Package 'scater'

April 15, 2020

Type Package

Version 1.14.6

Date 2019-12-13

License GPL-3

Title Single-Cell Analysis Toolkit for Gene Expression Data in R

Description A collection of tools for doing various analyses of single-cell RNA-seq gene expression data, with a focus on quality control and visualization.

Depends SingleCellExperiment, ggplot2

Imports BiocGenerics, SummarizedExperiment, Matrix, ggbeeswarm, grid, DelayedArray, DelayedMatrixStats, methods, S4Vectors, stats, utils, viridis, Rcpp, BiocNeighbors, BiocSingular, BiocParallel

Suggests BiocStyle, BiocFileCache, biomaRt, beachmat, cowplot, destiny, knitr, scRNAseq, robustbase, rmarkdown, Rtsne, uwot, testthat, pheatmap, Biobase, limma, DropletUtils

VignetteBuilder knitr

biocViews ImmunoOncology, SingleCell, RNASeq, QualityControl, Preprocessing, Normalization, Visualization, DimensionReduction, Transcriptomics, GeneExpression, Sequencing, Software, DataImport, DataRepresentation, Infrastructure, Coverage

LinkingTo Rcpp, beachmat

SystemRequirements C++11

BuildResaveData no

RoxygenNote 7.0.2

NeedsCompilation yes

URL http://bioconductor.org/packages/scater/

BugReports https://support.bioconductor.org/

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

git_branch RELEASE_3_10

git_last_commit ba37f4d

git_last_commit_date 2019-12-13

Date/Publication 2020-04-14

Author Davis McCarthy [aut, cre],
Kieran Campbell [aut],
Aaron Lun [aut, ctb],
Quin Wills [aut],
Vladimir Kiselev [ctb]

Maintainer Davis McCarthy <davis@ebi.ac.uk>

R topics documented:

`addPerCellQC` *Add QC to an SE*

Description

Convenient utilities to compute QC metrics and add them to a SummarizedExperiment's metadata.

Usage

```
addPerCellQC(x, ...)

addPerFeatureQC(x, ...)
```

Arguments

`x`	A SummarizedExperiment object or one of its subclasses.
`...`	For `addPerCellQC`, further arguments to pass to `perCellQCMetrics`. For `addPerFeatureQC`, further arguments to pass to `perFeatureQCMetrics`.

Details

These functions are simply wrappers around `perCellQCMetrics` and `perFeatureQCMetrics`, respectively. The computed QC metrics are automatically appended onto the existing `colData` or `rowData`. No protection is provided to avoid duplicated column names.

Value

An object like `x` but with the QC metrics added to the row or column metadata.

Author(s)

Aaron Lun

See Also

perCellQCMetrics and perFeatureQCMetrics, which do the actual work.

Examples

```
example_sce <- mockSCE()
example_sce <- addPerCellQC(example_sce)
colData(example_sce)

example_sce <- addPerFeatureQC(example_sce)
rowData(example_sce)
```

---

annotateBMFeatures *Get feature annotation information from Biomart*

Description

Use the **biomaRt** package to add feature annotation information to an SingleCellExperiment.

Usage

```
annotateBMFeatures(
  ids,
  biomart = "ENSEMBL_MART_ENSEMBL",
  dataset = "mmusculus_gene_ensembl",
  id.type = "ensembl_gene_id",
  symbol.type,
  attributes = c(id.type, symbol.type, "chromosome_name", "gene_biotype",
    "start_position", "end_position"),
  filters = id.type,
  ...
)

getBMFeatureAnnos(x, ids = rownames(x), ...)
```

Arguments

ids	A character vector containing feature identifiers.
biomart	String defining the biomaRt to be used, to be passed to useMart.
dataset	String defining the dataset to use, to be passed to useMart.
id.type	String specifying the type of identifier in ids.
symbol.type	String specifying the type of symbol to retrieve. If missing, this is set to "mgi_symbol" if dataset="mmusculus_gene_ensembl", or to "hgnc_symbol" if dataset="hsapiens_gene_ense
attributes	Character vector defining the attributes to pass to getBM.
filters	String defining the type of identifier in ids, to be used as a filter in getBM.
...	For annotateBMFeatures, further named arguments to pass to biomaRt::useMart. For getBMFeatureAnnos, further arguments to pass to annotateBMFeatures.
x	A SingleCellExperiment object.

Details

These functions provide convenient wrappers around **biomaRt** to quickly obtain annotation in the required format.

Value

For `annotateBMFeatures`, a DataFrame containing feature annotation, with one row per value in `ids`.

For `getBMFeatureAnnos`, `x` is returned containing the output of `annotateBMFeatures` appended to its `rowData`.

Author(s)

Aaron Lun, based on code by Davis McCarthy

Examples

```
## Not run:
# Making up Ensembl IDs for demonstration purposes.
mock_id <- paste0("ENSMUSG", sprintf("%011d", seq_len(1000)))
anno <- annotateBMFeatures(ids=mock_id)

## End(Not run)
```

---

bootstraps *Accessor and replacement for bootstrap results in a SingleCellExperiment object*

Description

`SingleCellExperiment` objects can contain bootstrap expression values (for example, as generated by the kallisto software for quantifying feature abundance). These functions conveniently access and replace the 'bootstrap' elements in the `assays` slot with the value supplied, which must be an matrix of the correct size, namely the same number of rows and columns as the `SingleCellExperiment` object as a whole.

Usage

```
bootstraps(object)

bootstraps(object) <- value

## S4 method for signature 'SingleCellExperiment'
bootstraps(object)

## S4 replacement method for signature 'SingleCellExperiment,array'
bootstraps(object) <- value
```

Arguments

`object`	a `SingleCellExperiment` object.
`value`	an array of class `"numeric"` containing bootstrap expression values

Value

If accessing bootstraps slot of an SingleCellExperiment, then an array with the bootstrap values, otherwise an SingleCellExperiment object containing new bootstrap values.

Author(s)

Davis McCarthy

Examples

```
example_sce <- mockSCE()
bootstraps(example_sce)
```

`calculateAverage` *Calculate per-feature average counts*

Description

Calculate average counts per feature after normalizing observations using size factors.

Usage

```
calculateAverage(x, ...)

## S4 method for signature 'ANY'
calculateAverage(
  x,
  size_factors = NULL,
  use_size_factors = NULL,
  subset_row = NULL,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
calculateAverage(x, ..., exprs_values = "counts")

## S4 method for signature 'SingleCellExperiment'
calculateAverage(x, size_factors = NULL, ...)

calcAverage(x, ...)
```

Arguments

x	A numeric matrix of counts where features are rows and Alternatively, a SummarizedExperiment or a SingleCellExperiment containing such counts.
...	For the generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.

`size_factors` A numeric vector containing size factors. If `NULL`, these are calculated or extracted from `x`.

`use_size_factors` Deprecated, same as `size_factors`.

`subset_row` A vector specifying the subset of rows of `object` for which to return a result.

`BPPARAM` A BiocParallelParam object specifying whether the calculations should be parallelized.

`exprs_values` A string specifying the assay of `x` containing the count matrix.

Details

The size-adjusted average count is defined by dividing each count by the size factor and taking the average across cells. All sizes factors are scaled so that the mean is 1 across all cells, to ensure that the averages are interpretable on the same scale of the raw counts.

If no size factors are supplied, they are determined automatically:

- For count matrices and SummarizedExperiment inputs, the sum of counts for each cell is used to compute a size factor via the `librarySizeFactors` function.
- For SingleCellExperiment instances, the function searches for `sizeFactors` from `x`. If none are available, it defaults to library size-derived size factors.

If `size_factors` are supplied, they will override any size factors present in `x`.

Value

A numeric vector of average count values with same length as number of features (or the number of features in `subset_row` if supplied).

Author(s)

Aaron Lun

See Also

`librarySizeFactors`, for the default calculation of size factors.

`logNormCounts`, for the calculation of normalized expression values.

Examples

```
example_sce <- mockSCE()
ave_counts <- calculateAverage(example_sce)
summary(ave_counts)
```

`calculateCPM` *Calculate counts per million (CPM)*

Description

Calculate count-per-million (CPM) values from the count data.

Usage

```
calculateCPM(x, ...)

## S4 method for signature 'ANY'
calculateCPM(
  x,
  size_factors = NULL,
  subset_row = NULL,
  use_size_factors = NULL
)

## S4 method for signature 'SummarizedExperiment'
calculateCPM(x, ..., exprs_values = "counts")

## S4 method for signature 'SingleCellExperiment'
calculateCPM(x, size_factors = NULL, ...)
```

Arguments

`x`	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment or a SingleCellExperiment containing such counts.
`...`	For the generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
`size_factors`	A numeric vector containing size factors to adjust the library sizes. If `NULL`, the library sizes are used directly.
`subset_row`	A vector specifying the subset of rows of `x` for which to return a result.
`use_size_factors`	Deprecated, same as `size_factors`.
`exprs_values`	A string or integer scalar specifying the assay of `x` containing the count matrix.

Details

If `size_factors` are provided or available in `x`, they are used to define the effective library sizes. This is done by scaling all size factors such that the mean factor is equal to the mean sum of counts across all features. The effective library sizes are then used as the denominator of the CPM calculation.

Value

A numeric matrix of CPM values.

Author(s)

Aaron Lun

See Also

normalizeCounts, on which this function is based.

Examples

```
example_sce <- mockSCE()
cpm(example_sce) <- calculateCPM(example_sce)
str(cpm(example_sce))
```

---

calculateDiffusionMap *Create a diffusion map from cell-level data*

Description

Produce a diffusion map for the cells, based on the data in a SingleCellExperiment object.

Usage

```
calculateDiffusionMap(x, ...)

## S4 method for signature 'ANY'
calculateDiffusionMap(
  x,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  feature_set = NULL,
  scale = FALSE,
  scale_features = NULL,
  transposed = FALSE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
calculateDiffusionMap(x, ..., exprs_values = "logcounts")

## S4 method for signature 'SingleCellExperiment'
calculateDiffusionMap(
  x,
  ...,
  exprs_values = "logcounts",
  dimred = NULL,
  use_dimred = NULL,
```

```
  n_dimred = NULL
)

runDiffusionMap(x, ..., altexp = NULL, name = "DiffusionMap")
```

Arguments

x	For calculateDiffusionMap, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For runDiffusionMap, a SingleCellExperiment object.
...	For the calculateDiffusionMap generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to DiffusionMap. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runDiffusionMap, additional arguments to pass to calculateDiffusionMap.
ncomponents	Numeric scalar indicating the number of UMAP dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for PCA, see ?"scater-red-dim-args".
subset_row	Vector specifying the subset of features to use for PCA, see ?"scater-red-dim-args".
feature_set	Deprecated, same as subset_row.
scale	Logical scalar, should the expression values be standardised? See ?"scater-red-dim-args" for details.
scale_features	Deprecated, same as scale but with a different default.
transposed	Logical scalar, is x transposed with cells in rows? See ?"scater-red-dim-args" for details.
exprs_values	Integer scalar or string indicating which assay of x contains the expression values, see ?"scater-red-dim-args".
dimred	String or integer scalar specifying the existing dimensionality reduction results to use, see ?"scater-red-dim-args".
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified, see ?"scater-red-dim-args".
altexp	String or integer scalar specifying an alternative experiment to use to compute the PCA, see ?"scater-red-dim-args".
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function DiffusionMap is used internally to compute the diffusion map. The behaviour of DiffusionMap seems to be non-deterministic, in a manner that is not responsive to any set.seed call. The reason for this is unknown.

Value

For calculateDiffusionMap, a matrix is returned containing the diffusion map coordinates for each cell (row) and dimension (column).

For runDiffusionMap, a modified x is returned that contains the diffusion map coordinates in reducedDim(x,name).

Author(s)

Aaron Lun, based on code by Davis McCarthy

References

Haghverdi L, Buettner F, Theis FJ (2015). Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics* 31(18), 2989-2998.

See Also

DiffusionMap, to perform the underlying calculations.

plotDiffusionMap, to quickly visualize the results.

?"scater-red-dim-args", for a full description of various options.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runDiffusionMap(example_sce, scale_features=NULL)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

calculateFPKM *Calculate FPKMs*

Description

Calculate fragments per kilobase of exon per million reads mapped (FPKM) values from the feature-level counts.

Usage

```
calculateFPKM(x, lengths, effective_length = NULL, ..., subset_row = NULL)
```

Arguments

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment or a SingleCellExperiment containing such counts.
lengths	Numeric vector providing the effective length for each feature in x.
effective_length	Deprecated, same as lengths.
...	Further arguments to pass to calculateCPM.
subset_row	A vector specifying the subset of rows of x for which to return a result.

Value

A numeric matrix of FPKM values.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

calculateCPM, for the initial calculation of CPM values.

Examples

```
example_sce <- mockSCE()
eff_len <- runif(nrow(example_sce), 500, 2000)
fout <- calculateFPKM(example_sce, eff_len)
str(fout)
```

---

calculateMDS *Perform MDS on cell-level data*

Description

Perform multi-dimensional scaling (MDS) on cells, based on the data in a SingleCellExperiment object.

Usage

```
calculateMDS(x, ...)

## S4 method for signature 'ANY'
calculateMDS(
  x,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  feature_set = NULL,
  scale = FALSE,
  scale_features = NULL,
  transposed = FALSE,
  method = "euclidean"
)

## S4 method for signature 'SummarizedExperiment'
calculateMDS(x, ..., exprs_values = "logcounts")

## S4 method for signature 'SingleCellExperiment'
calculateMDS(
  x,
  ...,
  exprs_values = "logcounts",
  dimred = NULL,
  use_dimred = NULL,
  n_dimred = NULL
)

runMDS(x, ..., altexp = NULL, name = "MDS")
```

Arguments

x	For calculateMDS, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For runMDS, a SingleCellExperiment object.
...	For the calculateMDS generic, additional arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runMDS, additional arguments to pass to calculateMDS.
ncomponents	Numeric scalar indicating the number of MDS?g dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for PCA, see ?"scater-red-dim-args".
subset_row	Vector specifying the subset of features to use for PCA, see ?"scater-red-dim-args".
feature_set	Deprecated, same as subset_row.
scale	Logical scalar, should the expression values be standardised? See ?"scater-red-dim-args" for details.
scale_features	Deprecated, same as scale but with a different default.
transposed	Logical scalar, is x transposed with cells in rows? See ?"scater-red-dim-args" for details.
method	String specifying the type of distance to be computed between cells.
exprs_values	Integer scalar or string indicating which assay of x contains the expression values, see ?"scater-red-dim-args".
dimred	String or integer scalar specifying the existing dimensionality reduction results to use, see ?"scater-red-dim-args".
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified, see ?"scater-red-dim-args".
altexp	String or integer scalar specifying an alternative experiment to use to compute the PCA, see ?"scater-red-dim-args".
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function cmdscale is used internally to compute the MDS components.

Value

For calculateMDS, a matrix is returned containing the MDS coordinates for each cell (row) and dimension (column).

For runMDS, a modified x is returned that contains the MDS coordinates in reducedDim(x,name).

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

cmdscale, to perform the underlying calculations.

plotMDS, to quickly visualize the results.

?"scater-red-dim-args", for a full description of various options.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runMDS(example_sce, scale_features=NULL)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

calculatePCA *Perform PCA on expression data*

Description

Perform a principal components analysis (PCA) on cells, based on the expression data in a SingleCellExperiment object.

Usage

```
calculatePCA(x, ...)

## S4 method for signature 'ANY'
calculatePCA(
  x,
  ncomponents = 50,
  ntop = 500,
  subset_row = NULL,
  feature_set = NULL,
  scale = FALSE,
  scale_features = NULL,
  transposed = FALSE,
  BSPARAM = bsparam(),
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
calculatePCA(x, ..., exprs_values = "logcounts")

## S4 method for signature 'SingleCellExperiment'
calculatePCA(
  x,
  ...,
  exprs_values = "logcounts",
  dimred = NULL,
  use_dimred = NULL,
```

```
  n_dimred = NULL
)

## S4 method for signature 'SingleCellExperiment'
runPCA(x, ..., use_coldata = FALSE, altexp = NULL, name = "PCA")
```

Arguments

x	For calculatePCA, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For runPCA, a SingleCellExperiment object containing such a matrix.
...	For the calculatePCA generic, additional arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runPCA, additional arguments to pass to calculatePCA.
ncomponents	Numeric scalar indicating the number of principal components to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for PCA, see ?"scater-red-dim-args".
subset_row	Vector specifying the subset of features to use for PCA, see ?"scater-red-dim-args".
feature_set	Deprecated, same as subset_row.
scale	Logical scalar, should the expression values be standardised? See ?"scater-red-dim-args" for details.
scale_features	Deprecated, same as scale but with a different default.
transposed	Logical scalar, is x transposed with cells in rows? See ?"scater-red-dim-args" for details.
BSPARAM	A BiocSingularParam object specifying which algorithm should be used to perform the PCA.
BPPARAM	A BiocParallelParam object specifying whether the PCA should be parallelized.
exprs_values	Integer scalar or string indicating which assay of x contains the expression values, see ?"scater-red-dim-args".
dimred	String or integer scalar specifying the existing dimensionality reduction results to use, see ?"scater-red-dim-args".
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified, see ?"scater-red-dim-args".
use_coldata	Deprecated, use runColDataPCA instead.
altexp	String or integer scalar specifying an alternative experiment to use to compute the PCA, see ?"scater-red-dim-args".
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

Fast approximate SVD algorithms like BSPARAM=IrlbaParam() or RandomParam() use a random initialization, after which they converge towards the exact PCs. This means that the result will change slightly across different runs. For full reproducibility, users should call set.seed prior to running runPCA with such algorithms. (Note that this includes BSPARAM=bsparam(), which uses approximate algorithms by default.)

Value

A SingleCellExperiment object containing the first `ncomponents` principal coordinates for each cell. By default, this is stored in the `"PCA"` entry of the `reducedDims`. The proportion of variance explained by each PC is stored as a numeric vector in the `"percentVar"` attribute of the reduced dimension matrix.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

`runPCA`, for the underlying calculations.

`plotPCA`, to conveniently visualize the results.

?`"scater-red-dim-args"`, for a full description of various options.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runPCA(example_sce, scale_features=NULL)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

calculateQCMetrics *Calculate QC metrics*

Description

Compute quality control (QC) metrics for each feature and cell in a SingleCellExperiment object, accounting for specified control sets.

Usage

```
calculateQCMetrics(
  object,
  exprs_values = "counts",
  feature_controls = NULL,
  cell_controls = NULL,
  percent_top = c(50, 100, 200, 500),
  detection_limit = 0,
  use_spikes = TRUE,
  compact = FALSE,
  BPPARAM = SerialParam()
)
```

Arguments

object	A SingleCellExperiment object containing expression values, usually counts.
exprs_values	A string indicating which assays in the object should be used to define expression.
feature_controls	A named list containing one or more vectors (a character vector of feature names, a logical vector, or a numeric vector of indices), used to identify feature controls such as ERCC spike-in sets or mitochondrial genes.
cell_controls	A named list containing one or more vectors (a character vector of cell (sample) names, a logical vector, or a numeric vector of indices), used to identify cell controls, e.g., blank wells or bulk controls.
percent_top	An integer vector. Each element is treated as a number of top genes to compute the percentage of library size occupied by the most highly expressed genes in each cell. See pct_X_top_Y_features below for more details.
detection_limit	A numeric scalar to be passed to nexprs, specifying the lower detection limit for expression.
use_spikes	A logical scalar indicating whether existing spike-in sets in object should be automatically added to feature_controls, see ?isSpike.
compact	A logical scalar indicating whether the metrics should be returned in a compact format as a nested DataFrame.
BPPARAM	A BiocParallelParam object specifying whether the QC calculations should be parallelized.

Details

This function calculates useful quality control metrics to help with pre-processing of data and identification of potentially problematic features and cells.

Underscores in assayNames(object) and in feature_controls or cell_controls can cause theoretically cause ambiguities in the names of the output metrics. While problems are highly unlikely, users are advised to avoid underscores when naming their controls/assays.

If the expression values are double-precision, the per-row means may not be *exactly* identity for different choices of BPPARAM. This is due to differences in rounding error when summation is performed across different numbers of cores. If it is important to obtain numerically identical results (e.g., when using the per-row means for sensitive procedures like t-SNE) across various parallelization schemes, we suggest manually calculating those statistics using rowMeans.

Value

A SingleCellExperiment object containing QC metrics in the row and column metadata.

Cell-level QC metrics

Denote the value of exprs_values as X. Cell-level metrics are:

total_X: Sum of expression values for each cell (i.e., the library size, when counts are the expression values).

log10_total_X: Log10-transformed total_X after adding a pseudo-count of 1.

total_features_by_X: The number of features that have expression values above the detection limit.

log10_total_features_by_X: Log10-transformed total_features_by_X after adding a pseudo-count of 1.

pct_X_in_top_Y_features: The percentage of the total that is contained within the top Y most highly expressed features in each cell. This is only reported when there are more than Y features. The top numbers are specified via percent_top.

If any controls are specified in feature_controls, the above metrics will be recomputed using only the features in each control set. The name of the set is appended to the name of the recomputed metric, e.g., total_X_F. A pct_X_F metric is also calculated for each set, representing the percentage of expression values assigned to features in F.

In addition to the user-specified control sets, two other sets are automatically generated when feature_controls is non-empty. The first is the "feature_control" set, containing a union of all feature control sets; and the second is an "endogenous" set, containing all genes not in any control set. Metrics are also computed for these sets in the same manner described above, suffixed with _feature_control and _endogenous instead of _F.

Finally, there is the is_cell_control field, which indicates whether each cell has been defined as a cell control by cell_controls. If multiple sets of cell controls are defined (e.g., blanks or bulk libraries), a metric is_cell_control_C is produced for each cell control set C. The union of all sets is stored in is_cell_control.

All of these cell-level QC metrics are added as columns to the colData slot of the SingleCellExperiment object. This allows them to be inspected by the user and makes them readily available for other functions to use.

Feature-level QC metrics

Denote the value of exprs_values as X. Feature-level metrics are:

mean_X: Mean expression value for each gene across all cells.

log10_mean_X: Log10-mean expression value for each gene across all cells.

n_cells_by_X: Number of cells with expression values above the detection limit for each gene.

pct_dropout_by_X: Percentage of cells with expression values below the detection limit for each gene.

total_X: Sum of expression values for each gene across all cells.

log10_total_X: Log10-sum of expression values for each gene across all cells.

If any controls are specified in cell_controls, the above metrics will be recomputed using only the cells in each control set. The name of the set is appended to the name of the recomputed metric, e.g., total_X_C. A pct_X_C metric is also calculated for each set, representing the percentage of expression values assigned to cells in C.

In addition to the user-specified control sets, two other sets are automatically generated when cell_controls is non-empty. The first is the "cell_control" set, containing a union of all cell control sets; and the second is an "non_control" set, containing all genes not in any control set. Metrics are computed for these sets in the same manner described above, suffixed with _cell_control and _non_control instead of_C.

Finally, there is the is_feature_control field, which indicates whether each feature has been defined as a control by feature_controls. If multiple sets of feature controls are defined (e.g., ERCCs, mitochondrial genes), a metric is_feature_control_F is produced for each feature control set F. The union of all sets is stored in is_feature_control.

These feature-level QC metrics are added as columns to the rowData slot of the SingleCellExperiment object. They can be inspected by the user and are readily available for other functions to use.

Compacted output

If `compact=TRUE`, the QC metrics are stored in the `"scater_qc"` field of the `colData` and `rowData` as a nested DataFrame. This avoids cluttering the metadata with QC metrics, especially if many results are to be stored in a single SingleCellExperiment object.

Assume we have a feature control set `F` and a cell control set `C`. The nesting structure in `scater_qc` in the `colData` is:

```
scater_qc
|-- is_cell_control
|-- is_cell_control_C
|-- all
|   |-- total_counts
|   |-- total_features_by_counts
|   \-- ...
+-- endogenous
|   |-- total_counts
|   |-- total_features_by_counts
    |-- pct_counts
|   \-- ...
+-- feature_control
|   |-- total_counts
|   |-- total_features_by_counts
    |-- pct_counts
|   \-- ...
\-- feature_control_F
    |-- total_counts
    |-- total_features_by_counts
    |-- pct_counts
    \-- ...
```

The nesting in `scater_qc` in the `rowData` is:

```
scater_qc
|-- is_feature_control
|-- is_feature_control_F
|-- all
|   |-- total_counts
|   |-- total_features_by_counts
|   \-- ...
+-- non_control
|   |-- total_counts
|   |-- total_features_by_counts
    |-- pct_counts
|   \-- ...
+-- cell_control
|   |-- total_counts
|   |-- total_features_by_counts
    |-- pct_counts
|   \-- ...
\-- cell_control_C
    |-- total_counts
    |-- total_features_by_counts
```

```
|-- pct_counts
\-- ...
```

No suffixing of the metric names by the control names is performed here. This is not necessary when each control set has its own nested DataFrame.

Renamed metrics

Several metric names have been changed in **scater** 1.7.5:

- `total_features` was changed to `total_features_by_X` where `X` is the `exprs_values`. This avoids ambiguities if `calculateQCMetrics` is called multiple times with different `exprs_values`.
- `n_cells_X` was changed to `n_cells_by_X`, to provide a more sensible name for the metric.
- `pct_dropout_X` was changed to `pct_dropout_by_X`.
- `pct_X_top_Y_features` was changed to `pct_X_in_top_Y_features`.

The old metric names have been removed in version 1.9.10.

Author(s)

Davis McCarthy, with (many!) modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- calculateQCMetrics(example_sce)

## with a set of feature controls defined
example_sce <- calculateQCMetrics(example_sce,
feature_controls = list(set1 = 1:40))

## with a named set of feature controls defined
example_sce <- calculateQCMetrics(example_sce,
     feature_controls = list(ERCC = 1:40))
```

`calculateTPM` *Calculate TPMs*

Description

Calculate transcripts-per-million (TPM) values for expression from feature-level counts.

Usage

```
calculateTPM(x, ...)

## S4 method for signature 'ANY'
calculateTPM(x, lengths = NULL, effective_length = NULL, ...)

## S4 method for signature 'SummarizedExperiment'
calculateTPM(x, ..., exprs_values = "counts")

## S4 method for signature 'SingleCellExperiment'
calculateTPM(x, lengths = NULL, size_factors = NULL, ...)
```

Arguments

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment or a SingleCellExperiment containing such counts.
...	For the generic, arguments to pass to specific methods. For the ANY method, further arguments to pass to calculateCPM. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
lengths	Numeric vector providing the effective length for each feature in x. Alternatively NULL, see Details.
effective_length	Deprecated, same as length.
exprs_values	A string specifying the assay of x containing the count matrix.
size_factors	A numeric vector containing size factors to adjust the library sizes. If NULL, the library sizes are used directly.

Details

For read count data, this function assumes uniform coverage along the (effective) length of the transcript. Thus, the number of transcripts for a gene is proportional to the read count divided by the transcript length. Here, the division is done before calculation of the library size to compute per-million values, where calculateFPKM will only divide by the length after library size normalization.

For UMI count data, this function should be run with effective_length=NULL, i.e., no division by the effective length. This is because the number of UMIs is a direct (albeit biased) estimate of the number of transcripts.

Value

A numeric matrix of TPM values.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

calculateCPM, on which this function is based.

Examples

```
example_sce <- mockSCE()
eff_len <- runif(nrow(example_sce), 500, 2000)
tout <- calculateTPM(example_sce, lengths = eff_len)
str(tout)
```

`calculateTSNE` *Perform t-SNE on cell-level data*

Description

Perform t-stochastic neighbour embedding (t-SNE) for the cells, based on the data in a SingleCellExperiment object.

Usage

```
calculateTSNE(x, ...)

## S4 method for signature 'ANY'
calculateTSNE(
  x,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  feature_set = NULL,
  scale = FALSE,
  scale_features = NULL,
  transposed = FALSE,
  perplexity = NULL,
  normalize = TRUE,
  theta = 0.5,
  ...,
  external_neighbors = FALSE,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
calculateTSNE(x, ..., exprs_values = "logcounts")

## S4 method for signature 'SingleCellExperiment'
calculateTSNE(
  x,
  ...,
  pca = is.null(dimred),
  exprs_values = "logcounts",
  dimred = NULL,
  use_dimred = NULL,
  n_dimred = NULL
)

runTSNE(x, ..., altexp = NULL, name = "TSNE")
```

Arguments

`x`	For `calculateTSNE`, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix.

	For runTSNE, a SingleCellExperiment object.
...	For the calculateTSNE generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to Rtsne. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runTSNE, additional arguments to pass to calculateTSNE.
ncomponents	Numeric scalar indicating the number of t-SNE dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for PCA, see ?"scater-red-dim-args".
subset_row	Vector specifying the subset of features to use for PCA, see ?"scater-red-dim-args".
feature_set	Deprecated, same as subset_row.
scale	Logical scalar, should the expression values be standardised? See ?"scater-red-dim-args" for details.
scale_features	Deprecated, same as scale but with a different default.
transposed	Logical scalar, is x transposed with cells in rows? See ?"scater-red-dim-args" for details.
perplexity	Numeric scalar defining the perplexity parameter, see ?Rtsne for more details.
normalize	Logical scalar indicating if input values should be scaled for numerical precision, see normalize_input.
theta	Numeric scalar specifying the approximation accuracy of the Barnes-Hut algorithm, see Rtsne for details.
external_neighbors	Logical scalar indicating whether a nearest neighbors search should be computed externally with findKNN.
BNPARAM	A BiocNeighborParam object specifying the neighbor search algorithm to use when external_neighbors=TRUE.
BPPARAM	A BiocParallelParam object specifying how the neighbor search should be parallelized when external_neighbors=TRUE.
exprs_values	Integer scalar or string indicating which assay of x contains the expression values, see ?"scater-red-dim-args".
pca	Logical scalar indicating whether a PCA step should be performed inside Rtsne.
dimred	String or integer scalar specifying the existing dimensionality reduction results to use, see ?"scater-red-dim-args".
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified, see ?"scater-red-dim-args".
altexp	String or integer scalar specifying an alternative experiment to use to compute the PCA, see ?"scater-red-dim-args".
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function Rtsne is used internally to compute the t-SNE. Note that the algorithm is not deterministic, so different runs of the function will produce differing results. Users are advised to test multiple random seeds, and then use set.seed to set a random seed for replicable results.

The value of the perplexity parameter can have a large effect on the results. By default, the function will set a "reasonable" perplexity that scales with the number of cells in x. (Specifically, it is the number of cells divided by 5, capped at a maximum of 50.) However, it is often worthwhile to manually try multiple values to ensure that the conclusions are robust.

If external_neighbors=TRUE, the nearest neighbor search step will use a different algorithm to that in the Rtsne function. This can be parallelized or approximate to achieve greater speed for large data sets. The neighbor search results are then used for t-SNE via the Rtsne_neighbors function.

If dimred is specified, the PCA step of the Rtsne function is automatically turned off by default. This presumes that the existing dimensionality reduction is sufficient such that an additional PCA is not required.

Value

For calculateTSNE, a numeric matrix is returned containing the t-SNE coordinates for each cell (row) and dimension (column).

For runTSNE, a modified x is returned that contains the t-SNE coordinates in reducedDim(x,name).

Author(s)

Aaron Lun, based on code by Davis McCarthy

References

van der Maaten LJP, Hinton GE (2008). Visualizing High-Dimensional Data Using t-SNE. *J. Mach. Learn. Res.* 9, 2579-2605.

See Also

Rtsne, for the underlying calculations.

plotTSNE, to quickly visualize the results.

?"scater-red-dim-args", for a full description of various options.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runTSNE(example_sce, scale_features=NULL)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

calculateUMAP *Perform UMAP on cell-level data*

Description

Perform uniform manifold approximation and projection (UMAP) for the cells, based on the data in a SingleCellExperiment object.

Usage

```
calculateUMAP(x, ...)

## S4 method for signature 'ANY'
calculateUMAP(
  x,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  feature_set = NULL,
  scale = FALSE,
  scale_features = NULL,
  transposed = FALSE,
  pca = if (transposed) NULL else 50,
  n_neighbors = 15,
  ...,
  external_neighbors = FALSE,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
calculateUMAP(x, ..., exprs_values = "logcounts")

## S4 method for signature 'SingleCellExperiment'
calculateUMAP(
  x,
  ...,
  pca = if (!is.null(dimred)) NULL else 50,
  exprs_values = "logcounts",
  dimred = NULL,
  use_dimred = NULL,
  n_dimred = NULL
)

runUMAP(x, ..., altexp = NULL, name = "UMAP")
```

Arguments

x	For calculateUMAP, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For runTSNE, a SingleCellExperiment object containing such a matrix.
...	For the calculateUMAP generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to umap. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runUMAP, additional arguments to pass to calculateUMAP.
ncomponents	Numeric scalar indicating the number of UMAP dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for PCA, see ?"scater-red-dim-args".

subset_row	Vector specifying the subset of features to use for PCA, see ?"scater-red-dim-args".
feature_set	Deprecated, same as subset_row.
scale	Logical scalar, should the expression values be standardised? See ?"scater-red-dim-args" for details.
scale_features	Deprecated, same as scale but with a different default.
transposed	Logical scalar, is x transposed with cells in rows? See ?"scater-red-dim-args" for details.
pca	Integer scalar specifying how many PCs should be used as input into the UMAP algorithm. By default, no PCA is performed if the input is a dimensionality reduction result.
n_neighbors	Integer scalar, number of nearest neighbors to identify when constructing the initial graph.
external_neighbors	Logical scalar indicating whether a nearest neighbors search should be computed externally with findKNN.
BNPARAM	A BiocNeighborParam object specifying the neighbor search algorithm to use when external_neighbors=TRUE.
BPPARAM	A BiocParallelParam object specifying how the neighbor search should be parallelized when external_neighbors=TRUE.
exprs_values	Integer scalar or string indicating which assay of x contains the expression values, see ?"scater-red-dim-args".
dimred	String or integer scalar specifying the existing dimensionality reduction results to use, see ?"scater-red-dim-args".
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified, see ?"scater-red-dim-args".
altexp	String or integer scalar specifying an alternative experiment to use to compute the PCA, see ?"scater-red-dim-args".
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function umap is used internally to compute the UMAP. Note that the algorithm is not deterministic, so different runs of the function will produce differing results. Users are advised to test multiple random seeds, and then use set.seed to set a random seed for replicable results.

If external_neighbors=TRUE, the nearest neighbor search is conducted using a different algorithm to that in the umap function. This can be parallelized or approximate to achieve greater speed for large data sets. The neighbor search results are then used directly to create the UMAP embedding.

Value

For calculateUMAP, a matrix is returned containing the UMAP coordinates for each cell (row) and dimension (column).

For runUMAP, a modified x is returned that contains the UMAP coordinates in reducedDim(x,name).

Author(s)

Aaron Lun

References

McInnes L, Healy J, Melville J (2018). UMAP: uniform manifold approximation and projection for dimension reduction. arXiv.

See Also

umap, for the underlying calculations.

plotUMAP, to quickly visualize the results.

?"scater-red-dim-args", for a full description of various options.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runUMAP(example_sce, scale_features=NULL)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

centreSizeFactors *Centre size factors at unity*

Description

Scales all size factors so that the average size factor across cells is equal to 1.

Usage

```
centreSizeFactors(object, centre = 1)
```

Arguments

object	A SingleCellExperiment object containing any number (or zero) sets of size factors.
centre	A numeric scalar, the value around which all sets of size factors should be centred.

Details

This function is deprecated as support for multiple size factors in SingleCellExperiment is deprecated, and scaling one set of size factors is largely trivial.

Centering of size factors at unity ensures that division by size factors yields values on the same scale as the raw counts. This is important for the interpretation of the normalized values, as well as comaprisons between features normalized with different size factors (e.g., spike-ins).

Value

A SingleCellExperiment with modified size factors that are centred at unity.

Author(s)

Aaron Lun

See Also

normalizeSCE

Examples

```
example_sce <- mockSCE()
sizeFactors(example_sce) <- runif(ncol(example_sce))
sizeFactors(example_sce, "ERCC") <- runif(ncol(example_sce))
example_sce <- centreSizeFactors(example_sce)

mean(sizeFactors(example_sce))
mean(sizeFactors(example_sce, "ERCC"))
```

---

getExplanatoryPCs *Per-PC variance explained by a variable*

Description

Compute, for each principal component, the percentage of variance that is explained by one or more variables of interest.

Usage

```
getExplanatoryPCs(
  x,
  dimred = "PCA",
  use_dimred = NULL,
  n_dimred = 10,
  ncomponents = NULL,
  rerun = FALSE,
  run_args = list(),
  ...
)
```

Arguments

x	A SingleCellExperiment object containing dimensionality reduction results.
dimred	String or integer scalar specifying the field in reducedDims(x) that contains the PCA results.
use_dimred	Deprecated, same as dimred.
n_dimred	Integer scalar specifying the number of the top principal components to use.
ncomponents	Deprecated, same as n_dimred.
rerun	Deprecated. Logical scalar indicating whether the PCA should be repeated, even if pre-computed results are already present.
run_args	Deprecated. A named list of arguments to pass to runPCA.
...	Additional arguments passed to getVarianceExplained.

Details

This function computes the percentage of variance in PC scores that is explained by variables in the sample-level metadata. It allows identification of important PCs that are driven by known experimental conditions, e.g., treatment, disease. PCs correlated with technical factors (e.g., batch effects, library size) can also be detected and removed prior to further analysis.

By default, the function will attempt to use pre-computed PCA results in `object`. This is done by taking the top `n_dimred` PCs from the matrix specified by `dimred`. If these are not available or if `rerun=TRUE`, the function will rerun the PCA using `runPCA`; however, this mode is deprecated and users are advised to explicitly call `runPCA` themselves.

Value

A matrix containing the percentage of variance explained by each factor (column) and for each PC (row).

Author(s)

Aaron Lun

See Also

`plotExplanatoryPCs`, to plot the results.

`getVarianceExplained`, to compute the variance explained.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
example_sce <- runPCA(example_sce)

r2mat <- getExplanatoryPCs(example_sce)
```

---

getVarianceExplained *Per-gene variance explained by a variable*

Description

Compute, for each gene, the percentage of variance that is explained by one or more variables of interest.

Usage

```
getVarianceExplained(x, ...)

## S4 method for signature 'ANY'
getVarianceExplained(x, variables, subset_row = NULL, chunk = 1000)

## S4 method for signature 'SummarizedExperiment'
getVarianceExplained(x, variables = NULL, ..., exprs_values = "logcounts")
```

Arguments

x	A numeric matrix of expression values, usually log-transformed and normalized. Alternatively, a SummarizedExperiment containing such a matrix.
...	For the generic, arguments to be passed to specific methods. For the SummarizedExperiment method, arguments to be passed to the ANY method.
variables	A DataFrame or data.frame containing one or more variables of interest. This should have number of rows equal to the number of columns in x. For the SummarizedExperiment method, this can also be a character vector specifying column names of colData(x) to use; or NULL, in which case all columns in colData(x) are used.
subset_row	A vector specifying the subset of rows of x for which to return a result.
chunk	Integer scalar specifying the chunk size for chunk-wise processing. Only affects the speed/memory usage trade-off.
exprs_values	String or integer scalar specifying the expression values for which to compute the variance.

Details

This function computes the percentage of variance in gene expression that is explained by variables in the sample-level metadata. It allows problematic factors to be quickly identified, as well as the genes that are most affected.

Value

A numeric matrix containing the percentage of variance explained by each factor (column) and for each gene (row).

Author(s)

Aaron Lun

See Also

getExplanatoryPCs, which calls this function.

plotExplanatoryVariables, to plot the results.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

r2mat <- getVarianceExplained(example_sce)
```

`isOutlier` *Identify outlier values*

Description

Convenience function to determine which values in a numeric vector are outliers based on the median absolute deviation (MAD).

Usage

```
isOutlier(
  metric,
  nmads = 3,
  type = c("both", "lower", "higher"),
  log = FALSE,
  subset = NULL,
  batch = NULL,
  share_medians = FALSE,
  share_mads = FALSE,
  share_missing = TRUE,
  min_diff = NA
)
```

Arguments

`metric`	Numeric vector of values.
`nmads`	A numeric scalar, specifying the minimum number of MADs away from median required for a value to be called an outlier.
`type`	String indicating whether outliers should be looked for at both tails (`"both"`), only at the lower tail (`"lower"`) or the upper tail (`"higher"`).
`log`	Logical scalar, should the values of the metric be transformed to the log2 scale before computing MADs?
`subset`	Logical or integer vector, which subset of values should be used to calculate the median/MAD? If `NULL`, all values are used.
`batch`	Factor of length equal to `metric`, specifying the batch to which each observation belongs. A median/MAD is calculated for each batch, and outliers are then identified within each batch.
`share_medians`	Logical scalar indicating whether the median calculation should be shared across batches. Only used if `batch` is specified.
`share_mads`	Logical scalar indicating whether the MAD calculation should be shared across batches. Only used if `batch` is specified.
`share_missing`	Logical scalar indicating whether values should be shared across batches if they cannot be computed for a batch, e.g., due to subsetting.
`min_diff`	A numeric scalar indicating the minimum difference from the median to consider as an outlier. Ignored if `NA`.

Details

Lower and upper thresholds are stored in the "threshold" attribute of the returned vector. By default, this is a numeric vector of length 2 for the threshold on each side. If type="lower", the higher limit is Inf, while if type="higher", the lower limit is -Inf.

If min_diff is not NA, the minimum distance from the median required to define an outlier is set as the larger of nmads MADs and min_diff. This aims to avoid calling many outliers when the MAD is very small, e.g., due to discreteness of the metric. If log=TRUE, this difference is defined on the log2 scale.

If subset is specified, the median and MAD are computed from a subset of cells and the values are used to define the outlier threshold that is applied to all cells. In a quality control context, this can be handy for excluding groups of cells that are known to be low quality (e.g., failed plates) so that they do not distort the outlier definitions for the rest of the dataset.

Missing values trigger a warning and are automatically ignored during estimation of the median and MAD. The corresponding entries of the output vector are also set to NA values.

Value

A logical vector of the same length as the metric argument, specifying the observations that are considered as outliers.

Handling batches

If batch is specified, outliers are defined within each batch separately using batch-specific median and MAD values. This gives the same results as if the input metrics were subsetted by batch and isOutlier was run on each subset, and is often useful when batches are known *a priori* to have technical differences (e.g., in sequencing depth).

If share_medians=TRUE, a shared median is computed across all cells. If shared_mads=TRUE, a shared MAD is computed using all cells (from either a batch-specific or shared median, depending on share_medians). These settings are useful to enforce a common location or spread across batches, e.g., we might set shared_mads=TRUE for log-library sizes if coverage varies across batches but the variance across cells is expected to be consistent across batches.

If a batch does not have sufficient cells to compute the median or MAD (e.g., after applying subset), the default setting of share_missing=TRUE will set these values to the shared median and MAD. This allows us to define thresholds for low-quality batches based on information in the rest of the dataset. (Note that the use of shared values only affects this batch and not others unless share_medians and share_mads are also set.) Otherwise, if share_missing=FALSE, all cells in that batch will have NA in the output.

If batch is specified, the "threshold" attribute in the returned vector is a matrix with one named column per level of batch and two rows (one per threshold).

Author(s)

Aaron Lun

See Also

quickPerCellQC, a convenience wrapper to perform outlier-based quality control.

perCellQCMetrics, to compute potential QC metrics.

Examples

```
example_sce <- mockSCE()
stats <- perCellQCMetrics(example_sce)

str(isOutlier(stats$sum))
str(isOutlier(stats$sum, type="lower"))
str(isOutlier(stats$sum, type="higher"))

str(isOutlier(stats$sum, log=TRUE))

b <- sample(LETTERS[1:3], ncol(example_sce), replace=TRUE)
str(isOutlier(stats$sum, log=TRUE, batch=b))
```

`librarySizeFactors` *Compute library size factors*

Description

Define per-cell size factors from the library sizes (i.e., total sum of counts per cell).

Usage

```
librarySizeFactors(x, ...)

## S4 method for signature 'ANY'
librarySizeFactors(
  x,
  subset_row = NULL,
  geometric = FALSE,
  pseudo_count = 1,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
librarySizeFactors(x, exprs_values = "counts", ...)

computeLibraryFactors(x, ...)
```

Arguments

`x`	For `librarySizeFactors`, a numeric matrix of counts with one row per feature and column per cell. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such counts. For `computeLibraryFactors`, only a SingleCellExperiment is accepted.
`...`	For the `librarySizeFactors` generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `computeLibraryFactors`, further arguments to pass to `librarySizeFactors`.
`subset_row`	A vector specifying whether the size factors should be computed from a subset of rows of `x`.

geometric	Logical scalar indicating whether the size factor should be defined using the geometric mean.
pseudo_count	Numeric scalar specifying the pseudo-count to add during log-transformation when geometric=TRUE.
BPPARAM	A BiocParallelParam object indicating how calculations are to be parallelized. Only relevant when x is a DelayedArray object.
exprs_values	String or integer scalar indicating the assay of x containing the counts.

Details

Library sizes are converted into size factors by scaling them so that their mean across cells is unity. This ensures that the normalized values are still on the same scale as the raw counts. Preserving the scale is useful for interpretation of operations on the normalized values, e.g., the pseudo-count used in logNormCounts can actually be considered an additional read/UMI. This is important for ensuring that the effect of the pseudo-count decreases with increasing sequencing depth.

When using the library size-derived size factor, we implicitly assume that sequencing coverage is the only difference between cells. This is reasonable for homogeneous cell populations but is compromised by composition biases introduced by DE genes between cell types. In such cases, normalization by library size factors will not be entirely correct though the effect on downstream conclusions will vary, e.g., clustering is usually unaffected by composition biases but log-fold change estimates will be less accurate.

A closely related alternative approach involves using the geometric mean of counts within each cell to define the size factor, instead of the library size (which is proportional to the arithmetic mean). This is enabled with geometric=TRUE with addition of pseudo_count to avoid undefined values with zero counts. The geometric mean is more robust to composition biases from upregulated features but is a poor estimator of the relative bias when there are many zero counts, and thus is best suited for deeply sequenced features, e.g., antibody-derived tags.

Value

For librarySizeFactors, a numeric vector of size factors is returned for all methods.

For computeLibraryFactors, a numeric vector is also returned for the ANY and SummarizedExperiment methods. For the SingleCellExperiment method, x is returned containing the size factors in sizeFactors(x).

Author(s)

Aaron Lun

See Also

logNormCounts, where these size factors are used by default.

Examples

```
example_sce <- mockSCE()
summary(librarySizeFactors(example_sce))
```

`logNormCounts` *Compute log-normalized expression values*

Description

Compute log-transformed normalized expression values from a count matrix in a SingleCellExperiment object.

Usage

```
logNormCounts(x, ...)

## S4 method for signature 'SummarizedExperiment'
logNormCounts(
  x,
  size_factors = NULL,
  log = TRUE,
  pseudo_count = 1,
  center_size_factors = TRUE,
  ...,
  exprs_values = "counts",
  name = NULL
)

## S4 method for signature 'SingleCellExperiment'
logNormCounts(
  x,
  size_factors = NULL,
  log = TRUE,
  pseudo_count = 1,
  center_size_factors = TRUE,
  ...,
  exprs_values = "counts",
  use_altexps = FALSE,
  name = NULL
)
```

Arguments

`x`	A SingleCellExperiment or SummarizedExperiment object containing a count matrix.
`...`	For the generic, additional arguments passed to specific methods. For the methods, additional arguments passed to `normalizeCounts`.
`size_factors`	A numeric vector of cell-specific size factors. Alternatively `NULL`, in which case the size factors are extracted or computed from `x`.
`log`	Logical scalar indicating whether normalized values should be log2-transformed.
`pseudo_count`	Numeric scalar specifying the pseudo_count to add when log-transforming expression values.

center_size_factors
: Logical scalar indicating whether size factors should be centered at unity before being used.

exprs_values
: A string or integer scalar specifying the assay of x containing the count matrix.

name
: String containing an assay name for storing the output normalized values. Defaults to "logcounts" when log=TRUE and "normcounts" otherwise.

use_altexps
: Logical scalar indicating whether normalization should be performed for alternative experiments in x.

  Alternatively, a character vector specifying the names of the alternative experiments to be normalized.

  Alternatively, NULL in which case alternative experiments are not used.

Details

This function is a convenience wrapper around normalizeCounts. It returns a SingleCellExperiment or SummarizedExperiment containing the normalized values in a separate assay. This makes it easier to perform normalization by avoiding book-keeping errors during a long analysis workflow.

If x is a SingleCellExperiment that contains alternative experiments, normalized values can be computed and stored within each alternative experiment by setting use_altexps appropriately. By default, use_altexps=FALSE to avoid problems from attempting to library size-normalize alternative experiments that have zero total counts for some cells.

If size_factors=NULL, size factors are obtained separately for each nested experiment following the rules in normalizeCounts. However, if size_factors is supplied, it will override any size factors available in the alternative experiments.

Value

x is returned containing the (log-)normalized expression values in an additional assay named as name.

If x is a SingleCellExperiment, the size factors used for normalization are stored in sizeFactors. These are centered if center_size_factors=TRUE.

If x contains alternative experiments and use_altexps=TRUE, each of the alternative experiments in x will also contain an additional assay. This can be limited to particular altExps entries by specifying them in use_altexps.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

normalizeCounts, which is used to compute the normalized expression values.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
assayNames(example_sce)
```

`medianSizeFactors` *Compute median-based size factors*

Description

Define per-cell size factors by taking the median of ratios to a reference expression profile (a la **DESeq**).

Usage

```
medianSizeFactors(x, ...)

## S4 method for signature 'ANY'
medianSizeFactors(x, subset_row = NULL, reference = NULL)

## S4 method for signature 'SummarizedExperiment'
medianSizeFactors(x, exprs_values = "counts", ...)

computeMedianFactors(x, ...)
```

Arguments

`x`	For `medianSizeFactors`, a numeric matrix of counts with one row per feature and column per cell. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such counts. For `computeMedianFactors`, only a SingleCellExperiment is accepted.
`...`	For the `medianSizeFactors` generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `computeMedianFactors`, further arguments to pass to `medianSizeFactors`.
`subset_row`	A vector specifying whether the size factors should be computed from a subset of rows of `x`.
`reference`	A numeric vector of length equal to `nrow(x)`, containing the reference expression profile. Defaults to `rowMeans(x)`.
`exprs_values`	String or integer scalar indicating the assay of `x` containing the counts.

Details

This function implements a modified version of the **DESeq2** size factor calculation. For each cell, the size factor is proportional to the median of the ratios of that cell's counts to `reference`. The assumption is that most genes are not DE between the cell and the reference, such that the median captures any systematic increase due to technical biases. The modification stems from the fact that we use the arithmetic mean instead of the geometric mean to compute `reference`, as the former is more robust to the many zeros in single-cell RNA sequencing data.

That said, the median-based approach tends to perform poorly for typical scRNA-seq datasets for various reasons:

- The high number of zeroes in the count matrix means that the median ratio for each cell is often zero. If this method must be used, we recommend subsetting to only the highest-abundance genes to avoid problems with zeroes. (Of course, the smaller the subset, the more sensitive the results are to noise or violations of the non-DE majority.)

- The default reference effectively requires a non-DE majority of genes between *any* pair of cells in the dataset. This is a strong assumption for heterogeneous populations containing many cell types; most genes are likely to exhibit DE between at least one pair of cell types.

For these reasons, the simpler librarySizeFactors is usually preferred, which is no less inaccurate but is guarantted to return a positive size factor for any cell with non-zero counts.

One valid application of this method lies in the normalization of antibody-derived tag counts for quantifying surface proteins. These counts are usually large enough to avoid zeroes yet are also susceptible to strong composition biases that preclude the use of librarySizeFactors. In such cases, we would also set reference to the ambient profile (where possible). This assumes that most proteins are not expressed in each cell; thus, counts for most tags for any given cell can be attributed to background contamination that should not be DE between cells.

Value

For medianSizeFactors, a numeric vector of size factors is returned for all methods.

For computeMedianFactors, a numeric vector is also returned for the ANY and SummarizedExperiment methods. For the SingleCellExperiment method, x is returned containing the size factors in sizeFactors(x).

Author(s)

Aaron Lun

See Also

logNormCounts, where these size factors can be used.

librarySizeFactors, for the default method for computing size factors.

Examples

```
example_sce <- mockSCE()
summary(medianSizeFactors(example_sce))
```

---

mockSCE *Mock up a SingleCellExperiment*

Description

Mock up a SingleCellExperiment containing simulated data, for use in documentation examples.

Usage

```
mockSCE(ncells = 200, ngenes = 2000, nspikes = 100)
```

Arguments

ncells	Integer scalar, number of cells to simulate.
ngenes	Integer scalar, number of genes to simulate.
nspikes	Integer scalar, number of spike-in transcripts to simulate.

Details

Users should set a seed to obtain reproducible results from this function.

Value

A SingleCellExperiment object containing a count matrix in the "counts" assay, a set of simulated colData fields, and spike-in data in the "Spikes" field of altExps.

Author(s)

Aaron Lun

See Also

SingleCellExperiment, for the constructor.

Examples

```
set.seed(1000)
sce <- mockSCE()
sce
```

---

multiplot *Multiple plot function for ggplot2 plots*

Description

Place multiple ggplot plots on one page.

Usage

```
multiplot(..., plotlist = NULL, cols = 1, layout = NULL)
```

Arguments

...	One or more ggplot objects.
plotlist	A list of ggplot objects, as an alternative to ....
cols	A numeric scalar giving the number of columns in the layout.
layout	A matrix specifying the layout. If present, cols is ignored.

Details

If the layout is something like matrix(c(1,2,3,3),nrow=2,byrow=TRUE), then:

- plot 1 will go in the upper left;
- plot 2 will go in the upper right;
- and plot 3 will go all the way across the bottom.

There is no way to tweak the relative heights or widths of the plots with this simple function. It was adapted from http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/

Value

A ggplot object.

Examples

```
library(ggplot2)

## This example uses the ChickWeight dataset, which comes with ggplot2
## First plot
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet, group = Chick)) +
   geom_line() +
   ggtitle("Growth curve for individual chicks")
## Second plot
p2 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet)) +
   geom_point(alpha = .3) +
   geom_smooth(alpha = .2, size = 1) +
   ggtitle("Fitted growth curve per diet")

## Third plot
p3 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, colour = Diet)) +
   geom_density() +
   ggtitle("Final weight, by diet")
## Fourth plot
p4 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, fill = Diet)) +
    geom_histogram(colour = "black", binwidth = 50) +
   facet_grid(Diet ~ .) +
   ggtitle("Final weight, by diet") +
   theme(legend.position = "none")        # No legend (redundant in this graph)

## Combine plots and display
multiplot(p1, p2, p3, p4, cols = 2)
```

`nexprs` *Count the number of non-zero counts per cell or feature*

Description

Counting the number of non-zero counts in each row (per feature) or column (per cell), without constructing an intermediate logical matrix.

Usage

```
nexprs(x, ...)

## S4 method for signature 'ANY'
nexprs(
  x,
  byrow = FALSE,
  detection_limit = 0,
  subset_row = NULL,
  subset_col = NULL,
  BPPARAM = SerialParam()
```

```
)

## S4 method for signature 'SummarizedExperiment'
nexprs(x, ..., exprs_values = "counts")
```

Arguments

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment containing such counts.
...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
byrow	Logical scalar indicating whether to count the number of detected cells per feature. If FALSE, the function will count the number of detected features per cell.
detection_limit	Numeric scalar providing the value above which observations are deemed to be expressed.
subset_row	Logical, integer or character vector indicating which rows (i.e. features) to use.
subset_col	Logical, integer or character vector indicating which columns (i.e., cells) to use.
BPPARAM	A BiocParallelParam object specifying whether the calculations should be parallelized.
exprs_values	String or integer specifying the assay of x to obtain the count matrix from.

Value

An integer vector containing counts per gene or cell, depending on the provided arguments.

Author(s)

Aaron Lun

See Also

numDetectedAcrossFeatures and numDetectedAcrossCells, to do this calculation for each group of features or cells, respectively.

Examples

```
example_sce <- mockSCE()

nexprs(example_sce)[1:10]
nexprs(example_sce, byrow = TRUE)[1:10]
```

normalize *Normalize a SingleCellExperiment object using pre-computed size factors*

Description

Compute normalized expression values from count data in a SingleCellExperiment object, using the size factors stored in the object. This function is now deprecated, use `logNormCounts` instead.

Usage

```
normalizeSCE(
  object,
  exprs_values = "counts",
  return_log = TRUE,
  log_exprs_offset = NULL,
  centre_size_factors = TRUE,
  preserve_zeroes = FALSE
)

## S4 method for signature 'SingleCellExperiment'
normalize(
  object,
  exprs_values = "counts",
  return_log = TRUE,
  log_exprs_offset = NULL,
  centre_size_factors = TRUE,
  preserve_zeroes = FALSE
)
```

Arguments

`object`	A SingleCellExperiment object.
`exprs_values`	String indicating which assay contains the count data that should be used to compute log-transformed expression values.
`return_log`	Logical scalar, should normalized values be returned on the log2 scale? If `TRUE`, output is stored as `"logcounts"` in the returned object; if `FALSE` output is stored as `"normcounts"`.
`log_exprs_offset`	Numeric scalar specifying the pseudo-count to add when log-transforming expression values. If `NULL`, the value is taken from `metadata(object)$log.exprs.offset` if defined, otherwise it is set to 1.
`centre_size_factors`	Logical scalar indicating whether size fators should be centred.
`preserve_zeroes`	Logical scalar indicating whether zeroes should be preserved when dealing with non-unity offsets.

Details

Normalized expression values are computed by dividing the counts for each cell by the size factor for that cell. This aims to remove cell-specific scaling biases, e.g., due to differences in sequencing coverage or capture efficiency. If `log=TRUE`, log-normalized values are calculated by adding `log_exprs_offset` to the normalized count and performing a log2 transformation.

Features marked as spike-in controls will be normalized with control-specific size factors, if these are available. This reflects the fact that spike-in controls are subject to different biases than those that are removed by gene-specific size factors (namely, total RNA content). If size factors for a particular spike-in set are not available, a warning will be raised.

If `centre_size_factors=TRUE`, all sets of size factors will be centred to have the same mean prior to calculation of normalized expression values. This ensures that abundances are roughly comparable between features normalized with different sets of size factors. By default, the centre mean is unity, which means that the computed `exprs` can be interpreted as being on the same scale as log-counts. It also means that the added `log_exprs_offset` can be interpreted as a pseudo-count (i.e., on the same scale as the counts).

If `preserve_zeroes=TRUE` and the pseudo-count is not unity, size factors are instead centered at the specified value of `log_exprs_offset`. The log-transformation is then performed on the normalized expression values with a pseudo-count of 1, which ensures that zeroes remain so in the output matrix. This yields the same results as `preserve_zeroes=FALSE` minus a matrix-wide constant of `log2(log_exprs_offset)`.

In some cases, the function will return a DelayedMatrix with delayed division and log-transformation operations. This requires that the assay specified by `exprs_values` contains a DelayedMatrix, and only one set of size factors is used for all features. This avoids the need to explicitly calculate normalized expression values across a very large (possibly file-backed) matrix.

Value

A SingleCellExperiment object containing normalized expression values in `"normcounts"` if `log=FALSE`, and log-normalized expression values in `"logcounts"` if `log=TRUE`. All size factors will also be centred in the output object if `centre_size_factors=TRUE`.

Author(s)

Davis McCarthy and Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- normalize(example_sce)
```

`normalizeCounts` *Compute normalized expression values*

Description

Compute (log-)normalized expression values by dividing counts for each cell by the corresponding size factor.

Usage

```
normalizeCounts(x, ...)

## S4 method for signature 'ANY'
normalizeCounts(
  x,
  size_factors = NULL,
  use_size_factors = NULL,
  log = TRUE,
  return_log = NULL,
  pseudo_count = 1,
  log_exprs_offset = NULL,
  center_size_factors = TRUE,
  subset_row = NULL,
  downsample = FALSE,
  down_target = NULL,
  down_prop = 0.01
)

## S4 method for signature 'SummarizedExperiment'
normalizeCounts(x, ..., exprs_values = "counts")

## S4 method for signature 'SingleCellExperiment'
normalizeCounts(x, size_factors = NULL, ...)
```

Arguments

x	A numeric matrix-like object containing counts for cells in the columns and features in the rows. Alternatively, a SingleCellExperiment or SummarizedExperiment object containing such a count matrix.
...	For the generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY or DelayedMatrix methods. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
size_factors	A numeric vector of cell-specific size factors. Alternatively NULL, in which case the size factors are extracted or computed from x.
use_size_factors	Deprecated, same as size_factors.
log	Logical scalar indicating whether normalized values should be log2-transformed.
return_log	Deprecated, same as log.
pseudo_count	Numeric scalar specifying the pseudo_count to add when log-transforming expression values.
log_exprs_offset	Deprecated, same as pseudo_count.
center_size_factors	Logical scalar indicating whether size factors should be centered at unity before being used.

subset_row	A vector specifying the subset of rows of x for which to return a result.
downsample	Logical scalar indicating whether downsampling should be performed prior to scaling and log-transformation.
down_target	Numeric scalar specifying the downsampling target when downsample=TRUE. If NULL, this is defined by down_prop and a warning is emitted.
down_prop	Numeric scalar between 0 and 1 indicating the quantile to use to define the downsampling target when downsample=TRUE.
exprs_values	A string or integer scalar specifying the assay of x containing the count matrix.

Details

Normalized expression values are computed by dividing the counts for each cell by the size factor for that cell. This aims to remove cell-specific scaling biases, e.g., due to differences in sequencing coverage or capture efficiency. If log=TRUE, log-normalized values are calculated by adding pseudo_count to the normalized count and performing a log2 transformation.

If no size factors are supplied, they are determined automatically from x:

- For count matrices and SummarizedExperiment inputs, the sum of counts for each cell is used to compute a size factor via the librarySizeFactors function.
- For SingleCellExperiment instances, the function searches for sizeFactors from x. If none are available, it defaults to library size-derived size factors.

If size_factors are supplied, they will override any size factors present in x.

If center_size_factors=TRUE, size factors are centred at unity prior to calculation of normalized expression values. This means that the computed expression values can be interpreted as being on the same scale as log-counts, and that the value of pseudo_count can be interpreted as being on the same scale as the counts. It also ensures that abundances are roughly comparable between features normalized with different sets of size factors.

Value

A matrix-like object of (log-)normalized expression values.

Downsampling instead of scaling

If downsample=TRUE, counts for each cell are randomly downsampled according to their size factors prior to log-transformation. This is occasionally useful for avoiding artifacts caused by scaling count data with a strong mean-variance relationship. Each cell is downsampled according to the ratio between down_target and that cell's size factor. (Cells with size factors below the target are not downsampled and are directly scaled by this ratio.) If log=TRUE, a log-transformation is also performed after adding pseudo_count to the downsampled counts.

Note that the normalized expression values in this mode cannot be interpreted as being on the same abundance as the original counts, but instead have abundance equivalent to counts after downsampling to the target size factor. This motivates the use of a fixed down_target to ensure that expression values are comparable across different normalizeCounts calls. We automatically set down_target to the 1st percentile of size factors across all cells involved in the analysis, but this is only appropriate if the resulting expression values are only compared within the same call to normalizeCounts. If expression values are to be compared across multiple calls (e.g., in modelGeneVarWithSpikes or multiBatchNorm), down_target should be manually set to a constant target value that can be considered a low size factor in every call.

Author(s)

Aaron Lun

See Also

logNormCounts, which wraps this function for convenient use with SingleCellExperiment instances.

downsampleMatrix, to perform the downsampling.

Examples

```
example_sce <- mockSCE()
normed <- normalizeCounts(example_sce)
str(normed)
```

---

norm_exprs *Additional accessors for the typical elements of a SingleCellExperiment object.*

Description

Convenience functions to access commonly-used assays of the SingleCellExperiment object.

Usage

```
norm_exprs(object)

norm_exprs(object) <- value

stand_exprs(object)

stand_exprs(object) <- value

fpkm(object)

fpkm(object) <- value
```

Arguments

object	SingleCellExperiment class object from which to access or to which to assign assay values. Namely: "exprs", norm_exprs", "stand_exprs", "fpkm". The following are imported from SingleCellExperiment: "counts", "normcounts", "logcounts", "cpm", "tpm".
value	a numeric matrix (e.g. for exprs)

Value

a matrix of normalised expression data

a matrix of standardised expressiond data

a matrix of FPKM values

A matrix of numeric, integer or logical values.

Author(s)

Davis McCarthy

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
head(logcounts(example_sce)[,1:10])
head(exprs(example_sce)[,1:10]) # identical to logcounts()

norm_exprs(example_sce) <- log2(calculateCPM(example_sce) + 1)

stand_exprs(example_sce) <- log2(calculateCPM(example_sce) + 1)

tpm(example_sce) <- calculateTPM(example_sce, lengths = 5e4)

cpm(example_sce) <- calculateCPM(example_sce)

fpkm(example_sce)
```

---

numDetectedAcrossCells

Number of detected expression values per group of cells

Description

Computes the number of detected expression values (default defined as non-zero counts) for each feature in each group of cells.

Usage

```
numDetectedAcrossCells(x, ...)

## S4 method for signature 'ANY'
numDetectedAcrossCells(
  x,
  ids,
  average = FALSE,
  subset_row = NULL,
  subset_col = NULL,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
numDetectedAcrossCells(x, ..., exprs_values = "counts")
```

Arguments

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment containing such counts.

...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the ANY method, further arguments to pass to the nexprs function.
ids	A vector of length ncol(x), specifying the group assignment for each cell.
average	Logical scalar indicating whether the proportion of non-zero counts in each group should be computed instead.
subset_row	A vector specifying the rows to use. Defaults to all rows.
subset_col	A vector specifying the columns to use. Defaults to all cells with non-NA entries of ids.
BPPARAM	A BiocParallelParam object specifying whether the calculations should be parallelized.
exprs_values	String or integer specifying the assay of x to obtain the count matrix from.

Value

An integer or numeric matrix containing the number or proportion of detected expression values for each feature (row) in each group of cells (column).

Author(s)

Aaron Lun

See Also

nexprs, on which this function is based.

Examples

```
example_sce <- mockSCE()

ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)
bycol <- numDetectedAcrossCells(example_sce, ids)
head(bycol)
```

---

numDetectedAcrossFeatures

Number of detected expression values per group of features

Description

Computes the number of detected expression values (default defined as non-zero counts) for each cell in each group of features.

Usage

```
numDetectedAcrossFeatures(x, ...)

## S4 method for signature 'ANY'
numDetectedAcrossFeatures(
  x,
  ids,
  average = FALSE,
  subset_row = NULL,
  subset_col = NULL,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
numDetectedAcrossFeatures(x, ..., exprs_values = "counts")
```

Arguments

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a SummarizedExperiment containing such counts.
...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the ANY method, further arguments to pass to the nexprs function.
ids	A vector of length nrow(x), specifying the group assignment for each feature.
average	Logical scalar indicating whether the proportion of non-zero counts in each group should be computed instead.
subset_row	A vector specifying the rows to use. Defaults to all rows with non-NA entries of ids.
subset_col	A vector specifying the columns to use. Defaults to all columns.
BPPARAM	A BiocParallelParam object specifying whether the calculations should be parallelized.
exprs_values	String or integer specifying the assay of x to obtain the count matrix from.

Value

An integer or numeric matrix containing the number of detected expression values in each group of features (row) and cell (column).

Author(s)

Aaron Lun

See Also

nexprs, on which this function is based.

Examples

```
example_sce <- mockSCE()

ids <- sample(paste0("GENE_", 1:100), nrow(example_sce), replace=TRUE)
byrow <- numDetectedAcrossFeatures(example_sce, ids)
head(byrow[,1:10])
```

`perCellQCMetrics` *Per-cell quality control metrics*

Description

Compute per-cell quality control metrics for a count matrix or a SingleCellExperiment.

Usage

```
perCellQCMetrics(x, ...)

## S4 method for signature 'ANY'
perCellQCMetrics(
  x,
  subsets = NULL,
  percent_top = c(50, 100, 200, 500),
  detection.limit = 0,
  BPPARAM = SerialParam(),
  flatten = TRUE
)

## S4 method for signature 'SummarizedExperiment'
perCellQCMetrics(x, ..., exprs_values = "counts")

## S4 method for signature 'SingleCellExperiment'
perCellQCMetrics(
  x,
  subsets = NULL,
  percent_top = c(50, 100, 200, 500),
  ...,
  flatten = TRUE,
  exprs_values = "counts",
  use_altexps = TRUE
)
```

Arguments

`x`	A numeric matrix of counts with cells in columns and features in rows. Alternatively, a SummarizedExperiment or SingleCellExperiment object containing such a matrix.
`...`	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, further arguments to pass to the ANY method.

`subsets` A named list containing one or more vectors (a character vector of feature names, a logical vector, or a numeric vector of indices), used to identify interesting feature subsets such as ERCC spike-in transcripts or mitochondrial genes.

`percent_top` An integer vector. Each element is treated as a number of top genes to compute the percentage of library size occupied by the most highly expressed genes in each cell.

`detection.limit` A numeric scalar specifying the lower detection limit for expression.

`BPPARAM` A BiocParallelParam object specifying whether the QC calculations should be parallelized.

`flatten` Logical scalar indicating whether the nested DataFrames in the output should be flattened.

`exprs_values` A string or integer scalar indicating which `assays` in the `x` contains the count matrix.

`use_altexps` Logical scalar indicating whether QC statistics should be computed for alternative Experiments in `x`. If `TRUE`, statistics are computed for all alternative experiments.

Alternatively, an integer or character vector specifying the alternative Experiments to use to compute QC statistics.

Alternatively `NULL`, in which case alternative experiments are not used.

Details

This function calculates useful QC metrics for identification and removal of potentially problematic cells. Obvious per-cell metrics are the sum of counts (i.e., the library size) and the number of detected features. The percentage of counts in the top features also provides a measure of library complexity.

If `subsets` is specified, these statistics are also computed for each subset of features. This is useful for investigating gene sets of interest, e.g., mitochondrial genes, Y chromosome genes. These statistics are stored as nested DataFrames in the `subsets` field of the output. For example, if the input `subsets` contained `"Mito"` and `"Sex"`, the output would look like:

```
output
|-- sum
|-- detected
|-- percent_top
+-- subsets
    |-- Mito
    |   |-- sum
    |   |-- detected
    |   +-- percent
    +-- Sex
        |-- sum
        |-- detected
        +-- percent
```

Here, the `percent` field contains the percentage of each cell's count sum assigned to each subset.

If `use_altexps` is `TRUE`, the same statistics are computed for each alternative experiment in `x`. This can also be an integer or character vector specifying the alternative Experiments to use. These statistics are also stored as nested DataFrames, this time in the `altexps` field of the output. For example, if `x` contained the alternative Experiments `"Spike"` and `"Ab"`, the output would look like:

```
output
|-- sum
|-- detected
|-- percent_top
+-- altexps
|   |-- Spike
|   |   |-- sum
|   |   |-- detected
|   |   +-- percent.total
|   +-- Ab
|       |-- sum
|       |-- detected
|       +-- percent.total
+-- total
```

The `total` field contains the total sum of counts for each cell across the main and alternative Experiments. The `percent` field contains the percentage of the total count in each alternative Experiment for each cell.

If `flatten=TRUE`, the nested DataFrames are flattened by concatenating the column names with underscores. This means that, say, the `subsets$Mito$sum` nested field becomes the top-level `subsets_Mito_sum` field. A flattened structure is more convenient for end-users performing interactive analyses, but less convenient for programmatic access as artificial construction of strings is required.

Value

A DataFrame of QC statistics where each row corresponds to a column in `x`. This contains the following fields:

- `sum`: numeric, the sum of counts for each cell.
- `detected`: numeric, the number of observations above `detection.limit`.

If `flatten=FALSE`, the DataFrame will contain the additional columns:

- `percent_top`: numeric matrix, the percentage of counts assigned to the percent_topage of most highly expressed genes. Each column of the matrix corresponds to an entry of the sorted `percent_top`, in increasing order.
- `subsets`: A nested DataFrame containing statistics for each subset, see Details.
- `altexps`: A nested DataFrame containing statistics for each alternative experiment, see Details. This is only returned for the SingleCellExperiment method.
- `total`: numeric, the total sum of counts for each cell across main and alternative Experiments. This is only returned for the SingleCellExperiment method.

If `flatten=TRUE`, nested matrices and DataFrames are flattened to remove the hierarchical structure from the output DataFrame.

Author(s)

Aaron Lun

See Also

`addPerCellQC`, to add the QC metrics to the column metadata.

Examples

```
example_sce <- mockSCE()
stats <- perCellQCMetrics(example_sce)
stats

# With subsets.
stats2 <- perCellQCMetrics(example_sce, subsets=list(Mito=1:10),
    flatten=FALSE)
stats2$subsets

# With alternative Experiments.
pretend.spike <- ifelse(seq_len(nrow(example_sce)) < 10, "Spike", "Gene")
alt_sce <- splitAltExps(example_sce, pretend.spike)
stats3 <- perCellQCMetrics(alt_sce, flatten=FALSE)
stats3$altexps
```

`perFeatureQCMetrics` *Per-feature quality control metrics*

Description

Compute per-feature quality control metrics for a count matrix or a SummarizedExperiment.

Usage

```
perFeatureQCMetrics(x, ...)

## S4 method for signature 'ANY'
perFeatureQCMetrics(
  x,
  subsets = NULL,
  detection_limit = 0,
  BPPARAM = SerialParam(),
  flatten = TRUE
)

## S4 method for signature 'SummarizedExperiment'
perFeatureQCMetrics(x, ..., exprs_values = "counts")
```

Arguments

`x`	A numeric matrix of counts with cells in columns and features in rows. Alternatively, a SummarizedExperiment object containing such a matrix.
`...`	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, further arguments to pass to the ANY method.
`subsets`	A named list containing one or more vectors (a character vector of cell names, a logical vector, or a numeric vector of indices), used to identify interesting sample subsets such as negative control wells.

detection_limit
: A numeric scalar specifying the lower detection_limit for expression.

BPPARAM
: A BiocParallelParam object specifying whether the QC calculations should be parallelized.

flatten
: Logical scalar indicating whether the nested DataFrames in the output should be flattened.

exprs_values
: A string or integer scalar indicating which assays in the x contains the count matrix.

Details

This function calculates useful QC metrics for features, including the mean across all cells and the number of expressed features (i.e., counts above the detection_limit).

If subsets is specified, the same statistics are computed for each subset of cells. This is useful for obtaining statistics for cell sets of interest, e.g., negative control wells. These statistics are stored as nested DataFrames in the output. For example, if subsets contained "empty" and "cellpool", the output would look like:

```
output
|-- mean
|-- detected
+-- subsets
    |-- empty
    |   |-- mean
    |   |-- detected
    |   +-- ratio
    +-- cellpool
        |-- mean
        |-- detected
        +-- ratio
```

The ratio field contains the ratio of the mean within each subset to the mean across all cells.

If flatten=TRUE, the nested DataFrames are flattened by concatenating the column names with underscores. This means that, say, the subsets$empty$mean nested field becomes the top-level subsets_empty_mean field. A flattened structure is more convenient for end-users performing interactive analyses, but less convenient for programmatic access as artificial construction of strings is required.

Value

A DataFrame of QC statistics where each row corresponds to a row in x. This contains the following fields:

- mean: numeric, the mean counts for each feature.
- detected: numeric, the percentage of observations above detection_limit.

If flatten=FALSE, the output DataFrame also contains the subsets field. This a nested DataFrame containing per-feature QC statistics for each subset of columns.

If flatten=TRUE, subsets is flattened to remove the hierarchical structure.

Author(s)

Aaron Lun

See Also

addPerFeatureQC, to add the QC metrics to the row metadata.

Examples

```
example_sce <- mockSCE()
stats <- perFeatureQCMetrics(example_sce)
stats

# With subsets.
stats2 <- perFeatureQCMetrics(example_sce, subsets=list(Empty=1:10))
stats2$subsets
```

---

plotColData *Plot column metadata*

Description

Plot column-level (i.e., cell) metadata in an SingleCellExperiment object.

Usage

```
plotColData(
  object,
  y,
  x = NULL,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = "logcounts",
  by_show_single = FALSE,
  other_fields = list(),
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information.
y	String specifying the column-level metadata field to show on the y-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveCellInfo.
x	String specifying the column-level metadata to show on the x-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveCellInfo. If NULL, nothing is shown on the x-axis.
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.

size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see ?retrieveCellInfo for details.
by_show_single	Deprecated and ignored.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

If y is continuous and x=NULL, a violin plot is generated. If x is categorical, a grouped violin plot will be generated, with one violin for each level of x. If x is continuous, a scatter plot will be generated.

If y is categorical and x is continuous, horizontal violin plots will be generated. If x is missing or categorical, rectangle plots will be generated where the area of a rectangle is proportional to the number of points for a combination of factors.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
colData(example_sce) <- cbind(colData(example_sce),
    perCellQCMetrics(example_sce))

plotColData(example_sce, y = "detected", x = "sum",
   colour_by = "Mutation_Status") + scale_x_log10()

plotColData(example_sce, y = "detected", x = "sum",
   colour_by = "Mutation_Status", size_by = "Gene_0001",
   shape_by = "Treatment") + scale_x_log10()

plotColData(example_sce, y = "Treatment", x = "sum",
   colour_by = "Mutation_Status") + scale_y_log10() # flipped violin.

plotColData(example_sce, y = "detected",
   x = "Cell_Cycle", colour_by = "Mutation_Status")
```

plotDots *Create a dot plot of expression values*

Description

Create a dot plot of expression values for a grouping of cells, where the size and color of each dot represents the proportion of detected expression values and the average expression, respectively, for each feature in each group of cells.

Usage

```
plotDots(
  object,
  features,
  group = NULL,
  exprs_values = "logcounts",
  detection_limit = 0,
  low_color = "white",
  high_color = "red",
  max_ave = NULL,
  max_detected = NULL,
  other_fields = list(),
  by_exprs_values = exprs_values
)
```

Arguments

object	A SingleCellExperiment object.
features	A character vector of feature names to show as rows of the dot plot.
group	Specification of a column metadata field or a feature to show as columns. Alternatively, an AsIs vector, see ?retrieveCellInfo for details.
exprs_values	A string or integer scalar specifying which assay in assays(object) to obtain expression values from.
detection_limit	Numeric scalar providing the value above which observations are deemed to be expressed. This is also used as the
low_color	String specifying the color to use for low expression. This is also used as the background color, see Details.
high_color	String specifying the color to use for high expression.
max_ave	Numeric value specifying the cap on the average expression.
max_detected	Numeric value specifying the cap on the proportion of detected expression values.
other_fields	Additional feature-based fields to include in the data.frame, see ?"scater-plot-args" for details. Note that any AsIs vectors or data.frames must be of length equal to nrow(object), not features.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, to use when extracting values according to each entry of other_fields.

Details

This implements a **Seurat**-style "dot plot" that creates a dot for each feature (row) in each group of cells (column). The proportion of detected expression values and the average expression for each feature in each group of cells is visualized efficiently using the size and colour, respectively, of each dot.

We impose two restrictions - the low end of the color scale must correspond to the detection limit, and the color at this end of the scale must be the same as the background color. These ensure that the visual cues from low average expression or low detected proportions are consistent, as both will result in a stronger `low_color`. (In the latter case, the reduced size of the dot means that the background color dominates.)

If these restrictions are violated, visualization can be misleading due to the difficulty of simultaneously interpreting both size and color. For example, if we colored by z-score on a conventional blue-white-red color axis, a gene that is downregulated in a group of cells would show up as a small blue dot. If the background color was also white, this might be mistaken for a gene that is not downregulated at all. On the other hand, any other background color would effectively require consideration of two color axes as expression decreases.

We can also cap the color and size scales at `max_ave` and `max_detected`, respectively. This aims to preserve resolution for low-abundance genes by preventing domination of the scales by high-abundance features.

Value

A ggplot object containing a dot plot.

Author(s)

Aaron Lun

See Also

`plotExpression` and `plotHeatmap`, for alternatives to visualizing group-level expression values.

Examples

```
sce <- mockSCE()
sce <- logNormCounts(sce)
plotDots(sce, features=rownames(sce)[1:10], group="Cell_Cycle")
```

---

plotExplanatoryPCs *Plot the explanatory PCs for each variable*

Description

Plot the explanatory PCs for each variable

Usage

```
plotExplanatoryPCs(
  object,
  nvars_to_plot = 10,
  npcs_to_plot = 50,
  theme_size = 10,
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information. Alternatively, a matrix containing the output of getExplanatoryPCs.
nvars_to_plot	Integer scalar specifying the number of variables with the greatest explanatory power to plot. This can be set to Inf to show all variables.
npcs_to_plot	Integer scalar specifying the number of PCs to plot.
theme_size	numeric scalar providing base font size for ggplot theme.
...	Parameters to be passed to getExplanatoryPCs.

Details

A density plot is created for each variable, showing the R-squared for each successive PC (up to npcs_to_plot PCs). Only the nvars_to_plot variables with the largest maximum R-squared across PCs are shown.

If object is a SingleCellExperiment object, getExplanatoryPCs will be called to compute the variance in expression explained by each variable in each gene. Users may prefer to run getExplanatoryPCs manually and pass the resulting matrix as object, in which case the R-squared values are used directly.

Value

A ggplot object.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
plotExplanatoryPCs(example_sce)
```

---

plotExplanatoryVariables

Plot explanatory variables ordered by percentage of variance explained

---

Description

Plot explanatory variables ordered by percentage of variance explained

Usage

```
plotExplanatoryVariables(
  object,
  nvars_to_plot = 10,
  min_marginal_r2 = 0,
  theme_size = 10,
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information. Alternatively, a matrix containing the output of getVarianceExplained.
nvars_to_plot	Integer scalar specifying the number of variables with the greatest explanatory power to plot. This can be set to Inf to show all variables.
min_marginal_r2	Numeric scalar specifying the minimal value required for median marginal R-squared for a variable to be plotted. Only variables with a median marginal R-squared strictly larger than this value will be plotted.
theme_size	Numeric scalar specifying the font size to use for the plotting theme
...	Parameters to be passed to getVarianceExplained.

Details

A density plot is created for each variable, showing the distribution of R-squared across all genes. Only the nvars_to_plot variables with the largest median R-squared across genes are shown. Variables are also only shown if they have median R-squared values above min_marginal_r2.

If object is a SingleCellExperiment object, getVarianceExplained will be called to compute the variance in expression explained by each variable in each gene. Users may prefer to run getVarianceExplained manually and pass the resulting matrix as object, in which case the R-squared values are used directly.

Value

A ggplot object.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
plotExplanatoryVariables(example_sce)
```

---

plotExpression *Plot expression values for all cells*

Description

Plot expression values for a set of features (e.g. genes or transcripts) in a SingleExperiment object, against a continuous or categorical covariate for all cells.

Usage

```
plotExpression(
  object,
  features,
  x = NULL,
  exprs_values = "logcounts",
  log2_values = FALSE,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = exprs_values,
  by_show_single = FALSE,
  xlab = NULL,
  feature_colours = TRUE,
  one_facet = TRUE,
  ncol = 2,
  scales = "fixed",
  other_fields = list(),
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and other metadata.
features	A character vector or a list specifying the features to plot. If a list is supplied, each entry of the list can be a string, an AsIs-wrapped vector or a data.frame - see ?retrieveCellInfo.
x	Specification of a column metadata field or a feature to show on the x-axis, see the by argument in ?retrieveCellInfo for possible values.
exprs_values	A string or integer scalar specifying which assay in assays(object) to obtain expression values from.
log2_values	Logical scalar, specifying whether the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes).
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.
size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.
by_show_single	Deprecated and ignored.
xlab	String specifying the label for x-axis. If NULL (default), x will be used as the x-axis label.
feature_colours	Logical scalar indicating whether violins should be coloured by feature when x and colour_by are not specified and one_facet=TRUE.

one_facet	Logical scalar indicating whether grouped violin plots for multiple features should be put onto one facet. Only relevant when x=NULL.
ncol	Integer scalar, specifying the number of columns to be used for the panels of a multi-facet plot.
scales	String indicating whether should multi-facet scales be fixed ("fixed"), free ("free"), or free in one dimension ("free_x", "free_y"). Passed to the scales argument in the facet_wrap when multiple facets are generated.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

This function plots expression values for one or more features. If x is not specified, a violin plot will be generated of expression values. If x is categorical, a grouped violin plot will be generated, with one violin for each level of x. If x is continuous, a scatter plot will be generated.

If multiple features are requested and x is not specified and one_facet=TRUE, a grouped violin plot will be generated with one violin per feature. This will be coloured by feature if colour_by=NULL and feature_colours=TRUE, to yield a more aesthetically pleasing plot. Otherwise, if x is specified or one_facet=FALSE, a multi-panel plot will be generated where each panel corresponds to a feature. Each panel will be a scatter plot or (grouped) violin plot, depending on the nature of x.

Note that this assumes that the expression values are numeric. If not, and x is continuous, horizontal violin plots will be generated. If x is missing or categorical, rectangle plots will be generated where the area of a rectangle is proportional to the number of points for a combination of factors.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

## default plot
plotExpression(example_sce, rownames(example_sce)[1:15])

## plot expression against an x-axis value
plotExpression(example_sce, c("Gene_0001", "Gene_0004"),
    x="Mutation_Status")
plotExpression(example_sce, c("Gene_0001", "Gene_0004"),
    x="Gene_0002")

## add visual options
plotExpression(example_sce, rownames(example_sce)[1:6],
    colour_by = "Mutation_Status")
plotExpression(example_sce, rownames(example_sce)[1:6],
    colour_by = "Mutation_Status", shape_by = "Treatment",
    size_by = "Gene_0010")
```

```
## plot expression against expression values for Gene_0004
plotExpression(example_sce, rownames(example_sce)[1:4],
    "Gene_0004", show_smooth = TRUE)
```

---

plotExprsFreqVsMean *Plot frequency against mean for each feature*

Description

Plot the frequency of expression (i.e., percentage of expressing cells) against the mean expression level for each feature in a SingleCellExperiment object. This is deprecated in favour of directly using plotRowData.

Usage

```
plotExprsFreqVsMean(
  object,
  freq_exprs,
  mean_exprs,
  controls,
  exprs_values = "counts",
  by_show_single = FALSE,
  show_smooth = TRUE,
  show_se = TRUE,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
freq_exprs	String specifying the column-level metadata field containing the number of expressing cells per feature. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo.
mean_exprs	String specifying the column-level metadata fielcontaining the mean expression of each feature. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo.
controls	Deprecated and ignored.
exprs_values	String specifying the assay used for the default freq_exprs and mean_exprs. This can be set to, e.g., "logcounts" so that freq_exprs defaults to "n_cells_by_logcounts".
by_show_single	Deprecated and ignored.
show_smooth	Logical scalar, should a smoothed fit be shown on the plot? See geom_smooth for details.
show_se	Logical scalar, should the standard error be shown for a smoothed fit?
...	Further arguments passed to plotRowData.

Details

This function plots gene expression frequency versus mean expression level, which can be useful to assess the effects of technical dropout in the dataset. We fit a non-linear least squares curve for the relationship between expression frequency and mean expression. We use this curve to define the number of genes above high technical dropout and the numbers of genes that are expressed in at least 50% and at least 25% of cells.

Value

A ggplot object.

See Also

plotRowData

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- calculateQCMetrics(example_sce,
    feature_controls = list(set1 = 1:500))
plotExprsFreqVsMean(example_sce)

plotExprsFreqVsMean(example_sce, size_by = "is_feature_control")
```

---

plotExprsVsTxLength *Plot expression against transcript length*

Description

Plot mean expression values for all features in a SingleCellExperiment object against transcript length values. This is deprecated in favour of directly using plotRowData.

Usage

```
plotExprsVsTxLength(
  object,
  tx_length = "median_feat_eff_len",
  length_is_assay = FALSE,
  exprs_values = "logcounts",
  log2_values = FALSE,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = exprs_values,
  by_show_single = FALSE,
  xlab = "Median transcript length",
  show_exprs_sd = FALSE,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
tx_length	Transcript lengths for all features, to plot on the x-axis. If length_is_assay=FALSE, this should be a stirng specifying the column-level metadata field containing the number of expressing cells per feature. Otherwise,

	if `length_is_assay=TRUE`, `tx_length` should be the name or index of an assay in `object`. Alternatively, an AsIs vector or data.frame, see ?`retrieveFeatureInfo`.
`length_is_assay`	Logical scalar indicating whether `tx_length` refers to an assay of `object` containing transcript lengths for all features in all cells.
`exprs_values`	A string or integer scalar specifying which assay in `assays(object)` to obtain expression values from.
`log2_values`	Logical scalar, specifying whether the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes).
`colour_by`	Specification of a row metadata field or a sample to colour by, see ?`retrieveFeatureInfo` for possible values.
`shape_by`	Specification of a row metadata field or a sample to shape by, see ?`retrieveFeatureInfo` for possible values.
`size_by`	Specification of a row metadata field or a sample to size by, see ?`retrieveFeatureInfo` for possible values.
`by_exprs_values`	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see ?`retrieveFeatureInfo` for details.
`by_show_single`	Deprecated and ignored.
`xlab`	String specifying the label for x-axis.
`show_exprs_sd`	Logical scalar indicating whether the standard deviation of expression values for each feature should be plotted.
`...`	Additional arguments for visualization, see ?`"scater-plot-args"` for details.

Details

If `length_is_assay=TRUE`, the median transcript length of each feature across all cells is used. This may be necessary if the effective transcript length differs across cells, e.g., as observed in the results from pseudo-aligners.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
rowData(example_sce) <- DataFrame(gene_id = rownames(example_sce),
    feature_id = paste("feature", rep(1:500, each = 4), sep = "_"),
    median_tx_length = rnorm(2000, mean = 5000, sd = 500),
    other = sample(LETTERS, 2000, replace = TRUE)
)
example_sce <- logNormCounts(example_sce)

plotExprsVsTxLength(example_sce, "median_tx_length")
plotExprsVsTxLength(example_sce, "median_tx_length", show_smooth = TRUE)
```

```
plotExprsVsTxLength(example_sce, "median_tx_length", show_smooth = TRUE,
    colour_by = "other", show_exprs_sd = TRUE)

## using matrix of tx length values in assays(object)
mat <- matrix(rnorm(ncol(example_sce) * nrow(example_sce), mean = 5000,
    sd = 500), nrow = nrow(example_sce))
dimnames(mat) <- dimnames(example_sce)
assay(example_sce, "tx_len") <- mat

plotExprsVsTxLength(example_sce, "tx_len", show_smooth = TRUE,
    length_is_assay = TRUE, show_exprs_sd = TRUE)

## using a vector of tx length values
plotExprsVsTxLength(example_sce,
    data.frame(rnorm(2000, mean = 5000, sd = 500)))
```

---

plotHeatmap *Plot heatmap of gene expression values*

Description

Create a heatmap of expression values for each cell and specified features in a SingleCellExperiment object.

Usage

```
plotHeatmap(
  object,
  features,
  columns = NULL,
  exprs_values = "logcounts",
  center = FALSE,
  zlim = NULL,
  symmetric = FALSE,
  color = NULL,
  colour_columns_by = NULL,
  order_columns_by = NULL,
  by_exprs_values = exprs_values,
  by_show_single = FALSE,
  show_colnames = FALSE,
  cluster_cols = is.null(order_columns_by),
  ...
)
```

Arguments

object	A SingleCellExperiment object.
features	A character vector of row names, a logical vector of integer vector of indices specifying rows of object to show in the heatmap.
columns	A vector specifying the subset of columns in object to show as columns in the heatmap. Also specifies the column order if cluster_cols=FALSE and order_columns_by=NULL. By default, all columns are used.

exprs_values
: A string or integer scalar indicating which assay of object should be used as expression values for colouring in the heatmap.

center
: A logical scalar indicating whether each row should have its mean expression centered at zero prior to plotting.

zlim
: A numeric vector of length 2, specifying the upper and lower bounds for the expression values. This winsorizes the expression matrix prior to plotting (but after centering, if center=TRUE). If NULL, it defaults to the range of the expression matrix.

symmetric
: A logical scalar specifying whether the default zlim should be symmetric around zero. If TRUE, the maximum absolute value of zlim will be computed and multiplied by c(-1,1) to redefine zlim.

color
: A vector of colours specifying the palette to use for mapping expression values to colours. This defaults to the default setting in pheatmap.

colour_columns_by
: A list of values specifying how the columns should be annotated with colours. Each entry of the list can be any acceptable input to the by argument in ?retrieveCellInfo. A character vector can also be supplied and will be treated as a list of strings.

order_columns_by
: A list of values specifying how the columns should be ordered. Each entry of the list can be any acceptable input to the by argument in ?retrieveCellInfo. A character vector can also be supplied and will be treated as a list of strings. This argument is automatically appended to colour_columns_by.

by_exprs_values
: A string or integer scalar specifying which assay to obtain expression values from, for colouring of column-level data - see the exprs_values argument in ?retrieveCellInfo.

by_show_single
: Deprecated and ignored.

show_colnames, cluster_cols, ...
: Additional arguments to pass to pheatmap.

Details

Setting center=TRUE is useful for examining log-fold changes of each cell's expression profile from the average across all cells. This avoids issues with the entire row appearing a certain colour because the gene is highly/lowly expressed across all cells.

Setting zlim preserves the dynamic range of colours in the presence of outliers. Otherwise, the plot may be dominated by a few genes, which will "flatten" the observed colours for the rest of the heatmap.

Setting order_columns_by is useful for automatically ordering the heatmap by one or more factors of interest, e.g., cluster identity. This the need to set colour_columns_by, cluster_cols and columns to achieve the same effect.

Value

A heatmap is produced on the current graphics device. The output of pheatmap is invisibly returned.

Author(s)

Aaron Lun

See Also

pheatmap

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

plotHeatmap(example_sce, features=rownames(example_sce)[1:10])

plotHeatmap(example_sce, features=rownames(example_sce)[1:10],
    center=TRUE, symmetric=TRUE)

plotHeatmap(example_sce, features=rownames(example_sce)[1:10],
    colour_columns_by=c("Mutation_Status", "Cell_Cycle"))
```

---

plotHighestExprs *Plot the highest expressing features*

Description

Plot the features with the highest average expression across all cells, along with their expression in each individual cell.

Usage

```
plotHighestExprs(
  object,
  n = 50,
  colour_cells_by = NULL,
  controls = NULL,
  drop_features = NULL,
  exprs_values = "counts",
  by_exprs_values = exprs_values,
  by_show_single = TRUE,
  feature_names_to_plot = NULL,
  as_percentage = TRUE
)
```

Arguments

object	A SingleCellExperiment object.
n	A numeric scalar specifying the number of the most expressed features to show.
colour_cells_by	Specification of a column metadata field or a feature to colour by, see ?retrieveCellInfo for possible values.
controls	Deprecated and ignored.
drop_features	A character, logical or numeric vector indicating which features (e.g. genes, transcripts) to drop when producing the plot. For example, spike-in transcripts might be dropped to examine the contribution from endogenous genes.

exprs_values A integer scalar or string specifying the assay to obtain expression values from.

by_exprs_values
A string or integer scalar specifying which assay to obtain expression values from, for use in colouring - see ?retrieveCellInfo for details.

by_show_single Deprecated and ignored. Default is NULL, in which case rownames(object) are used.

feature_names_to_plot
String specifying which row-level metadata column contains the feature names. Alternatively, an AsIs-wrapped vector or a data.frame, see ?retrieveFeatureInfo for possible values.

as_percentage logical scalar indicating whether percentages should be plotted. If FALSE, the raw exprs_values are shown instead.

Details

This function will plot the percentage of counts accounted for by the top n most highly expressed features across the dataset. Each row on the plot corresponds to a feature and is sorted by average expression (denoted by the point). The distribution of expression across all cells is shown as tick marks for each feature. These ticks can be coloured according to cell-level metadata, as specified by colour_cells_by.

Value

A ggplot object.

Examples

```
example_sce <- mockSCE()
colData(example_sce) <- cbind(colData(example_sce),
     perCellQCMetrics(example_sce))

plotHighestExprs(example_sce, colour_cells_by="detected")
plotHighestExprs(example_sce, colour_cells_by="Mutation_Status")
```

---

plotPlatePosition *Plot cells in plate positions*

Description

Plots cells in their position on a plate, coloured by metadata variables or feature expression values from a SingleCellExperiment object.

Usage

```
plotPlatePosition(
  object,
  plate_position = NULL,
  colour_by = NULL,
  size_by = NULL,
  shape_by = NULL,
```

```
  by_exprs_values = "logcounts",
  by_show_single = FALSE,
  add_legend = TRUE,
  theme_size = 24,
  point_alpha = 0.6,
  point_size = 24,
  other_fields = list()
)
```

Arguments

object	A SingleCellExperiment object.
plate_position	A character vector specifying the plate position for each cell (e.g., A01, B12, and so on, where letter indicates row and number indicates column). If NULL, the function will attempt to extract this from object$plate_position. Alternatively, a list of two factors ("row" and "column") can be supplied, specifying the row (capital letters) and column (integer) for each cell in object.
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.
by_show_single	Deprecated and ignored.
add_legend	Logical scalar specifying whether a legend should be shown.
theme_size	Numeric scalar, see ?"scater-plot-args" for details.
point_alpha	Numeric scalar specifying the transparency of the points, see ?"scater-plot-args" for details.
point_size	Numeric scalar specifying the size of the points, see ?"scater-plot-args" for details.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.

Details

This function expects plate positions to be given in a charcter format where a letter indicates the row on the plate and a numeric value indicates the column. Each cell has a plate position such as "A01", "B12", "K24" and so on. From these plate positions, the row is extracted as the letter, and the column as the numeric part. Alternatively, the row and column identities can be directly supplied by setting plate_position as a list of two factors.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

## define plate positions
example_sce$plate_position <- paste0(
    rep(LETTERS[1:5], each = 8),
    rep(formatC(1:8, width = 2, flag = "0"), 5)
)

## plot plate positions
plotPlatePosition(example_sce, colour_by = "Mutation_Status")

plotPlatePosition(example_sce, shape_by = "Treatment",
    colour_by = "Gene_0004")

plotPlatePosition(example_sce, shape_by = "Treatment", size_by = "Gene_0001",
    colour_by = "Cell_Cycle")
```

`plotReducedDim` *Plot reduced dimensions*

Description

Plot cell-level reduced dimension results stored in a SingleCellExperiment object.

Usage

```
plotReducedDim(
  object,
  dimred,
  use_dimred = NULL,
  ncomponents = 2,
  percentVar = NULL,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = "logcounts",
  by_show_single = NULL,
  text_by = NULL,
  text_size = 5,
  text_colour = "black",
  label_format = c("%s %i", " (%i%%)"),
  other_fields = list(),
  ...
)
```

Arguments

`object`	A SingleCellExperiment object.

dimred	A string or integer scalar indicating the reduced dimension result in reducedDims(object) to plot.
use_dimred	Deprecated, same as dimred.
ncomponents	A numeric scalar indicating the number of dimensions to plot, starting from the first dimension. Alternatively, a numeric vector specifying the dimensions to be plotted.
percentVar	A numeric vector giving the proportion of variance in expression explained by each reduced dimension. Only expected to be used in PCA settings, e.g., in the plotPCA function.
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.
size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.
by_show_single	Deprecated and ignored.
text_by	String specifying the column metadata field with which to add text labels on the plot. This must refer to a categorical field, i.e., coercible into a factor. Alternatively, an AsIs vector or data.frame, see ?retrieveCellInfo.
text_size	Numeric scalar specifying the size of added text.
text_colour	String specifying the colour of the added text.
label_format	Character vector of length 2 containing format strings to use for the axis labels. The first string expects a string containing the result type (e.g., "PCA") and an integer containing the component number, while the second string shows the rounded percentage of variance explained and is only relevant when this information is provided in object.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

If ncomponents is a scalar equal to 2, a scatterplot of the first two dimensions is produced. If ncomponents is greater than 2, a pairs plots for the top dimensions is produced.

Alternatively, if ncomponents is a vector of length 2, a scatterplot of the two specified dimensions is produced. If it is of length greater than 2, a pairs plot is produced containing all pairwise plots between the specified dimensions.

The text_by option will add factor levels as labels onto the plot, placed at the median coordinate across all points in that level. This is useful for annotating position-related metadata (e.g., clusters) when there are too many levels to distinguish by colour. It is only available for scatterplots.

Value

A ggplot object

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

example_sce <- runPCA(example_sce, ncomponents=5)
plotReducedDim(example_sce, "PCA")
plotReducedDim(example_sce, "PCA", colour_by="Cell_Cycle")
plotReducedDim(example_sce, "PCA", colour_by="Gene_0001")

plotReducedDim(example_sce, "PCA", ncomponents=5)
plotReducedDim(example_sce, "PCA", ncomponents=5, colour_by="Cell_Cycle",
    shape_by="Treatment")
```

---

plotRLE *Plot relative log expression*

Description

Produce a relative log expression (RLE) plot of one or more transformations of cell expression values.

Usage

```
plotRLE(
  object,
  exprs_values = "logcounts",
  exprs_logged = TRUE,
  style = "minimal",
  legend = TRUE,
  ordering = NULL,
  colour_by = NULL,
  by_exprs_values = exprs_values,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
exprs_values	A string or integer scalar specifying the expression matrix in object to use.
exprs_logged	A logical scalar indicating whether the expression matrix is already log-transformed. If not, a log2-transformation (+1) will be performed prior to plotting.
style	String defining the boxplot style to use, either "minimal" (default) or "full"; see Details.
legend	Logical scalar specifying whether a legend should be shown.
ordering	A vector specifying the ordering of cells in the RLE plot. This can be useful for arranging cells by experimental conditions or batches.

colour_by — Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.

by_exprs_values — A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.

... — further arguments passed to geom_boxplot when style="full".

Details

Relative log expression (RLE) plots are a powerful tool for visualising unwanted variation in high dimensional data. These plots were originally devised for gene expression data from microarrays but can also be used on single-cell expression data. RLE plots are particularly useful for assessing whether a procedure aimed at removing unwanted variation (e.g., scaling normalisation) has been successful.

If style is "full", the usual **ggplot2** boxplot is created for each cell. Here, the box shows the inter-quartile range and whiskers extend no more than 1.5 times the IQR from the hinge (the 25th or 75th percentile). Data beyond the whiskers are called outliers and are plotted individually. The median (50th percentile) is shown with a white bar. This approach is detailed and flexible, but can take a long time to plot for large datasets.

If style is "minimal", a Tufte-style boxplot is created for each cell. Here, the median is shown with a circle, the IQR in a grey line, and "whiskers" (as defined above) for the plots are shown with coloured lines. No outliers are shown for this plot style. This approach is more succinct and faster for large numbers of cells.

Value

A ggplot object

Author(s)

Davis McCarthy, with modifications by Aaron Lun

References

Gandolfo LC, Speed TP (2017). RLE plots: visualising unwanted variation in high dimensional data. *arXiv*.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

plotRLE(example_sce, colour_by = "Mutation_Status", style = "minimal")

plotRLE(example_sce, colour_by = "Mutation_Status", style = "full",
       outlier.alpha = 0.1, outlier.shape = 3, outlier.size = 0)
```

plotRowData *Plot row metadata*

Description

Plot row-level (i.e., gene) metadata from a SingleCellExperiment object.

Usage

```
plotRowData(
  object,
  y,
  x = NULL,
  colour_by = NULL,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = "logcounts",
  by_show_single = FALSE,
  other_fields = list(),
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information.
y	String specifying the column-level metadata field to show on the y-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo.
x	String specifying the column-level metadata to show on the x-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo. If NULL, nothing is shown on the x-axis.
colour_by	Specification of a row metadata field or a cell to colour by, see ?retrieveFeatureInfo for possible values.
shape_by	Specification of a row metadata field or a cell to shape by, see ?retrieveFeatureInfo for possible values.
size_by	Specification of a row metadata field or a cell to size by, see ?retrieveFeatureInfo for possible values.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see ?retrieveFeatureInfo for details.
by_show_single	Deprecated and ignored.
other_fields	Additional feature-based fields to include in the data.frame, see ?"scater-plot-args" for details.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

If y is continuous and x=NULL, a violin plot is generated. If x is categorical, a grouped violin plot will be generated, with one violin for each level of x. If x is continuous, a scatter plot will be generated.

If y is categorical and x is continuous, horizontal violin plots will be generated. If x is missing or categorical, rectangle plots will be generated where the area of a rectangle is proportional to the number of points for a combination of factors.

Value

A ggplot object.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
rowData(example_sce) <- cbind(rowData(example_sce),
    perFeatureQCMetrics(example_sce))

plotRowData(example_sce, y="detected", x="mean") +
    scale_x_log10()
```

---

plotScater *Plot an overview of expression for each cell*

Description

Plot the relative proportion of the library size that is accounted for by the most highly expressed features for each cell in a SingleCellExperiment object.

Usage

```
plotScater(
  x,
  nfeatures = 500,
  exprs_values = "counts",
  colour_by = NULL,
  by_exprs_values = exprs_values,
  by_show_single = FALSE,
  block1 = NULL,
  block2 = NULL,
  ncol = 3,
  line_width = 1.5,
  theme_size = 10
)
```

Arguments

x	A SingleCellExperiment object.
nfeatures	Numeric scalar indicating the number of top-expressed features to show n the plot.
exprs_values	String or integer scalar indicating which assay of object should be used to obtain the expression values for this plot.
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values. The curve for each cell will be coloured according to this specification.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the exprs_values argument in ?retrieveCellInfo.
by_show_single	Deprecated and ignored.
block1	String specifying the column-level metadata field by which to separate the cells into separate panels in the plot. Alternatively, an AsIs vector or data.frame, see ?retrieveCellInfo. Default is NULL, in which case there is no blocking.
block2	Same as block1, providing another level of blocking.
ncol	Number of columns to use for facet_wrap if only one block is defined.
line_width	Numeric scalar specifying the line width.
theme_size	Numeric scalar specifying the font size to use for the plotting theme.

Details

For each cell, the features are ordered from most-expressed to least-expressed. The cumulative proportion of the total expression for the cell is computed across the top nfeatures features. These plots can flag cells with a very high proportion of the library coming from a small number of features; such cells are likely to be problematic for downstream analyses.

Using the colour and blocking arguments can flag overall differences in cells under different experimental conditions or affected by different batch and other variables. If only one of block1 and block2 are specified, each panel corresponds to a separate level of the specified blocking factor. If both are specified, each panel corresponds to a combination of levels.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
example_sce <- mockSCE()
plotScater(example_sce)
plotScater(example_sce, exprs_values = "counts", colour_by = "Cell_Cycle")
plotScater(example_sce, block1 = "Treatment", colour_by = "Cell_Cycle")
```

quickPerCellQC *Quick cell-level QC*

Description

A convenient utility that identifies low-quality cells based on frequently used QC metrics.

Usage

```
quickPerCellQC(
  df,
  lib_size = "sum",
  n_features = "detected",
  percent_subsets = NULL,
  ...
)
```

Arguments

df	A DataFrame containing per-cell QC statistics, as computed by perCellQCMetrics.
lib_size	String specifying the column of df containing the library size for each cell.
n_features	String specifying the column of df containing the number of detected features per cell.
percent_subsets	String specifying the column(s) of df containing the percentage of counts in subsets of "control features".
...	Further arguments to pass to isOutlier.

Details

This function simply calls isOutlier on the various QC metrics in df.

- For lib_size, small outliers are detected on the log-scale to remove cells with low library sizes.
- For n_features, small outliers are detected on the log-scale to remove cells with few detected features.
- For each field in percent_subsets, large outliers are detected on the original scale. This aims to remove cells with high spike-in or mitochondrial content.

Users can change the number of MADs used to define an outlier or specify batches by passing appropriate arguments to ....

Value

A DataFrame with one row per cell and containing columns of logical vectors. Each column specifies a reason for why a cell was considered to be low quality, with the final discard column indicating whether the cell should be discarded.

Author(s)

Aaron Lun

See Also

perCellQCMetrics, for calculation of these metrics.

isOutlier, to identify outliers with a MAD-based approach.

Examples

```
example_sce <- mockSCE()
df <- perCellQCMetrics(example_sce, subsets=list(Mito=1:100))

discarded <- quickPerCellQC(df, percent_subsets=c(
    "subsets_Mito_percent", "altexps_Spikes_percent"))
colSums(as.data.frame(discarded))
```

---

readSparseCounts *Read sparse count matrix from file*

Description

Reads a sparse count matrix from file containing a dense tabular format.

Usage

```
readSparseCounts(
  file,
  sep = "\t",
  quote = NULL,
  comment.char = "",
  row.names = TRUE,
  col.names = TRUE,
  ignore.row = 0L,
  skip.row = 0L,
  ignore.col = 0L,
  skip.col = 0L,
  chunk = 1000L
)
```

Arguments

file	A string containing a file path to a count table, or a connection object opened in read-only text mode.
sep	A string specifying the delimiter between fields in file.
quote	A string specifying the quote character, e.g., in column or row names.
comment.char	A string specifying the comment character after which values are ignored.
row.names	A logical scalar specifying whether row names are present.
col.names	A logical scalar specifying whether column names are present.
ignore.row	An integer scalar specifying the number of rows to ignore at the start of the file, *before* the column names.

skip.row	An integer scalar specifying the number of rows to ignore at the start of the file, *after* the column names.
ignore.col	An integer scalar specifying the number of columns to ignore at the start of the file, *before* the column names.
skip.col	An integer scalar specifying the number of columns to ignore at the start of the file, *after* the column names.
chunk	A integer scalar indicating the chunk size to use, i.e., number of rows to read at any one time.

Details

This function provides a convenient method for reading dense arrays from flat files into a sparse matrix in memory. Memory usage can be further improved by setting chunk to a smaller positive value.

The ignore.* and skip.* parameters allow irrelevant rows or columns to be skipped. Note that the distinction between the two parameters is only relevant when row.names=FALSE (for skipping/ignoring columns) or col.names=FALSE (for rows).

Value

A dgCMatrix containing double-precision values (usually counts) for each row (gene) and column (cell).

Author(s)

Aaron Lun

See Also

read.table, readMM

Examples

```
outfile <- tempfile()
write.table(data.frame(A=1:5, B=0, C=0:4, row.names=letters[1:5]),
    file=outfile, col.names=NA, sep="\t", quote=FALSE)

readSparseCounts(outfile)
```

Reduced dimension plots

Plot specific reduced dimensions

Description

Wrapper functions to create plots for specific types of reduced dimension results in a SingleCellExperiment object.

Usage

```
plotPCASCE(object, ..., rerun = FALSE, ncomponents = 2, run_args = list())

plotTSNE(object, ..., rerun = FALSE, ncomponents = 2, run_args = list())

plotUMAP(object, ..., rerun = FALSE, ncomponents = 2, run_args = list())

plotDiffusionMap(
  object,
  ...,
  rerun = FALSE,
  ncomponents = 2,
  run_args = list()
)

plotMDS(object, ..., rerun = FALSE, ncomponents = 2, run_args = list())

## S4 method for signature 'SingleCellExperiment'
plotPCA(object, ..., rerun = FALSE, ncomponents = 2, run_args = list())
```

Arguments

`object`	A SingleCellExperiment object.
`...`	Additional arguments to pass to `plotReducedDim`.
`rerun`	Logical, should the reduced dimensions be recomputed even if `object` contains an appropriately named set of results in the `reducedDims` slot?
`ncomponents`	Numeric scalar indicating the number of dimensions components to (calculate and) plot. This can also be a numeric vector, see ?`plotReducedDim` for details.
`run_args`	Arguments to pass to `runPCA`, `runTSNE`, etc.

Details

Each function is a convenient wrapper around `plotReducedDim` that searches the `reducedDims` slot for an appropriately named dimensionality reduction result:

- `"PCA"` for `plotPCA`
- `"TSNE"` for `plotTSNE`
- `"DiffusionMap"` for `plotDiffusionMap`
- `"MDS"` for `"plotMDS"`
- `"UMAP"` for `"plotUMAP"`

Its only purpose is to streamline workflows to avoid the need to specify the `dimred` argument.

Previous versions of these functions would recompute the dimensionality reduction results if they were not already present. This has been deprecated in favour of users explicitly calling the relevant `run*` function, to avoid uncertainties about what was actually being plotted.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

See Also

runPCA, runDiffusionMap, runTSNE, runMDS, and runUMAP, for the functions that actually perform the calculations.

plotReducedDim, for the underlying plotting function.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
example_sce <- runPCA(example_sce)

## Examples plotting PC1 and PC2
plotPCA(example_sce)
plotPCA(example_sce, colour_by = "Cell_Cycle")
plotPCA(example_sce, colour_by = "Cell_Cycle", shape_by = "Treatment")
plotPCA(example_sce, colour_by = "Cell_Cycle", shape_by = "Treatment",
    size_by = "Mutation_Status")

## Force legend to appear for shape:
example_subset <- example_sce[, example_sce$Treatment == "treat1"]
plotPCA(example_subset, colour_by = "Cell_Cycle", shape_by = "Treatment",
    by_show_single = TRUE)

## Examples plotting more than 2 PCs
plotPCA(example_sce, ncomponents = 4, colour_by = "Treatment",
    shape_by = "Mutation_Status")

## Same for TSNE:
example_sce <- runTSNE(example_sce)
plotTSNE(example_sce, run_args=list(perplexity = 10))

## Same for DiffusionMaps:
example_sce <- runDiffusionMap(example_sce)
plotDiffusionMap(example_sce)

## Same for MDS plots:
example_sce <- runMDS(example_sce)
plotMDS(example_sce)
```

---

retrieveCellInfo *Cell-based data retrieval*

Description

Retrieves a per-cell (meta)data field from a SingleCellExperiment based on a single keyword, typically for use in visualization functions.

Usage

```
retrieveCellInfo(
  x,
  by,
  search = c("colData", "assays", "altExps"),
  exprs_values = "logcounts"
)
```

Arguments

x	A SingleCellExperiment object.
by	A string specifying the field to extract (see Details). Alternatively, a data.frame, DataFrame or an AsIs vector.
search	Character vector specifying the types of data or metadata to use.
exprs_values	String or integer scalar specifying the assay from which expression values should be extracted.

Details

Given an AsIs-wrapped vector in by, this function will directly return the vector values as value, while name is set to an empty string. For data.frame or DataFrame instances with a single column, this function will return the vector from that column as value and the column name as name. This allows downstream visualization functions to accommodate arbitrary inputs for adjusting aesthetics.

Given a character string in by, this function will:

1. Search colData for a column named by, and return the corresponding field as the output value. We do not consider nested elements within the colData.
2. Search assay(x,exprs_values) for a row named by, and return the expression vector for this feature as the output value.
3. Search each alternative experiment in altExps(x) for a row names by, and return the expression vector for this feature at exprs_values as the output value.

Any match will cause the function to return without considering later possibilities. The search can be modified by changing the presence and ordering of elements in search.

If there is a name clash that results in retrieval of an unintended field, users should explicitly set by to a data.frame, DataFrame or AsIs-wrapped vector containing the desired values. Developers can also consider setting search to control the fields that are returned.

Value

A list containing name, a string with the name of the extracted field (usually identically to by); and value, a vector of length equal to ncol(x) containing per-cell (meta)data values. If by=NULL or was not found in x, both name and value are set to NULL.

Author(s)

Aaron Lun

See Also

plotColData, plotReducedDim, plotExpression, plotPlatePosition, and most other cell-based plotting functions.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)

retrieveCellInfo(example_sce, "Cell_Cycle")
retrieveCellInfo(example_sce, "Gene_0001")

arbitrary.field <- rnorm(ncol(example_sce))
retrieveCellInfo(example_sce, I(arbitrary.field))
retrieveCellInfo(example_sce, data.frame(stuff=arbitrary.field))
```

---

retrieveFeatureInfo *Feature-based data retrieval*

Description

Retrieves a per-feature (meta)data field from a SingleCellExperiment based on a single keyword, typically for use in visualization functions.

Usage

```
retrieveFeatureInfo(
  x,
  by,
  search = c("rowData", "assays"),
  exprs_values = "logcounts"
)
```

Arguments

x	A SingleCellExperiment object.
by	A string specifying the field to extract (see Details). Alternatively, a data.frame, DataFrame or an AsIs vector.
search	Character vector specifying the types of data or metadata to use.
exprs_values	String or integer scalar specifying the assay from which expression values should be extracted.

Details

Given a AsIs-wrapped vector in by, this function will directly return the vector values as value, while name is set to an empty string. For data.frame or DataFrame instances with a single column, this function will return the vector from that column as value and the column name as name. This allows downstream visualization functions to accommodate arbitrary inputs for adjusting aesthetics.

Given a character string in by, this function will:

1. Search rowData for a column named by, and return the corresponding field as the output value. We do not consider nested elements within the rowData.
2. Search assay(x,exprs_values) for a column named by, and return the expression vector for this feature as the output value.

Any match will cause the function to return without considering later possibilities. The search can be modified by changing the presence and ordering of elements in `search`.

If there is a name clash that results in retrieval of an unintended field, users should explicitly set `by` to a data.frame, DataFrame or AsIs-wrapped vector containing the desired values. Developers can also consider setting `search` to control the fields that are returned.

Value

A list containing `name`, a string with the name of the extracted field (usually identically to `by`); and `value`, a vector of length equal to `ncol(x)` containing per-feature (meta)data values. If `by=NULL` or was not found in `x`, both `name` and `value` are set to `NULL`.

Author(s)

Aaron Lun

See Also

`plotRowData` and other feature-based plotting functions.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
rowData(example_sce)$blah <- sample(LETTERS,
    nrow(example_sce), replace=TRUE)

str(retrieveFeatureInfo(example_sce, "blah"))
str(retrieveFeatureInfo(example_sce, "Cell_001"))

arbitrary.field <- rnorm(nrow(example_sce))
str(retrieveFeatureInfo(example_sce, I(arbitrary.field)))
str(retrieveFeatureInfo(example_sce, data.frame(stuff=arbitrary.field)))
```

---

runColDataPCA *Perform PCA on column metadata*

Description

Perform a principal components analysis (PCA) on cells, based on the column metadata in a SingleCellExperiment object.

Usage

```
runColDataPCA(
  x,
  ncomponents = 2,
  variables = NULL,
  selected_variables = NULL,
  scale = TRUE,
  scale_features = NULL,
```

```
  outliers = FALSE,
  detect_outliers = NULL,
  BSPARAM = ExactParam(),
  BPPARAM = SerialParam(),
  name = "PCA_coldata"
)
```

Arguments

x	A SingleCellExperiment object.
ncomponents	Numeric scalar indicating the number of principal components to obtain.
variables	List of strings or a character vector indicating which variables in colData(x) to use. If a list, each entry can also be an AsIs vector or a data.frame, as described in ?retrieveCellInfo.
selected_variables	Deprecated, same as variables.
scale	Logical scalar, should the expression values be standardised so that each feature has unit variance? This will also remove features with standard deviations below 1e-8.
scale_features	Deprecated, same as scale.
outliers	Logical indicating whether outliers should be detected based on PCA coordinates.
detect_outliers	Deprecated, same as outliers.
BSPARAM	A BiocSingularParam object specifying which algorithm should be used to perform the PCA.
BPPARAM	A BiocParallelParam object specifying whether the PCA should be parallelized.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

This function performs PCA on variables from the column-level metadata instead of the gene expression matrix. Doing so can be occasionally useful when other forms of experimental data are stored in the colData, e.g., protein intensities from FACs or other cell-specific phenotypic information.

This function is particularly useful for identifying low-quality cells based on QC metrics with outliers=TRUE. This uses an "outlyingness" measure computed by adjOutlyingness in the **robustbase** package. Outliers are defined those cells with outlyingness values more than 5 MADs above the median, using isOutlier.

Value

A SingleCellExperiment object containing the first ncomponent principal coordinates for each cell. By default, these are stored in the "PCA_coldata" entry of the reducedDims slot. The proportion of variance explained by each PC is stored as a numeric vector in the "percentVar" attribute.

If outliers=TRUE, the output colData will also contain a logical outlier field. This specifies the cells that correspond to the identified outliers.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

runPCA, for the corresponding method operating on expression data.

Examples

```
example_sce <- mockSCE()
qc.df <- perCellQCMetrics(example_sce, subset=list(Mito=1:10))
colData(example_sce) <- cbind(colData(example_sce), qc.df)

# Can supply names of colData variables to 'variables',
# as well as AsIs-wrapped vectors of interest.
example_sce <- runColDataPCA(example_sce, variables=list(
    "sum", "detected", "subsets_Mito_percent", "altexps_Spikes_percent"
))
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

---

runMultiUMAP *Multi-modal UMAP*

Description

Perform UMAP with multiple input matrices by intersecting their simplicial sets. Typically used to combine results from multiple data modalities into a single embedding.

Usage

```
runMultiUMAP(inputs, ..., metric = "euclidean")
```

Arguments

inputs	A list of numeric matrices where each row is a cell and each column is some dimension/variable. For gene expression data, this is usually the matrix of PC coordinates.
...	Further arguments to pass to umap.
metric	String specifying the type of distance to use.

Details

This is simply a convenience wrapper around umap for multi-modal analysis. All modes use the distance metric of metric to construct the simplicial sets *within* each mode. Comparisons across modes are then performed after intersecting the sets to obtain a single graph.

Value

A numeric matrix containing the low-dimensional UMAP embedding.

Author(s)

Aaron Lun

See Also

runUMAP, for the more straightforward application of UMAP.

Examples

```
# Mocking up a gene expression + ADT dataset:
exprs_sce <- mockSCE()
exprs_sce <- logNormCounts(exprs_sce)
exprs_sce <- runPCA(exprs_sce)

adt_sce <- mockSCE(ngenes=20)
adt_sce <- logNormCounts(adt_sce)
altExp(exprs_sce, "ADT") <- adt_sce

# Running a multimodal analysis using PCs for expression
# and log-counts for the ADTs:
output <- runMultiUMAP(
    list(
        reducedDim(exprs_sce, "PCA"),
        t(logcounts(altExp(exprs_sce, "ADT")))
    )
)

reducedDim(exprs_sce, "combinedUMAP") <- output
plotReducedDim(exprs_sce, "combinedUMAP")
```

---

scater-plot-args *General visualization parameters*

Description

scater functions that plot points share a number of visualization parameters, which are described on this page.

Aesthetic parameters

`add_legend`**:** Logical scalar, specifying whether a legend should be shown. Defaults to TRUE.

`theme_size`**:** Integer scalar, specifying the font size. Defaults to 10.

`point_alpha`**:** Numeric scalar in [0, 1], specifying the transparency. Defaults to 0.6.

`point_size`**:** Numeric scalar, specifying the size of the points. Defaults to `NULL`.

`jitter_type`**:** String to define how points are to be jittered in a violin plot. This is either with random jitter on the x-axis (`"jitter"`) or in a "beeswarm" style (if `"swarm"`, default). The latter usually looks more attractive, but for datasets with a large number of cells, or for dense plots, the jitter option may work better.

Distributional calculations

`show_median`: Logical, should the median of the distribution be shown for violin plots? Defaults to `FALSE`.

`show_violin`: Logical, should the outline of a violin plot be shown? Defaults to `TRUE`.

`show_smooth`: Logical, should a smoother be fitted to a scatter plot? Defaults to `FALSE`.

`show_se`: Logical, should standard errors for the fitted line be shown on a scatter plot when `show_smooth=TRUE`? Defaults to `TRUE`.

Miscellaneous fields

Addititional fields can be added to the data.frame passed to ggplot by setting the `other_fields` argument. This allows users to easily incorporate additional metadata for use in further **ggplot** operations.

The `other_fields` argument should be character vector where each string is passed to `retrieveCellInfo` (for cell-based plots) or `retrieveFeatureInfo` (for feature-based plots). Alternatively, `other_fields` can be a named list where each element is of any type accepted by `retrieveCellInfo` or `retrieveFeatureInfo`. This includes AsIs-wrapped vectors, data.frames or DataFrames.

Each additional column of the output data.frame will be named according to the `name` returned by `retrieveCellInfo` or `retrieveFeatureInfo`. If these clash with inbuilt names (e.g., `X`, `Y`, `colour_by`), a warning will be raised and the additional column will not be added to avoid overwriting an existing column.

See Also

`plotColData`, `plotRowData`, `plotReducedDim`, `plotExpression`, `plotPlatePosition`, and most other plotting functions.

---

scater-red-dim-args *Dimensionality reduction options*

Description

An overview of the common options for dimensionality reduction methods in **scater**. The following sections consider an input `x` to the various `run*` methods, where `x` can be a numeric matrix or a SingleCellExperiment.

Feature selection

This section is relevant if `x` is a numeric matrix of (log-)expression values with features in rows and cells in columns; or if `x` is a SingleCellExperiment and `dimred=NULL`. In the latter, the expression values are obtained from the assay specified by `exprs_values`.

The `subset_row` argument specifies the features to use in a dimensionality reduction algorithm. This can be set to any user-defined vector containing, e.g., highly variable features or genes in a pathway of interest. It can be a character vector of row names, an integer vector of row indices or a logical vector.

If `subset_row=NULL`, the `ntop` features with the largest variances are used instead. This literally computes the variances from the expression values without considering any mean-variance trend. Note that the value of `ntop` is ignored if `subset_row` is specified.

If `scale=TRUE`, the expression values for each feature are standardized so that their variance is unity. This will also remove features with standard deviations below 1e-8.

Using reduced dimensions

This section is relevant if `x` is a SingleCellExperiment and `dimred` is not `NULL`.

All dimensionality reduction methods can be applied on existing dimensionality reduction results in `x` by setting `dimred`. This is typically used to run non-linear algorithms like t-SNE or UMAP on the PCA results. It may also be desirable in cases where the existing reduced dimensions are computed from *a priori* knowledge (e.g., gene set scores). In such cases, further reduction with PCA could be used to compress the data.

The matrix of existing reduced dimensions is taken from `reducedDims(x,dimred)`. By default, all dimensions are used to compute the second set of reduced dimensions. If `n_dimred` is also specified, only the first `n_dimred` columns are used. Alternatively, `n_dimred` can be an integer vector specifying the column indices of the dimensions to use.

When `dimred` is specified, no additional feature selection or standardization is performed. This means that any settings of `ntop`, `subset_row` and `scale` are ignored.

Transposed inputs

This section is relevant if `x` is a numeric matrix and `transposed=TRUE`, such that cells are the rows and the various dimensions are the columns.

Here, the aim is to allow users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., `ntop`, `subset_row` and `scale` are ignored.

Alternative experiments

This section is relevant if `x` is a SingleCellExperiment and `altexp` is not `NULL`.

If `altexp` is specified, the method is run on data from an alternative SummarizedExperiment nested within `x`. This is useful for performing dimensionality reduction on other features stored in `altExp(x,altexp)`, e.g., antibody tags.

Setting `altexp` with `exprs_values` will use the specified assay from the alternative SummarizedExperiment. If the alternative is a SingleCellExperiment, setting `dimred` will use the specified dimensionality reduction results from the alternative. This option will also interact as expected with `n_dimred`.

Note that the output is still stored in the `reducedDims` of the output SingleCellExperiment. It is advisable to use a different `name` to distinguish this from PCA results obtained from the main experiment's assay values.

Author(s)

Aaron Lun

See Also

These arguments are used throughout `runPCA`, `runTSNE`, `runUMAP`, `runMDS` and `runDiffusionMap`.

SCESet *The "Single Cell Expression Set" (SCESet) class*

Description

S4 class and the main class used by scater to hold single cell expression data. SCESet extends the basic Bioconductor ExpressionSet class.

Details

This class is initialized from a matrix of expression values.

Methods that operate on SCESet objects constitute the basic scater workflow.

Slots

`logExprsOffset`: Scalar of class `"numeric"`, providing an offset applied to expression data in the 'exprs' slot when undergoing log2-transformation to avoid trying to take logs of zero.

`lowerDetectionLimit`: Scalar of class `"numeric"`, giving the lower limit for an expression value to be classified as "expressed".

`cellPairwiseDistances`: Matrix of class `"numeric"`, containing pairwise distances between cells.

`featurePairwiseDistances`: Matrix of class `"numeric"`, containing pairwise distances between features.

`reducedDimension`: Matrix of class `"numeric"`, containing reduced-dimension coordinates for cells (generated, for example, by PCA).

`bootstraps`: Array of class `"numeric"` that can contain bootstrap estimates of the expression or count values.

`sc3`: List containing results from consensus clustering from the SC3 package.

`featureControlInfo`: Data frame of class `"AnnotatedDataFrame"` that can contain information/metadata about sets of control features defined for the `SCESet` object. bootstrap estimates of the expression or count values.

References

Thanks to the Monocle package (github.com/cole-trapnell-lab/monocle-release/) for their CellDataSet class, which provided the inspiration and template for SCESet.

sumCountsAcrossCells *Sum counts across sets of cells*

Description

Sum together expression values (by default, counts) for each set of cells and for each feature.

Usage

```
sumCountsAcrossCells(x, ...)

aggregateAcrossCells(x, ...)

## S4 method for signature 'ANY'
sumCountsAcrossCells(
  x,
  ids,
  subset_row = NULL,
  subset_col = NULL,
  average = FALSE,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
sumCountsAcrossCells(x, ..., exprs_values = "counts")

## S4 method for signature 'SummarizedExperiment'
aggregateAcrossCells(
  x,
  ids,
  ...,
  coldata_merge = NULL,
  use_exprs_values = "counts"
)

## S4 method for signature 'SingleCellExperiment'
aggregateAcrossCells(
  x,
  ids,
  ...,
  subset_row = NULL,
  coldata_merge = NULL,
  use_exprs_values = "counts",
  use_altexps = TRUE
)
```

Arguments

x	For sumCountsAcrossCells, a numeric matrix of counts containing features in rows and cells in columns. Alternatively, a SummarizedExperiment object containing such a count matrix. For aggregateAcrossCells, a SingleCellExperiment or SummarizedExperiment containing a count matrix.
...	For the generics, further arguments to be passed to specific methods. For the sumCountsAcrossCells SummarizedExperiment method, further arguments to be passed to the ANY method. For aggregateAcrossCells, further arguments to be passed to sumCountsAcrossCells.
ids	A factor specifying the set to which each cell in x belongs.

Alternatively, a DataFrame of such vectors or factors, in which case each unique combination of levels defines a set.

`subset_row` An integer, logical or character vector specifying the features to use. Defaults to all features.

For the SingleCellExperiment method, this argument will not affect alternative Experiments, where summation is always performed for all features (or not at all, depending on `use_alt_exps`).

`subset_col` An in teger, logical or character vector specifying the cells to use. Defaults to all cells with non-`NA` entries of `ids`.

`average` Logical scalar indicating whether the average should be computed instead of the sum.

`BPPARAM` A BiocParallelParam object specifying whether summation should be parallelized.

`exprs_values` A string or integer scalar specifying the assay of `x` containing the matrix of counts (or any other expression quantity that can be meaningfully summed).

`coldata_merge` A named list of functions specifying how each column metadata field should be aggregated. For any unspecified field, metadata is retained for the first instance of a cell from each set in `ids`. If `NULL`, the first instance is retained for all fields.

`use_exprs_values`
A character or integer vector specifying the assay(s) of `x` containing count matrices.

`use_altexps` Logical scalar indicating whether aggregation should be performed for alternative experiments in `x`.

Alternatively, a character vector specifying the names of the alternative experiments to be aggregated.

Details

This function provides a convenient method for aggregating counts across multiple columns for each feature. A typical application would be to sum counts across all cells in each cluster to obtain "pseudo-bulk" samples for further analysis.

The behaviour of this function is equivalent to that of `colsum`. However, this function can operate on any matrix representation in `object`; can do so in a parallelized manner for large matrices without resorting to block processing; and can natively support combinations of multiple factors in `ids`.

Any `NA` values in `ids` are implicitly ignored and will not be considered during summation. This may be useful, e.g., to remove undesirable cells by setting their entries in `ids` to `NA`. Alternatively, we can explicitly select the cells of interest with `subset_col`.

Setting `average=TRUE` will compute the average in each set rather than the sum. This is particularly useful if `x` contains expression values that have already been normalized in some manner, as computing the average avoids another round of normalization to account for differences in the size of each set.

Value

For `sumCountsAcrossCells` with a factor `ids`, a count matrix is returned with one column per level of `ids`. For each feature, counts for all cells in the same set are summed together. Columns are ordered by `levels(ids)`.

For `sumCountsAcrossCells` with a DataFrame `ids`, a SummarizedExperiment is returned containing a similar count matrix in the first assay. Each column corresponds to a unique combination of

levels in ids and contains the sum of counts for all cells with that combination. The identities of the levels for each column are reported in the colData.

For aggregateAcrossCells, a SummarizedExperiment of the same class as x is returned, containing summed matrices generated by sumCountsAcrossCell on all assays specified by use_exprs_values. By default, column metadata is retained for the first instance of a cell from each set in ids, but this behavior can be customized by supplying appropriate functions to coldata_merge. If ids is a DataFrame, the combination of levels corresponding to each column is also reported in the column metadata.

Author(s)

Aaron Lun

Examples

```
example_sce <- mockSCE()
ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)

out <- sumCountsAcrossCells(example_sce, ids)
head(out)

batches <- sample(1:3, ncol(example_sce), replace=TRUE)
out2 <- sumCountsAcrossCells(example_sce,
      DataFrame(label=ids, batch=batches))
out2

# Using another column metadata merge strategy.
example_sce$stuff <- runif(ncol(example_sce))
example_merged <- aggregateAcrossCells(example_sce, ids,
     coldata_merge=list(stuff=sum))
```

---

sumCountsAcrossFeatures

Sum counts across feature sets

Description

Sum together expression values (by default, counts) for each feature set in each cell.

Usage

```
sumCountsAcrossFeatures(x, ...)

## S4 method for signature 'ANY'
sumCountsAcrossFeatures(x, ids, average = FALSE, BPPARAM = SerialParam())

## S4 method for signature 'SummarizedExperiment'
sumCountsAcrossFeatures(x, ..., exprs_values = "counts")

aggregateAcrossFeatures(x, ids, ..., use_exprs_values = "counts")
```

Arguments

x	For sumCountsAcrossFeatures, a numeric matrix of counts containing features in rows and cells in columns. Alternatively, a SummarizedExperiment object containing such a count matrix. For aggregateAcrossFeatures, a SummarizedExperiment containing a count matrix.
...	For the sumCountsAcrossFeatures generic, further arguments to be passed to specific methods. For the SummarizedExperiment method, further arguments to be passed to the ANY method. For aggregateAcrossFeatures, further arguments to be passed to sumCountsAcrossFeatures.
ids	A factor specifying the set to which each feature in x belongs. Alternatively, a list of integer or character vectors, where each vector specifies the indices or names of features in a set.
average	Logical scalar indicating whether the average should be computed instead of the sum.
BPPARAM	A BiocParallelParam object specifying whether summation should be parallelized.
exprs_values	A string or integer scalar specifying the assay of x containing the matrix of counts (or any other expression quantity that can be meaningfully summed).
use_exprs_values	A character or integer vector specifying the assay(s) of x containing count matrices.

Details

This function provides a convenient method for aggregating counts across multiple rows for each cell. Several possible applications are listed below:

- Using a list of genes in ids, we can obtain a summary expression value for all genes in one or more gene sets. This allows the activity of various pathways to be compared across cells.
- Genes with multiple mapping locations in the reference will often manifest as multiple rows with distinct Ensembl/Entrez IDs. These counts can be aggregated into a single feature by setting the shared identifier (usually the gene symbol) as ids.
- It is theoretically possible to aggregate transcript-level counts to gene-level counts with this function. However, it is often better to do so with dedicated functions (e.g., from the **tximport** or **tximeta** packages) that account for differences in length across isoforms.

The behaviour of this function is equivalent to that of rowsum. However, this function can operate on any matrix representation in object, and can do so in a parallelized manner for large matrices without resorting to block processing.

If ids is a factor, any NA values are implicitly ignored and will not be considered or reported. This may be useful, e.g., to remove undesirable feature sets by setting their entries in ids to NA.

Setting average=TRUE will compute the average in each set rather than the sum. This is particularly useful if x contains expression values that have already been normalized in some manner, as computing the average avoids another round of normalization to account for differences in the size of each set.

Value

For sumCountsAcrossFeatures, a count matrix is returned with one row per level of ids. In each cell, counts for all features in the same set are summed together. Rows are ordered according to levels(ids).

For aggregateAcrossFeatures, a SummarizedExperiment of the same class as x is returned, containing summed matrices generated by sumCountsAcrossFeatures on all assays in use_exprs_values. Row metadata is retained for the first instance of a feature from each set in ids.

Author(s)

Aaron Lun

Examples

```
example_sce <- mockSCE()
ids <- sample(LETTERS, nrow(example_sce), replace=TRUE)
out <- sumCountsAcrossFeatures(example_sce, ids)
dimnames(out)
```

---

uniquifyFeatureNames *Make feature names unique*

Description

Combine a user-interpretable feature name (e.g., gene symbol) with a standard identifier that is guaranteed to be unique and valid (e.g., Ensembl) for use as row names.

Usage

```
uniquifyFeatureNames(ID, names)
```

Arguments

ID	A character vector of unique identifiers.
names	A character vector of feature names.

Details

This function will attempt to use names if it is unique. If not, it will append the _ID to any non-unique value of names. Missing names will be replaced entirely by ID.

The output is guaranteed to be unique, assuming that ID is also unique. This can be directly used as the row names of a SingleCellExperiment object.

Value

A character vector of unique-ified feature names.

Author(s)

Aaron Lun

Examples

```
uniquifyFeatureNames(
  ID=paste0("ENSG0000000", 1:5),
  names=c("A", NA, "B", "C", "A")
)
```

---

updateSCESet *Convert an SCESet object to a SingleCellExperiment object*

Description

Convert an SCESet object produced with an older version of the package to a SingleCellExperiment object compatible with the current version.

Usage

```
updateSCESet(object)

toSingleCellExperiment(object)
```

Arguments

object	an SCESet object to be updated

Value

a SingleCellExperiment object

Examples

```
## Not run:
updateSCESet(example_sceset)

## End(Not run)
## Not run:
toSingleCellExperiment(example_sceset)

## End(Not run)
```

Index