Package 'flowQB'

October 12, 2016

Type Package

Title Automated Quadratic Characterization of Flow Cytometer Instrument Sensitivity: Q, B and CV instrinsic calculations

Version 1.18.4

Author Josef Spidlen, Faysal El Khettabi, Wayne Moore, David Parks, Ryan Brinkman

Maintainer Josef Spidlen <jspidlen@bccrc.ca>

Description flowQB is a fully automated R Bioconductor package to calculate automatically the detector efficiency (Q), optical background (B) and intrinsic CV of the beads.

Imports methods, flowCore (>= 1.32.0), stats, extremevalues

License Artistic-2.0

Suggests FlowRepositoryR, xlsx, RUnit, BiocGenerics

biocViews FlowCytometry, Regression, PeakDetection, QualityControl, MultiChannel, OneChannel

LazyLoad yes

Collate helper_functions.R fitted_ellipse_gate.R split_in_two.R peak_gate.R pick_parameters.R calc_mean_sd_functions.R fit_functions.R

NeedsCompilation no

R topics documented:

`calc_mean_sd_197` *Calculate the mean and the standard deviation of Standford's "197 calibration beads".*

Description

This method calculated the mean and the standard deviation from peaks identified in date generated by running Stanford's "197 calibration beads". Back in 1993, Spherotech pruduced calibration beads for Stanford and these are the 197th bead sample that Spherotech sent to Stanford. These have then been used at Stanford until 2013. Later on, Spherotech sent a new batch of similar 3 micron single-level multi-dye bead lot called "RCP-30-5A LotAE01", but, since the FACS Facility users were used to looking for 197 beads, these were simply called 197B. These beads were used as one of the reference particles in the 23 instrument multi-site instrument comparison that flowQB was originally designed to analyze. These beads provides a convenient stable single peak reference with reasonable signal levels on all of the usual fluorescence channels. Unfortunately, the amount of 197B available was small, so in 2014, Spherotech sent a new batch, which was later called 197C at Stanford.

Usage

```
calc_mean_sd_197(fcs_file_path, scatter_channels, ignore_channels)
```

Arguments

`fcs_file_path`	A character string specifying the file path to the FCS file with the acquired bead data.
`scatter_channels`	A vector of 2 short channel names (values of the $PnN keywords) specifying the 2 channels that should not be used to gate the main bead population. The first channel should be a forward scatter channel, the second one should be a side scatter channel.
`ignore_channels`	A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

Details

The method fits an ellipse gate on the 2 specified scatter channels and then locates the peak of that populations in each of the fluorescence channels and finally uses the getOutliers method from the extremevalues package in order to calculate the mean and the standard deviation of the result for each of the fluorescence channels.

Value

The result is a data frame with columns corresponding to short channel names of channels from the input FCS file except those specified by the ignore_channels parameter. The rows include the total number of events, the number of events in the FSC/SSC ellipse gate, the number of events in the peak gate (which can vary slighly among the different channels), the mean and the standard deviation.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

fitted_ellipse_gate, calc_mean_sd_duke,

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#
#   test_column <- 14
#   test_row <- 14
#   folder <- xlsx[[test_column]][[test_row]]
#   beads_file_name <- xlsx[[test_column]][[test_row+1]]
#   scatter_channels <- c(
```

```
#       xlsx[[test_column]][[test_row+2]],
#       xlsx[[test_column]][[test_row+3]])
#
#   fcs_path <- system.file("extdata", instrument_folder,
#       folder, beads_file_name, package="flowQBData")
#
#   results <- calc_mean_sd_197(fcs_path, scatter_channels, ignore_channels)
#
#   ## Same thing as above with providing the arguments directly without
#   ## parsing it from the spreadsheet.
#   fcs_path <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "933743.fcs", package="flowQBData")
#   scatter_channels <- c("FSC-A", "SSC-A")
#   ignore_channels <- c(
#       "Time", "FSC-A", "FSC-W", "FSC-H", "SSC-A", "SSC-W", "SSC-H")
#   results <- calc_mean_sd_197(fcs_path, scatter_channels, ignore_channels)
```

`calc_mean_sd_background` *Calculate the mean and the standard deviation.*

Description

This method looks at all channels except those specified in the ignore_channel list and calculated the mean and the standard deviation for those channels.

Usage

```
calc_mean_sd_background(fcs_file_path, ignore_channels)
```

Arguments

`fcs_file_path`	A character string specifying the file path to the FCS file with the acquired bead data.
`ignore_channels`	A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

Details

The getOutliers method from the extremevalues package is used to calculate the mean and the standard deviation values for all the FCS channels in the file except those specified in the ignore channels list.

Value

The result is a data frame with columns corresponding to short channel names of channels from the input FCS file except those specified by the ignore_channels parameter. The rows include the total number of events, the mean and the standard deviation.

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

See Also

calc_mean_sd_197, calc_mean_sd_duke, calc_mean_sd_capture, calc_mean_sd_capture_all

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#
#   test_column <- 15
#   test_row <- 14
#   folder <- xlsx[[test_column]][[test_row]]
#   file_name <- xlsx[[test_column]][[test_row+1]]
#
#   fcs_path <- system.file("extdata",
#       instrument_folder, folder, file_name, package="flowQBData")
#
#   results <- calc_mean_sd_background(fcs_path, ignore_channels)
#
#   ## Same thing as above with providing the arguments directly without
#   ## parsing it from the spreadsheet.
#   fcs_path <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "935319.fcs", package="flowQBData")
#   ignore_channels <- c(
```

```
#       "Time", "FSC-A", "FSC-W", "FSC-H", "SSC-A", "SSC-W", "SSC-H")
#   results <- calc_mean_sd_background(fcs_path, ignore_channels)
```

`calc_mean_sd_capture` *Calculate the mean and the standard deviation for the stained and unstained population of a specified channel in an FCS file.*

Description

This function calculates the mean and the standard deviation of two populations - one unstaned (low) and one stained (high) for a specified channel (specified by the value of the detector parameter). These populations are derived by first fitting an ellipse gate on the 2 specified scatter channels and then splitting for low and high based on the specified detector.

Usage

```
calc_mean_sd_capture(fcs_file_path, scatter_channels, detector, dye)
```

Arguments

`fcs_file_path`	A character string specifying the file path to the FCS file with the acquired bead data.
`scatter_channels`	A vector of 2 short channel names (values of the $PnN keywords) specifying the 2 channels that should not be used to gate the main bead population. The first channel should be a forward scatter channel, the second one should be a side scatter channel.
`detector`	A character string specifying which channel to split for the low/high gate, which is also the channel that we calculate the mean and the standard deviation for.
`dye`	A character string specifying the desired column heading of the result.

Details

This function first fits an ellipse gate on the 2 specified scatter channels. This scatter gated population is then split to 2 (low=unstained and high=stained) in each of the fluorescence channels, and a peak gate is applied in order to isolate the high and the low peaks. Finally, the getOutliers method from the extremevalues package is used in order to calculate the mean and the standard deviation of both the stained and unstained population of each of the fluorescence channels. The value of the detector argument is used to determine which channel to work with, the value of the dye argument is only used to specify the column name of the result. These shall correspond to each other, e.g., "FITC-A" may be the value of the detector, which shall correspond to the short channel name of a channel in the FCS file. FITC may then be the desidered dye name, which will end be used for the column heading.

Value

The result is a data frame with a single column, the heading of the column corresponds to the value of the dye argument. The rows include the total number of events, the number of events in the FSC/SSC ellipse gate, the number of events in the high peak gate and low peak gate, the stained mean and stained standard deviation (based on the high peak gate), and finally the unstained mean and unstained standard deviation (based on the low peak gate).

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

calc_mean_sd_capture_all

Examples

```
    library(flowCore)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   fcs_file_path <- system.file("extdata", "SSFF_LSRII", "SU_2B",
#       "933723.fcs", package="flowQBData")
#
#   scatter_channels <- c("FSC-A", "SSC-A")
#   detector <- "APC-A"
#   dye <- "APC"
#
#   results <- calc_mean_sd_capture(
#       fcs_file_path, scatter_channels, detector, dye)
```

---

calc_mean_sd_capture_all

Calculate the mean and the standard deviation for the stained and unstained population of specified channels in specified FCS files.

Description

This methods performs the `calc_mean_sd_capture` function on a list of FCS files, list of scatter channel pairs, list of detectors and a list of dyes, and collates the results. The order of the arguments in the input lists matters, i.e., the first FCS file will be matched with the first pair of FSC/SSC channel names, the first detector name and the first dye name.

Usage

```
calc_mean_sd_capture_all(fcs_file_path_list, scatter_channels_list,
detector_list, dye_list)
```

Arguments

fcs_file_path_list	A list of n FCS files, one for each detector.
scatter_channels_list	A list of n pairs of forward and side scatter channel names.
detector_list	A list of n detector names; those shall correspond to specific detector in the n specified FCS files.
dye_list	A list of n dye names; those will be used to name the columns of the resulting data frame.

Details

This method assumes that each of the FCS files have useful data only in the specified channel. Therefore, we perform the calc_mean_sd_capture on all these FCS files separatelly and then put the results together into a single data frame.

Value

The result is a data frame with n columns, the headings of the columns correspond to the values in the list provided by the dye_list argument. The rows include the total number of events, the number of events in the FSC/SSC ellipse gate, the number of events in the high peak gate and low peak gate, the stained mean and stained standard deviation (based on the high peak gate), and finally the unstained mean and unstained standard deviation (based on the low peak gate).

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

See Also

calc_mean_sd_capture

Examples

```
    library(flowCore)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   file_directory <- system.file("extdata", "SSFF_LSRII", "SU_2B",
#       package="flowQBData")
#   fcs_file_path_list <- as.list(file.path(
#       file_directory, c("933723.fcs","933725.fcs")))
#   scatter_channels_list <- list(c("FSC-A", "SSC-A"), c("FSC-A", "SSC-A"))
#   detector_list <- list("APC-A", "APC-Cy7-A")
#   dye_list <- list("APC", "APC-Cy7")
#
#   results <- calc_mean_sd_capture_all(
#       fcs_file_path_list,
#       scatter_channels_list,
```

```
#       detector_list,
#       dye_list
#   )
#
#   ## Now the same thing again, but we will show how to extract information
#   ## from the spreadsheet and run the appropriate calculations
#   library(xlsx)
#   xls_path <- system.file("extdata", "140126_InstEval_Stanford_LSRIIA2.xlsx",
#       package="flowQBData")
#   xls <- read.xlsx(xls_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#   insfolder <- instrument.folder <- xls[[2]][[9]]
#
#   dyes <- list()
#   detectors <- list()
#   filepaths <- list()
#   scatters <- list()
#
#   for (i in 1:10)
#   {
#       folder <- xls[[i+2]][[14]]
#       filename <- xls[[i+2]][[15]]
#
#       if (is.na(filename)) next
#       filepath <- system.file("extdata", insfolder, folder, filename,
#           package="flowQBData")
#       ## Spreadsheet may describe additional FCS files not included
#       ## with the library, so skip if file doesn't exist
#       if (nchar(filepath) == 0) next
#
#       filepaths <- c(filepaths, filepath)
#       dyes <- c(dyes, xls[[i+2]][[11]])
#       detectors <- c(detectors, xls[[i+2]][[13]])
#       scatters[[length(scatters)+1]] <- c(xls[[i+2]][[16]], xls[[i+2]][[17]])
#   }
#
#   results2 <- calc_mean_sd_capture_all(filepaths, scatters, detectors, dyes)
```

`calc_mean_sd_duke` *Calculate the mean and the standard deviation of calibration beads from Duke.*

Description

Currently, this is the same calculation as `calc_mean_sd_197`.

Usage

```
calc_mean_sd_duke(fcs_file_path, scatter_channels, ignore_channels)
```

Arguments

fcs_file_path	A character string specifying the file path to the FCS file with the acquired bead data.
scatter_channels	A vector of 2 short channel names (values of the $PnN keywords) specifying the 2 channels that should not be used to gate the main bead population. The first channel should be a forward scatter channel, the second one should be a side scatter channel.
ignore_channels	A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

Details

Currently, this is the same calculation as `calc_mean_sd_197`.

Value

The result is a data frame with columns corresponding to short channel names of channels from the input FCS file except those specified by the ignore_channels parameter. The rows include the total number of events, the number of events in the FSC/SSC ellipse gate, the number of events in the peak gate (which can vary slighly among the different channels), the mean and the standard deviation.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

`fitted_ellipse_gate`, `calc_mean_sd_197`

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
```

```
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#
#   test_column <- 13
#   test_row <- 14
#   folder <- xlsx[[test_column]][[test_row]]
#   beads_file_name <- xlsx[[test_column]][[test_row+1]]
#   scatter_channels <- c(
#       xlsx[[test_column]][[test_row+2]],
#       xlsx[[test_column]][[test_row+3]])
#
#   fcs_path <- system.file("extdata", instrument_folder, folder,
#       beads_file_name, package="flowQBData")
#
#   results <- calc_mean_sd_duke(fcs_path, scatter_channels, ignore_channels)
```

`find_peak` *Find density peak in the provided vector of values*

Description

Find a density peak in the provided vector of values and return the lower and upper bounds around that peak.

Usage

```
find_peak(data, width=0.5, fraction=0.1)
```

Arguments

`data`	Vector of numbers.
`width`	The returned lower and upper bounds for the peak are calculated based on how much larger is the difference between the values at the "i"" and "i + faction size" index of sorted data is; i is itterated from the index of the most dense region to the left (for lower bound) and to the right (for upper bound), which increases the above mentioned difference as the region is becomming less dense. The itteration is stopped when the difference gets larger than the minimum difference times 1/width. This means, by default with width = 0.5, the difference can reach double the minimum difference.
`fraction`	How large is the proportion of values that we look at when distances as mentioned above. By default with fraction = 0.1, we look at 10% of values.

Details

This functions finds a density peak in the provided vector of values and returns the lower and upper bounds around that peak. First, we sort the input data, let's call it x, and look at a fraction of values at a time. Say we have 1,000 values in our input vector and fraction is 0.1 (10%), meaning we look at regions spanning 100 values. We will find the most dense region by itterating the possible starting index from 1 to 899 and minimizing the difference between the sorted values at potential starting index and corresponding ending index (100 values apart). Finding the minimum difference equals to finding the most dense region. Next we find the lower and upper bounds for the peak based on how much larger is the difference between the values at the "i"" and "i + faction size" index of sorted data is; i is itterated from the index of the most dense region to the left (for lower bound) and to the right (for upper bound), which increases the above mentioned difference as the region is becomming less dense. The itteration is stopped when the difference gets larger than the minimum difference times 1/width. This means, by default with width = 0.5, the difference can reach double the minimum difference.

Value

A list with 2 components, "lo" and "hi", identifying the lower and the upper bounds for the peak.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

fitted_ellipse_gate

Examples

```
my_peak_info <- find_peak(rnorm(1000, mean=5))
```

---

fitted_ellipse_gate *Fit and apply an ellipse gate*

Description

Fit an ellipse (or ellipsoid) gate on the most dense region of the selected channels of a flowFrame object.

Usage

```
fitted_ellipse_gate(object, channels, R=1)
```

Arguments

object	A flowFrame object that will be gated by an automatically fitted ellipse (or ellipsoid) gate.
channels	A vector if indices or short channel names (valued of the \$PnN keywords) of channels that shall be used to fit an ellipse (or ellipsoid) gate. Typically, these would be the indeces or names of the forward and scatter channels in the input flowFrame, which will then be gated by a 2D ellipse gate (since 2 channels were provided). A multidimensional ellipsoid gate is fitted analogically if more than 2 channels are provided. A 1D "ellipsoid" gate (i.e., a range gate) is applied if only a single channel is specified.
R	An additional scaling factor for the radii of the ellipse/ellipsoid gate.

Details

First, events with negative values in the specified channels are removed. Next, the find_peak function is called on log transformed values of each of the specified channels. The returned lower and upper boundaries of each of the channels are used to fit an ellipse/ellipsoid gate. The radii of this gate are scalled by the value of the provided R argument. A new flowFrame object with only those events in this gate is returned.

Value

A flowFrame object gated by an automatically fitted ellipse or ellipsoid gate created in the specified channels.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

find_peak

Examples

```
    library('flowCore')
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   fcsFilePath <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "933745.fcs", package="flowQBData")
#   myFlowFrame <- read.FCS(fcsFilePath)
#   gatedFlowFrame <- fitted_ellipse_gate(myFlowFrame, c('FSC-H', 'SSC-H'))
```

`fit_beads` *Fit multi-level bead data.*

Description

Fit observed means and variances of data generated by a sample of multi-level beads to a quadratic model involving the Poisson distribution expectations for the relation between them.

Usage

```
fit_beads(fcs_file_path, scatter_channels, ignore_channels,
    N_peaks, dyes, detectors, bounds,
    signal_type, instrument_name, minimum_useful_peaks = 3,
    max_iterations = 10, logicle_width = 1.0, ...)

fit_spherotech(fcs_file_path, scatter_channels, ignore_channels,
    dyes, detectors, bounds,
    signal_type, instrument_name, minimum_useful_peaks = 3,
    max_iterations = 10, logicle_width = 1.0, ...)

fit_thermo_fisher(fcs_file_path, scatter_channels, ignore_channels,
    dyes, detectors, bounds,
    signal_type, instrument_name, minimum_useful_peaks = 3,
    max_iterations = 10, logicle_width = 1.0, ...)
```

Arguments

`fcs_file_path`	A character string specifying the file path to the FCS file with the acquired bead data.
`scatter_channels`	A vector of 2 short channel names (values of the $PnN keywords) specifying the 2 channels that should not be used to gate the main bead population. The first channel should be a forward scatter channel, the second one should be a side scatter channel.
`ignore_channels`	A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.
`N_peaks`	The number of peaks (different beads) to look for. This argument is applicable to the `fit_beads` function only; the `fit_spherotech` and `fit_thermo_fisher` functions have the number of peaks predefined to 8 and 6, resp.
`dyes`	A vector of dye names. This value does not affect the fitting, but those dyes will be "highlighted" in the provided results.

`detectors` A vector of short channel names (values of the $PnN keywords) specifying channels matching to the dyes specified above. The length of this vector shall correspond to the length of the dyes vector. These channels should be all of the same type as specified by the `signal_type` below, i.e., area, height or width of the measured signal.

`bounds` On some instruments, the lowest LED peaks may be cut off at a data baseline so that the peak statistics will not be valid. Therefore, peaks too close to the baseline need to be excluded from the fitting. Also, many instruments do not maintain good linearity to the full top of scale, so it is also important to specify a maximum level for good linearity and, on each fluorescence channel, exclude any peak that is above that maximum. The bounds argument shall provide a list specifying the minimum and maximum value for the means of valid peaks; peaks with means outsize of this range will be ignored for that particular channel.

`signal_type` he type of the signal specified as the "area", "height" or "width". This should match to the signal type that is being captured by the channels specified in the detectors argument. The signal type is being used in order to trigger type-specific peak validity checks. Currently, if signal type equals to "height" then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. In addition, peaks that are not sufficiently narrow (i.e., exceeding a specific maximum CV) are also omitted from the fitting. Currently, the maximum allowed CV is set to 0.65, but the code is designed to make this user-configurable and signal type dependent eventually.

`instrument_name`
The make/model of the instrument. The purpose if this argument is to allow for instrument-specific peak validity checks. At this point, if BD Accuri is passed as the instrument type, then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. Additional instrument-specific peak validity checks may be implemented in the future.

`minimum_useful_peaks`
Different peaks may be omitted for different channels due to various validity checks described above. This argument specifies the minimal number of valid peaks required in order for the fitting procedure to be performed on a particular fluorescence channel.

`max_iterations` The maximum number of iterations for the iterative fitting approach with appropriate weight recalculations.

`logicle_width` The width parameter for the Logicle transformation. The data clustering part is performed on data transformed with the Logicle transformation. Generally, the Logicle width (w parameter) of 1.0 has been working well for all our data, but users can change the default by providing a different value.

`...` Additional arguments that will be passed to the get_peak_statistics function used internally to calculate peak statistics, such as the maximum.cv.area and maximum.cv.height values.

Details

The `fit_beads` function performs quadratic fitting for multi-level, multi-dye bead sets. In addition, the `fit_spherotech` function performs fitting for the Sph8 particle sets from Spherotech, and the

`fit_thermo_fisher` function performs fitting for the 6-level (TF6) Thermo Fisher set. Internally, this is the same `fit_beads` function except that the number of expected peaks is predefined to 8 and 6, resp. The parameters for the bead data fitting functions are similar to those required for the LED fitting. The main difference is that a single FCS file is expected because the bead sets are provided as a mixture of the different populations and therefore, acquiring data from a single sample will naturally result in all the peaks contained within a single FCS file. All the beads are expected to have the same (or very similar) light scatter properties. Therefore, we perform automated gating on the forward and side scatter channels in order to isolate the main population. In order to do that, the method requires a `scatter_channels` argument that specifies which 2 channels shall be used for the scatter gating. After the main population is isolated, we use K-means clustering to separate the expression peaks generated by different beads. The number of clusters is pre-defined as 8 for the `fit_spherotech` function, 6 for the `fit_thermo_fisher` function, and provided by the user in the form of the `N_peaks` argument in case of the `fit_beads` function. This clustering is performed on data transformed with the Logicle transformation.

Value

The value is a list, see the vignette for a detailed description.

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

See Also

`fit_led`

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#
```

```
#   folder_column <- 16
#   folder_row <- 14
#   folder <- xlsx[[folder_column]][[folder_row]]
#   filename <- xlsx[[folder_column]][[folder_row+1]]
#   scatter_channels <- c(
#       xlsx[[folder_column]][[folder_row+2]],
#       xlsx[[folder_column]][[folder_row+3]])
#
#   fcs_file_path <- system.file("extdata", instrument_folder, folder,
#       filename, package="flowQBData")
#
#   bounds_min_col <- 6
#   bounds_min_row <- 7
#   bounds_max_col <- 7
#   bounds_max_row <- 7
#   bounds <- list()
#   if (is.na(xlsx[[bounds_min_col]][[bounds_min_row]])) {
#       bounds$minimum <- -100
#   } else {
#       bounds$minimum <- as.numeric(xlsx[[bounds_min_col]][[bounds_min_row]])
#   }
#   if (is.na(xlsx[[bounds_max_col]][[bounds_max_row]])) {
#       bounds$maximum <- 100000
#   } else {
#       bounds$maximum <- as.numeric(xlsx[[bounds_max_col]][[bounds_max_row]])
#   }
#   signal_type_col <- 3
#   signal_type_row <- 19
#   signal_type <- xlsx[[signal_type_col]][[signal_type_row]]
#
#   instrument_name_col <- 2
#   instrument_name_row <- 5
#   instrument_name <- xlsx[[instrument_name_col]][[instrument_name_row]]
#
#   channel_cols <- 3:12
#   dye_row <- 11
#   detector_row <- 13
#   dyes <- as.character(xlsx[dye_row,channel_cols])
#   detectors <- as.character(xlsx[detector_row,channel_cols])
#
#   multipeak_results <- fit_spherotech(fcs_file_path, scatter_channels,
#       ignore_channels, dyes, detectors, bounds,
#       signal_type, instrument_name, minimum_useful_peaks = 3,
#       max_iterations = 10, logicle_width = 1.0)
#
#   ## The above is the same as this:
#   ## N_peaks <- 8
#   ## multipeak_results <- fit_beads(fcs_file_path, scatter_channels,
#   ##     ignore_channels, N_peaks, dyes, detectors, bounds,
#   ##     signal_type, instrument_name, minimum_useful_peaks = 3,
#   ##     max_iterations = 10, logicle_width = 1.0)
#
#   plot(
```

```
#       exprs(multipeak_results$transformed_data[,"FITC-A"]),
#       exprs(multipeak_results$transformed_data[,"Pacific Blue-A"]),
#       col=multipeak_results$peak_clusters$cluster, pch='.')
#
#   ## Thermo-Fisher Example:
#   folder_column <- 17
#   folder <- xlsx[[folder_column]][[folder_row]]
#   filename <- xlsx[[folder_column]][[folder_row+1]]
#
#   fcs_file_path <- system.file("extdata", instrument_folder, folder,
#       filename, package="flowQBData")
#
#   beads_results_tf <- fit_thermo_fisher(fcs_file_path, scatter_channels,
#       ignore_channels, dyes, detectors, bounds,
#       signal_type, instrument_name, minimum_useful_peaks = 3,
#       max_iterations = 10, logicle_width = 1.0)
#
#   ## The above is the same as this:
#   ## N_peaks <- 6
#   ## beads_results_tf <- fit_beads(fcs_file_path, scatter_channels,
#   ##     ignore_channels, N_peaks, dyes, detectors, bounds,
#   ##     signal_type, instrument_name, minimum_useful_peaks = 3,
#   ##     max_iterations = 10, logicle_width = 1.0)
#
#   plot(
#       exprs(beads_results_tf$transformed_data[,"FITC-A"]),
#       exprs(beads_results_tf$transformed_data[,"Pacific Blue-A"]),
#       col=beads_results_tf$peak_clusters$cluster, pch='.')
```

`fit_led` *Fit LED data.*

Description

Fit observed means and variances of data generated by an LED pulser to a quadratic and a linear model involving the Poisson distribution expectations for the relation between them. The function assumes that data generated by different LED levels are provided as separate FCS files. These files are passed to the function in the form of a vector of FCS file paths. In addition, house keeping details about the data and the way the fitting procedure should be performed need to be provided, see the description of the arguments below.

Usage

```
fit_led(fcs_file_path_list, ignore_channels, dyes, detectors,
    signal_type, instrument_name,
    bounds = list(minimum = -100, maximum = 100000),
    minimum_useful_peaks = 3, max_iterations = 10, ...)
```

Arguments

`fcs_file_path_list`
: A vector of FCS file paths pointing to data generated by an LED pulser set to a range of LED levels; different levels generated different FCS files, all data coming from a single instrument.

`ignore_channels`
: A vector of short channel names (values of the $PnN keywords) specifying channels that should not be considered for the fitting procedure. Normally, those should be all non-fluorescence channels, such as the time and the (forward and side) scatter channels.

`dyes`
: A vector of dye names that you would normally use with the detectors specified below. This value does not affect the fitting, but those dyes will be "highlighted" in the provided results.

`detectors`
: A vector of short channel names (values of the $PnN keywords) specifying channels matching to the dyes specified above. The length of this vector shall correspond to the length of the dyes vector. These channels should be all of the same type as specified by the signal_type below, i.e., area, height or width of the measured signal.

`signal_type`
: The type of the signal specified as the "area", "height" or "width". This should match to the signal type that is being captured by the channels specified in the detectors argument. The signal type is being used in order to trigger type-specific peak validity checks. Currently, if signal type equals to "height" then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. In addition, peaks that are not sufficiently narrow (i.e., exceeding a specific maximum CV) are also omitted from the fitting. Currently, the maximum allowed CV is set to 0.65, but the code is designed to make this user-configurable and signal type dependent eventually.

`instrument_name`
: The make/model of the instrument. The purpose if this argument is to allow for instrument-specific peak validity checks. At this point, if "BD Accuri" is passed as the instrument type, then peaks with a mean value lower than the lowest peak mean value are omitted from the fitting. Additional instrument-specific peak validity checks may be implemented in the future.

`bounds`
: On some instruments, the lowest LED peaks may be cut off at a data baseline so that the peak statistics will not be valid. Therefore, peaks too close to the baseline need to be excluded from the fitting. Also, many instruments do not maintain good linearity to the full top of scale, so it is also important to specify a maximum level for good linearity and, on each fluorescence channel, exclude any peak that is above that maximum. The bounds argument shall provide a list specifying the minimum and maximum value for the means of valid peaks; peaks with means outsize of this range will be ignored for that particular channel.

`minimum_useful_peaks`
: Different peaks may be omitted for different channels due to various validity checks described above. This argument specifies the minimal number of valid peaks required in order for the fitting procedure to be performed on a particular

fluorescence channel. Generally, fitting the three quadratic parameters requires three valid points to obtain a fit at all, and 4 or more points are needed to obtain error estimates. Requiring higher values would exclude some of your data but likely produce better results.

`max_iterations` The peaks have a wide range of variances, so unweighted least squares fitting is not appropriate, and we need to apply appropriate weights in the fitting procedure. In particular, the populations with lower variances get more weight since having the fit miss them by any particular amount is worse than missing a high variance population by the same amount. This argument specifies the maximum number of iterations for the iterative fitting approach with appropriate weight recalculations. In most cases, the fitting converges relatively fast. The iterating stops when either the maximum of iterations is used or if none of the coefficients of the model changed more than 0.00005. The default maximum of 10 iterations seems to be enough in most cases. You can also explore your results in order to see how many iterations were actually done for each of the all of the fitting.

`...` Additional arguments that will be passed to the get_peak_statistics function used internally to calculate peak statistics, such as the maximum.cv.area and maximum.cv.height values.

Details

An LED light pulser is producing very uniform pulses at adjustable signal levels. White LEDs provide some signal at all visible wavelengths, but the far-red emission is weak. A given LED pulse level will generate quite different photoelectron signals on different detectors, so it is important to collect data over a wide range of LED levels to assure that the measurement series on each detector will include the low, middle and high level signals needed for optimal results in the fitting procedure.

Value

`fit_led` returns a list, see the vignette for a detailed description.

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

See Also

`fit_beads`

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
```

```
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#   folder_column <- 18
#   folder_row <- 14
#   folder <- xlsx[[folder_column]][[folder_row]]
#   fcs_directory <- system.file("extdata", instrument_folder,
#       folder, package="flowQBData")
#   fcs_file_path_list <- list.files(fcs_directory, "*.fcs", full.names= TRUE)
#
#   bounds_min_col <- 6
#   bounds_min_row <- 7
#   bounds_max_col <- 7
#   bounds_max_row <- 7
#   bounds <- list()
#   if (is.na(xlsx[[bounds_min_col]][[bounds_min_row]])) {
#       bounds$minimum <- -100
#   } else {
#       bounds$minimum <- as.numeric(xlsx[[bounds_min_col]][[bounds_min_row]])
#   }
#   if (is.na(xlsx[[bounds_max_col]][[bounds_max_row]])) {
#       bounds$maximum <- 100000
#   } else {
#       bounds$maximum <- as.numeric(xlsx[[bounds_max_col]][[bounds_max_row]])
#   }
#   signal_type_col <- 3
#   signal_type_row <- 19
#   signal_type <- xlsx[[signal_type_col]][[signal_type_row]]
#
#   instrument_name_col <- 2
#   instrument_name_row <- 5
#   instrument_name <- xlsx[[instrument_name_col]][[instrument_name_row]]
#
#   channel_cols <- 3:12
#   dye_row <- 11
#   detector_row <- 13
#   dyes <- as.character(xlsx[dye_row,channel_cols])
#   detectors <- as.character(xlsx[detector_row,channel_cols])
#
#   led_results <- fit_led(fcs_file_path_list, ignore_channels, dyes,
#       detectors, signal_type, instrument_name, bounds = bounds,
#       minimum_useful_peaks = 3, max_iterations = 10)
```

`get_results_for_dyes` *Extract dye results from a data frame with detector results.*

Description

This function takes a data frame where columns are named based on detectors and extracts a subset of the data frame it by selecting only specified detectors. In addition, the columns will be renamed based on the specified dyes argument.

Usage

```
get_results_for_dyes(dyes, detectors, results)
```

Arguments

`dyes`	A vector of n dye names which shall correspond to the dyes specified in the dyes argument. These will be the column names of the resulting data frame. The detector-dye mapping is done based on the order of values in the two vectors, i.e., the first dye shall correspond to the first detector, etc.
`detectors`	A vector of n detector names which shall correspond to the dyes specified in the dyes argument. These shall correspond to the column names in the input data frame. The detector-dye mapping is done based on the order of values in the two vectors, i.e., the first dye shall correspond to the first detector, etc.
`results`	An input data frame that shall contain columns corresponding to all the different values specified by the detectors vector.

Details

This function is used to select a subset of columns from a data frame by specifying the columns of interest (detectors). In addition, the columns will be renamed to dyes corresponding to those detectors.

Value

A data frame with n columns, column names corresponding to the specified dyes and rows/values extracted from the input data frame.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

See Also

`calc_mean_sd_duke`

Examples

```
    library(flowCore)
    library(xlsx)
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   inst_xlsx_path <- system.file("extdata",
#       "140126_InstEval_Stanford_LSRIIA2.xlsx", package="flowQBData")
#   xlsx <- read.xlsx(inst_xlsx_path, 1, headers=FALSE, stringsAsFactors=FALSE)
#
#   ignore_channels_row <- 9
#   ignore_channels <- vector()
#   i <- 1
#   while(!is.na(xlsx[[i+4]][[ignore_channels_row]])) {
#       ignore_channels[[i]] <- xlsx[[i+4]][[ignore_channels_row]]
#       i <- i + 1
#   }
#
#   instrument_folder_row <- 9
#   instrument_folder_col <- 2
#   instrument_folder <- xlsx[[instrument_folder_col]][[instrument_folder_row]]
#
#   test_column <- 13
#   test_row <- 14
#   folder <- xlsx[[test_column]][[test_row]]
#   beads_file_name <- xlsx[[test_column]][[test_row+1]]
#   scatter_channels <- c(
#       xlsx[[test_column]][[test_row+2]],
#       xlsx[[test_column]][[test_row+3]])
#
#   fcs_path <- system.file("extdata",
#       instrument_folder, folder, beads_file_name, package="flowQBData")
#
#   results <- calc_mean_sd_duke(fcs_path, scatter_channels, ignore_channels)
#
#   channel_cols <- 3:12
#   dye_row <- 11
#   detector_row <- 13
#   dyes <- as.character(xlsx[dye_row,channel_cols])
#   detectors <- as.character(xlsx[detector_row,channel_cols])
#   dye_results <- get_results_for_dyes(dyes, detectors, results)
```

---

peak_gate *Gate a 1D density peak in the provided object*

Description

This method finds a density peak in the provided object (which shall be either a matrix or flowCore's flowFrame object) and returns a vector of TRUE/FALSE depending on whether each of the events

(rows) are in the density peak. An FCS channel shall be specified if a flowFrame object with multiple channels in provided on the input.

Usage

```
peak_gate(object, ...)
```

Arguments

object	Object of class flowFrame or a matrix.
...	Additional options, see the details section.

Details

Additional parameters of the method:

channel Which FCS channel shall be used in order to look for the density peak? This is applicable if a flowFrame object with several channels is used on the input.

R The radius to be used when finding the peak; R=1 by default.

Value

A vector of TRUE/FALSE values depending on whether each of the events (rows) are located in the identified density peak.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

Examples

```
library('flowCore')
## library(flowQBData)
## flowQBData is available since BioConductor 3.4, please install it
## manually in order to be able to these examples

#   fcsFilePath <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "933745.fcs", package="flowQBData")
#
#   myFlowFrame <- read.FCS(fcsFilePath)
#   r1 <- peak_gate(myFlowFrame, 'FSC-H')
#   r2 <- peak_gate(exprs(myFlowFrame[,'SSC-H']))
#   ## r3 will have more events than r2
#   r3 <- peak_gate(exprs(myFlowFrame[,'SSC-H']), R=1.5)
```

pick_parameters *Pick channel names*

Description

Extract all channel names from a flowFrame object or column names from a matrix except those specified in a provided ignore list.

Usage

```
pick_parameters(object, ignore)
```

Arguments

object	Object of class flowFrame or a matrix.
ignore	A vector of channel names that we want to ignore

Details

This method simply looks at all channel names in the provided flowFrame object or all column names of a matrix, then subtracts those specified in the ignore list and returns the resulting vector of channel/column names.

Value

A vector of character strings containing channel names of channels that were in the input flowFrame object (or columns of the input matrix) but were not included in the provided ignore list.

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

Examples

```
    library('flowCore')
    ## library(flowQBData)
    ## flowQBData is available since BioConductor 3.4, please install it
    ## manually in order to be able to these examples

#   fcsFilePath <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "933745.fcs", package="flowQBData")
#   myFlowFrame <- read.FCS(fcsFilePath)
#   ignore <- c("Time", "FSC-H", "FSC-A", "FSC-W", "SSC-H", "SSC-A", "SSC-W")
#   fluorescences <- pick_parameters(myFlowFrame, ignore)
```

qb_from_fits *Extract Q, B and the intrinsic CV0 from fitting results*

Description

This function can be used to compute (1) flow cytometer's detection efficiency (i.e., Q, the statistical number of single photoelectrons (Spe) generated per unit of dye in the sample), (2) background illumination (i.e., B, the background light in dye equivalents that sets the minimum variance that underlies all measurements), and (3) the intrinsic CV0 (i.e, variance in the signal produced by the variation in dye amount of beads containing a "fixed" level of dye plus the illumination variations due to particles taking different flow paths through the laser beam) from fitting results produced by either LED data fitting (`fit_led` function) or bead data fitting (`fit_beads`, `fit_spherotech` or `fit_thermo_fisher` functions). One can calculate based on any of the fits (i.e, the `fits`, `dye_fits`, `iterated_fits` or `iterated_dye_fits` items from the result list).

Usage

```
qb_from_fits(fits)
```

Arguments

`fits`	Fitting results as produced by either LED data fitting (`fit_led` function) or bead data fitting (`fit_beads`, `fit_spherotech` or `fit_thermo_fisher` functions). One can calculate based on any of the fits provided as part of the results of these functions, namely the `fits`, `dye_fits`, `iterated_fits` or `iterated_dye_fits` items from the result list.

Details

As explained in our paper, this method calculats QI as 1/c1, BSpe = c0/c1^2 and CV0^2 = c2 from the results of quadratic fit for the measured means and variances to a statistical model involving the Poisson distribution expectations for the relation between them. For fitting results containing both, a quadratic and a linear fit coeficients, (i.e, results of LED fitting), this method also includes QI and BSpe from the linear model.

Value

The value is a matrix with columns corresponding to the columns of the input data frame (i.e., the names of the dyes). The rows are as follows: q_QI as the QI from the fits of the quadratic model, q_BSpe as the BSpe from the fits of the quadratic model, q_CV0sq as the CV0^2 from the fits of the quadratic model, l_QI as the QI from the fits of the linear model (provided only if linear fit coefficients are also present in the input data frame, i.e., for LED fits only), and l_BSpe as the BSpe from the fits of the linear model (again, provided only if linear fit coefficients are also present in the input data frame, i.e., for LED fits only)

Author(s)

Josef Spidlen, Wayne Moore, Faysal El Khettabi

See Also

fit_led, fit_beads, fit_spherotech, fit_thermo_fisher

Examples

```
library(flowCore)
## Example is based on LED data from the flowQBData package
## library(flowQBData)
## flowQBData is available since BioConductor 3.4, please install it
## manually in order to be able to these examples
#
# fcs_directory <- system.file("extdata", "SSFF_LSRII", "LED_Series",
#     package="flowQBData")
# fcs_file_path_list <- list.files(fcs_directory, "*.fcs", full.names= TRUE)
# ## We are working with these FCS files:
# basename(fcs_file_path_list)
#
# ## Various house keeping information
# ## - Which channels should be ignored, typically the non-fluorescence
# ##    channels, such as the time and the scatter channels
# ignore_channels <- c("Time",
#     "FSC-A", "FSC-W", "FSC-H",
#     "SSC-A", "SSC-W", "SSC-H")
# ## - Which dyes would you typically use with the detectors
# dyes <- c("APC", "APC-Cy7", "APC-H7", "FITC", "PE", "PE-Cy7", "PerCP",
#     "PerCP-Cy55", "V450", "V500-C")
# ## - What are the corresponding detectors, provide a vector of short channel
# ## names, i.e., values of the $PnN FCS keywords.
# detectors <- c("APC-A", "APC-Cy7-A", "APC-Cy7-A", "FITC-A", "PE-A",
#     "PE-Cy7-A",
#     "PerCP-Cy5-5-A", "PerCP-Cy5-5-A", "Pacific Blue-A", "Aqua Amine-A")
# ## - The signal type that you are looking at (Area, Height or Width)
# signal_type <- "Area"
# ## - The instrument make/model
# instrument_name <- 'LSRII'
# ## - Set the minimum and maximum values, peaks with mean outsize of this range
# ## will be ignored
# bounds <- list(minimum = -100, maximum = 100000)
# ## - The minimum number of usable peaks (represented by different FCS files
# ## in case of an LED pulser) required in order for a fluorescence channel
# ## to be included in the fitting. Peaks with mean expression outside of the
# ## bounds specified above are omitted and therefore not considered useful
# minimum_fcs_files <- 3 # The default 3 seems to be work well in typical cases
# ## - What is the maximum number of iterations for iterative fitting with
# ## weight adjustments
# max_iterations <- 10 # The default 10 seems to be enough in typical cases
#
# ## Now, let's calculate the fitting
# led_results <- fit_led(fcs_file_path_list, ignore_channels, dyes,
#     detectors, signal_type, instrument_name, bounds = bounds,
#     minimum_useful_peaks = minimum_fcs_files, max_iterations = max_iterations)
#
```

```
# qb_from_fits(led_results$iterated_dye_fits)
```

`split_in_two` *Split an object in a low density region*

Description

This method finds a split in the low density region of the provided object (which shall be either a matrix or flowCore's flowFrame object) and returns a vector of TRUE/FALSE depending on whether each of the events (rows) are left or righ (i.e, lower or higher) than density peak. An FCS channel shall be specified if a flowFrame object with multiple channels in provided on the input. If matrix is the input then it shall contain one column only. This method is designed to work well for 2 level beads, such as stained and unstained, but will not provide meaningful results for multi-level beads or other data in general.

Usage

```
split_in_two(object, ...)
```

Arguments

`object`	Object of class `flowFrame` or a matrix.
`...`	Additional options, see the details section.

Details

Additional parameter of the method:

`channel` Which FCS channel shall be used in order to look for the split in the data? This is applicable if a flowFrame object with several channels is used on the input.

Value

A vector of TRUE/FALSE values depending on whether each of the events (rows) are below or above the identified split value.

Author(s)

Wayne Moore, Faysal El Khettabi, Josef Spidlen

Examples

```
library('flowCore')
## library(flowQBData)
## flowQBData is available since BioConductor 3.4, please install it
## manually in order to be able to these examples

#   fcsFilePath <- system.file("extdata", "SSFF_LSRII", "Other_Tests",
#       "933745.fcs", package="flowQBData")
#   myFlowFrame <- read.FCS(fcsFilePath)
#   ## Note that this is just to demonstrate the syntax, but doing this on the
#   ## FSC and SSC channels of this particular FCS file is not very meaningful
#   r1 <- split_in_two(myFlowFrame, 'FSC-H')
#   r2 <- split_in_two(exprs(myFlowFrame[,'SSC-H']))
```

Index