Package 'beadplexr'

May 7, 2026

Type Package

Title Analysis of Multiplex Cytometric Bead Assays

Version 0.5.0

Description Reproducible and automated analysis of multiplex bead assays such as CBA (Morgan et al. 2004; <doi:10.1016/j.clim.2003.11.017>), LEGENDplex (Yu et al. 2015; <doi:10.1084/jem.20142318>), and MACSPlex (Miltenyi Biotec 2014; Application note: Data acquisition and analysis without the MACSQuant analyzer; <https://www.miltenyibiotec.com/upload/assets/IM0021608.PDF>). The package provides functions for streamlined reading of fcs files, and identification of bead clusters and analyte expression. The package eases the calculation of standard curves and the subsequent calculation of the analyte concentration.

License MIT + file LICENSE

URL https://gitlab.com/ustervbo/beadplexr

BugReports https://gitlab.com/ustervbo/beadplexr/-/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.3)

Suggests spelling, gridExtra, hexbin, knitr, rmarkdown, stringr, testthat

VignetteBuilder knitr

RoxygenNote 7.2.3

Imports cluster, dplyr, drc, fpc, ggplot2, mclust, purrr, tibble, tidyr, yaml, rlang

Language en-US

NeedsCompilation no

Author Ulrik Stervbo [aut, cre] (ORCID: <https://orcid.org/0000-0002-2831-8868>)

Maintainer Ulrik Stervbo <ulrik.stervbo@gmail.com>

Repository CRAN

Date/Publication 2023-06-16 11:30:07 UTC

Contents

`approx_adjust` *Approximate bandwidth adjustment.*

Description

Approximates the adjust argument to `stats::density()` needed to find the required number of clusters.

Usage

```
approx_adjust(.x, .k, .lower = 0.4, .upper = 2, .step = 0.2)
```

Arguments

`.x`	A numeric vector.
`.k`	Numeric giving the number of expected clusters.
`.lower`, `.upper`	The interval for possible value of adjust.
`.step`	A numeric giving the increment to adjust. Sometimes low values are needed to find a proper adjust value.

Details

This function finds the first value of the `stats::density()` `adjust` argument which gives the `.k` number of clusters. It it quite crude in that every value of adjust from `.lower` to `.upper` is tested until the desired number of clusters is found. A cluster is defined by a peak, and should no suitable `adjust` value be found, NA is returned.

Value

A numeric.

Examples

```
set.seed(1234)
.x <- c(rnorm(100, 2, 1), rnorm(100, 9, 1))
approx_adjust(.x, 2)
```

---

as_data_frame_analyte *Cast list of analytes to* `data.frame`

Description

A well structured list, such at those loaded by `load_panel()`, is cast to a data.frame.

Usage

```
as_data_frame_analyte(
  .analytes,
  .id_bead = "Bead group",
  .id_analyte = "Analyte ID"
)
```

Arguments

`.analytes`	The named list to be cast. It usually is loaded using `load_panel()`. See Details for expected structure.
`.id_bead, .id_analyte`	The name of the column to hold the bead group and the analyte ID, respectively.

Details

Each analyte in the `list` passed to the function is expected to be a named list with named elements `name` and `concentration`. The name of the `list` with the analyte specific information is the analyte ID.

Because of the particular setup of the LEGENDplex assay with two bead groups, the analytes are expected to be wrapped in another `list`.

Value

A data-frame

Examples

```
.analytes <-  list(A = list(
                A1 = list(name = "name_a1", concentration = 500),
                A2 = list(name = "name_a2", concentration = 50000)))

as_data_frame_analyte(.analytes)
```

---

calculate_concentration

Calculate concentration.

Description

Calculate the concentration in a sample

Usage

```
calculate_concentration(
  df,
  .model,
  .parameter = "FL2.H",
  .value = "Calc.conc",
  .data = NULL
)
```

Arguments

df	A tidy data.frame.
.model	An object of class drc with the fitted dose-response model.
.parameter	A character giving the name of column(s) where populations are identified.
.value	A character giving the name of the column to store the calculated concentration
.data	Deprecated. Use df.

Value

The df with the calculated concentration and error added in two columns.

Examples

```
library(beadplexr)
library(drc)
data(ryegrass)

ryegrass_m <-
  fit_standard_curve(df = ryegrass,
                     .parameter = "rootl",
                     .concentration = "conc")

sample_data <-
  calculate_concentration(df = ryegrass[sample(1:nrow(ryegrass), 5),],
                          .model = ryegrass_m,
                          .parameter = "rootl")
```

`calc_analyte_mfi` *Calculate the analyte intensity*

Description

The mean fluorescence intensity (MFI) of the analyte

Usage

```
calc_analyte_mfi(
  df,
  .parameter,
  .column_name = "analyte",
  .mean_fun = c("geometric", "harmonic", "arithmetic"),
  .data = NULL
)
```

Arguments

`df`	A tidy data.frame.
`.parameter`	A character giving the name of column(s) where populations are identified.
`.column_name`	A character giving the name of the column to store the population information.
`.mean_fun`	A character giving the mean function to use.
`.data`	Deprecated. Use `df`.

Value

A summarized `data.frame`

Mean calculation

The calculation of the harmonic mean is as follows:

$$n \frac{1}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

NAs are removed before calculation

The geometric mean is given by:

$$\left(\prod_{i=1}^{n} x_i \right)^{\frac{1}{n}}$$

but implemented as:

$$\frac{1}{n} \exp \sum_{i=1}^{n} \log(x_i)$$

NAs are removed before calculation

Examples

```
library(beadplexr)
library(dplyr)

data("lplex")

df <- lplex[[1]] |>
  filter(`FSC-A` > 4e5L, `FSC-A` < 6.3e5L) |>
  identify_analyte(.parameter = "FL6-H",
                   .analyte_id = as.character(c(1:7)))

df |>
  calc_analyte_mfi(.parameter = "FL2-H")

df |>
  calc_analyte_mfi(.parameter = "FL2-H",
              .mean_fun = "harmonic")
```

---

calc_std_conc *Calculate standard concentration*

Description

Given a start concentration and dilution factor, the concentration of the given standard samples is calculated

Usage

```
calc_std_conc(.standard_sample, .start_concentration, .dilution_factor = 4L)
```

Arguments

`.standard_sample`	a vector giving the standard samples. The sample with the highest value is given the start concentration, and a `.standard_sample` with the value of 0, is set to 0 if it exists. See Details for details on how order is assessed.
`.start_concentration`	a numeric vector giving the initial standard concentration. If longer than one the maximum value is taken as start concentration.
`.dilution_factor`	a numeric vector giving the dilution factor. If a single element is passed, this is applied to all standard samples as a dilution series. If more then one value is given, it must be of equal length as the `.standard_sample`, and each element is taken as the dilution factor to the previous element, using 1 for the first element. The order of dilution factors must match that of the ordered `.standard_sample`.

Details

In the manuals to the LEGENDplex system, standard are labeled 0 to 8, where 8 indicate the highest concentration and 0 the background (no analyte). The standard is diluted at 1:4 so that

```
[s7] = [start]
[s6] = [s7]/4
[s5] = [s6]/4
[s4] = [s5]/4
[s3] = [s4]/4
[s2] = [s3]/4
[s1] = [s2]/4
[s0] = 0
```

It might happen, that a dilution step is missing in which case the dilution is corrected to accommodate the missing step. However, since it is inspired guess work and out of the ordinary, a warning is thrown, see Examples.

Value

A numeric vector

Standard sample order

If the vector is numeric, the values are ordered numerically from high to low.

If the vector is not numeric, things become a little more difficult, because sorting a vector like `c("a", "c", "0", "b")` by default results in `c("0", "a", "b", "c")`, which means that '0' is the highest value and will be assigned the start concentration and the sample 'a' is then the first dilution.

To avoid this problem, the vector is split into two: one containing numerical values and one containing alphabetical. Each vector is then sorted appropriately and combined, see Examples.

Examples

```
calc_std_conc(.standard_sample = c(7:0),
                                  .start_concentration = 5000)

suppressWarnings(
# Sample 5 is missing - raises a warning
calc_std_conc(.standard_sample = c(7, 6, 4, 3, 2, 1, 0),
                                  .start_concentration = 5000)
)

calc_std_conc(.standard_sample = rep(c(7:0), 2),
                                  .start_concentration = 5000)
calc_std_conc(.standard_sample = c(9:0),
                                 .start_concentration = 5000)

calc_std_conc(.standard_sample = c(letters[1:7], 0),
                                  .start_concentration = 5000)
calc_std_conc(.standard_sample = c(letters[1:7], 0, 1),
                                  .start_concentration = 5000)

calc_std_conc(.standard_sample = c(7:1, 0),
                                  .start_concentration = 5000,
                                  .dilution_factor = c(1, 2, 2, 2, 4, 6, 6, 0))

# If 0 exists it is always set to 0
calc_std_conc(.standard_sample = c(7:1, 0),
                                  .start_concentration = 5000,
                                  .dilution_factor = c(1, 2, 2, 2, 4, 6, 6, 100000))
calc_std_conc(.standard_sample = c(8:1),
                                  .start_concentration = 5000,
                                  .dilution_factor = c(1, 2, 2, 2, 4, 6, 6, 100000))
```

---

cluster_events *Clustering with trimming*

Description

Cluster identification with various algorithms and subsequent trimming of each cluster

Usage

```
bp_kmeans(df, .parameter, .column_name, .k, .trim = 0, .data = NULL, ...)

bp_clara(df, .parameter, .column_name, .k, .trim = 0, .data = NULL, ...)

bp_dbscan(
  df,
  .parameter,
  .column_name,
```

```
  .eps = 0.2,
  .MinPts = 50,
  .data = NULL,
  ...
)

bp_mclust(
  df,
  .parameter,
  .column_name,
  .k,
  .trim = 0,
  .sample_frac = 0.05,
  .max_subset = 500,
  .data = NULL,
  ...
)

bp_density_cut(df, .parameter, .column_name, .k, .trim = 0, .data = NULL, ...)
```

Arguments

df	A tidy data.frame.
.parameter	A character giving the name of column(s) where populations are identified.
.column_name	A character giving the name of the column to store the population information.
.k	Numeric giving the number of expected clusters, or a set of initial cluster centers.
.trim	A numeric between 0 and 1, giving the fraction of points to remove by marking them NA.
.data	Deprecated. Use df.
...	Additional arguments passed to appropriate methods, see below.
.eps	Reachability distance, see fpc::dbscan().
.MinPts	Reachability minimum no. of points, see fpc::dbscan().
.sample_frac	A numeric between 0 and 1 giving the fraction of points to use in initialisation of Mclust().
.max_subset	A numeric giving the maximum of events to use in initialisation of Mclust(), see below.

Value

The data.frame in df with the cluster classification added in the column given by .column_name.

Additional parameters

Information on additional arguments passed, can be found here:

clara cluster::clara()

kmeans `kmeans()`

dbscan `fpc::dbscan()`

mclust `mclust::Mclust()`

density_cut `approx_adjust()`

Default parameters to `clara()`

`cluster::clara()` is by default called with the following parameters:

samples 100

pamLike TRUE

Parameters to dbscan

It requires some trial and error to get the right parameters for the density based clustering, but the parameters usually stay stable throughout an entire experiment and over time (assuming that there is only little drift in the flow cytometer). There is no guarantee that the correct number of clusters are returned, and it might be better to use this on the forward - side scatter discrimination.

Scaling of the parameters seems to be appropriate in most cases for the forward - side scatter discrimination and is automatically performed.

Parameters to mclust

Mclust is is slow and memory hungry on large datasets. Using a subset of the data to initialise the clustering greatly improves the speed. I have found that a subset sample of 500 even works well and gives no markedly better clustering than a subset of 5000 events, but initialisation with 500 makes the clustering complete about 12 times faster than with 5000 events.

Parameters to density_cut

This simple function works by smoothing a density function until the desired number of clusters are found. The segregation of the clusters follows at the lowest point between two clusters.

See Also

`trim_population()`, `identify_analyte()`.

Mclust and dbscan seems to do an excellent job at separating on the forward and side scatter parameters. Mclust and clara both perform well separating beads in the APC channel, but clara is about 3 times faster than Mclust.

Examples

```
library(beadplexr)
library(dplyr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
```

```
  # Speed things up a bit by selecting one fourth of the events.
  # Probably not something you'd usually do
  dplyr::sample_frac(0.25) |>
  bp_kmeans(.parameter = c("FSC-A", "SSC-A"),
            .column_name = "population", .trim = 0.1, .k = 2) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()

library(beadplexr)
library(dplyr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  # Speed things up a bit by selecting one fourth of the events.
  # Probably not something you'd usually do
  dplyr::sample_frac(0.25) |>
  bp_clara(.parameter = c("FSC-A", "SSC-A"),
           .column_name = "population", .trim = 0.1, .k = 2) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()

lplex[[1]] |>
  # Speed things up a bit by selecting one fourth of the events.
  # Probably not something you'd usually do
  dplyr::sample_frac(0.25) |>
  bp_clara(.parameter = c("FSC-A", "SSC-A"),
           .column_name = "population", .trim = 0, .k = 2) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()

## Not run:
library(beadplexr)
library(dplyr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  # Speed things up a bit by selecting one fourth of the events.
  # Probably not something you'd usually do
  dplyr::sample_frac(0.25) |>
  bp_dbscan(.parameter = c("FSC-A", "SSC-A"), .column_name = "population",
            eps = 0.2, MinPts = 50) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()

pop1 <- lplex[[1]] |>
```

```
  # Speed things up a bit by selecting one fourth of the events.
  # Probably not something you'd usually do
  dplyr::sample_frac(0.25) |>
  bp_dbscan(.parameter = c("FSC-A", "SSC-A"), .column_name = "population",
    eps = 0.2, MinPts = 50) |>
  dplyr::filter(population == "1")

pop1 |>
  bp_dbscan(.parameter = c("FL6-H", "FL2-H"), .column_name = "population",
    eps = 0.2, MinPts = 50) |>
  pull(population) |>
  unique()

pop1 |>
  bp_dbscan(.parameter = c("FL6-H", "FL2-H"), .column_name = "population",
    eps = 0.2, MinPts = 50, scale = FALSE) |>
  pull(population) |>
  unique()

## End(Not run)
library(beadplexr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  bp_mclust(.parameter = c("FSC-A", "SSC-A"),
            .column_name = "population", .trim = 0, .k = 2) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()
library(beadplexr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  bp_density_cut(.parameter = c("FSC-A"),
            .column_name = "population", .trim = 0, .k = 2) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()
```

`despeckle` *Despeckle parameters*

Description

Remove lonely, noisy data points in a 2D scatter matrix

Usage

```
despeckle(df, .parameters, .bins = 256L, .neighbours = 4L, .data = NULL, ...)
```

Arguments

df	A tidy data.frame.
.parameters	A character of the length of two giving the parameters to despeckle.
.bins	A numeric giving the resolution of the raster matrix. Increasing the resolution results in more isolated events.
.neighbours	A numeric giving the minimum number of neighbours. Points with fewer neighbours are removed.
.data	Deprecated. Use df.
...	Deprecated. It's use has no effect.

Details

The values of the two parameters are binned into the given number of bins. They are then cast into a 2D matrix, with the bins of the first of the parameters ending up as rows, the bins of the second parameter as columns, and combinations are marked by 1.

The rows of the df where lonely points are found in .parameters are removed.

Value

A data.frame with noisy points removed.

Examples

```
library(beadplexr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  ggplot() +
  aes(x = `FL6-H`, y = `FL2-H`) +
  geom_point()

lplex[[1]] |>
  despeckle(.parameters = c("FL6-H", "FL2-H"), .neighbours = 8) |>
  ggplot() +
  aes(x = `FL6-H`, y = `FL2-H`) +
  geom_point()

lplex[[1]] |>
  despeckle(.parameters = c("FL6-H", "FL2-H"), .bin = 128) |>
  ggplot() +
  aes(x = `FL6-H`, y = `FL2-H`) +
  geom_point()
```

dist_chebyshev *Chebyshev distance*

Description

Chebyshev distance

Usage

```
dist_chebyshev(x, diag = FALSE, upper = FALSE)
```

Arguments

x	a numeric matrix or data frame.
diag	logical value indicating whether the diagonal of the distance matrix should be printed by print.dist.
upper	logical value indicating whether the upper triangle of the distance matrix should be printed by print.dist .

Value

Chebyshev distance returns an object of class "dist".

Examples

```
x <- matrix(rnorm(100), nrow = 5)
dist_chebyshev(x)
```

facs_plot *Plot FACS data.*

Description

Wrappers around building a ggplot with geom_point, geom_density_2d, and geom_hex.

Usage

```
facs_plot(
  df,
  .x = "FSC-A",
  .y = "SSC-A",
  .type = c("scatter", "density1d", "density2d", "hexbin"),
  .data = NULL,
  ...
```

```
)

facs_scatter(
  df,
  .x = "FSC-A",
  .y = "SSC-A",
  .beads = NULL,
  .plot_distinct = TRUE,
  .data = NULL
)

facs_density2d(df, .x = "FSC-A", .y = "SSC-A", .beads = NULL, .data = NULL)

facs_density1d(df, .x = "FSC-A", .beads = NULL, .data = NULL)

facs_hexbin(df, .x = "FSC-A", .y = "SSC-A", .bins = 75, .data = NULL)
```

Arguments

df	The data to be plotted in a data.frame.
.x, .y	Character vector with the column name for the variable to plot on the x or y-axis.
.type	Character vector giving the type of plot being used. Options are one of "scatter", "density", "hexbin".
.data	Deprecated. Use df.
...	Arguments passed to the individual functions.
.beads	Character vector to with the column name with identification of beads. If used it will show up with the aesthetic 'color'. Defaults to not being used.
.plot_distinct	Boolean to decide if only distinct events should be plotted. If used, the number of data points might be greatly reduced which could make for faster plotting. Defaults to TRUE.
.bins	Numeric vector giving number of bins in both vertical and horizontal directions. Set to 75 by default.

Details

These plot functions are meant to provide a quick way of viewing the FACS data. For more control, use ggplot2 directly.

Value

A ggplot

Examples

```
## Not run:
library(beadplexr)
data("lplex")
```

```
df <-  lplex[[1]]
df$bead_group <- ifelse(df$`FSC-A` < 4e5L, "A", "B")

# Using facs_plot
facs_plot(df, .type = "scatter")
facs_plot(df, .type = "density1d")
facs_plot(df, .type = "density2d")
facs_plot(df, .type = "hexbin")

facs_plot(df, .type = "scatter", .beads = "bead_group")
facs_plot(df, .type = "density1d", .beads = "bead_group")
facs_plot(df, .type = "hexbin", .bins = 50)

facs_plot(df, .x = "FL2-H", .type = "scatter", .beads = "bead_group")

# Individual functions
facs_scatter(df)

facs_scatter(df, .beads = "bead_group", .plot_distinct = FALSE)
facs_scatter(df, .beads = "bead_group")

facs_scatter(df, .x = "FL2-H", .y = "FL6-H", .beads = "bead_group")

facs_density1d(df)
facs_density1d(df, .beads = "bead_group")

facs_density2d(df)
facs_density2d(df, .beads = "bead_group")

facs_hexbin(df)
facs_hexbin(df, .bins = 30)

## End(Not run)
```

`fit_standard_curve` *Fit a standard curve*

Description

Fit a logistic function to the standard concentrations.

Usage

```
fit_standard_curve(
  df,
  .parameter = "FL2.H",
  .concentration = "Concentration",
  .fct = "LL.5",
  .data = NULL,
  ...
)
```

Arguments

df	A tidy data.frame.
.parameter	A character giving the name of column(s) where populations are identified.
.concentration	A character giving the name of the column with the standard concentration.
.fct	A character giving the name of the logistic function to use in the fit, see drc::drm() for details.
.data	Deprecated. Use df.
...	Other arguments to drc::drm()

Value

An object of class drc

Examples

```
library(beadplexr)
library(drc)
data(ryegrass)

ryegrass_m <-
  fit_standard_curve(df = ryegrass,
                     .parameter = "rootl",
                     .concentration = "conc")

summary(ryegrass_m)
```

---

identify_analyte *Identify analyte*

Description

Identify analyte

Usage

```
identify_analyte(
  df,
  .parameter,
  .analyte_id,
  .column_name = "analyte",
  .k = length(.analyte_id),
  .trim = 0,
  .desc = FALSE,
  .method = c("clara", "kmeans", "dbscan", "mclust", "density_cut"),
  .data = NULL,
  ...
)
```

Arguments

df	A tidy data.frame.
.parameter	A character giving the name of column(s) where populations are identified.
.analyte_id	A character vector giving the ID of the analyte. The **order** is important and must match the expected order of analytes.
.column_name	A character giving the name of the column to store the population information.
.k	Numeric giving the number of expected clusters, or a set of initial cluster centers.
.trim	A numeric between 0 and 1, giving the fraction of points to remove by marking them NA.
.desc	A boolean to indicate if the centers of the analytes should be arranged in a descending fashion before assigning the names.
.method	A character giving the clustering method to use.
.data	Deprecated. Use df.
...	Additional arguments passed to appropriate methods, see below.

Details

This function is a wrapper around the process of:

- Finding analyte clusters
- Trimming the clusters by removing the cluster members most distant from the cluster center
- Sorting the analyte clusters based on their centers
- Giving each analyte cluster a useful name

Value

A data.frame with analyte IDs in a separate column

Additional parameters

Information on additional arguments passed, can be found here:

clara `cluster::clara()`

kmeans `kmeans()`

dbscan `fpc::dbscan()`

mclust `mclust::Mclust()`

density_cut `approx_adjust()`

See Also

`cluster_events()`

Examples

```
## Not run:
library(beadplexr)
library(ggplot2)

data("lplex")

df <- lplex[[1]]
df |>
  identify_analyte(.parameter = c("FSC-A", "SSC-A"),
                       .analyte_id = c("A", "B"),
                       .column_name = "analyte",
                       .method = "clara", .trim = 0.02) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = analyte) +
  geom_point()

df |>
  identify_analyte(.parameter = c("FSC-A", "SSC-A"),
                       .analyte_id = c("A", "B"),
                       .column_name = "analyte",
                       .method = "clara", .desc = TRUE) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = analyte) +
  geom_point()

df |>
  identify_analyte(.parameter = c("FSC-A", "SSC-A"),
                       .analyte_id = c("A", "B"),
                       .column_name = "analyte",
                       .method = "dbscan") |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = analyte) +
  geom_point()

## End(Not run)
```

`identify_assay_analyte` *Identify multiplex assay analytes*

Description

Convenience functions to identify analytes in different multiplex systems.

Usage

```
identify_legendplex_analyte(df, .analytes, .method_args, .data = NULL)
```

```
identify_cba_analyte(
  df,
  .analytes,
  .method_args,
  .trim_fs = NULL,
  .parameter_fs = NULL,
  .data = NULL
)

identify_macsplex_analyte(
  df,
  .analytes,
  .method_args,
  .trim_fs = NULL,
  .parameter_fs = NULL,
  .data = NULL
)
```

Arguments

df	A tidy data.frame.
.analytes	A vector or list giving the IDs of the analytes. The **order** is important and must match the expected order of analytes.
.method_args	A list giving the parameters passed on to identify_analyte().
.data	Deprecated. Use df.
.trim_fs	A numeric between 0 and 1, giving the fraction of points to remove from the forward side scatter.
.parameter_fs	A character giving the names of the forward and side scatter parameters.

Details

These functions wraps around the process of:

- Trim or subset on forward side scatter
- Identifying analytes. For LEGENDplex in both bead groups

If the forward side scatter events are not trimmed, the function is equivalent to call identify_analyte() with CBA or MACSPlex data.

Value

A data.frame

Analytes

The parameter .analytes is either a simple vector with the IDs or, in the case of the LEGENDplex system, a list giving the IDs of analytes among the groups A and B.

A list for the LEGENDplex system might look like this:

```
  list(A = c("A1", "A2"),
       B = c("B1", "B2"))
```

The **order** of analyte IDs is important and must match the expected order of analytes.

Method arguments

The parameter `.method_args` is a list of key-value pairs passed to `identify_analyte()`.

Examples

```
## Not run:
library(beadplexr)
library(dplyr)
data("lplex")
df <- lplex[[1]]

panel_info <- load_panel(.panel_name = "Human Growth Factor Panel (13-plex)")

args_ident_analyte <- list(fs = list(.parameter = c("FSC-A", "SSC-A"),
                                     .column_name = "Bead group",
                                     .trim = 0.1,
                                     .method = "clara"),
                           analytes = list(.parameter = "FL6-H",
                                           .column_name = "Analyte ID",
                                           .trim = 0,
                                           .method = "clara"))

annot_events <- identify_legendplex_analyte(df = df,
                                            .analytes = panel_info$analytes,
                                            .method_args = args_ident_analyte)

annot_events |> facs_plot(.beads = "Bead group")

annot_events |>
  filter(`Bead group` == "A") |>
  facs_plot(.x = "FL2-H", .y = "FL6-H", .beads = "Analyte ID")

annot_events |>
  filter(`Bead group` == "B") |>
  facs_plot(.x = "FL2-H", .y = "FL6-H", .beads = "Analyte ID")

## End(Not run)
## Not run:
library(beadplexr)
data(simplex)

df <- simplex[["cba"]]

analytes <- vector("list", 30) |> setNames(as.character(c(1:30)))

args_ident_analyte <- list(.parameter = c("APC", "APC-Cy7"),
```

```
                          .column_name = "Analyte ID",
                          .trim = 0.1,
                          .method = "clara")
annot_events <- identify_cba_analyte(df = df,
                    .analytes = analytes,
                    .method_args = args_ident_analyte)

annot_events |> facs_plot(.x = "FSC", .y = "SSC")

annot_events |>
  facs_plot(.x = "APC", .y = "APC-Cy7", .beads = "Analyte ID")

annot_events <- identify_cba_analyte(df = df,
                    .analytes = analytes,
                    .method_args = args_ident_analyte,
                    .trim_fs = 0.1,
                    .parameter_fs = c("FSC", "SSC"))

annot_events |> facs_plot(.x = "FSC", .y = "SSC", .beads = "Bead events")

# Looks strange because some true beads events have randomly been placed far
# from the center in the forward-side scatter when the data was created
annot_events |>
  facs_plot(.x = "APC", .y = "APC-Cy7", .beads = "Analyte ID")

## End(Not run)
## Not run:
library(beadplexr)
data(simplex)

df <- simplex[["mplex"]]
analytes <- vector("list", 10) |> setNames(as.character(c(1:10)))

args_ident_analyte <- list(.parameter = c("FITC", "PE"),
                          .column_name = "Analyte ID",
                          .trim = 0.1,
                          .method = "clara")

annot_events <- identify_macsplex_analyte(df = df,
                                    .analytes = analytes,
                                    .method_args = args_ident_analyte)

annot_events |> facs_plot(.x = "FSC", .y = "SSC")

annot_events |>
  facs_plot(.x = "FITC", .y = "PE", .beads = "Analyte ID")

annot_events <- identify_macsplex_analyte(df = df,
                                    .analytes = analytes,
                                    .method_args = args_ident_analyte,
                                    .trim_fs = 0.1,
                                    .parameter_fs = c("FSC", "SSC"))
```

```
annot_events |> facs_plot(.x = "FSC", .y = "SSC", .beads = "Bead events")
# Looks strange because some true beads events have randomly been placed far
# from the center in the forward-side scatter when the data was created
annot_events |>
  facs_plot(.x = "FITC", .y = "PE", .beads = "Analyte ID")

## End(Not run)
```

`load_panel` *Load panel information*

Description

The panel information are stored in `resources` in the package directory and can be loaded by providing a file or panel name, or a search pattern

Usage

```
load_panel(.file_name = NULL, .panel_name = NULL, .panel_pattern = NULL)
```

Arguments

`.file_name`	Character vector giving the name of an external panel info file. See below for the expected components.
`.panel_name`	Character vector giving the name of the panel. See below for accepted panel names.
`.panel_pattern`	The pattern to look for. Can be a regular expression.

Details

If an explicit `.file_name` is given, `.panel_name` and `.panel_pattern` are ignored. If no file is given but a `.panel_name` is, the `.panel_pattern` i ignored. Only if no `.file_name` and `.panel_name` are given, is the `.panel_pattern` used.

Value

A `list`

Included panels

- Human Adipokine Panel* (13-plex)
- Human Anti-Virus Response Panel (13-plex)
- Human CD8/NK Panel (13-plex)
- Human Cytokine Panel 2 (13-plex)
- Human Growth Factor Panel (13-plex)
- Human Inflammation Panel (13-plex)

- Human Metabolic Panel 1 (4-plex)
- Human Proinflammatory Chemokine Panel (13-plex)
- Human T Helper Cytokine Panels (13-plex)
- Mouse Anti-Virus Response Panel (13-plex)
- Mouse Cytokine Panel 2 (13-plex)
- Mouse Free Active/Total TGF-b1 Panel (Mouse/Rat) (1-plex)
- Mouse HSC Panel (13-plex)
- Mouse IgE Panel (1-plex)
- Mouse Immunoglobulin Isotyping Panel (6-plex)
- Mouse Inflammation Panel (13-plex)
- Mouse Proinflammatory Chemokine Panel (13-plex)
- Mouse T Helper Cytokine Panels (13-plex)

Example panel file

The panel information files are formatted in YAML. It has three main parts:

- Some general information about the panel - The order of the major bead groups in the forward and side scatter - Analytes, where each bead ID is listed for each major bead group. The bead IDs are further complemented with the name of the analyte and the start concentration of the standard value

The 'Human Th Cytokine Panel (13-plex)' YAML file is found in `/resources/legendplex_human_th_cytokine_panel_13` of the package directory.

Examples

```
library(beadplexr)

.panel_name <- "Human T Helper Cytokine Panels (13-plex)"
panel_info <- load_panel(.panel_name = .panel_name)
panel_info$panel_name

.file_name <- system.file("resources",
                          "legendplex_human_cytokine_panel_2_13-plex.yml",
                          package = "beadplexr")
panel_info <- load_panel(.file_name = .file_name)
panel_info$panel_name

panel_info <- load_panel(.file_name = .file_name, .panel_name = .panel_name)
panel_info$panel_name

suppressWarnings(
# The pattern matches several files, which raises a warning
panel_info <- load_panel(.panel_pattern = "panel_2_13-plex")
)
panel_info$panel_name
```

lplex *LEGENDplex example data*

Description

Data from a "Human Growth Factor Panel (13-plex)" LEGENDplex experiment, with 8 controls and 1 human serum samples, all in duplicates. The beads were measured on a CytoFLEX cytometer, and the fcs-files were processed using `read_fcs()`, with default settings.

Usage

```
data("lplex")
```

Format

A list with 18 elements. Each element is a data.frame about 5000 rows and 4 columns (the exact number varies a little due to the data acquisition):

FSC-A The forward scatter parameter

SSC-A The side scatter parameter

FL6-H Intensity in the FL6 channel

FL2-H Intensity in the FL2 channel

The list contains 8 standard samples in duplicates, and one serum sample, also in duplicate. The names of each element have the format K3 (internal panel shorthand), C:number: for standards and S:number: for serum sample, and a number indicating the replicate (1 or 2).

Source

Ulrik Stervbo, 2016, Unpublished

plot_concentrations *Plot concentrations*

Description

Plot concentrations

Usage

```
plot_std_curve(
  df,
  .model,
  .title = NULL,
  .parameter = "FL2.H",
  .concentration = "Concentration",
  .data = NULL
)

plot_target_est_conc(
  df,
  .title = NULL,
  .concentration = "Calc.conc",
  .std_concentration = "Concentration",
  .data = NULL
)

plot_estimate(
  .sample_data,
  .standard_data,
  .model,
  .title = NULL,
  .parameter = "FL2.H",
  .concentration = "Concentration"
)
```

Arguments

df	A data.frame with the data to be plotted.
.model	An object of class drc with the fitted dose-response model.
.title	A character giving the title of the plot.
.parameter	A character giving the name of the column with the MFI
.concentration	A character giving the name of the column with the with the calculated concentrations.
.data	Deprecated. Use df.
.std_concentration	A character giving the name of the column with the standard concentration.
.sample_data	A data.frame with the calculated sample concentrations.
.standard_data	A data.frame with the calculated standard concentrations.

Value

A ggplot

Examples

```
library(beadplexr)
library(drc)
data(ryegrass)

ryegrass_m <-
  fit_standard_curve(df = ryegrass,
                     .parameter = "rootl",
                     .concentration = "conc")
recalc_std <-
  calculate_concentration(df = ryegrass,
                          .model = ryegrass_m,
                          .parameter = "rootl")
sample_data <-
  calculate_concentration(df = ryegrass[sample(1:nrow(ryegrass), 5),],
                          .model = ryegrass_m,
                          .parameter = "rootl")

plot_std_curve(ryegrass,
               ryegrass_m,
               .parameter = "rootl",
               .concentration = "conc")

plot_target_est_conc(df = recalc_std,
                     .concentration = "Calc.conc",
                     .std_concentration = "conc")
plot_estimate(
  .sample_data = sample_data,
  .standard_data = ryegrass,
  .model = ryegrass_m,
  .parameter = "rootl",
  .concentration = "conc")
```

---

read_fcs *Read a fcs file.*

Description

Is deprecated. See the vignette "Preparing flow-data for use with with beadplexr" for an example of preparing flow-data to be used with beadplexr.

Usage

```
read_fcs(
  .file_name,
  .fsc_ssc = c("FSC-A", "SSC-A"),
  .bead_channels = c("FL6-H", "FL2-H"),
 .filter = list(`FSC-A` = c(200000L, 800000L), `SSC-A` = c(200000L, 1000000L), `FL6-H` =
```

```
    c(7.3, Inf)),
  .compensation = "guess",
  ...
)
```

Arguments

`.file_name`	The path and name of the file to be read.
`.fsc_ssc`	The names of the forward and side scatter channels. A character vector of length of two.
`.bead_channels`	The names of the channels with bead events. A character vector of length of at least two.
`.filter`	Optional list of upper and lower cutoff for individual channels. Use `.filter = NULL` for no filtering at all.
`.compensation`	A character vector, a compensation matrix, or `NULL`. See 'Details' for extended information of the argument.
`...`	additional arguments passed to `flowCore::read.FCS`

`simplex` *Simulated beadplex data*

Description

Very simple, simulated multiplex data to demonstrate the clustering functionality of the **beadplexr** package on CBA and MACSPlex assays.

Usage

```
data(simplex)
```

Format

A list with three elements. Each element is a `data.frame` of 3000 to 9000 rows. The exact format depends on the assay simulated:

lplex Simulated LEGENDplex data. A single `data.frame` with the columns:

- **FSC** The forward scatter parameter
- **SSC** The side scatter parameter
- **APC** Intensity in the APC channel
- **PE** Intensity in the PE channel

mplex Simulated MACSPlex data. A single `data.frame` with the columns:

- **FSC** The forward scatter parameter
- **SSC** The side scatter parameter
- **FITC** Intensity in the FITC channel

PE Intensity in the PE channel

APC Intensity in the PE channel

cba Simulated CBA data. A single `data.frame` with the columns:

FSC The forward scatter parameter

SSC The side scatter parameter

APC Intensity in the APC channel

APC-Cy7 Intensity in the APC-Cy7 channel

PE Intensity in the PE channel

Source

Artificial

`trim_population` *Trim cluster.*

Description

Remove the points furthest form the center of the cluster.

Usage

```
trim_population(
  df,
  .parameter,
  .column_name = "population",
  .trim = 0.1,
  .data = NULL
)
```

Arguments

`df`	The tidy data.frame with clusters to be modified.
`.parameter`	A character giving the name of dimensions to calculate distance on.
`.column_name`	A character giving the name of the column with the cluster information.
`.trim`	A numeric between 0 and 1, giving the fraction of points to remove.
`.data`	Deprecated. Use `df`.

Details

The euclidean distance is calculated for each point defined by `.parameter` to the center of the cluster. The cluster designation of the `.trim` most distant points are changed to `NA`.

Value

A data.frame

Examples

```
library(beadplexr)
library(dplyr)
library(ggplot2)

data("lplex")

lplex[[1]] |>
  filter(`FSC-A` > 3.2e5L) |>
  mutate(population = "1") |>
 trim_population(.parameter = c("FSC-A", "SSC-A"), .column_name = "population", .trim = 0.1) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()

lplex[[1]] |>
  filter(`FSC-A` > 3.2e5L) |>
  mutate(population = "1") |>
  trim_population(.parameter = c("FSC-A", "SSC-A"),
                  .column_name = "population", .trim = 0.8) |>
  ggplot() +
  aes(x = `FSC-A`, y = `SSC-A`, colour = population) +
  geom_point()
```

---

turning_point *Turning points*

Description

Find turning points (minima and maxima) in a vector.

Usage

```
turning_point(
  .x,
  .which = c("both", "minima", "maxima"),
  .return = c("value", "index"),
  .adjust = 1.5,
  .k = NULL,
  ...
)
```

Arguments

.x	A numeric vector or a list of numeric vectors. If the list is named, the names become column names in the returned data.frames
.which	A character indicating the values of interest.
.return	A character giving the desired return type.

.adjust A numeric giving the adjustment to the adjust argument of stats::density().

.k Numeric giving the number of expected clusters.

... Arguments passed on to approx_adjust

.lower,.upper The interval for possible value of adjust.

.step A numeric giving the increment to adjust. Sometimes low values are needed to find a proper adjust value.

Value

A list with the two elements maxima and minima. each element consist of a single data.frame.

Examples

```
set.seed(1234)
.x <- c(rnorm(100, 2, 1), rnorm(100, 9, 1))

turning_point(.x = .x, .adjust = 1)
turning_point(.x = .x, .k = 2)

turning_point(.x = .x, .which = "minima")
turning_point(.x = .x, .which = "maxima")

turning_point(.x = .x, .return = "index")
```

Index