Package 'diffcyt'

November 11, 2026

Version 1.32.1

Title Differential discovery in high-dimensional cytometry via high-resolution clustering

Description Statistical methods for differential discovery analyses in high-dimensional cytometry data (including flow cytometry, mass cytometry or CyTOF, and oligonucleotide-tagged cytometry), based on a combination of high-resolution clustering and empirical Bayes moderated tests adapted from transcriptomics.

URL https://github.com/lmweber/diffcyt

BugReports https://github.com/lmweber/diffcyt/issues

License MIT + file LICENSE

biocViews ImmunoOncology, FlowCytometry, Proteomics, SingleCell, CellBasedAssays, CellBiology, Clustering, FeatureExtraction, Software

Depends R (>= 3.4.0)

Imports flowCore, FlowSOM, SummarizedExperiment, S4Vectors, limma, edgeR, lme4, multcomp, dplyr, tidyr, reshape2, magrittr, stats, methods, utils, grDevices, graphics, ComplexHeatmap, circlize, grid

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown, testthat, HDCytoData, CATALYST

RoxygenNote 7.1.1

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

git_branch RELEASE_3_23

git_last_commit 22a19f8

git_last_commit_date 2026-07-14

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Lukas M. Weber [aut, cre] (ORCID: <https://orcid.org/0000-0002-3282-1730>)

Maintainer Lukas M. Weber <weberlm3@gmail.com>

Contents

calcCounts . . . 2
calcMedians . . . 4
calcMediansByClusterMarker . . . 5
calcMediansBySampleMarker . . . 7
createContrast . . . 9
createDesignMatrix . . . 10
createFormula . . . 12
diffcyt . . . 14
generateClusters . . . 20
plotHeatmap . . . 22
prepareData . . . 25
testDA_edgeR . . . 28
testDA_GLMM . . . 31
testDA_voom . . . 34
testDS_limma . . . 37
testDS_LMM . . . 41
topClusters . . . 44
topTable . . . 45
transformData . . . 48

Index **51**

`calcCounts` *Calculate cluster cell counts*

Description

Calculate number of cells per cluster-sample combination

Usage

```
calcCounts(d_se)
```

Arguments

`d_se`	Data object from previous steps, in `SummarizedExperiment` format, containing cluster labels as a column in the row meta-data (from `generateClusters`).

Details

Calculate number of cells per cluster-sample combination (referred to as cluster cell 'counts', 'abundances', or 'frequencies').

The cluster cell counts are required for testing for differential abundance of cell populations, and are also used for weights and filtering when testing for differential states within cell populations.

Results are returned as a new `SummarizedExperiment` object, where rows = clusters, columns = samples, `assay` = values (counts). (Note that this structure differs from the input data object.)

Value

d_counts: SummarizedExperiment object, where rows = clusters, columns = samples, assay = values (counts).

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)
```

calcMedians *Calculate cluster medians*

Description

Calculate cluster medians (median expression for each cluster-sample-marker combination)

Usage

```
calcMedians(d_se)
```

Arguments

d_se	Data object from previous steps, in SummarizedExperiment format, containing cluster labels as a column in the row meta-data (from generateClusters). Column meta-data is assumed to contain a factor marker_class.

Details

Calculate median marker expression for each cluster and sample (i.e. medians for each cluster-sample-marker combination).

The data object is assumed to contain a factor marker_class in the column meta-data (see prepareData), which indicates the protein marker class for each column of data ("type", "state", or "none").

The cluster medians are required for testing for differential states within cell populations, and for plotting purposes.

Variables id_type_markers and id_state_markers are saved in the metadata slot of the output object. These can be used to identify the 'cell type' and 'cell state' markers in the list of assays in the output SummarizedExperiment object, which is useful in later steps of the 'diffcyt' pipeline.

Results are returned as a new SummarizedExperiment object, where rows = clusters, columns = samples, sheets (assays slot) = markers. Note that there is a separate table of values (assay) for each marker. The metadata slot also contains variables id_type_markers and id_state_markers, which can be used to identify the sets of cell type and cell state markers in the list of assays.

Value

d_medians: SummarizedExperiment object, where rows = clusters, columns = samples, sheets (assays slot) = markers. The metadata slot contains variables id_type_markers and id_state_markers, which can be accessed with metadata(d_medians)$id_type_markers and metadata(d_medians)$id_state_markers.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
```

```
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate medians
d_medians <- calcMedians(d_se)
```

`calcMediansByClusterMarker` *Calculate medians (by cluster and marker)*

Description

Calculate medians for each cluster-marker combination

Usage

```
calcMediansByClusterMarker(d_se)
```

Arguments

d_se	Data object from previous steps, in SummarizedExperiment format, containing cluster labels as a column in the row meta-data (from generateClusters). Column meta-data is assumed to contain a factor marker_class.

Details

Calculate median marker expression for each cluster, across all samples (i.e. medians for each cluster-marker combination).

The data object is assumed to contain a factor marker_class in the column meta-data (see prepareData), which indicates the protein marker class for each column of data ("type", "state", or "none"). Cluster medians are calculated for all markers.

The medians by cluster and marker are required for plotting purposes.

Variables id_type_markers and id_state_markers are saved in the metadata slot of the output object. These can be used to identify the 'cell type' and 'cell state' markers in the sequence of markers (columns) in the output object, which is useful in later steps of the 'diffcyt' pipeline.

Results are returned as a new SummarizedExperiment object, where rows = clusters, columns = markers, assay = values (marker expression values). The metadata slot also contains variables id_type_markers and id_state_markers, which can be used to identify the sets of cell type and cell state markers in the columns.

Value

d_medians_by_cluster_marker: SummarizedExperiment object, where rows = clusters, columns = markers, assay = values (marker expression values). The metadata slot contains variables id_type_markers and id_state_markers, which can be accessed with metadata(d_medians)$id_type_markers and metadata(d_medians)$id_state_markers.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)
```

```
experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate medians (by cluster and marker)
d_medians_by_cluster_marker <- calcMediansByClusterMarker(d_se)
```

`calcMediansBySampleMarker` *Calculate medians (by sample and marker)*

Description

Calculate medians for each sample-marker combination

Usage

```
calcMediansBySampleMarker(d_se)
```

Arguments

`d_se`	Data object from previous steps, in `SummarizedExperiment` format, containing cluster labels as a column in the row meta-data (from `generateClusters`). Column meta-data is assumed to contain a factor `marker_class`.

Details

Calculate overall median marker expression for each sample (i.e. medians for each sample-marker combination).

The data object is assumed to contain a factor `marker_class` in the column meta-data (see `prepareData`), which indicates the protein marker class for each column of data (`"type"`, `"state"`, or `"none"`). Cluster medians are calculated for all markers.

The medians by sample and marker are required for plotting purposes.

Variables `id_type_markers` and `id_state_markers` are saved in the `metadata` slot of the output object. These can be used to identify the 'cell type' and 'cell state' markers in the sequence of markers (columns) in the output object, which is useful in later steps of the 'diffcyt' pipeline.

Results are returned as a new `SummarizedExperiment` object, where rows = samples, columns = markers, `assay` = values (marker expression values). The `metadata` slot also contains variables `id_type_markers` and `id_state_markers`, which can be used to identify the sets of cell type and cell state markers in the columns.

Value

`d_medians_by_sample_marker`: `SummarizedExperiment` object, where rows = samples, columns = markers, `assay` = values (marker expression values). The `metadata` slot contains variables `id_type_markers` and `id_state_markers`, which can be accessed with `metadata(d_medians)$id_type_markers` and `metadata(d_medians)$id_state_markers`.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
```

```
                          levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate medians (by sample and marker)
d_medians_by_sample_marker <- calcMediansBySampleMarker(d_se)
```

`createContrast` *Create contrast matrix*

Description

Create contrast matrix for differential testing

Usage

```
createContrast(contrast)
```

Arguments

`contrast`	Vector defining the contrast of interest. This should be a numeric vector specifying the combination of model parameters to test whether they are equal to zero. The entries correspond to the columns of the design matrix, or the levels of the fixed effect terms in the model formula. For example, using a design matrix: `c(0, 1, 0, 0, 0)` to test whether a single parameter corresponding to the second column in the design matrix is equal to zero.

Details

Creates a contrast matrix specifying the comparison of interest, in the correct format for the differential testing functions. This can then be provided to the differential testing functions, together with either a design matrix or model formula, and the data object.

The argument `contrast` defines the contrast of interest. This should be a numeric vector specifying the combination of model parameters to test whether they are equal to zero. In many cases, this will simply be a vector of zeros and a single entry equal to one; this will test whether a single parameter is equal to zero (e.g. c(0, 1, 0, 0, 0)).

If a design matrix has been used, the entries of `contrast` correspond to the columns of the design matrix; and the length of `contrast` equals the number of columns in the design matrix. If a model

formula has been used, the entries correspond to the levels of the fixed effect terms; and the length equals the number of levels of the fixed effect terms.

The contrast matrix is formatted as a matrix with a single column containing the contrast of interest. To perform tests for multiple contrasts, run this function and the corresponding differential testing function multiple times.

Value

`contrast`: Returns a contrast matrix containing the contrast of interest, formatted as a matrix with a single column.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Example: contrast matrix
createContrast(c(0, 1, 0, 0, 0))
```

---

createDesignMatrix *Create design matrix*

Description

Create design matrix for model fitting

Usage

```
createDesignMatrix(experiment_info, cols_design = NULL)
```

Arguments

`experiment_info`	`data.frame`, `DataFrame`, or `tbl_df` of experiment information (which was also previously provided to `prepareData`). This should be a data frame containing all factors and covariates of interest; e.g. group IDs, block IDs, batch IDs, and continuous covariates.
`cols_design`	Argument specifying the columns of `experiment_info` to include in the design matrix. This can be provided as a character vector of column names, a numeric vector of column indices, or a logical vector. Default = all columns.

Details

Creates a design matrix specifying the models to be fitted. (Alternatively, `createFormula` can be used to generate a model formula instead of a design matrix.)

The design matrix can then be provided to the differential testing functions, together with the data object and contrast matrix.

The `experiment_info` input (which was also previously provided to `prepareData`) should be a data frame containing all factors and covariates of interest. For example, depending on the experimental design, this may include the following columns:

- group IDs (e.g. groups for differential testing)
- block IDs (e.g. patient IDs in a paired design)
- batch IDs (batch effects)
- continuous covariates

The argument `cols_design` specifies which columns in `experiment_info` to include in the design matrix. (For example, there may be an additional column of sample IDs, which should not be included.) This can be provided as a character vector of column names, a numeric vector of column indices, or a logical vector. By default, all columns are included.

Columns of indicator variables (e.g. group IDs, block IDs, and batch IDs) in `experiment_info` must be formatted as factors (otherwise they will be treated as numeric values). The indicator columns will be expanded into the design matrix format. The names for each parameter are taken from the column names of `experiment_info`.

All factors provided here will be included as fixed effect terms in the design matrix. Alternatively, to use random effects for some factors (e.g. for block IDs), see `createFormula`; or, depending on the method used, provide them directly to the differential testing function (`testDA_voom` and `testDS_limma`).

Value

`design`: Returns a design matrix (numeric matrix), with one row per sample, and one column per model parameter.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Example: simple design matrix
experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)
createDesignMatrix(experiment_info, cols_design = "group_id")

# Example: more complex design matrix: patient IDs and batch IDs
experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:8)),
```

```
  group_id = factor(rep(paste0("group", 1:2), each = 4)),
  patient_id = factor(rep(paste0("patient", 1:4), 2)),
  batch_id = factor(rep(paste0("batch", 1:2), 4)),
  stringsAsFactors = FALSE
)
createDesignMatrix(experiment_info, cols_design = c("group_id", "patient_id", "batch_id"))

# Example: more complex design matrix: continuous covariate
experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  age = c(52, 35, 71, 60),
  stringsAsFactors = FALSE
)
createDesignMatrix(experiment_info, cols_design = c("group_id", "age"))
```

`createFormula` *Create model formula and corresponding data frame of variables*

Description

Create model formula and corresponding data frame of variables for model fitting

Usage

```
createFormula(experiment_info, cols_fixed = NULL, cols_random = NULL)
```

Arguments

`experiment_info`	`data.frame`, `DataFrame`, or `tbl_df` of experiment information (which was also previously provided to `prepareData`). This should be a data frame containing all factors and covariates of interest; e.g. group IDs, block IDs, batch IDs, and continuous covariates.
`cols_fixed`	Argument specifying columns of `experiment_info` to include as fixed effect terms in the model formula. This can be provided as a character vector of column names, a numeric vector of column indices, or a logical vector.
`cols_random`	Argument specifying columns of `experiment_info` to include as random intercept terms in the model formula. This can be provided as a character vector of column names, a numeric vector of column indices, or a logical vector. Default = none.

Details

Creates a model formula and corresponding data frame of variables specifying the models to be fitted. (Alternatively, `createDesignMatrix` can be used to generate a design matrix instead of a model formula.)

The output is a list containing the model formula and corresponding data frame of variables (one column per formula term). These can then be provided to differential testing functions that require a model formula, together with the main data object and contrast matrix.

The `experiment_info` input (which was also previously provided to `prepareData`) should be a data frame containing all factors and covariates of interest. For example, depending on the experimental design, this may include the following columns:

- group IDs (e.g. groups for differential testing)
- block IDs (e.g. patient IDs in a paired design; these may be included as either fixed effect or random effects)
- batch IDs (batch effects)
- continuous covariates
- sample IDs (e.g. to include random intercept terms for each sample, to account for overdispersion typically seen in high-dimensional cytometry data; this is known as an 'observation-level random effect' (OLRE); see see Nowicka et al., 2017, *F1000Research* for more details)

The arguments `cols_fixed` and `cols_random` specify the columns in `experiment_info` to include as fixed effect terms and random intercept terms respectively. These can be provided as character vectors of column names, numeric vectors of column indices, or logical vectors. The names for each formula term are taken from the column names of `experiment_info`.

Note that for some methods, random effect terms (e.g. for block IDs) must be provided directly to the differential testing function instead (`testDA_voom` and `testDS_limma`).

If there are no random effect terms, it will usually be simpler to use a design matrix instead of a model formula; see `createDesignMatrix`.

Value

`formula`: Returns a list with three elements:

- `formula`: model formula
- `data`: data frame of variables corresponding to the model formula
- `random_terms`: TRUE if model formula contains any random effect terms

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Example: model formula
experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:8)),
  group_id = factor(rep(paste0("group", 1:2), each = 4)),
  patient_id = factor(rep(paste0("patient", 1:4), 2)),
  stringsAsFactors = FALSE
)
createFormula(experiment_info, cols_fixed = "group_id", cols_random = c("sample_id", "patient_id"))
```

`diffcyt` *Run 'diffcyt' pipeline*

Description

Wrapper function to run complete 'diffcyt' pipeline

Usage

```
diffcyt(
  d_input,
  experiment_info = NULL,
  marker_info = NULL,
  design = NULL,
  formula = NULL,
  contrast,
  analysis_type = c("DA", "DS"),
  method_DA = c("diffcyt-DA-edgeR", "diffcyt-DA-voom", "diffcyt-DA-GLMM"),
  method_DS = c("diffcyt-DS-limma", "diffcyt-DS-LMM"),
  markers_to_test = NULL,
  clustering_to_use = NULL,
  cols_to_include = NULL,
  subsampling = FALSE,
  n_sub = NULL,
  seed_sub = NULL,
  transform = TRUE,
  cofactor = 5,
  cols_clustering = NULL,
  xdim = 10,
  ydim = 10,
  meta_clustering = FALSE,
  meta_k = 40,
  seed_clustering = NULL,
  min_cells = 3,
  min_samples = NULL,
  normalize = FALSE,
  norm_factors = "TMM",
  trend_method = "none",
  block_id = NULL,
  trend = TRUE,
  weights = TRUE,
  plot = FALSE,
  path = ".",
  verbose = TRUE
)
```

Arguments

d_input	Input data. Must be either: (i) a flowSet or list of flowFrames, DataFrames, data.frames, or matrices as input (one flowFrame or list item per sample) (see prepareData); or (ii) a CATALYST daFrame (containing cluster labels in rowData; see vignette for an example).
experiment_info	data.frame, DataFrame, or tbl_df of experiment information, for example sample IDs and group IDs. Must contain a column named sample_id. See prepareData. (Not required when providing a CATALYST daFrame for d_input.)
marker_info	data.frame, DataFrame, or tbl_df of marker information for each column of data. This should contain columns named marker_name and marker_class. The columns contain: (i) marker names (and any other column names); and (ii) a factor indicating the marker class for each column (with entries "type", "state", or "none"). See prepareData. (Not required when providing a CATALYST daFrame for d_input.)
design	Design matrix, created with createDesignMatrix. See createDesignMatrix.
formula	Model formula object, created with createFormula. See createFormula.
contrast	Contrast matrix, created with createContrast. See createContrast.
analysis_type	Type of differential analysis to perform: differential abundance (DA) of cell populations, or differential states (DS) within cell populations. Options are "DA" and "DS". See testDA_edgeR, testDA_voom, testDA_GLMM, testDS_limma, or testDS_LMM.
method_DA	Method to use for calculating differential abundance (DA) tests. Options are "diffcyt-DA-edgeR", "diffcyt-DA-voom", and "diffcyt-DA-GLMM". Default = "diffcyt-DA-edgeR". See testDA_edgeR, testDA_voom, or testDA_GLMM.
method_DS	Method to use for calculating differential state (DS) tests. Options are "diffcyt-DS-limma" and "diffcyt-DS-LMM". Default = "diffcyt-DS-limma". See testDS_limma or testDS_LMM.
markers_to_test	(Optional) Logical vector specifying which markers to test for differential expression (from the set of markers stored in the assays of d_medians; for method testDS_limma or testDS_LMM). Default = all 'cell state' markers, which are identified by the logical vector id_state_markers stored in the meta-data of d_medians. See testDS_limma or testDS_LMM.
clustering_to_use	(Optional) Column name indicating which set of cluster labels to use for differential testing, when input data are provided as a CATALYST daFrame object containing multiple sets of cluster labels. (In this case, the metadata of the daFrame object is assumed to contain a data frame named cluster_codes; clustering_to_use is the column name of the selected column in cluster_codes. If clustering_to_use is provided, an identifier clustering_name to identify this column will also be saved in the metadata of the output object.) Default = NULL, in which case cluster labels stored in column named cluster_id in the rowData of the daFrame object are used.

cols_to_include
: Logical vector indicating which columns to include from the input data. Default = all columns. See prepareData.

subsampling
: Whether to use random subsampling to select an equal number of cells from each sample. Default = FALSE. See prepareData.

n_sub
: Number of cells to select from each sample by random subsampling, if subsampling = TRUE. Default = number of cells in smallest sample. See prepareData.

seed_sub
: Random seed for subsampling. Set to an integer value to generate reproducible results. Default = NULL. See prepareData.

transform
: Whether to apply 'arcsinh' transform. This may be set to FALSE if the input data has already been transformed. Default = TRUE. See transformData.

cofactor
: Cofactor parameter for 'arcsinh' transform. Default = 5, which is appropriate for mass cytometry (CyTOF) data. For fluorescence flow cytometry, we recommend cofactor = 150 instead. See transformData.

cols_clustering
: Columns to use for clustering. Default = NULL, in which case markers identified as 'cell type' markers (with entries "type") in the vector marker_class in the column meta-data of d_se will be used. See generateClusters.

xdim
: Horizontal length of grid for self-organizing map for FlowSOM clustering (number of clusters = xdim * ydim). Default = 10 (i.e. 100 clusters). See generateClusters.

ydim
: Vertical length of grid for self-organizing map for FlowSOM clustering (number of clusters = xdim * ydim). Default = 10 (i.e. 100 clusters). See generateClusters.

meta_clustering
: Whether to include FlowSOM 'meta-clustering' step. Default = FALSE. See generateClusters.

meta_k
: Number of meta-clusters for FlowSOM, if meta-clustering = TRUE. Default = 40. See generateClusters.

seed_clustering
: Random seed for clustering. Set to an integer value to generate reproducible results. Default = NULL. See generateClusters.

min_cells
: Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples. See testDA_edgeR, testDA_voom, testDA_GLMM, testDS_limma, or testDS_LMM.

min_samples
: Filtering parameter. Default = number of samples / 2, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples. See testDA_edgeR, testDA_voom, testDA_GLMM, testDS_limma, or testDS_LMM.

normalize
: Whether to include optional normalization factors to adjust for composition effects. Default = FALSE. See testDA_edgeR, testDA_voom, or testDA_GLMM.

norm_factors
: Normalization factors to use, if normalize = TRUE. Default = "TMM", in which case normalization factors are calculated automatically using the 'trimmed mean of M-values' (TMM) method from the edgeR package. Alternatively, a vector of values can be provided (the values should multiply to 1). See testDA_edgeR, testDA_voom, or testDA_GLMM.

trend_method — Method for estimating dispersion trend; passed to function estimateDisp from edgeR package (for method testDA_edgeR). Default = "none". (See estimateDisp help file from edgeR package for other options.) See testDA_edgeR.

block_id — (Optional) Vector or factor of block IDs (e.g. patient IDs) for paired experimental designs, to be included as random effects (for method testDA_voom or testDS_limma). If provided, the block IDs will be included as random effects using the limma duplicateCorrelation methodology. Alternatively, block IDs can be included as fixed effects in the design matrix (createDesignMatrix). See testDA_voom or testDS_limma.

trend — (Optional) Whether to fit a mean-variance trend when calculating moderated tests with function eBayes from limma package (for method testDS_limma). When trend = TRUE, this is known as the limma-trend method (Law et al., 2014; Phipson et al., 2016). Default = TRUE. See testDS_limma.

weights — (Optional) Whether to include precision weights (for method testDS_limma or testDS_LMM). For method testDS_limma, cluster cell counts will be used as precision weights (across all samples and clusters); this allows the limma model fitting functions to account for uncertainty due to the total number of cells per sample (library sizes) and total number of cells per cluster. For methods testDS_LMM, cluster cell counts will be used as precision weights within each model (across samples, i.e. within the model for each cluster); these represent the relative uncertainty in calculating each median value (within each model). Default = TRUE. See testDS_limma or testDS_LMM.

plot — Whether to save diagnostic plots (for method testDA_voom or testDS_limma). Default = FALSE. See testDA_voom or testDS_limma.

path — Path for diagnostic plots, if plot = TRUE (for method testDA_voom or testDS_limma). Default = current working directory. See testDA_voom or testDS_limma.

verbose — Whether to print status messages during each step of the pipeline. Default = TRUE.

Details

This wrapper function runs the complete 'diffcyt' analysis pipeline, by calling the functions for the individual steps in the pipeline in the correct sequence.

For more details about the functions for the individual steps, see the package vignette and the function help pages. Running the individual functions may provide additional flexibility, especially for complex analyses.

The input data can be provided as a flowSet or a list of flowFrames, DataFrames, data.frames, or matrices (one flowFrame or list item per sample). Alternatively, it is also possible to provide the input as a daFrame object from the CATALYST Bioconductor package (Chevrier, Crowell, Zanotelli et al., 2018). This can be useful when initial exploratory analyses and clustering have been performed using CATALYST; the daFrame object from CATALYST (containing cluster labels in the rowData) can then be provided directly to the diffcyt functions for differential testing.

Minimum required arguments when not providing a flowSet or list of flowFrames, DataFrames, data.frames, or matrices:

- d_input

- experiment_info
- marker_info
- either design or formula (depending on the differential testing method used)
- contrast
- analysis_type

Minimum required arguments when providing a CATALYST daFrame object:

- d_input
- either design or formula (depending on the differential testing method used)
- contrast
- analysis_type

Value

Returns a list containing the results object res, as well as the data objects d_se, d_counts, d_medians, d_medians_by_cluster_marker, and d_medians_by_sample_marker. (If a CATALYST daFrame object was used as input, the output list contains objects res, d_counts, and d_medians.) The structure of res depends on the differential testing method used. See testDA_edgeR, testDA_voom, testDA_GLMM, testDS_limma, or testDS_LMM.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

# Add differential states (DS) signal
```

```
ix_DS <- 901:1000
ix_cols_DS <- 19:20
d_input[[1]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[2]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[3]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)
d_input[[4]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters (using default method 'diffcyt-DA-edgeR')
out_DA <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DA", method_DA = "diffcyt-DA-edgeR",
                  seed_clustering = 123, verbose = FALSE)

# Test for differential states (DS) within clusters (using default method 'diffcyt-DS-limma')
out_DS <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DS", method_DS = "diffcyt-DS-limma",
                  seed_clustering = 123, verbose = FALSE)

# Display results for top DA clusters
topTable(out_DA, format_vals = TRUE)

# Display results for top DS cluster-marker combinations
topTable(out_DS, format_vals = TRUE)

# Plot heatmap for DA tests
plotHeatmap(out_DA, analysis_type = "DA")

# Plot heatmap for DS tests
plotHeatmap(out_DS, analysis_type = "DS")
```

`generateClusters` *Generate clusters*

Description

Generate high-resolution clusters for `diffcyt` analysis

Usage

```
generateClusters(
  d_se,
  cols_clustering = NULL,
  xdim = 10,
  ydim = 10,
  meta_clustering = FALSE,
  meta_k = 40,
  seed_clustering = NULL,
  ...
)
```

Arguments

`d_se`	Transformed input data, from `prepareData` and `transformData`.
`cols_clustering`	Columns to use for clustering. Default = `NULL`, in which case markers identified as 'cell type' markers (with entries `"type"`) in the vector `marker_class` in the column meta-data of `d_se` will be used.
`xdim`	Horizontal length of grid for self-organizing map for FlowSOM clustering (number of clusters = `xdim` * `ydim`). Default = 10 (i.e. 100 clusters).
`ydim`	Vertical length of grid for self-organizing map for FlowSOM clustering (number of clusters = `xdim` * `ydim`). Default = 10 (i.e. 100 clusters).
`meta_clustering`	Whether to include FlowSOM 'meta-clustering' step. Default = `FALSE`.
`meta_k`	Number of meta-clusters for FlowSOM, if `meta-clustering` = `TRUE`. Default = 40.
`seed_clustering`	Random seed for clustering. Set to an integer value to generate reproducible results. Default = `NULL`.
`...`	Other parameters to pass to the FlowSOM clustering algorithm (through the function `BuildSOM`).

Details

Performs clustering to group cells into clusters representing cell populations or subsets, which can then be further analyzed by testing for differential abundance of cell populations or differential

states within cell populations. By default, we use high-resolution clustering or over-clustering (i.e. we generate a large number of small clusters), which helps ensure that rare populations are adequately separated from larger ones.

Data is assumed to be in the form of a SummarizedExperiment object generated with prepareData and transformed with transformData.

The input data object d_se is assumed to contain a vector marker_class in the column meta-data. This vector indicates the marker class for each column ("type", "state", or "none"). By default, clustering is performed using the 'cell type' markers only. For example, in immunological data, this may be the lineage markers. The choice of cell type markers is an important design choice for the user, and will depend on the underlying experimental design and research questions. It may be made based on prior biological knowledge or using data-driven methods. For an example of a data-driven method of marker ranking and selection, see Nowicka et al. (2017), *F1000Research*.

By default, we use the FlowSOM clustering algorithm (Van Gassen et al. 2015, *Cytometry Part A*, available from Bioconductor) to generate the clusters. We previously showed that FlowSOM gives very good clustering performance for high-dimensional cytometry data, for both major and rare cell populations, and is extremely fast (Weber and Robinson, 2016, *Cytometry Part A*).

The clustering is run at high resolution to give a large number of small clusters (i.e. over-clustering). This is done by running only the initial 'self-organizing map' clustering step in the FlowSOM algorithm, i.e. without the final 'meta-clustering' step. This ensures that small or rare populations are adequately separated from larger populations, which is crucial for detecting differential signals for extremely rare populations.

The minimum spanning tree (MST) object from BuildMST is stored in the experiment metadata slot in the SummarizedExperiment object d_se, and can be accessed with metadata(d_se)$MST.

Value

d_se: Returns the SummarizedExperiment input object, with cluster labels for each cell stored in an additional column of row meta-data. Row meta-data can be accessed with rowData. The minimum spanning tree (MST) object is also stored in the metadata slot, and can be accessed with metadata(d_se)$MST.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
```

```
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)
```

---

plotHeatmap *Plot heatmap*

Description

Plot heatmap showing top clusters or cluster-marker combinations

Usage

```
plotHeatmap(
  out = NULL,
  analysis_type = c("DA", "DS"),
  top_n = 20,
  threshold = 0.1,
  res = NULL,
  d_se = NULL,
  d_counts = NULL,
  d_medians = NULL,
  d_medians_by_cluster_marker = NULL,
  sample_order = NULL
)
```

Arguments

out	Output object from diffcyt wrapper function, containing results object res and data objects d_se, d_counts, d_medians, and d_medians_by_cluster_marker. Alternatively, the results and data objects can be provided individually.
analysis_type	Whether to plot heatmap for differential abundance (DA) or differential state (DS) test results.
top_n	Number of top clusters (DA tests) or cluster-marker combinations (DS tests) to display. Default = 20.
threshold	Threshold for significant adjusted p-values. Default = 0.1.
res	Object containing differential test results. Alternatively, the combined output object from the wrapper function diffcyt can be provided.
d_se	Data object. Alternatively, the combined output object from the wrapper function diffcyt can be provided.
d_counts	Data object. Alternatively, the combined output object from the wrapper function diffcyt can be provided.
d_medians	Data object. (Required for DS tests only.) Alternatively, the combined output object from the wrapper function diffcyt can be provided.
d_medians_by_cluster_marker	Data object. Alternatively, the combined output object from the wrapper function diffcyt can be provided.
sample_order	(Optional) Custom ordering for samples (columns) in right-hand panel of heatmap. (This is useful when the default ordering does not group samples by condition; e.g. samples are ordered alphabetically by sample IDs instead.)

Details

Display heatmap to visualize results for the top (most highly significant) detected clusters or cluster-marker combinations.

For DA tests, the heatmap consists of the following panels:

- median (arcsinh-transformed) expression (across all samples) for 'cell type' markers
- cluster abundances by sample
- row annotation indicating significant detected clusters

For DS tests, the heatmap consists of:

- median (arcsinh-transformed) expression (across all samples) for 'cell type' markers
- median (arcsinh-transformed) expression (across all samples) for 'cell state' markers
- median (arcsinh-transformed) expression (by sample) for 'cell state' markers for the top cluster-marker combinations
- row annotation indicating significant detected cluster-marker combinations

Heatmaps are generated using the ComplexHeatmap package (Gu et al., 2016), and color scales are generated using the circlize package (Gu et al., 2014). Both packages are available from Bioconductor.

Value

Displays a heatmap.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

# Add differential states (DS) signal
ix_DS <- 901:1000
ix_cols_DS <- 19:20
d_input[[1]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[2]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[3]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)
d_input[[4]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)
```

```
# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters (using default method 'diffcyt-DA-edgeR')
out_DA <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DA", method_DA = "diffcyt-DA-edgeR",
                  seed_clustering = 123, verbose = FALSE)

# Test for differential states (DS) within clusters (using default method 'diffcyt-DS-limma')
out_DS <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DS", method_DS = "diffcyt-DS-limma",
                  seed_clustering = 123, verbose = FALSE)

# Display results for top DA clusters
topTable(out_DA, format_vals = TRUE)

# Display results for top DS cluster-marker combinations
topTable(out_DS, format_vals = TRUE)

# Plot heatmap for DA tests
plotHeatmap(out_DA, analysis_type = "DA")

# Plot heatmap for DS tests
plotHeatmap(out_DS, analysis_type = "DS")
```

`prepareData` *Prepare data*

Description

Prepare data into format for `diffcyt` pipeline

Usage

```
prepareData(
  d_input,
  experiment_info,
  marker_info,
  cols_to_include = NULL,
  subsampling = FALSE,
  n_sub = NULL,
  seed_sub = NULL
)
```

Arguments

d_input	Input data. Must be a flowSet or list of flowFrames, DataFrames, data.frames, or matrices as input (one flowFrame or list item per sample).
experiment_info	data.frame, DataFrame, or tbl_df of experiment information, for example sample IDs and group IDs. Must contain a column named sample_id.
marker_info	data.frame, DataFrame, or tbl_df of marker information for each column of data. This should contain columns named marker_name and marker_class. The columns contain: (i) marker names (and any other column names); and (ii) a factor indicating the marker class for each column (with entries "type", "state", or "none").
cols_to_include	Logical vector indicating which columns to include from the input data. Default = all columns.
subsampling	Whether to use random subsampling to select an equal number of cells from each sample. Default = FALSE.
n_sub	Number of cells to select from each sample by random subsampling, if subsampling = TRUE. Default = number of cells in smallest sample.
seed_sub	Random seed for subsampling. Set to an integer value to generate reproducible results. Default = NULL.

Details

Functions in the diffcyt analysis pipeline assume that input data is provided as a SummarizedExperiment object, which contains a single matrix of expression values, together with row and column meta-data.

This function accepts a flowSet or a list of flowFrames, data.frames, or matrices as input (i.e. one flowFrame or list item per sample). The function then concatenates the data tables into a single matrix of values, and adds row and column meta-data.

Row meta-data should be provided as a data frame named experiment_info, containing columns of relevant experiment information, such as sample IDs and group IDs (for each sample). This must contain at least a column named sample_id.

Column meta-data should be provided as a data frame named marker_info, containing the following columns of marker information. The column names must be as shown.

- marker_name: protein marker names (and column names for any other columns)
- marker_class: factor indicating the protein marker class for each column of data (usually, entries will be either "type", "state", or "none")

The split into 'cell type' and 'cell state' markers is crucial for the analysis. Cell type markers are used to define cell populations by clustering, and to test for differential abundance of cell populations; while cell state markers are used to test for differential states within cell populations.

The optional argument cols_to_include allows unnecessary columns (e.g. any columns not containing protein markers) to be discarded.

Optionally, random subsampling can be used to select an equal number of cells from each sample (`subsampling = TRUE`). This can be useful when there are large differences in total numbers of cells per sample, since it ensures that samples with relatively large numbers of cells do not dominate the clustering. However, subsampling should generally not be used when rare cell populations are of interest, due to the significant loss of information if cells from the rare population are discarded.

Value

`d_se`: Returns data as a `SummarizedExperiment` containing a single matrix of data (expression values) in the `assays` slot, together with row meta-data (experiment information) and column meta-data (marker information). The `metadata` slot also contains the `experiment_info` data frame, and a vector `n_cells` of the number of cells per sample; these can be accessed with `metadata(d_se)$experiment_info` and `metadata(d_se)$n_cells`.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)
```

testDA_edgeR *Test for differential abundance: method 'diffcyt-DA-edgeR'*

Description

Calculate tests for differential abundance of cell populations using method 'diffcyt-DA-edgeR'

Usage

```
testDA_edgeR(
  d_counts,
  design,
  contrast,
  trend_method = "none",
  min_cells = 3,
  min_samples = NULL,
  normalize = FALSE,
  norm_factors = "TMM"
)
```

Arguments

d_counts	SummarizedExperiment object containing cluster cell counts, from calcCounts.
design	Design matrix, created with createDesignMatrix. See createDesignMatrix for details.
contrast	Contrast matrix, created with createContrast. See createContrast for details.
trend_method	Method for estimating dispersion trend; passed to function estimateDisp from edgeR package. Default = "none". (See estimateDisp help file from edgeR package for other options.)
min_cells	Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.
min_samples	Filtering parameter. Default = number of samples / 2, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.
normalize	Whether to include optional normalization factors to adjust for composition effects (see details). Default = FALSE.
norm_factors	Normalization factors to use, if normalize = TRUE. Default = "TMM", in which case normalization factors are calculated automatically using the 'trimmed mean of M-values' (TMM) method from the edgeR package. Alternatively, a vector of values can be provided (the values should multiply to 1).

Details

Calculates tests for differential abundance of clusters, using functions from the edgeR package.

This method uses the edgeR package (Robinson et al. 2010, *Bioinformatics*; McCarthy et al. 2012, *Nucleic Acids Research*) to fit models and calculate moderated tests at the cluster level. Moderated tests improve statistical power by sharing information on variability (i.e. variance across samples for a single cluster) between clusters. By default, we use the option `trend.method = "none"` to calculate dispersions, since the dispersion-mean relationship typically does not resemble RNA-sequencing data; see edgeR User's Guide. The statistical methods implemented in the edgeR package were originally designed for the analysis of gene expression data such as RNA-sequencing counts. Here, we apply these methods to cluster cell counts.

The experimental design must be specified using a design matrix, which can be created with createDesignMatrix. Flexible experimental designs are possible, including blocking (e.g. paired designs), batch effects, and continuous covariates. See createDesignMatrix for more details.

The contrast matrix specifying the contrast of interest can be created with createContrast. See createContrast for more details.

Filtering: Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples. This removes clusters with very low cell counts across conditions, to improve power.

Normalization for the total number of cells per sample (library sizes) and total number of cells per cluster is automatically performed by the edgeR functions. Optional normalization factors can also be included to adjust for composition effects in the cluster cell counts per sample. For example, in an extreme case, if several additional clusters are present in only one condition, while all other clusters are approximately equally abundant between conditions, then simply normalizing by the total number of cells per sample will create a false positive differential abundance signal for the non-differential clusters. (For a detailed explanation in the context of RNA sequencing gene expression, see Robinson and Oshlack, 2010.) Normalization factors can be calculated automatically using the 'trimmed mean of M-values' (TMM) method (Robinson and Oshlack, 2010), implemented in the edgeR package (see also the edgeR User's Guide for details). Alternatively, a vector of values can be provided (the values should multiply to 1).

Value

Returns a new SummarizedExperiment object, with differential test results stored in the `rowData` slot. Results include raw p-values (`p_val`) and adjusted p-values (`p_adj`) from the edgeR moderated tests, which can be used to rank clusters by evidence for differential abundance. Additional output columns from the edgeR tests are also included. The results can be accessed with the rowData accessor function.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
```

```
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)

# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters
res_DA <- testDA_edgeR(d_counts, design, contrast)
```

testDA_GLMM *Test for differential abundance: method 'diffcyt-DA-GLMM'*

Description

Calculate tests for differential abundance of cell populations using method 'diffcyt-DA-GLMM'

Usage

```
testDA_GLMM(
  d_counts,
  formula,
  contrast,
  min_cells = 3,
  min_samples = NULL,
  normalize = FALSE,
  norm_factors = "TMM"
)
```

Arguments

d_counts	SummarizedExperiment object containing cluster cell counts, from calcCounts.
formula	Model formula object, created with createFormula. This should be a list containing three elements: formula, data, and random_terms: the model formula, data frame of corresponding variables, and variable indicating whether the model formula contains any random effect terms. See createFormula for details.
contrast	Contrast matrix, created with createContrast. See createContrast for details.
min_cells	Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.
min_samples	Filtering parameter. Default = number of samples / 2, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.
normalize	Whether to include optional normalization factors to adjust for composition effects (see details). Default = FALSE.
norm_factors	Normalization factors to use, if normalize = TRUE. Default = "TMM", in which case normalization factors are calculated automatically using the 'trimmed mean of M-values' (TMM) method from the edgeR package. Alternatively, a vector of values can be provided (the values should multiply to 1).

Details

Calculates tests for differential abundance of clusters, using generalized linear mixed models (GLMMs).

This methodology was originally developed and described by Nowicka et al. (2017), *F1000Research*, and has been modified here to make use of high-resolution clustering to enable investigation of rare cell populations. Note that unlike the original method by Nowicka et al., we do not attempt to manually merge clusters into canonical cell populations. Instead, results are reported at the high-resolution cluster level, and the interpretation of significant differential clusters is left to the user via visualizations such as heatmaps (see the package vignette for an example).

This method fits generalized linear mixed models (GLMMs) for each cluster, and calculates differential tests separately for each cluster. The response variables in the models are the cluster cell counts, which are assumed to follow a binomial distribution. There is one model per cluster. We also include a filtering step to remove clusters with very small numbers of cells, to improve statistical power.

For more details on the statistical methodology, see Nowicka et al. (2017), *F1000Research* (section 'Differential cell population abundance'.)

The experimental design must be specified using a model formula, which can be created with `createFormula`. Flexible experimental designs are possible, including blocking (e.g. paired designs), batch effects, and continuous covariates. Blocking variables can be included as either random intercept terms or fixed effect terms (see `createFormula`). For paired designs, we recommend using random intercept terms to improve statistical power; see Nowicka et al. (2017), *F1000Research* for details. Batch effects and continuous covariates should be included as fixed effects. In addition, we include random intercept terms for each sample to account for overdispersion typically seen in high-dimensional cytometry count data. The sample-level random intercept terms are known as 'observation-level random effects' (OLREs); see Nowicka et al. (2017), *F1000Research* for more details.

The contrast matrix specifying the contrast of interest can be created with `createContrast`. See `createContrast` for more details.

Filtering: Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples. This removes clusters with very low cell counts across conditions, to improve power.

Normalization: Optional normalization factors can be included to adjust for composition effects in the cluster cell counts per sample. For example, in an extreme case, if several additional clusters are present in only one condition, while all other clusters are approximately equally abundant between conditions, then simply normalizing by the total number of cells per sample will create a false positive differential abundance signal for the non-differential clusters. (For a detailed explanation in the context of RNA sequencing gene expression, see Robinson and Oshlack, 2010.) Normalization factors can be calculated automatically using the 'trimmed mean of M-values' (TMM) method (Robinson and Oshlack, 2010), implemented in the `edgeR` package (see also the `edgeR` User's Guide for details). Alternatively, a vector of values can be provided (the values should multiply to 1).

Value

Returns a new `SummarizedExperiment` object, with differential test results stored in the `rowData` slot. Results include raw p-values (`p_val`) and adjusted p-values (`p_adj`), which can be used to rank clusters by evidence for differential abundance. The results can be accessed with the `rowData` accessor function.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
```

```
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)

# Create model formula
formula <- createFormula(experiment_info, cols_fixed = "group_id", cols_random = "sample_id")
```

```
# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters
res_DA <- testDA_GLMM(d_counts, formula, contrast)
```

`testDA_voom` *Test for differential abundance: method 'diffcyt-DA-voom'*

Description

Calculate tests for differential abundance of cell populations using method 'diffcyt-DA-voom'

Usage

```
testDA_voom(
  d_counts,
  design,
  contrast,
  block_id = NULL,
  min_cells = 3,
  min_samples = NULL,
  normalize = FALSE,
  norm_factors = "TMM",
  plot = FALSE,
  path = "."
)
```

Arguments

`d_counts`	`SummarizedExperiment` object containing cluster cell counts, from `calcCounts`.
`design`	Design matrix, created with `createDesignMatrix`. See `createDesignMatrix` for details.
`contrast`	Contrast matrix, created with `createContrast`. See `createContrast` for details.
`block_id`	(Optional) Vector or factor of block IDs (e.g. patient IDs) for paired experimental designs, to be included as random effects. If provided, the block IDs will be included as random effects using the `limma` `duplicateCorrelation` methodology. Alternatively, block IDs can be included as fixed effects in the design matrix (`createDesignMatrix`). See details.
`min_cells`	Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples.
`min_samples`	Filtering parameter. Default = `number of samples / 2`, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples.

normalize	Whether to include optional normalization factors to adjust for composition effects (see details). Default = FALSE.
norm_factors	Normalization factors to use, if normalize = TRUE. Default = "TMM", in which case normalization factors are calculated automatically using the 'trimmed mean of M-values' (TMM) method from the edgeR package. Alternatively, a vector of values can be provided (the values should multiply to 1).
plot	Whether to save diagnostic plots for the limma voom transformations. Default = FALSE.
path	Path for diagnostic plots, if plot = TRUE. Default = current working directory.

Details

Calculates tests for differential abundance of clusters, using functions from the limma package and voom method.

This method uses the limma package (Ritchie et al. 2015, *Nucleic Acids Research*) to fit models and calculate moderated tests at the cluster level. Moderated tests improve statistical power by sharing information on variability (i.e. variance across samples for a single cluster) between clusters. Since count data are often heteroscedastic, we use the voom method (Law et al. 2014, *Genome Biology*) to transform the raw cluster cell counts and estimate observation-level weights to stabilize the mean-variance relationship. Diagnostic plots are shown if plot = TRUE.

The experimental design must be specified using a design matrix, which can be created with createDesignMatrix. Flexible experimental designs are possible, including blocking (e.g. paired designs), batch effects, and continuous covariates. See createDesignMatrix for more details.

For paired designs, either fixed effects or random effects can be used. Fixed effects are simpler, but random effects may improve power in data sets with unbalanced designs or very large numbers of samples. To use fixed effects, provide the block IDs (e.g. patient IDs) to createDesignMatrix. To use random effects, provide the block_id argument here instead. This will make use of the limma duplicateCorrelation methodology. Note that >2 measures per sample are not possible in this case (fixed effects should be used instead). Block IDs should not be included in the design matrix if the limma duplicateCorrelation methodology is used.

The contrast matrix specifying the contrast of interest can be created with createContrast. See createContrast for more details.

Filtering: Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples. This removes clusters with very low cell counts across conditions, to improve power.

Normalization for the total number of cells per sample (library sizes) and total number of cells per cluster is automatically performed by the limma and voom functions. Optional normalization factors can also be included to adjust for composition effects in the cluster cell counts per sample. For example, in an extreme case, if several additional clusters are present in only one condition, while all other clusters are approximately equally abundant between conditions, then simply normalizing by the total number of cells per sample will create a false positive differential abundance signal for the non-differential clusters. (For a detailed explanation in the context of RNA sequencing gene expression, see Robinson and Oshlack, 2010.) Normalization factors can be calculated automatically using the 'trimmed mean of M-values' (TMM) method (Robinson and Oshlack, 2010), implemented in the edgeR package (see also the edgeR User's Guide for details). Alternatively, a vector of values can be provided (the values should multiply to 1).

Value

Returns a new SummarizedExperiment object, with differential test results stored in the rowData slot. Results include raw p-values (p_val) and adjusted p-values (p_adj) from the limma moderated tests, which can be used to rank clusters by evidence for differential abundance. Additional output columns from the limma tests are also included. The results can be accessed with the rowData accessor function.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
```

```
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)

# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters
res_DA <- testDA_voom(d_counts, design, contrast)
```

`testDS_limma` *Test for differential states: method 'diffcyt-DS-limma'*

Description

Calculate tests for differential states within cell populations using method 'diffcyt-DS-limma'

Usage

```
testDS_limma(
  d_counts,
  d_medians,
  design,
  contrast,
  block_id = NULL,
  trend = TRUE,
  weights = TRUE,
  markers_to_test = NULL,
  min_cells = 3,
  min_samples = NULL,
  plot = FALSE,
  path = "."
)
```

Arguments

`d_counts`	`SummarizedExperiment` object containing cluster cell counts, from `calcCounts`.
`d_medians`	`SummarizedExperiment` object containing cluster medians (median marker expression for each cluster-sample combination), from `calcMedians`. Assumed to contain a logical vector `id_state_markers` in the meta-data (accessed with

metadata(d_medians)$id_state_markers), which identifies the set of 'cell state' markers in the list of assays.

design — Design matrix, created with createDesignMatrix. See createDesignMatrix for details.

contrast — Contrast matrix, created with createContrast. See createContrast for details.

block_id — (Optional) Vector or factor of block IDs (e.g. patient IDs) for paired experimental designs, to be included as random effects. If provided, the block IDs will be included as random effects using the limma duplicateCorrelation methodology. Alternatively, block IDs can be included as fixed effects in the design matrix (createDesignMatrix). See details.

trend — (Optional) Whether to fit a mean-variance trend when calculating moderated tests with function eBayes from limma package. When trend = TRUE, this is known as the limma-trend method (Law et al., 2014; Phipson et al., 2016). Default = TRUE.

weights — (Optional) Whether to use cluster cell counts as precision weights (across all samples and clusters); this allows the limma model fitting functions to account for uncertainty due to the total number of cells per sample (library sizes) and total number of cells per cluster. Default = TRUE.

markers_to_test — (Optional) Logical vector specifying which markers to test for differential expression (from the set of markers stored in the assays of d_medians). Default = all 'cell state' markers, which are identified by the logical vector id_state_markers stored in the meta-data of d_medians.

min_cells — Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.

min_samples — Filtering parameter. Default = number of samples / 2, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples.

plot — Whether to save diagnostic plot. Default = FALSE.

path — Path for diagnostic plot, if plot = TRUE. Default = current working directory.

Details

Calculates tests for differential states within cell populations (i.e. differential expression of cell state markers within clusters). Clusters are defined using cell type markers, and cell states are characterized by the median transformed expression of cell state markers.

This method uses the limma package (Ritchie et al. 2015, *Nucleic Acids Research*) to fit models and calculate moderated tests at the cluster level. Moderated tests improve statistical power by sharing information on variability (i.e. variance across samples for a single cluster) between clusters. By default, we provide option trend = TRUE to the limma eBayes function; this fits a mean-variance trend when calculating moderated tests, which is also known as the limma-trend method (Law et al., 2014; Phipson et al., 2016). Diagnostic plots are shown if plot = TRUE.

The experimental design must be specified using a design matrix, which can be created with createDesignMatrix. Flexible experimental designs are possible, including blocking (e.g. paired designs), batch effects, and continuous covariates. See createDesignMatrix for more details.

For paired designs, either fixed effects or random effects can be used. Fixed effects are simpler, but random effects may improve power in data sets with unbalanced designs or very large numbers of samples. To use fixed effects, provide the block IDs (e.g. patient IDs) to createDesignMatrix. To use random effects, provide the block_id argument here instead. This will make use of the limma duplicateCorrelation methodology. Note that >2 measures per sample are not possible in this case (fixed effects should be used instead). Block IDs should not be included in the design matrix if the limma duplicateCorrelation methodology is used.

The contrast matrix specifying the contrast of interest can be created with createContrast. See createContrast for more details.

By default, differential tests are performed for all cell state markers (which are identified with the vector id_state_markers stored in the meta-data of the cluster medians input object). The optional argument markers_to_test allows the user to specify a different set of markers to test (e.g. to investigate differences for cell type markers).

Filtering: Clusters are kept for differential testing if they have at least min_cells cells in at least min_samples samples. This removes clusters with very low cell counts across conditions, to improve power.

Weights: By default, cluster cell counts are used as precision weights (across all samples and clusters); allowing the limma model fitting functions to account for uncertainty due to the total number of cells per sample (library sizes) and total number of cells per cluster. This option can also be disabled with weights = FALSE, if required.

Value

Returns a new SummarizedExperiment object, where rows = cluster-marker combinations, and columns = samples. In the rows, clusters are repeated for each cell state marker (i.e. the sheets or assays from the previous d_medians object are stacked into a single matrix). Differential test results are stored in the rowData slot. Results include raw p-values (p_val) and adjusted p-values (p_adj) from the limma moderated tests, which can be used to rank cluster-marker combinations by evidence for differential states within cell populations. Additional output columns from the limma tests are also included. The results can be accessed with the rowData accessor function.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
```

```
  sample4 = d_random()
)

# Add differential states (DS) signal
ix_DS <- 901:1000
ix_cols_type <- 1:10
ix_cols_DS <- 19:20
d_input[[1]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[2]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[3]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)
d_input[[4]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)

# Calculate medians
d_medians <- calcMedians(d_se)

# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential states (DS) within clusters
res_DS <- testDS_limma(d_counts, d_medians, design, contrast)
```

testDS_LMM *Test for differential states: method 'diffcyt-DS-LMM'*

Description

Calculate tests for differential states within cell populations using method 'diffcyt-DS-LMM'

Usage

```
testDS_LMM(
  d_counts,
  d_medians,
  formula,
  contrast,
  weights = TRUE,
  markers_to_test = NULL,
  min_cells = 3,
  min_samples = NULL
)
```

Arguments

d_counts	SummarizedExperiment object containing cluster cell counts, from calcCounts.
d_medians	SummarizedExperiment object containing cluster medians (median marker expression for each cluster-sample combination), from calcMedians. Assumed to contain a logical vector id_state_markers in the meta-data (accessed with metadata(d_medians)$id_state_markers), which identifies the set of 'cell state' markers in the list of assays.
formula	Model formula object, created with createFormula. This should be a list containing three elements: formula, data, and random_terms: the model formula, data frame of corresponding variables, and variable indicating whether the model formula contains any random effect terms. See createFormula for details.
contrast	Contrast matrix, created with createContrast. See createContrast for details.
weights	(Optional) Whether to include precision weights within each model (across samples, i.e. within the model for each cluster); these represent the relative uncertainty in calculating each median value (within each model). Accepts values of TRUE, FALSE, or a numeric vector of custom weights. Default = TRUE, in which case cluster cell counts are used as weights.
markers_to_test	(Optional) Logical vector specifying which markers to test for differential expression (from the set of markers stored in the assays of d_medians). Default = all 'cell state' markers, which are identified by the logical vector id_state_markers stored in the meta-data of d_medians.

`min_cells` Filtering parameter. Default = 3. Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples.

`min_samples` Filtering parameter. Default = `number of samples / 2`, which is appropriate for two-group comparisons (of equal size). Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples.

Details

Calculates tests for differential states within cell populations (i.e. differential expression of cell state markers within clusters), using linear mixed models (LMMs). Clusters are defined using cell type markers, and cell states are characterized by the median transformed expression of cell state markers.

This methodology was originally developed and described by Nowicka et al. (2017), *F1000Research*, and has been modified here to make use of high-resolution clustering to enable investigation of rare cell populations. Note that unlike the original method by Nowicka et al., we do not attempt to manually merge clusters into canonical cell populations. Instead, results are reported at the high-resolution cluster level, and the interpretation of significant differential clusters is left to the user via visualizations such as heatmaps (see the package vignette for an example).

This method fits linear mixed models (LMMs) for each cluster-marker combination (cell state markers only), and calculates differential tests separately for each cluster-marker combination. The response variable in each model is the median arcsinh-transformed marker expression of the cell state marker, which is assumed to follow a Gaussian distribution. There is one model per cluster per cell state marker. Within each model, sample-level weights are included (by default) for the number of cells per sample; these weights represent the relative uncertainty in calculating each median value. (Additional uncertainty exists due to variation in the total number of cells per cluster; however, it is not possible to account for this, since there are separate models for each cluster-marker combination.) We also include a filtering step to remove clusters with very small numbers of cells, to improve statistical power.

For more details on the statistical methodology, see Nowicka et al. (2017), *F1000Research* (section 'Differential analysis of marker expression stratified by cell population'.)

The experimental design must be specified using a model formula, which can be created with `createFormula`. Flexible experimental designs are possible, including blocking (e.g. paired designs), batch effects, and continuous covariates. Blocking variables can be included as either random intercept terms or fixed effect terms (see `createFormula`). For paired designs, we recommend using random intercept terms to improve statistical power; see Nowicka et al. (2017), *F1000Research* for details. Batch effects and continuous covariates should be included as fixed effects.

If no random intercept terms are included in the model formula, model fitting is performed using a linear model (LM) instead of a LMM.

The contrast matrix specifying the contrast of interest can be created with `createContrast`. See `createContrast` for more details.

By default, differential tests are performed for all cell state markers (which are identified with the vector `id_state_markers` stored in the meta-data of the cluster medians input object). The optional argument `markers_to_test` allows the user to specify a different set of markers to test (e.g. to investigate differences for cell type markers).

Filtering: Clusters are kept for differential testing if they have at least `min_cells` cells in at least `min_samples` samples. This removes clusters with very low cell counts across conditions, to improve power.

Weights: By default, cluster cell counts are used as precision weights within each model (across samples only, i.e. within the model for each cluster); these represent the relative uncertainty in calculating each median value (within each model). See above for details.

Value

Returns a new `SummarizedExperiment` object, where rows = cluster-marker combinations, and columns = samples. In the rows, clusters are repeated for each cell state marker (i.e. the sheets or `assays` from the previous `d_medians` object are stacked into a single matrix). Differential test results are stored in the `rowData` slot. Results include raw p-values (`p_val`) and adjusted p-values (`p_adj`), which can be used to rank cluster-marker combinations by evidence for differential states within cell populations. The results can be accessed with the `rowData` accessor function.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential states (DS) signal
ix_DS <- 901:1000
ix_cols_type <- 1:10
ix_cols_DS <- 19:20
d_input[[1]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[2]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[3]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)
d_input[[4]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)
```

```
marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)

# Generate clusters
d_se <- generateClusters(d_se)

# Calculate counts
d_counts <- calcCounts(d_se)

# Calculate medians
d_medians <- calcMedians(d_se)

# Create model formula
formula <- createFormula(experiment_info, cols_fixed = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential states (DS) within clusters
res_DS <- testDS_LMM(d_counts, d_medians, formula, contrast)
```

`topClusters` *Alias for 'topTable' (deprecated)*

Description

Alias for function 'topTable' (deprecated)

Usage

```
topClusters(...)
```

Arguments

`...`	See arguments for function `topTable`

Details

The function `topClusters` has been renamed to `topTable`, to more accurately reflect the structure of the results (results are returned for either clusters or cluster-marker combinations, depending on the type of differential tests performed).

This alias is provided for backward compatibility. The new function name `topTable` should be used whenever possible.

See `topTable` for details.

`topTable` *Show table of results for top clusters or cluster-marker combinations*

Description

Show table of results for top (most highly significant) clusters or cluster-marker combinations

Usage

```
topTable(
  res,
  d_counts = NULL,
  d_medians = NULL,
  order = TRUE,
  order_by = "p_adj",
  all = FALSE,
  top_n = 20,
  show_counts = FALSE,
  show_props = FALSE,
  show_meds = FALSE,
  show_logFC = FALSE,
  show_all_cols = FALSE,
  sort_cols = TRUE,
  format_vals = FALSE,
  digits = 3
)
```

Arguments

`res`	Output object from either the `diffcyt` wrapper function or one of the individual differential testing functions (`testDA_edgeR`, `testDA_voom`, `testDA_GLMM`, `testDS_limma`, or `testDS_LMM`). If the output object is from the wrapper function, the objects `res` and `d_counts` will be automatically extracted. Alternatively, these can be provided directly.
`d_counts`	(Optional) `SummarizedExperiment` object containing cluster cell counts, from `calcCounts`. (If the output object from the wrapper function is provided, this will be be automatically extracted.)

d_medians	(Optional) SummarizedExperiment object containing cluster medians (median marker expression for each cluster-sample combination), from calcMedians. Assumed to contain a logical vector id_state_markers in the meta-data (accessed with metadata(d_medians)$id_state_markers), which identifies the set of 'cell state' markers in the list of assays. (If the output object from the wrapper function is provided, this will be be automatically extracted.)
order	Whether to order results by values in column order_by (default: column p_adj containing adjusted p-values). Default = TRUE.
order_by	Name of column to use to order rows by values, if order = TRUE. Default = "p_adj" (adjusted p-values); other options include "p_val", "cluster_id", and "marker_id".
all	Whether to display all clusters or cluster-marker combinations (instead of top top_n). Default = FALSE.
top_n	Number of clusters or cluster-marker combinations to display (if all = FALSE). Default = 20.
show_counts	Whether to display cluster cell counts by sample (from d_counts). Default = FALSE.
show_props	Whether to display cluster cell count proportions by sample (calculated from d_counts). Default = FALSE.
show_meds	Whether to display median expression values for each cluster-marker combination (from d_medians). Default = FALSE.
show_logFC	Whether to display log fold change (logFC) values. Default = FALSE.
show_all_cols	Whether to display all columns from output object (e.g. logFC, logCPM, LR, etc.) Default = FALSE.
sort_cols	Whether to sort columns of counts, proportions, and medians; by levels of factor sample_id in colData of d_medians (requires object d_medians to be provided). Default = TRUE.
format_vals	Whether to display rounded values in numeric columns. This improves readability of the summary table, but should not be used when exact numeric values are required for subsequent steps (e.g. plotting). Default = FALSE.
digits	Number of significant digits to show, if format_vals = TRUE. Default = 3. (Note: for percentages shown if show_props = TRUE, digits = 1 is used.)

Details

Summary function to display table of results for top (most highly significant) detected clusters or cluster-marker combinations.

The differential testing functions return results in the form of p-values and adjusted p-values for each cluster (DA tests) or cluster-marker combination (DS tests), which can be used to rank the clusters or cluster-marker combinations by their evidence for differential abundance or differential states. The p-values and adjusted p-values are stored in the rowData of the output SummarizedExperiment object generated by the testing functions.

This function displays a summary table of results. By default, the top_n clusters or cluster-marker combinations are shown, ordered by adjusted p-values. Optionally, cluster counts, proportions, and

median expression by cluster-marker combination can also be included. The `format_vals` and `digits` arguments can be used to display rounded values to improve readability of the summary table.

Value

Returns a `DataFrame` table of results for the `top_n` clusters or cluster-marker combinations, ordered by values in column `order_by` (default: adjusted p-values). Optionally, cluster counts, proportions, and median expression by cluster-marker combination are also included.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

# Add differential abundance (DA) signal
ix_DA <- 801:900
ix_cols_type <- 1:10
d_input[[3]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)
d_input[[4]][ix_DA, ix_cols_type] <- d_random(n = 1000, mean = 2, ncol = 10)

# Add differential states (DS) signal
ix_DS <- 901:1000
ix_cols_DS <- 19:20
d_input[[1]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[2]][ix_DS, ix_cols_type] <- d_random(n = 1000, mean = 3, ncol = 10)
d_input[[3]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)
d_input[[4]][ix_DS, c(ix_cols_type, ix_cols_DS)] <- d_random(n = 1200, mean = 3, ncol = 12)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)

marker_info <- data.frame(
```

```
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Create design matrix
design <- createDesignMatrix(experiment_info, cols_design = "group_id")

# Create contrast matrix
contrast <- createContrast(c(0, 1))

# Test for differential abundance (DA) of clusters (using default method 'diffcyt-DA-edgeR')
out_DA <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DA", method_DA = "diffcyt-DA-edgeR",
                  seed_clustering = 123, verbose = FALSE)

# Test for differential states (DS) within clusters (using default method 'diffcyt-DS-limma')
out_DS <- diffcyt(d_input, experiment_info, marker_info,
                  design = design, contrast = contrast,
                  analysis_type = "DS", method_DS = "diffcyt-DS-limma",
                  seed_clustering = 123, verbose = FALSE)

# Display results for top DA clusters
topTable(out_DA, format_vals = TRUE)

# Display results for top DS cluster-marker combinations
topTable(out_DS, format_vals = TRUE)
```

`transformData` *Transform data*

Description

Transform data prior to clustering

Usage

```
transformData(d_se, cofactor = 5)
```

Arguments

`d_se`	Input data. Assumed to be in the form of a `SummarizedExperiment`, prepared with the function `prepareData`. Column meta-data is assumed to contain a factor `marker_class`, where entries `"none"` indicate non-marker columns.
`cofactor`	Cofactor parameter for 'arcsinh' transform. Default = 5, which is appropriate for mass cytometry (CyTOF) data. For fluorescence flow cytometry, we recommend cofactor = 150 instead.

Details

Flow and mass cytometry data should be transformed prior to clustering. The raw data follows an approximately log-normal distribution. Transforming with a log (or similar) function brings the data closer to a normal distribution, which improves clustering performance and allows positive and negative populations to be distinguished more clearly.

This function implements an inverse hyperbolic sine ('arcsinh') transform with adjustable 'cofactor' parameter. The arcsinh transform is widely used for CyTOF data. It behaves similarly to a log transform at high values, but is approximately linear near zero; so unlike the log, it can handle zeros or small negative values. The cofactor parameter controls the width of the linear region. Zero values and small negatives occur in CyTOF data when no ions are detected in a given channel (negatives are due to background subtraction and randomization of integer count values, which are performed by default by the CyTOF instrument software).

Recommended values for the cofactor parameter are 5 (mass cytometry, CyTOF) or 150 (fluorescence flow cytometry); see Bendall et al. (2011), *Science*, Supplementary Figure S2.

The transform should be applied to protein marker columns only. The `SummarizedExperiment` object created in the previous step (`prepareData`) is assumed to contain a factor `marker_class` in the column meta-data, where entries `"none"` indicate non-marker columns. (If this is not available, all columns will be transformed instead.)

Value

`d_se`: Data with transform applied to protein marker columns.

Examples

```
# For a complete workflow example demonstrating each step in the 'diffcyt' pipeline,
# see the package vignette.

# Function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
  d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
  colnames(d) <- paste0("marker", sprintf("%02d", 1:ncol))
  d
}

# Create random data (without differential signal)
set.seed(123)
d_input <- list(
  sample1 = d_random(),
  sample2 = d_random(),
  sample3 = d_random(),
  sample4 = d_random()
)

experiment_info <- data.frame(
  sample_id = factor(paste0("sample", 1:4)),
  group_id = factor(c("group1", "group1", "group2", "group2")),
  stringsAsFactors = FALSE
)
```

```
marker_info <- data.frame(
  channel_name = paste0("channel", sprintf("%03d", 1:20)),
  marker_name = paste0("marker", sprintf("%02d", 1:20)),
  marker_class = factor(c(rep("type", 10), rep("state", 10)),
                        levels = c("type", "state", "none")),
  stringsAsFactors = FALSE
)

# Prepare data
d_se <- prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- transformData(d_se)
```

Index

BuildMST, *21*
BuildSOM, *20*

calcCounts, 2, *28*, *31*, *34*, *37*, *41*, *45*
calcMedians, 4, *37*, *41*, *46*
calcMediansByClusterMarker, 5
calcMediansBySampleMarker, 7
createContrast, 9, *15*, *28, 29*, *31, 32*, *34, 35*, *38, 39*, *41, 42*
createDesignMatrix, 10, *12, 13*, *15*, *17*, *28, 29*, *34, 35*, *38, 39*
createFormula, *11*, 12, *15*, *31, 32*, *41, 42*

DataFrame, *15*, *17*, *26*, *47*
diffcyt, 14, *23*, *45*
diffcyt-package *(*diffcyt*)*, 14
duplicateCorrelation, *17*, *34, 35*, *38, 39*

eBayes, *17*, *38*
edgeR, *29*
estimateDisp, *17*, *28*

flowFrame, *15*, *17*, *26*
flowSet, *15*, *17*, *26*
FlowSOM, *21*

generateClusters, *2*, *4*, *6, 7*, *16*, 20

limma, *35*, *38*

plotHeatmap, 22
prepareData, *4*, *6*, *8*, *10–13*, *15, 16*, *20, 21*, 25, *48, 49*

rowData, *21*, *29*, *32*, *36*, *39*, *43*

SummarizedExperiment, *2–4*, *6–8*, *21*, *26–29*, *31, 32*, *34*, *36, 37*, *39*, *41*, *43*, *45, 46*, *48, 49*

tbl_df, *15*, *26*
testDA_edgeR, *15–18*, 28, *45*
testDA_GLMM, *15, 16*, *18*, 31, *45*
testDA_voom, *11*, *13*, *15–18*, 34, *45*
testDS_limma, *11*, *13*, *15–18*, 37, *45*
testDS_LMM, *15–18*, 41, *45*
topClusters, 44
topTable, *44, 45*, 45
transformData, *16*, *20, 21*, 48

voom, *35*