number of clusters to find

trans, min.count, max.count, nstart — some flowClust parameters. see flowClust

prior — list of prior parameters for the Bayesian flowClust. If NULL, no prior is used.

criterion — a character string stating the criterion used to choose the best model. May take either "BIC" or "ICL". This argument is only relevant when K is NULL or if length(K) > 1. The value selected is passed to flowClust.

cutpoint_method — How should the cutpoint be chosen from the fitted flowClust model? See Details.

neg_cluster — integer. The index of the negative cluster. The cutpoint is computed between clusters neg_cluster and neg_cluster + 1.

cutpoint_min — numeric value that sets a minimum thresold for the cutpoint. If a value is provided, any cutpoint below this value will be set to the given minimum value. If NULL (default), there is no minimum cutpoint value.

cutpoint_max — numeric value that sets a maximum thresold for the cutpoint. If a value is provided, any cutpoint above this value will be set to the given maximum value. If NULL (default), there is no maximum cutpoint value.

min — a numeric value that sets the lower bound for data filtering. If NULL (default), no truncation is applied.

max — a numeric value that sets the upper bound for data filtering. If NULL (default), no truncation is applied.

quantile — the quantile for which we will find the cutpoint using the quantile cutpoint_method. If the cutpoint_method is not set to quantile, this argument is ignored.

quantile_interval — a vector of length 2 containing the end-points of the interval of values to find the quantile cutpoint. If the cutpoint_method is not set to quantile, this argument is ignored.

plot — logical value indicating that the fitted flowClust model should be plotted along with the cutpoint

debug — logical indicating whether to carry the prior and posterious with the gate for debugging purpose. Default is FALSE.

... — additional arguments that are passed to flowClust

Details

By default, the cutpoint is chosen to be the boundary of the first two clusters. That is, between the first two cluster centroids, we find the midpoint between the largest observation from the first cluster and the smallest observations from the second cluster. Alternatively, if the cutpoint_method is min_density, then the cutpoint is the point at which the density between the first and second smallest cluster centroids is minimum.

Value

a rectangleGate object consisting of all values beyond the cutpoint calculated

Examples

```
## Not run:
 gate <- gate_flowclust_1d(fr, params = "APC-A", K =2) # fr is a flowFrame

## End(Not run)
```

`gate_flowclust_2d` *Automatic identification of a population of interest via flowClust based on two markers*

Description

We cluster the observations in `fr` into `K` clusters. We set the cutpoint to be the point at which the density between the first and second smallest cluster centroids is minimum.

Usage

```
gate_flowclust_2d(
  fr,
  xChannel,
  yChannel,
  filterId = "",
  K = 2,
  usePrior = "no",
  prior = list(NA),
  trans = 0,
  min.count = -1,
  max.count = -1,
  nstart = 1,
  plot = FALSE,
  target = NULL,
  transitional = FALSE,
  quantile = 0.9,
  translation = 0.25,
  transitional_angle = NULL,
  min = NULL,
  max = NULL,
  ...
)
```

Arguments

`fr`	a `flowFrame` object
`xChannel, yChannel`	`character` specifying channels to be gated on
`filterId`	A `character` string that identifies the filter created.
`K`	the number of clusters to find
`usePrior`	Should we use the Bayesian version of `flowClust`? Answers are "yes", "no", or "vague". The answer is passed along to `flowClust`.

prior	list of prior parameters for the Bayesian version of flowClust. If usePrior is set to no, then the list is unused.
trans, min.count, max.count, nstart	some flowClust parameters. see flowClust
plot	a logical value indicating if the fitted mixture model should be plotted. By default, no.
target	a numeric vector of length 2 (number of dimensions) containing the location of the cluster of interest. See details.
transitional	logical value indicating if a transitional gate should be constructed from the target flowClust cluster. By default, no.
quantile	the contour level of the target cluster from the flowClust fit to construct the gate
translation	a numeric value between 0 and 1 used to position a transitional gate if transitional = TRUE. This argument is ignored if transitional = FALSE. See details
transitional_angle	the angle (in radians) of the transitional gate. It is also used to determine which quadrant the final gate resides in. See details. Ignored if transitional = FALSE.
min	A vector of length 2. Truncate observations less than this minimum value. The first value truncates the xChannel, and the second value truncates the yChannel. By default, this vector is NULL and is ignored.
max	A vector of length 2. Truncate observations greater than this maximum value. The first value truncates the xChannel, and the second value truncates the yChannel. By default, this vector is NULL and is ignored.
...	additional arguments that are passed to flowClust

Details

The cluster for the population of interest is selected as the one with cluster centroid nearest the target in Euclidean distance. By default, the largest cluster (i.e., the cluster with the largest proportion of observations) is selected as the population of interest.

We also provide the option of constructing a transitional gate from the selected population of interest. The location of the gate can be controlled with the translation argument, which translates the gate along the major axis of the targest cluster as a function of the appropriate chi-squared coefficient. The larger translation is, the more gate is shifted in a positive direction. Furthermore, the width of the transitional gate can be controlled with the quantile argument.

The direction of the transitional gate can be controlled with the transitional_angle argument. By default, it is NULL, and we use the eigenvector of the target cluster that points towards the first quadrant (has positive slope). If transitional_angle is specified, we rotate the eigenvectors so that the angle between the x-axis (with the cluster centroid as the origin) and the major eigenvector (i.e., the eigenvector with the larger eigenvalue) is transitional_angle. So based on range that the angle falls in, the final rectangleGate will be constructed at the corresponding quadrant. i.e. Clockwise, [0,pi/2] UR, (pi/2, pi] LR, (pi, 3/2 * pi] LL, (3/2 * pi, 2 * pi] UL

Value

a polygonGate object containing the contour (ellipse) for 2D gating.

Examples

```
## Not run:
 gate <- gate_flowclust_2d(fr, xChannel = "FSC-A", xChannel = "SSC-A", K = 3) # fr is a flowFrame

## End(Not run)
```

`gate_mindensity` *Determines a cutpoint as the minimum point of a kernel density estimate between two peaks*

Description

We fit a kernel density estimator to the cells in the `flowFrame` and identify the two largest peaks. We then select as the cutpoint the value at which the minimum density is attained between the two peaks of interest.

Usage

```
gate_mindensity(
  fr,
  channel,
  filterId = "",
  positive = TRUE,
  gate_range = NULL,
  min = NULL,
  max = NULL,
  peaks = NULL,
  ...
)
```

Arguments

`fr`	a `flowFrame` object
`channel`	TODO
`filterId`	TODO
`positive`	If `TRUE`, then the gate consists of the entire real line to the right of the cutpoint. Otherwise, the gate is the entire real line to the left of the cutpoint. (Default: `TRUE`)
`gate_range`	numeric vector of length 2. If given, this sets the bounds on the gate applied. If no gate is found within this range, we set the gate to the minimum value within this range if `positive` is `TRUE` and the maximum value of the range otherwise.
`min`	a numeric value that sets the lower boundary for data filtering
`max`	a numeric value that sets the upper boundary for data filtering
`peaks`	`numeric` vector. If not given , then perform peak detection first by .find_peaks
`...`	Additional arguments for peak detection.

Details

In the default case, the two peaks of interest are the two largest peaks obtained from the `link{density}` function.

In the special case that there is only one peak, we are conservative and set the cutpoint as the `min(x)` if `positive` is `TRUE`, and the `max(x)` otherwise.

Value

a `rectangleGate` object based on the minimum density cutpoint

Examples

```
## Not run:
 gate <- gate_mindensity(fr, channel = "APC-A") # fr is a flowFrame

## End(Not run)
```

`gate_mindensity2` *An improved version of mindensity used to determines a cutpoint as the minimum point of a kernel density estimate between two peaks.*

Description

Analogous to the original openCyto::mindensity(), mindensity2 operates on a standard flowFrame. Its behavior is closely modeled on the original mindensity() whenever possible. However, the underlying peak-finding algorithm (improvedMindensity) behaves significantly differently.

Usage

```
gate_mindensity2(
  fr,
  channel,
  filterId = "",
  gate_range = NULL,
  min = NULL,
  max = NULL,
  peaks = NULL,
  ...
)
```

Arguments

`fr`	a `flowFrame` object
`channel`	the channel to operate on
`filterId`	a name to refer to this filter
`gate_range`	numeric vector of length 2. If given, this sets the bounds on the gate applied.
`min`	a numeric value that sets the lower boundary for data filtering
`max`	a numeric value that sets the upper boundary for data filtering
`peaks`	`numeric` vector. If not given , then perform peak detection first by .find_peaks
`...`	Additional arguments for peak detection.

Value

a `rectangleGate` object based on the minimum density cutpoint

Author(s)

Greg Finak, Phu T. Van

Examples

```
## Not run:
 gate <- gate_mindensity2(fr, channel = "APC-A") # fr is a flowFrame

## End(Not run)
```

`gate_quad_sequential` *sequential quadrant gating function*

Description

The order of 1d-gating is determined so that the gates better capture the distributions of flow data.

Usage

```
gate_quad_sequential(fr, channels, gFunc, min = NULL, max = NULL, ...)
```

Arguments

`fr`	`flowFrame`
`channels`	`character` two channels used for gating
`gFunc`	the name of the 1d-gating function to be used for either dimension
`min`	a numeric vector that sets the lower bounds for data filtering
`max`	a numeric vector that sets the upper bounds for data filtering
`...`	other arguments passed to `.find_peak` (e.g. 'num_peaks' and 'adjust'). see tailgate

Value

a `filters` that contains four rectangleGates `gate_quad_tmix` *quadGate based on flowClust::tmixFiler*

Description

This gating method identifies two quadrants (first, and third quadrants) by fitting the data with tmixture model. It is particually useful when the two markers are not well resolved thus the regular quadGate method based on 1d gating will not find the perfect cut points on both dimensions.

Usage

```
gate_quad_tmix(
  fr,
  channels,
  K,
  usePrior = "no",
  prior = list(NA),
  quantile1 = 0.8,
  quantile3 = 0.8,
  trans = 0,
  plot = FALSE,
  ...
)
```

Arguments

`fr`	`flowFrame`
`channels`	`character` vector specifies two channels
`K`	see gate_flowclust_2d
`usePrior`	see gate_flowclust_2d
`prior`	see gate_flowclust_2d
`quantile1`	`numeric` specifies the quantile level(see 'level' in flowClust) for the first quadrant (x-y+)
`quantile3`	`numeric` specifies the quantile level see 'level' in flowClust for third quadrant (x+y-)
`trans`	see gate_flowclust_2d
`plot`	logical whether to plot flowClust clustering results
`...`	other arguments passed to flowClust

Value

a `filters` object that contains four `polygonGates` following the order of (-+,++,+-,–)

`gate_quantile` *Determine the cutpoint by the events quantile.*

Description

It is possible that the cutpoint calculated by quantile function may not produce the exact the probability set by 'probs' argument if there are not enough cell events to reach that precision. Sometime the difference could be significant.

Usage

```
gate_quantile(
  fr,
  channel,
  probs = 0.999,
  plot = FALSE,
  filterId = "",
  min = NULL,
  max = NULL,
  ...
)
```

Arguments

`fr`	a `flowFrame` object
`channel`	the channel from which the cytokine gate is constructed
`probs`	probabilities passed to 'stats::quantile' function.
`plot`	whether to plot the gate result
`filterId`	the name of the filter
`min`	a numeric value that sets the lower boundary for data filtering
`max`	a numeric value that sets the upper boundary for data filtering
`...`	additional arguments passed to 'stats::quantile' function.

Value

a `rectangleGate`

Examples

```
## Not run:
 gate <- gate_quantile(fr, Channel = "APC-A", probs = 0.995) # fr is a flowFrame

## End(Not run)
```

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. 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,
  ...
)
```

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
`...`	additional arguments (not used)

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

`gatingTemplate-class` *a class storing the gating method and population information in a graphNEL object*

Description

Each cell population is stored in graph node and is connected with its parent population or its reference node for boolGate or refGate.

It parses the csv file that specifies the gating scheme for a particular staining pannel.

Usage

```
gatingTemplate(x, ...)

## S4 method for signature 'character'
gatingTemplate(
  x,
  name = "default",
  strict = TRUE,
  strip_extra_quotes = FALSE,
  ...
)

## S4 method for signature 'data.table'
gatingTemplate(
  x,
  name = "default",
  strict = TRUE,
  strip_extra_quotes = FALSE,
  ...
)
```

Arguments

`x`	`character` csv file name or a `data.table`
`...`	other arguments passed to `data.table::fread`
`name`	`character` the label of the gating template

strict
: `logical` whether to perform validity check(special characters) on the alias column. By default it is(and should be) turned on for the regular template parsing. But sometime it is useful to turned it off to bypass the check for the dummy nodes(e.g. the csv template generated by 'gh_generate_template' with some existing boolean gates that has '!' or ':' symbol).

strip_extra_quotes
: `logical` Extra quotes are added to strings by fread. This causes problems with parsing R strings to expressions in some cases. Default FALSE for usual behaviour. TRUE should be passed if parsing gating_args fails.

Details

This csv must have the following columns:

'alias': a name used label the cell population, the path composed by the alias and its precedent nodes (e.g. /root/A/B/alias) has to be uniquely identifiable. So alias can not contain '/' character, which is reserved as path delimiter.

'pop': population patterns of '+/-' or '+/-+/-', which tells the algorithm which side (postive or negative) of 1d gate or which quadrant of 2d gate to be kept.

'parent': the parent population alias, its path has to be uniquely identifiable.

'dims': characters seperated by comma specifying the dimensions(1d or 2d) used for gating. It can be either channel name or stained marker name (or the substrings of channel/marker names as long as they are uniquely identifiable.).

'gating_method': the name of the gating function (e.g. 'flowClust'). It is invoked by a wrapper function that has the identical function name prefixed with a dot.(e.g. '.flowClust')

'gating_args': the named arguments passed to gating function (Note that double quotes are often used as text delimiter by some csv editors. So try to use single quote instead if needed.)

'collapseDataForGating': When TRUE, data is collapsed (within groups if 'groupBy' specified) before gating and the gate is replicated across collapsed samples. When set FALSE (or blank),then 'groupBy' argument is only used by 'preprocessing' and ignored by gating.

'groupBy': If given, samples are split into groups by the unique combinations of study variable (i.e. column names of pData,e.g."PTID:VISITNO"). when split is numeric, then samples are grouped by every N samples

'preprocessing_method': the name of the preprocessing function(e.g. 'prior_flowclust'). It is invoked by a wrapper function that has the identical function name prefixed with a dot.(e.g. '.prior_flowclust') the preprocessing results are then passed to gating wrapper function through 'pps_res' argument.

'preprocessing_args': the named arguments passed to preprocessing function.

Examples

```
## Not run:
 gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
  plot(gt)

## End(Not run)
```

getGate,fcTree,character-method

get gates saved in `fcTree`

Description

get gates saved in `fcTree`

Usage

```
## S4 method for signature 'fcTree,character'
getGate(obj, y, ...)
```

Arguments

`obj`	`fcTree`
`y`	`character` node name
`...`	other arguments (not used)

getNodes,fcTree-method

get nodes from `fcTree`

Description

get nodes from `fcTree`

Usage

```
## S4 method for signature 'fcTree'
getNodes(x, y)
```

Arguments

`x`	`fcTree`
`y`	`character` node name

gh_generate_template — *generate a partially complete csv template from the existing gating hierarchy*

Description

To ease the process of replicating the existing (usually a manual one) gating schemes, this function populate an empty gating template with the 'alias', 'pop', 'parent' and 'dims' columns that exacted from an `GatingHierarchy`, and leave the other columns (e.g. 'gating_method') blank. So users can make changes to that template instead of writing from scratch.

Usage

```
gh_generate_template(gh)
```

Arguments

`gh`	a `GatingHierarchy` likely parsed from a xml workspace

Value

a gating template in `data.frame` format that requires further edition after output to csv

Examples

```
library(flowWorkspace)
dataDir <- system.file("extdata",package="flowWorkspaceData")
gs <- load_gs(list.files(dataDir, pattern = "gs_manual",full = TRUE))
gh_generate_template(gs[[1]])
```

groupBy,gtMethod-method — *get the grouping variable for the gating method*

Description

When specified, the flow data is grouped by the grouping variable (column names in pData). Within each group, when `isCollapse` is set to TRUE, the gating method is applied to the collapsed data. Otherwise, it is done indepentently for each indiviudal sample(flowFrame). Grouping variable is also used by `preprocessing` `method`.

Usage

```
## S4 method for signature 'gtMethod'
groupBy(object)
```

Arguments

`object`	`gtMethod`

gs_add_gating_method *apply a gating method to the* GatingSet

Description

When interacting with the existing gated data, this function provides an alternative way to interact with the GatingSet by supplying the gating description directly through arguments without the need to write the complete csv gating template.

Usage

```
gs_add_gating_method(
  gs,
  alias = "*",
  pop = "+",
  parent,
  dims = NA,
  gating_method,
  gating_args = NA,
  collapseDataForGating = NA,
  groupBy = NA,
  preprocessing_method = NA,
  preprocessing_args = NA,
  strip_extra_quotes = FALSE,
  ...
)
```

Arguments

`gs` GatingSet or GatingSetList

`alias, pop, parent, dims, gating_method, gating_args, collapseDataForGating, groupBy, preprocessing_method, preprocessing_args`
see details in gatingTemplate

`strip_extra_quotes`
`logical` Extra quotes are added to strings by fread. This causes problems with parsing R strings to expressions in some cases. Default FALSE for usual behaviour. TRUE should be passed if parsing gating_args fails.

`...` other arguments

- mc.cores passed to `multicore` package for parallel computing
- parallel_type `character` specifying the parallel type. The valid options are "none", "multicore", "cluster".
- cl `cluster` object passed to `parallel` package (when `parallel_type` is "cluster")

Details

Calls to `gs_add_gating_method` can also be easily reversed with `gs_remove_gating_method`. Note, however, that it is not possible to differentiate between different `GatingSet` objects loaded from the same directory with `load_gs` within a session. Thus, to guarantee a clean history for gs_remove_gating_method, it is necessary to call gs_add_gating_method_init on the loaded GatingSet immediately after re-loading it. See the documentation for gs_add_gating_method_init for more details. This will not be an issue for GatingSet objects created directly using the constructor.

See Also

gs_remove_gating_method gs_add_gating_method_init

Examples

```
## Not run:
# add quad gates
gs_add_gating_method(gs, gating_method = "mindensity", dims = "CCR7,CD45RA", parent = "cd4-cd8+", pop = "CCR7+/

# polyfunctional gates (boolean combinations of exsiting marginal gates)
gs_add_gating_method(gs, gating_method = "polyFunctions", parent = "cd8", gating_args = "cd8/IFNg:cd8/IL2:cd8/

#boolGate method
gs_add_gating_method(gs, alias = "IL2orIFNg", gating_method = "boolGate", parent = "cd4", gating_args = "cd4/IL

## End(Not run)
```

---

gs_add_gating_method_init

Clear history of gs_add_gating_method *calls for a given* GatingSet *or* GatingSetList

---

Description

Repeated calls to the load_gs method in the same session will yield indistinguishable objects that can result in overlapping history of gs_add_gating_method calls. This method allows for the history to be cleared if the user would like to reload the GatingSet and start fresh. Calling gs_add_gating_method_init without an argument will clear the entire gs_add_gating_method history.

Usage

```
gs_add_gating_method_init(gs)
```

Arguments

gs	a GatingSet or GatingSetList. Can be omitted to clean entire gs_add_gating_method history.

Examples

```
## Not run:
# load in a GatingSet
gs <- load_gs(path)
# Add some nodes using gs_add_gating_method
gs_add_gating_method(gs, gating_method = "mindensity", dims = "CCR7,CD45RA", parent = "cd4-cd8+", pop = "CCR7+/
gs_add_gating_method(gs, gating_method = "polyFunctions", parent = "cd8", gating_args = "cd8/IFNg:cd8/IL2:cd8/
```

```
# Remove the effect of the last gs_add_gating_method call using gs_remove_gating_method (note that the first cal
gs_remove_gating_method(gs)
# Re-load the GatingSet to start over
gs <- load_gs(path)

# At this point, gs will still see the history of the first gs_add_gating_method call above
# which will cause problems for later calls to gs_remove_gating_method.
# To fix that, just call gs_add_gating_method_init() to start a clean history
gs_add_gating_method_init(gs)
# Now you can continue using gs_add_gating_method and gs_remove_gating_method from scratch
gs_add_gating_method(gs, gating_method = "mindensity", dims = "CCR7,CD45RA", parent = "cd4-cd8+", pop = "CCR7+/

## End(Not run)
```

---

gs_remove_gating_method   *Reverse the action of gating methods applied via* gs_add_gating_method

---

Description

This function provides an easy way to remove the gates and nodes created by the most recent call to gs_add_gating_method on the specified GatingSet or GatingSetList, with a separate history being maintained for each such object. gs_remove_gating_method allows for repeated use, effectively serving as a multi-level undo function for gs_add_gating_method.

Usage

```
gs_remove_gating_method(gs)
```

Arguments

gs	The GatingSet or GatingSetList for which the most recent gs_add_gating_method call should be reversed.

See Also

gs_add_gating_method gs_add_gating_method_init

Examples

```
## Not run:
 # add quad gates
 gs_add_gating_method(gs, gating_method = "mindensity", dims = "CCR7,CD45RA", parent = "cd4-cd8+", pop = "CCR7+
 # Remove the gates and nodes resulting from that gs_add_gating_method call
 gs_remove_gating_method(gs)

## End(Not run)
```

gtMethod-class *A class to represent a gating method.*

Description

A gating method object contains the specifics for generating the gates.

Slots

name a `character` specifying the name of the gating method

dims a `character` vector specifying the dimensions (channels or markers) of the gate

args a `list` specifying the arguments passed to gating function

groupBy a `character` or `integer` specifying how to group the data. If `character`, group the data by the study variables (columns in `pData`). If `integer`, group the data by every `N` samples.

collapse a `logical` specifying wether to collapse the data within group before gating. it is only valid when `groupBy` is specified

Examples

```
 ## Not run:
   gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
     gh_pop_get_gate(gt, '2', '3')

## End(Not run)
```

gtPopulation-class *A class to represent a cell population that will be generated by a gating method.*

Description

A class to represent a cell population that will be generated by a gating method.

Slots

id `numeric` unique ID that is consistent with node label of graphNEL in gating template

name `character` the name of population

alias `character` the more user friendly name of population

Examples

```
 ## Not run:
   gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))

     gt_get_nodes(gt, '2')

## End(Not run)
```

gtSubsets-class *A class representing a group of cell populations.*

Description

It extends gtPopulation class.

gt_gating *Applies a gatingTemplate to a GatingSet.*

Description

It loads the gating methods by topological order and applies them to GatingSet.

Usage

```
gt_gating(x, y, ...)
```

Arguments

x	a gatingTemplate object
y	a GatingSet object
...	• start a character that specifies the population (correspoding to 'alias' column in csv template) where the gating process will start from. It is useful to quickly skip some gates and go directly to the target population in the testing run. Default is "root". • stop.at a character that specifies the population (correspoding to 'alias' column in csv template) where the gating prcoess will stop at. Default is NULL, indicating the end of gating tree. • keep.helperGatesa logical flag indicating whether to keep the intermediate helper gates that are automatically generated by openCyto. Default is TRUE. • mc.cores passed to multicore package for parallel computing • parallel_type character specifying the parallel type. The valid options are "none", "multicore", "cluster". • cl cluster object passed to parallel package (when parallel_type is "cluster")

Value

Nothing. As the side effect, gates generated by gating methods are saved in GatingSet.

Examples

```
## Not run:
 gt <- gatingTemplate(file.path(path, "data/ICStemplate.csv"), "ICS")
 gs <- GatingSet(fs) #fs is a flowSet/ncdfFlowSet
 gt_gating(gt, gs)
 gt_gating(gt, gs, stop.at = "v") #proceed the gating until population 'v'
 gt_gating(gt, gs, start = "v") # start from 'v'
 gt_gating(gt, gs, parallel_type = "multicore", mc.cores = 8) #parallel gating using multicore
 #parallel gating by using cluster
 cl1 <- makeCluster (8, type = "MPI")
 gt_gating(gt, gs, parallel_type = "cluster", cl = cl1)
 stopCluster ( cl1 )

## End(Not run)
```

`gt_get_children` *get children nodes*

Description

get children nodes

Usage

```
gt_get_children(obj, y)
```

Arguments

obj	gatingTemplate
y	character parent node path

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))

gt_get_nodes(gt, "/nonDebris")
gt_get_children(gt, "/nonDebris")

## End(Not run)
```

`gt_get_gate` *get gating method from the node*

Description

get gating method from the node

Usage

```
gt_get_gate(obj, y, z)
```

Arguments

obj	gatingTemplate
y	character parent node path
z	character child node path

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
gt_get_nodes(gt, only.names = TRUE)
gt_get_nodes(gt, "/nonDebris")
gt_get_children(gt, "/nonDebris")
gt_get_gate(gt, "/nonDebris", "/nonDebris/singlets")

## End(Not run)
```

---

gt_get_nodes *get nodes from gatingTemplate object*

Description

get nodes from gatingTemplate object

Usage

```
gt_get_nodes(
  x,
  y,
  order = c("default", "bfs", "dfs", "tsort"),
  only.names = FALSE
)
```

Arguments

x	gatingTemplate
y	character node index. When missing, return all the nodes
order	character specifying the order of nodes. options are "default", "bfs", "dfs", "tsort"
only.names	logical specifiying whether user wants to get the entire gtPopulation object or just the name of the population node

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
gt_get_nodes(gt)[1:2]
gt_get_nodes(gt, only.names = TRUE)
gt_get_nodes(gt, "/nonDebris")

## End(Not run)
```

gt_get_parent — *get parent nodes*

Description

get parent nodes

Usage

```
gt_get_parent(obj, y, isRef = FALSE)
```

Arguments

obj	gatingTemplate
y	character child node path
isRef	logical whether show the reference node besides the parent node

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))

gt_get_nodes(gt, "/nonDebris")
gt_get_parent(gt, "/nonDebris/singlets")

## End(Not run)
```

gt_list_methods — *Print a list of the registered gating methods*

Description

Print a list of the registered gating methods

Usage

```
gt_list_methods()
```

Value

Does not return anything. Prints a list of the available gating methods.

gt_toggle_helpergates *toggle/delete the hidden flag of the helper gates*

Description

The helper gates are defined as the referred gates in csv template. And all the chidlren of referred gates are also referred gates thus they are considered the helper gates and can usually be hidden to simply the final gating tree.

Usage

```
gt_toggle_helpergates(gt, gs)

gt_get_helpergates(gt, gs)

gt_delete_helpergates(gt, gs)
```

Arguments

gt	gatingTemplate object
gs	GatingSet

Details

Note that delete action is NOT reversible.

Examples

```
## Not run:
gt <- gatingTemplate(gtFile)
#run the gating
gt_gating(gt, gs)
#hide the gates that are not of interest
gt_toggle_helpergates(gt, gs)
#or simply remove them if you are sure they will not be useful in future
gt_delete_helpergates(gt, gs)

## End(Not run)
```

isCollapse,gtMethod-method *get the flag that determines whether gating method is applied on collapsed data*

Description

When TRUE, the flow data(multiple flowFrames) is collapsed into one and the gating method is applied on the collapsed data. Once the gate is generated, it is thenreplicated and applied to the each single flowFrame.

Usage

```
## S4 method for signature 'gtMethod'
isCollapse(object)
```

Arguments

object	gtMethod

Value

logical

---

names,gtMethod-method *get gating method name*

Description

get gating method name

Usage

```
## S4 method for signature 'gtMethod'
names(x)
```

Arguments

x	gtMethod

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))

gtMthd <- gt_get_gate(gt, "/nonDebris/singlets",  "/nonDebris/singlets/lymph")
names(gtMthd)
dims(gtMthd)
parameters(gtMthd)
isCollapse(gtMthd)
groupBy(gtMthd)

gtPop <- gt_get_nodes(gt, "/nonDebris/singlets/lymph/cd3/cd4+cd8-/CD38+")
names(gtPop)
alias(gtPop)

## End(Not run)
```

`names,gtPopulation-method` *get population name*

Description

get population name

Usage

```
## S4 method for signature 'gtPopulation'
names(x)
```

Arguments

`x`	`gtPopulation` object

`ocRectangleGate-class` *the class that carries event indices as well*

Description

the class that carries event indices as well

`ocRectRefGate` *constructor for ocRectRefGate*

Description

constructor for ocRectRefGate

Usage

```
ocRectRefGate(rectGate, boolExprs)
```

Arguments

`rectGate`	`rectangleGate`
`boolExprs`	`character` boolean expression of reference nodes

`ocRectRefGate-class` *special gate type that mix the rectangleGate with boolean gate*

Description

special gate type that mix the rectangleGate with boolean gate

openCyto — *Hierarchical Gating Pipeline for flow cytometry data*

Description

Hierarchical Gating Pipeline for flow cytometry data.

Details

openCyto is a package designed to facilitate the automated gating methods in sequential way to mimic the manual gating strategy.

Package:	openCyto
Type:	Package
Version:	1.2.8
Date:	2014-04-10
License:	GPL (>= 2)
LazyLoad:	yes

Author(s)

Mike Jiang <wjiang2@fhcrc.org>, John Ramey <jramey@fhcrc.org>, Greg Finak <gfinak@fhcrc.org>

Maintainer: Mike Jiang <wjiang2@fhcrc.org>

See Also

See gt_gating, gate_flowclust_1d, for an overview of gating functions.

Examples

```
## Not run: gatingTemplate('test.csv')
```

openCyto-deprecated — *Deprecated functions in package* **openCyto**.

Description

add_pop –> gs_add_gating_method

add_pop_init –> gs_add_gating_method_init

prior_flowClust –> prior_flowclust

templateGen –> gh_generate_template

gate_flowClust_1d –> gate_flowclust_1d

gate_flowClust_2d –> gate_flowclust_2d

quantileGate –> gate_quantile

quadGate.seq –> gate_quad_sequential

quadGate.tmix –> gate_quad_tmix

gating –> gt_gating

getNodes –> gt_get_nodes

getChildren –> gt_get_children

getParent –> gt_get_parent

getGate –> gt_get_gate

listgtMethods –> gt_list_methods

registerPlugins –> register_plugins

remove_pop –> gs_remove_gating_method

toggle.helperGates –> gt_toggle_helpergates

get.helperGates –> gt_get_helpergates

delete.helperGates –> gt_delete_helpergates

---

openCyto.options *Some global options for openCyto See examples for the meaning of these options and how to get/set them.*

Description

Get/set some global options for openCyto

Examples

```
opt <- getOption("openCyto")
#the threshold of minimum cell events required for the gating algorithm to proceed
opt[["gating"]][["minEvents"]]
#to change the threshold
opt[["gating"]][["minEvents"]] <- 100
options(openCyto = opt)

#switch off the validity check flags(Not recommended)
opt[["check.pop"]] <- FALSE
options(openCyto = opt)
```

---

parameters,gtMethod-method
get parameters of the gating method/function

Description

get parameters of the gating method/function

Usage

```
## S4 method for signature 'gtMethod'
parameters(object)
```

Arguments

object	gtMethod

---

plot,fcFilterList,ANY-method
plot a fcFilterList

Description

It is usually called by plot method for fcTree instead of directly by users.

Usage

```
## S4 method for signature 'fcFilterList,ANY'
plot(
  x,
  y,
  samples = NULL,
  posteriors = FALSE,
  xlim = NULL,
  ylim = NULL,
  node = NULL,
  data = NULL,
  breaks = 20,
  lwd = 1,
  ...
)
```

Arguments

x	fcFilterList
y	character channel name
samples	character a vector of sample names to be plotted
posteriors	logical indicating whether posteriors should be plotted
xlim, ylim	scale settings for x,y axises
node	character population name associated with the fcFilterList
data	GatingSet object
breaks	passed to hist
lwd	line width
...	other arguments passed to base plot

Examples

```
## Not run:
 env1<-new.env(parent=emptyenv())
 #gt is a gatingTemplate, gs is a GatingSet
 gt_gating(gt,gs,env1) #the flowClust gating results are stored in env1
plot(env1$fct,"nonDebris",post=T) #plot the priors as well as posteriors for the "nonDebris" gate

## End(Not run)
```

plot,fcTree,character-method

plot the flowClust gating results

Description

This provides the priors and posteriors as well as the gates for the purpose of debugging flowClust gating algorithm

Usage

```
## S4 method for signature 'fcTree,character'
plot(x, y, channel = NULL, data = NULL, ...)
```

Arguments

x	fcTree
y	character node name in the fcTree
channel	character specifying the channel.
data	GatingSet that the fcTree is associated with
...	other arguments

plot,gatingTemplate,missing-method

plot the gating scheme

Description

plot the gating scheme using Rgraphviz

Usage

```
## S4 method for signature 'gatingTemplate,missing'
plot(x, y, ...)
```

Arguments

x	gatingTemplate object
y	either character specifying the root node which can be used to visualize only the subgraph or missing which display the entire gating scheme
...	other arguments graphAttr, nodeAttr: graph rendering attributes passed to renderGraph showRef logical: whether to display the reference gates. Sometime it maybe helpful to hide all those reference gates which are not the cell population of interest and used primarily for generating other population nodes.

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
plot(gt) #plot entire tree
plot(gt, "lymph") #only plot the subtree rooted from "lymph"

## End(Not run)
```

---

polyFunctions-class   *A class to represent a polyFunctions gating method.*

Description

It extends boolMethod class and will be expanded to multiple boolMethod object.

---

pop_add.ocRectangleGate   *bypass the default flowWorkspace:::.addGate*

Description

to support adding gate along with indices without loading flow data and computing

to support adding rectangleGate yet gating through boolean operations without loading flow data

Usage

```
## S3 method for class 'ocRectangleGate'
pop_add(gate, gh, recompute, ...)

## S3 method for class 'ocRectRefGate'
pop_add(gate, gh, recompute, ...)
```

Arguments

gate	ocRectangleGate or logicalFilterResult
gh	GatingHierarchy see add in flowWorkspace package
recompute	logical see add in flowWorkspace package
...	see add in flowWorkspace package

Details

however it is proven that logical indices are too big to be efficiently passed around

posteriors,fcFilter,ANY-method

get posteriors from a `fcFilter` *object*

Description

get posteriors from a `fcFilter` object

Usage

```
## S4 method for signature 'fcFilter,ANY'
posteriors(x, y = "missing")
```

Arguments

x	`fcFilter`
y	`character` or `missing` that specifiy which channel to look for

ppMethod,gatingTemplate,character-method

get preprocessing method from the node

Description

get preprocessing method from the node

Usage

```
## S4 method for signature 'gatingTemplate,character'
ppMethod(obj, y, z)
```

Arguments

obj	`gatingTemplate`
y	`character` parent node path
z	`character` child node path

Examples

```
## Not run:
gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
ppMethod(gt,  "/nonDebris/singlets",  "/nonDebris/singlets/lymph")

## End(Not run)
```

ppMethod-class — *A class to represent a preprocessing method.*

Description

It extends `gtMethod` class.

Examples

```
 ## Not run:
   gt <- gatingTemplate(system.file("extdata/gating_template/tcell.csv",package = "openCyto"))
     ppMethod(gt, '3', '4')

## End(Not run)
```

preprocessing,ppMethod,GatingSet-method — *apply a ppMethod to the* `GatingSet`

Description

apply a ppMethod to the `GatingSet`

Usage

```
## S4 method for signature 'ppMethod,GatingSet'
preprocessing(x, y, ...)
```

Arguments

x	`ppMethod`
y	`GatingSet` or `GatingSetList`
...	other arguments

priors,fcFilter,ANY-method — *get priors from a* `fcFilter` *object*

Description

get priors from a `fcFilter` object

Usage

```
## S4 method for signature 'fcFilter,ANY'
priors(x, y = "missing")
```

Arguments

`x`	`fcFilter` object
`y`	`character` specifying channel name. if `missing` then extract priors for all the channels

---

`prior_flowclust` *Elicits data-driven priors from a flowSet object for specified channels*

Description

We elicit data-driven prior parameters from a `flowSet` object for specified channels. For each sample in the `flowSet` object, we apply the given `prior_method` to elicit the priors parameters.

Usage

```
prior_flowclust(
  flow_set,
  channels,
  prior_method = c("kmeans"),
  K = 2,
  nu0 = 4,
  w0 = c(10, 10),
  shrink = 1e-06,
  ...
)
```

Arguments

`flow_set`	a `flowSet` object
`channels`	a character vector containing the channels in the `flowSet` from which we elicit the prior parameters for the Student's t mixture
`prior_method`	the method to elicit the prior parameters
`K`	the number of mixture components to identify
`nu0`	prior degrees of freedom of the Student's t mixture components.
`w0`	the number of prior pseudocounts of the Student's t mixture components. (only the first element is used and the rest is ignored at the moment)
`shrink`	the amount of eigenvalue shrinkage to add in the case the prior covariance matrices are singular. See details.
`...`	Additional arguments passed to the prior elicitation method selected

Details

Currently, we have implemented only two methods. In the case that one channel is given, we use the kernel-density estimator (KDE) approach for each sample to obtain `K` peaks from which we elicit prior parameters. Otherwise, if more than one channel is specified, we apply K-Means to each of the samples in the `flowSet` and aggregate the clusters to elicit the prior parameters.

In the rare case that a prior covariance matrix is singular, we shrink the eigenvalues of the matrix slightly to ensure that it is positive definite. For instance, if the `flow_set` has two samples, this case can occur. The amount of shrinkage is controlled in `shrink`.

Value

list of the necessary prior parameters

Examples

```
## Not run:
library(flowCore)
data(GvHD)
prior_flowclust(GvHD[1:3], c("FSC-H", "SSC-H"))

## End(Not run)
```

---

refGate-class — *A class to represent a reference gating method.*

Description

It extends `gtMethod` class.

Slots

refNodes `character` specifying the reference nodes

---

register_plugins — *Register a gating or preprocessing function with OpenCyto*

Description

Function registers a new gating or preprocessing method with openCyto so that it may be used in the csv template.

Usage

```
register_plugins(fun = NA, methodName, dep = NA, ...)
```

Arguments

`fun`	`function` to be registered
`methodName`	`character` name of the gating or preprocessing method
`dep`	`character` name of the library dependency required for the plugin method to work.
`...`	other arguments type `character` specifying the type of registering method. Should be either "gating" or "preprocessing".

Details

The `fun` argument should be a wrapper function definition for the gating or preprocessing method. Gating method must have formal arguments:

fr a `flowFrame`

pp_res a pre-processing result

xChannel `character` (optional)

yChannel `character` (required)

filterId `character`

... ellipses for the additional parameters.

Preprocessing method must have formal arguments:

fs a `flowSet` that stores the flow data (could be subgrouped data if `groupBy` column is defined in the csv template

gs a `GatingSet`

gm a `gtMethod` object that stores the information from gating method

xChannel `character` (required)

yChannel `character` (required)

... ellipses for the additional parameters.

The gating function must return a filter (i.e. polygonGate or other instance) from flowCore. The preprocessing can return anything and it will be passed on to the gating function. So it is up to gating function to use and interpret the results of preprocessing. Not all formal parameters need to be used. Additional arguments are passed via the ... and can be processed in the wrapper

Value

`logical` TRUE if successful and prints a message. FALSE otherwise.

`robust_m_estimator` *rewrite huber estimator*

Description

rewrite huber estimator

Usage

```
robust_m_estimator(x, sd)
```

show,boolMethod-method   *show method for boolMethod*

Description

show method for boolMethod

Usage

```
## S4 method for signature 'boolMethod'
show(object)
```

Arguments

object	boolMethod

show,fcFilter-method   *show method for fcFilter*

Description

show method for fcFilter

Usage

```
## S4 method for signature 'fcFilter'
show(object)
```

Arguments

object	fcFilter show method for fcFilter

show,gatingTemplate-method   *show method for gatingTemplate*

Description

show method for gatingTemplate

Usage

```
## S4 method for signature 'gatingTemplate'
show(object)
```

Arguments

object	gatingTemplate

`show,gtMethod-method` *show method for gtMethod*

Description

show method for gtMethod

Usage

```
## S4 method for signature 'gtMethod'
show(object)
```

Arguments

`object`	`gtMethod` show method for gtMethod

Index