

Package ‘scater’

June 23, 2019

Type Package

Version 1.12.2

Date 2019-05-23

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 R (>= 3.6), SingleCellExperiment, ggplot2

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

Suggests BiocStyle, biomaRt, beachmat, cowplot, destiny, knitr, mvoutlier, rmarkdown, Rtsne, uwot, testthat, magrittr, pheatmap, Biobase, limma

VignetteBuilder knitr

LazyData true

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

LinkingTo Rcpp, beachmat

SystemRequirements C++11

RoxygenNote 6.1.1

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_9

git_last_commit 1518dc2

git_last_commit_date 2019-05-23

Date/Publication 2019-06-22

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:

scater-package	3
arrange	3
bootstraps	4
calculateAverage	5
calculateCPM	6
calculateFPKM	7
calculateQCMetrics	8
calculateTPM	12
centreSizeFactors	13
filter	14
getBMFeatureAnnos	15
getExplanatoryPCs	16
getVarianceExplained	17
isOutlier	18
librarySizeFactors	19
multiplot	20
mutate	21
nexprs	21
normalize	22
normalizeCounts	24
norm_exprs	25
plotColData	26
plotExplanatoryPCs	28
plotExplanatoryVariables	29
plotExpression	30
plotExprsFreqVsMean	32
plotExprsVsTxLength	33
plotHeatmap	35
plotHighestExprs	37
plotPlatePosition	38
plotReducedDim	40
plotRLE	42
plotRowData	43
plotScater	45
readSparseCounts	46
Reduced dimension plots	47
rename	49
runDiffusionMap	50
runMDS	51
runPCA,SingleCellExperiment-method	53
runTSNE	55

runUMAP 57
 scater-plot-args 59
 scater-vis-var 59
 SCESet 61
 sc_example_cell_info 62
 sc_example_counts 62
 sumCountsAcrossCells 63
 sumCountsAcrossFeatures 64
 unquifyFeatureNames 65
 updateSCESet 66

Index 67

scater-package *Single-cell analysis toolkit for expression in R*

Description

scater provides a class and numerous functions for the quality control, normalisation and visualisation of single-cell RNA-seq expression data.

Details

In particular, **scater** provides easy generation of quality control metrics and simple functions to visualise quality control metrics and their relationships.

arrange *Arrange columns (cells) of a SingleCellExperiment object (deprecated).*

Description

The SingleCellExperiment returned will have cells ordered by the corresponding variable in colData(object).

Usage

```
arrange(object, ...)

## S4 method for signature 'SingleCellExperiment'
arrange(object, ...)
```

Arguments

- object A SingleCellExperiment object.
- ... Additional arguments to be passed to dplyr::arrange to act on colData(object).

Details

Refer to <https://github.com/sa-lee/plyexperiment> for replacement functionality.

Value

An SingleCellExperiment object.

bootstraps	<i>Accessor and replacement for bootstrap results in a SingleCellExperiment object</i>
------------	--

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

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
bootstraps(example_sce)
```

calculateAverage	<i>Calculate average counts, adjusting for size factors or library size</i>
------------------	---

Description

Calculate average counts per feature, adjusting them to account for normalization due to size factors or library sizes.

Usage

```
calculateAverage(object, exprs_values = "counts",
  use_size_factors = TRUE, subset_row = NULL,
  BPPARAM = SerialParam())
```

```
calcAverage(object, exprs_values = "counts", use_size_factors = TRUE,
  subset_row = NULL, BPPARAM = SerialParam())
```

Arguments

object	A SingleCellExperiment object or count matrix.
exprs_values	A string specifying the assay of object containing the count matrix, if object is a SingleCellExperiment.
use_size_factors	a logical scalar specifying whether the size factors in object should be used to construct effective library sizes.
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.

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 scale of the raw counts.

Assuming that object is a SingleCellExperiment:

- If use_size_factors=TRUE, size factors are automatically extracted from the object. Note that different size factors may be used for features marked as spike-in controls. This is due to the presence of control-specific size factors in object, see [normalizeSCE](#) for more details.
- If use_size_factors=FALSE, all size factors in object are ignored. Size factors are instead computed from the library sizes, using [librarySizeFactors](#).
- If use_size_factors is a numeric vector, it will override the any size factors for non-spike-in features in object. The spike-in size factors will still be used for the spike-in transcripts.

If no size factors are available, they will be computed from the library sizes using [librarySizeFactors](#).

If object is a matrix or matrix-like object, size factors can be supplied by setting use_size_factors to a numeric vector. Otherwise, the sum of counts for each cell is used as the size factor through [librarySizeFactors](#).

Value

Vector of average count values with same length as number of features, or the number of features in `subset_row` if supplied.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  list(counts = sc_example_counts),
  colData = sc_example_cell_info)

## calculate average counts
ave_counts <- calculateAverage(example_sce)
```

calculateCPM	<i>Calculate counts per million (CPM)</i>
--------------	---

Description

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

Usage

```
calculateCPM(object, exprs_values = "counts", use_size_factors = TRUE,
  subset_row = NULL)
```

Arguments

<code>object</code>	A <code>SingleCellExperiment</code> object or count matrix.
<code>exprs_values</code>	A string specifying the assay of object containing the count matrix, if object is a <code>SingleCellExperiment</code> .
<code>use_size_factors</code>	A logical scalar indicating whether size factors in object should be used to compute effective library sizes. If not, all size factors are deleted and library size-based factors are used instead (see librarySizeFactors). Alternatively, a numeric vector containing a size factor for each cell, which is used in place of <code>sizeFactor(object)</code> .
<code>subset_row</code>	A vector specifying the subset of rows of object for which to return a result.

Details

If requested, size factors are used to define the effective library sizes. This is done by scaling all size factors such that the mean scaled size factor is equal to the mean sum of counts across all features. The effective library sizes are then used to in the denominator of the CPM calculation.

Assuming that `object` is a `SingleCellExperiment`:

- If `use_size_factors=TRUE`, size factors are automatically extracted from the object. Note that effective library sizes may be computed differently for features marked as spike-in controls. This is due to the presence of control-specific size factors in object, see [normalizeSCE](#) for more details.

- If `use_size_factors=FALSE`, all size factors in object are ignored. The total count for each cell will be used as the library size for all features (endogenous genes and spike-in controls).
- If `use_size_factors` is a numeric vector, it will override the any size factors for non-spike-in features in object. The spike-in size factors will still be used for the spike-in transcripts.

If no size factors are available, the library sizes will be used.

If object is a matrix or matrix-like object, size factors will only be used if `use_size_factors` is a numeric vector. Otherwise, the sum of counts for each cell is directly used as the library size.

Value

Numeric matrix of CPM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  list(counts = sc_example_counts),
  colData = sc_example_cell_info)

cpm(example_sce) <- calculateCPM(example_sce, use_size_factors = FALSE)
```

calculateFPKM	<i>Calculate fragments per kilobase of exon per million reads mapped (FPKM)</i>
---------------	---

Description

Calculate fragments per kilobase of exon per million reads mapped (FPKM) values for expression from counts for a set of features.

Usage

```
calculateFPKM(object, effective_length, ..., subset_row = NULL)
```

Arguments

<code>object</code>	A <code>SingleCellExperiment</code> object or a numeric matrix of counts.
<code>effective_length</code>	Numeric vector providing the effective length for each feature in object.
<code>...</code>	Further arguments to pass to calculateCPM .
<code>subset_row</code>	A vector specifying the subset of rows of object for which to return a result.

Value

A numeric matrix of FPKM values.

Examples

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  list(counts = sc_example_counts),
  colData = sc_example_cell_info)

eff_len <- runif(nrow(example_sce), 500, 2000)
fout <- calculateFPKM(example_sce, eff_len, use_size_factors = FALSE)

```

calculateQCMetrics	<i>Calculate QC metrics</i>
--------------------	-----------------------------

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

<code>object</code>	A <code>SingleCellExperiment</code> object containing expression values, usually counts.
<code>exprs_values</code>	A string indicating which assays in the object should be used to define expression.
<code>feature_controls</code>	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.
<code>cell_controls</code>	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.
<code>percent_top</code>	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 <code>pct_X_top_Y_features</code> below for more details.
<code>detection_limit</code>	A numeric scalar to be passed to <code>nexprs</code> , specifying the lower detection limit for expression.
<code>use_spikes</code>	A logical scalar indicating whether existing spike-in sets in object should be automatically added to <code>feature_controls</code> , see <code>?isSpike</code> .
<code>compact</code>	A logical scalar indicating whether the metrics should be returned in a compact format as a nested <code>DataFrame</code> .
<code>BPPARAM</code>	A <code>BiocParallelParam</code> 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 a "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

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
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	<i>Calculate transcripts-per-million (TPM)</i>
--------------	--

Description

Calculate transcripts-per-million (TPM) values for expression from counts for a set of features.

Usage

```

calculateTPM(object, effective_length = NULL, exprs_values = "counts",
  subset_row = NULL)

```

Arguments

object	A SingleCellExperiment object or a count matrix.
effective_length	Numeric vector containing the effective length for each feature in object. If NULL, it is assumed that exprs_values has already been adjusted for transcript length.
exprs_values	String or integer specifying the assay containing the counts in object, if it is a SingleCellExperiment.
subset_row	A vector specifying the subset of rows of object for which to return a result.

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.

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 probably biased) estimate of the number of transcripts.

Value

A numeric matrix of TPM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)

eff_len <- runif(nrow(example_sce), 500, 2000)
tout <- calculateTPM(example_sce, effective_length = eff_len)
```

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

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 comparisons 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

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)

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"))
```

filter	<i>Return SingleCellExperiment with cells matching conditions (deprecated).</i>
--------	---

Description

Subsets the columns (cells) of a `SingleCellExperiment` based on matching conditions in the rows of `colData(object)`.

Usage

```
filter(object, ...)

## S4 method for signature 'SingleCellExperiment'
filter(object, ...)
```

Arguments

object	A <code>SingleCellExperiment</code> object.
...	Additional arguments to be passed to <code>dplyr::filter</code> to act on <code>colData(object)</code> .

Details

Refer to <https://github.com/sa-lee/plyexperiment> for replacement functionality.

Value

An `SingleCellExperiment` object.

getBMFeatureAnnos *Get feature annotation information from Biomart*

Description

Use the **biomaRt** package to add feature annotation information to an [SingleCellExperiment](#).

Usage

```
getBMFeatureAnnos(object, ids = rownames(object),
  filters = "ensembl_gene_id", attributes = c(filters, "mgi_symbol",
  "chromosome_name", "gene_biotype", "start_position", "end_position"),
  biomaRt = "ENSEMBL_MART_ENSEMBL", dataset = "mmusculus_gene_ensembl",
  host = "www.ensembl.org")
```

Arguments

object	A SingleCellExperiment object.
ids	A character vector containing the identifiers for all rows of object, of the same type specified by filters.
filters	Character vector defining the filters to pass to the getBM function.
attributes	Character vector defining the attributes to pass to getBM .
biomaRt	String defining the biomaRt to be used, to be passed to useMart . Default is "ENSEMBL_MART_ENSEMBL".
dataset	String defining the dataset to use, to be passed to useMart . Default is "mmusculus_gene_ensembl", which should be changed if the organism is not mouse.
host	Character string argument which can be used to select a particular "host" to pass to useMart . Useful for accessing archived versions of biomaRt data. Default is "www.ensembl.org", in which case the current version of the biomaRt (now hosted by Ensembl) is used.

Value

A [SingleCellExperiment](#) object containing feature annotation. The input feature_symbol appears as the feature_symbol field in the rowData of the output object.

Examples

```
## Not run:
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)

mock_id <- paste0("ENSMUSG", sprintf("%011d", seq_len(nrow(example_sce))))
example_sce <- getBMFeatureAnnos(example_sce, ids=mock_id)

## End(Not run)
```

getExplanatoryPCs *Estimate the percentage of variance explained for each PC.*

Description

Estimate the percentage of variance explained for each PC.

Usage

```
getExplanatoryPCs(object, use_dimred = "PCA", ncomponents = 10,  
  rerun = FALSE, run_args = list(), ...)
```

Arguments

object	A SingleCellExperiment object containing expression values and per-cell experimental information.
use_dimred	String specifying the field in reducedDims(object) that contains the PCA results.
ncomponents	Integer scalar specifying the number of the top principal components to use.
rerun	Logical scalar indicating whether the PCA should be repeated, even if pre-computed results are already present.
run_args	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 ncomponents PCs from the matrix identified by use_dimred. If these are not available or if rerun=TRUE, the function will rerun the PCA using [runPCA](#).

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](#), [getVarianceExplained](#)

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)
example_sce <- normalize(example_sce)

r2mat <- getExplanatoryPCs(example_sce)
```

getVarianceExplained *Estimate the percentage of variance explained for each gene.*

Description

Estimate the percentage of variance explained for each gene.

Usage

```
getVarianceExplained(object, exprs_values = "logcounts",
  variables = NULL, chunk = 1000)
```

Arguments

object	A SingleCellExperiment object containing expression values and per-cell experimental information.
exprs_values	String specifying the expression values for which to compute the variance.
variables	Character vector specifying the explanatory factors in colData(object) to use. Default is NULL, in which case all variables in colData(object) are considered.
chunk	Integer scalar specifying the chunk size for chunk-wise processing. Only affects the speed/memory usage trade-off.

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

Author(s)

Aaron Lun

See Also

[plotExplanatoryVariables](#)

Examples

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)
example_sce <- normalize(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 = 5, type = c("both", "lower", "higher"),
  log = FALSE, subset = NULL, batch = NULL, 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 log10 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. Missing values will trigger a warning and will be automatically ignored.
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.
min_diff	A numeric scalar indicating the minimum difference from the median to consider as an outlier. The outlier threshold is defined from the larger of nmads MADs and min_diff, to avoid calling many outliers when the MAD is very small. If NA, it is ignored.

Details

Lower and upper thresholds are stored in the "threshold" attribute of the returned vector. This is a numeric vector of length 2 when batch=NULL for the threshold on each side. Otherwise, it is a matrix with one named column per level of batch and two rows (one per threshold).

Value

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

Author(s)

Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- calculateQCMetrics(example_sce)

## with a set of feature controls defined
example_sce <- calculateQCMetrics(example_sce,
  feature_controls = list(set1 = 1:40))
isOutlier(example_sce$total_counts, nmads = 3)
```

librarySizeFactors *Compute library size factors*

Description

Define size factors from the library sizes after centering. This ensures that the library size adjustment yields values comparable to those generated after normalization with other sets of size factors.

Usage

```
librarySizeFactors(object, exprs_values = "counts", subset_row = NULL)
```

Arguments

<code>object</code>	A count matrix or <code>SingleCellExperiment</code> object containing counts.
<code>exprs_values</code>	A string indicating the assay of <code>object</code> containing the counts, if <code>object</code> is a <code>SingleCellExperiment</code> .
<code>subset_row</code>	A vector specifying whether the rows of <code>object</code> should be (effectively) subsetted before calculating library sizes.

Value

A numeric vector of size factors.

Examples

```
data("sc_example_counts")
summary(librarySizeFactors(sc_example_counts))
```

`multiplot`*Multiple plot function for ggplot2 plots*

Description

Place multiple `ggplot` plots on one page.

Usage

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

Arguments

<code>...</code>	One or more <code>ggplot</code> objects.
<code>plotlist</code>	A list of <code>ggplot</code> objects, as an alternative to <code>...</code>
<code>cols</code>	A numeric scalar giving the number of columns in the layout.
<code>layout</code>	A matrix specifying the layout. If present, <code>cols</code> 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_\(ggplot2\)/](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)

```

mutate	<i>Add new variables to colData(object) (deprecated).</i>
--------	---

Description

Add new variables to `colData(object)` (deprecated).

Usage

```

mutate(object, ...)

## S4 method for signature 'SingleCellExperiment'
mutate(object, ...)

```

Arguments

`object` a `SingleCellExperiment` object.
`...` Additional arguments to be passed to `dplyr::mutate` to act on `colData(object)`.

Details

Refer to <https://github.com/sa-lee/plyexperiment> for replacement functionality.

Value

An `SingleCellExperiment` object.

nexprs	<i>Count the number of non-zero counts per cell or feature</i>
--------	--

Description

An efficient internal function that counts the number of non-zero counts in each row (per feature) or column (per cell). This avoids the need to construct an intermediate logical matrix.

Usage

```

nexprs(object, detection_limit = 0, exprs_values = "counts",
  byrow = FALSE, subset_row = NULL, subset_col = NULL,
  BPPARAM = SerialParam())

```

Arguments

<code>object</code>	A <code>SingleCellExperiment</code> object or a numeric matrix of expression values.
<code>detection_limit</code>	Numeric scalar providing the value above which observations are deemed to be expressed.
<code>exprs_values</code>	String or integer specifying the assay of <code>object</code> to obtain the count matrix from, if <code>object</code> is a <code>SingleCellExperiment</code> .
<code>byrow</code>	Logical scalar indicating whether to count the number of detected cells per feature. If <code>FALSE</code> , the function will count the number of detected features per cell.
<code>subset_row</code>	Logical, integer or character vector indicating which rows (i.e. features) to use.
<code>subset_col</code>	Logical, integer or character vector indicating which columns (i.e., cells) to use.
<code>BPPARAM</code>	A <code>BiocParallelParam</code> object specifying whether the calculations should be parallelized.

Details

Setting `subset_row` or `subset_col` is equivalent to subsetting `object` before calling `nexprs`, but more efficient as a new copy of the matrix is not constructed.

Value

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

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)

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

<code>normalize</code>	<i>Normalize a <code>SingleCellExperiment</code> object using pre-computed size factors</i>
------------------------	---

Description

Compute normalized expression values from count data in a `SingleCellExperiment` object, using the size factors stored in the object.

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

<code>object</code>	A <code>SingleCellExperiment</code> object.
<code>exprs_values</code>	String indicating which assay contains the count data that should be used to compute log-transformed expression values.
<code>return_log</code>	Logical scalar, should normalized values be returned on the log ₂ scale? If TRUE, output is stored as "logcounts" in the returned object; if FALSE output is stored as "normcounts".
<code>log_exprs_offset</code>	Numeric scalar specifying the pseudo-count to add when log-transforming expression values. If NULL, the value is taken from <code>metadata(object)\$log_exprs_offset</code> if defined, otherwise it is set to 1.
<code>centre_size_factors</code>	Logical scalar indicating whether size factors should be centred.
<code>preserve_zeroes</code>	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

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)

example_sce <- normalize(example_sce)
```

normalizeCounts

Divide columns of a count matrix by the size factors

Description

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

Usage

```
normalizeCounts(x, size_factors, return_log = TRUE,
  log_exprs_offset = 1, centre_size_factors = FALSE,
  subset_row = NULL)
```

Arguments

<code>x</code>	A count matrix, with cells in the columns and genes in the rows.
<code>size_factors</code>	A numeric vector of size factors for all cells.
<code>return_log</code>	Logical scalar, should normalized values be returned on the log ₂ scale?
<code>log_exprs_offset</code>	Numeric scalar specifying the offset to add when log-transforming expression values.
<code>centre_size_factors</code>	Logical scalar indicating whether size factors should be centred.
<code>subset_row</code>	A vector specifying the subset of rows of <code>x</code> for which to return a result.

Details

This function will compute log-normalized expression values from `x`. It will endeavour to return an object of the same class as `x`, with particular focus on [DelayedMatrix](#) inputs/outputs.

Note that the default `centre_size_factors` differs from that in [normalizeSCE](#). Users of this function are assumed to know what they're doing with respect to normalization.

Value

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

Author(s)

Aaron Lun

Examples

```
data("sc_example_counts")
normed <- normalizeCounts(sc_example_counts,
  librarySizeFactors(sc_example_counts))
```

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 expression data
 a matrix of FPKM values
 A matrix of numeric, integer or logical values.

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts), colData = sc_example_cell_info)

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

example_sce <- SingleCellExperiment(
  assays = list(norm_counts = sc_example_counts), colData = sc_example_cell_info)

counts(example_sce) <- sc_example_counts
norm_exprs(example_sce) <- log2(calculateCPM(example_sce, use_size_factors = FALSE) + 1)

stand_exprs(example_sce) <- log2(calculateCPM(example_sce, use_size_factors = FALSE) + 1)

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

cpm(example_sce) <- calculateCPM(example_sce, use_size_factors = FALSE)

fpkm(example_sce)
```

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, ...)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information.
y	Specification of the column-level metadata to show on the y-axis, see ?"scater-vis-var" for possible values. Note that only metadata fields will be searched, assays will not be used.
x	Specification of the column-level metadata to show on the x-axis, see ?"scater-vis-var" for possible values. Again, only metadata fields will be searched, assays will not be used.
colour_by	Specification of a column metadata field or a feature to colour by, see ?"scater-vis-var" for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see ?"scater-vis-var" for possible values.
size_by	Specification of a column metadata field or a feature to size by, see ?"scater-vis-var" 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 ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see ?"scater-vis-var" 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.

Note that plotPhenoData and plotCellData are synonyms for plotColData. These are artifacts of the transition from the old SCESet class, and will be deprecated in future releases.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- calculateQCMetrics(example_sce)
```

```

example_sce <- normalize(example_sce)

plotColData(example_sce, y = "total_features_by_counts",
  x = "log10_total_counts", colour_by = "Mutation_Status")

plotColData(example_sce, y = "total_features_by_counts",
  x = "log10_total_counts", colour_by = "Mutation_Status",
  size_by = "Gene_0001", shape_by = "Treatment")

plotColData(example_sce, y = "Treatment",
  x = "log10_total_counts", colour_by = "Mutation_Status")

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

```

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

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)
example_sce <- normalize(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

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)
example_sce <- normalize(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", ...)

```

Arguments

object	A SingleCellExperiment object containing expression values and other meta-data.
features	A character vector (of feature names), a logical vector or numeric vector (of indices) specifying the features to plot.
x	Specification of a column metadata field or a feature to show on the x-axis, see ?"scater-vis-var" 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 ?"scater-vis-var" for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see ?"scater-vis-var" for possible values.
size_by	Specification of a column metadata field or a feature to size by, see ?"scater-vis-var" 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 ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see ?"scater-vis-var" for details.

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 <code>facet_wrap</code> when multiple facets are generated.
...	Additional arguments for visualization, see <code>?"scatter-plot-args"</code> 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

```
## prepare data
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- calculateQCMetrics(example_sce)
sizeFactors(example_sce) <- colSums(counts(example_sce))
example_sce <- normalize(example_sce)

## default plot
plotExpression(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, 1:6, colour_by = "Mutation_Status")
plotExpression(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, 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.

Usage

```

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

```

Arguments

<code>object</code>	A <code>SingleCellExperiment</code> object.
<code>freq_exprs</code>	Specification of the row-level metadata field containing the number of expressing cells per feature, see ?" scater-vis-var " for possible values. Note that only metadata fields will be searched, assays will not be used. If not supplied or <code>NULL</code> , this defaults to <code>"n_cells_by_counts"</code> or equivalent for compacted data.
<code>mean_exprs</code>	Specification of the row-level metadata field containing the mean expression of each feature, see ?" scater-vis-var " for possible values. Again, only metadata fields will be searched, assays will not be used. If not supplied or <code>NULL</code> , this defaults to <code>"mean_counts"</code> or equivalent for compacted data.
<code>controls</code>	Specification of the row-level metadata column indicating whether a feature is a control, see ?" scater-vis-var " for possible values. Only metadata fields will be searched, assays will not be used. If not supplied, this defaults to <code>"is_feature_control"</code> or equivalent for compacted data.
<code>exprs_values</code>	String specifying the assay used for the default <code>freq_exprs</code> and <code>mean_exprs</code> . This can be set to, e.g., <code>"logcounts"</code> so that <code>freq_exprs</code> defaults to <code>"n_cells_by_logcounts"</code> .
<code>by_show_single</code>	Logical scalar specifying whether a single-level factor for controls should be used for colouring, see ?" scater-vis-var " for details.
<code>show_smooth</code>	Logical scalar, should a smoothed fit (through feature controls if available; all features otherwise) be shown on the plot? See geom_smooth for details.
<code>show_se</code>	Logical scalar, should the standard error be shown for a smoothed fit?
<code>...</code>	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.

The plot will attempt to colour the points based on whether the corresponding features are labelled as feature controls in object. This can be turned off by setting `controls=NULL`.

Value

A ggplot object.

See Also

[plotRowData](#)

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(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.

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 can take any of the values described in <code>?"scater-vis-var"</code> for feature-level metadata; data in assays(object) will <i>not</i> be searched. Otherwise, if length_is_assay=TRUE, tx_length should be the name or index of an assay in object.
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 column metadata field or a feature to colour by, see <code>?"scater-vis-var"</code> for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see <code>?"scater-vis-var"</code> for possible values.
size_by	Specification of a column metadata field or a feature to size by, see <code>?"scater-vis-var"</code> 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 <code>?"scater-vis-var"</code> for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see <code>?"scater-vis-var"</code> for details.
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 <code>?"scater-plot-args"</code> 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

```
data("sc_example_counts")
data("sc_example_cell_info")
rd <- DataFrame(gene_id = rownames(sc_example_counts),
  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)
  )
rownames(rd) <- rownames(sc_example_counts)
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info, rowData = rd
)
example_sce <- normalize(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,
  by_exprs_values = exprs_values, by_show_single = FALSE,
  show_colnames = TRUE, ...)

```

Arguments

<code>object</code>	A <code>SingleCellExperiment</code> object.
<code>features</code>	A character vector of row names, a logical vector of integer vector of indices specifying rows of object to show in the heatmap.
<code>columns</code>	A vector specifying the subset of columns in object to show as columns in the heatmap. By default, all columns are used in their original order.

<code>exprs_values</code>	A string or integer scalar indicating which assay of object should be used as expression values for colouring in the heatmap.
<code>center</code>	A logical scalar indicating whether each row should have its mean expression centered at zero prior to plotting.
<code>zlim</code>	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 <code>center=TRUE</code>). If <code>NULL</code> , it defaults to the range of the expression matrix.
<code>symmetric</code>	A logical scalar specifying whether the default <code>zlim</code> should be symmetric around zero. If <code>TRUE</code> , the maximum absolute value of <code>zlim</code> will be computed and multiplied by <code>c(-1, 1)</code> to redefine <code>zlim</code> .
<code>color</code>	A vector of colours specifying the palette to use for mapping expression values to colours. This defaults to the default setting in pheatmap .
<code>colour_columns_by</code>	A list of values specifying how the columns should be annotated with colours. Each entry of the list can be of the form described by <code>?"scater-vis-var"</code> . A character vector can also be supplied and will be treated as a list of strings.
<code>by_exprs_values</code>	A string or integer scalar specifying which assay to obtain expression values from, for colouring of column-level data - see <code>?"scater-vis-var"</code> for details.
<code>by_show_single</code>	Logical scalar specifying whether single-level factors should be used for column-level colouring, see <code>?"scater-vis-var"</code> for details.
<code>show_colnames</code>	Logical scalar specifying whether column names should be shown, if available in object.
<code>...</code>	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.

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(normalizeSCE) # borrowing the example objects in here.
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, controls, colour_cells_by,
                 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.
controls	Specification of the row-level metadata column indicating whether a feature is a control, see ?"scater-vis-var" for possible values. Only metadata fields will be searched, assays will not be used. If not supplied, this defaults to "is_feature_control" or equivalent for compacted data.
colour_cells_by	Specification of a column metadata field or a feature to colour by, see ?"scater-vis-var" for possible values. If not supplied, this defaults to "total_features_by_counts" or equivalent for compacted data.
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 ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for colouring, see ?"scater-vis-var" for details. Default is NULL, in which case rownames(object) are used.
feature_names_to_plot	Specification of which row-level metadata column contains the feature names, see ?"scater-vis-var" 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 feature corresponds to a row on the plot, sorted by average expression (denoted by the point).

The plot will attempt to colour the points based on whether the corresponding feature is labelled as a control in object. This can be turned off by setting `controls=NULL`.

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`. Setting `colour_cells_by=NULL` will disable all tick colouring.

Value

A ggplot object.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- calculateQCMetrics(example_sce,
  feature_controls = list(set1 = 1:500)
)

plotHighestExprs(example_sce, colour_cells_by = "total_features_by_counts")
plotHighestExprs(example_sce, controls = NULL)
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)
```

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 ?"scater-vis-var" for possible values.
size_by	Specification of a column metadata field or a feature to size by, see ?"scater-vis-var" for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see ?"scater-vis-var" 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 ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see ?"scater-vis-var" for details.
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.

Details

This function expects plate positions to be given in a character 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

```
## prepare data
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(example_sce)
example_sce <- calculateQCMetrics(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, use_dimred, ncomponents = 2, percentVar = NULL,
  colour_by = NULL, shape_by = NULL, size_by = NULL,
  by_exprs_values = "logcounts", by_show_single = FALSE,
  text_by = NULL, text_size = 5, text_colour = "black", ...)

```

Arguments

object	A SingleCellExperiment object.
use_dimred	A string or integer scalar indicating the reduced dimension result in reducedDims(object) to plot.
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 ?" scatter-vis-var " for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see ?" scatter-vis-var " for possible values.
size_by	Specification of a column metadata field or a feature to size by, see ?" scatter-vis-var " 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 ?" scatter-vis-var " for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see ?" scatter-vis-var " for details.

text_by	Specification of a column metadata field for which to add text - see <code>?"scater-vis-var"</code> for possible values. This must refer to a categorical field, i.e., coercible into a factor.
text_size	Numeric scalar specifying the size of added text.
text_colour	String specifying the colour of the added text.
...	Additional arguments for visualization, see <code>?"scater-plot-args"</code> 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

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(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 a relative log expression (RLE) plot***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 ?"scater-vis-var" 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 ?"scater-vis-var" for details.
...	further arguments passed to <code>geom_boxplot</code> when <code>style="full"</code> .

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. RLE Plots: Visualising Unwanted Variation in High Dimensional Data. arXiv [stat.ME]. 2017. Available: <http://arxiv.org/abs/1704.03590>

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(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, ...)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information.
y	Specification of the row-level metadata to show on the y-axis, see ?"scater-vis-var" for possible values. Note that only metadata fields will be searched, assays will not be used.
x	Specification of the row-level metadata to show on the x-axis, see ?"scater-vis-var" for possible values. Again, only metadata fields will be searched, assays will not be used.

colour_by	Specification of a row metadata field or a cell to colour by, see ?"scatter-vis-var" for possible values.
shape_by	Specification of a row metadata field or a cell to shape by, see ?"scatter-vis-var" for possible values.
size_by	Specification of a row metadata field or a cell to size by, see ?"scatter-vis-var" 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 ?"scatter-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for point aesthetics, see ?"scatter-vis-var" for details.
...	Additional arguments for visualization, see ?"scatter-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.

Note that plotFeatureData is a synonym for plotRowData. This is an artifact of the transition from the old SCESet class, and will be deprecated in future releases.

Value

A ggplot object.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- calculateQCMetrics(example_sce,
  feature_controls = list(ERCC=1:40))
example_sce <- normalize(example_sce)

plotRowData(example_sce, y="n_cells_by_counts", x="log10_total_counts")
plotRowData(example_sce, y="n_cells_by_counts",
  size_by = "log10_total_counts",
  colour_by = "is_feature_control")
```

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 ?"scater-vis-var" 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 line colouring - see ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for line colouring, see ?"scater-vis-var" for details.
block1	Specification of a factor by which to separate the cells into blocks (separate panels) in the plot. This can be any type of value described in ?"scater-vis-var" for column-level metadata. 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 plot object

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)

plotScater(example_sce)
plotScater(example_sce, exprs_values = "counts", colour_by = "Cell_Cycle")
plotScater(example_sce, block1 = "Treatment", colour_by = "Cell_Cycle")

cpm(example_sce) <- calculateCPM(example_sce, use_size_factors = FALSE)
plotScater(example_sce, exprs_values = "cpm", block1 = "Treatment",
  block2 = "Mutation_Status", colour_by = "Cell_Cycle")
```

readSparseCounts	<i>Read sparse count matrix from file</i>
------------------	---

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, <i>before</i> the column names.

<code>skip.row</code>	An integer scalar specifying the number of rows to ignore at the start of the file, <i>after</i> the column names.
<code>ignore.col</code>	An integer scalar specifying the number of columns to ignore at the start of the file, <i>before</i> the column names.
<code>skip.col</code>	An integer scalar specifying the number of columns to ignore at the start of the file, <i>after</i> the column names.
<code>chunk</code>	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, or, if they are not already present, to calculate those results and then plot them.

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 will search the [reducedDims](#) slot for an appropriately named set of results and pass those coordinates onto [plotReducedDim](#). If the results are not present or rerun=TRUE, they will be computed using the relevant run* function. The result name and run* function for each plot* function are:

- "PCA" and [runPCA](#) for plotPCA
- "TSNE" and [runTSNE](#) for plotTSNE
- "DiffusionMap" and [runDiffusionMap](#) for plotDiffusionMap
- "MDS" and [runMDS](#) for "plotMDS"

Users can specify arguments to the run* functions via run_args.

If ncomponents is a numeric vector, the maximum value will be used to determine the required number of dimensions to compute in the run* functions. However, only the specified dimensions in ncomponents will be plotted.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

See Also

[runPCA](#), [runDiffusionMap](#), [runTSNE](#), [runMDS](#), [plotReducedDim](#)

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(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:
plotTSNE(example_sce, run_args=list(perplexity = 10))

## Same for DiffusionMaps:
plotDiffusionMap(example_sce)

## Same for MDS plots:
plotMDS(example_sce)
```

rename

Rename variables of colData(object) (deprecated).

Description

Rename variables of colData(object) (deprecated).

Usage

```
rename(object, ...)

## S4 method for signature 'SingleCellExperiment'
rename(object, ...)
```

Arguments

object A SingleCellExperiment object.
 ... Additional arguments to be passed to `dplyr::rename` to act on `colData(object)`.

Details

Refer to <https://github.com/sa-lee/plyexperiment> for replacement functionality.

Value

An SingleCellExperiment object.

runDiffusionMap	<i>Create a diffusion map from cell-level data</i>
-----------------	--

Description

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

Usage

```
runDiffusionMap(object, ncomponents = 2, ntop = 500,
  feature_set = NULL, exprs_values = "logcounts",
  scale_features = TRUE, use_dimred = NULL, n_dimred = NULL, ...)
```

Arguments

object A SingleCellExperiment object
 ncomponents Numeric scalar indicating the number of diffusion components to obtain.
 ntop Numeric scalar specifying the number of most variable features to use for constructing the diffusion map.
 feature_set Character vector of row names, a logical vector or a numeric vector of indices indicating a set of features to use to construct the diffusion map. This will override any ntop argument if specified.
 exprs_values Integer scalar or string indicating which assay of object should be used to obtain the expression values for the calculations.
 scale_features Logical scalar, should the expression values be standardised so that each feature has unit variance?
 use_dimred String or integer scalar specifying the entry of `reducedDims(object)` to use as input to `DiffusionMap`. Default is to not use existing reduced dimension results.
 n_dimred Integer scalar, number of dimensions of the reduced dimension slot to use when `use_dimred` is supplied. Defaults to all available dimensions.
 ... Additional arguments to pass to `DiffusionMap`.

Details

The function `DiffusionMap` is used internally to compute the diffusion map.

Setting `use_dimred` allows users to easily construct a diffusion map from low-rank approximations of the original expression matrix (e.g., after PCA). In such cases, arguments such as `ntop`, `feature_set`, `exprs_values` and `scale_features` will be ignored.

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

A `SingleCellExperiment` object containing the coordinates of the first `ncomponent` diffusion map components for each cell. This is stored in the "DiffusionMap" entry of the `reducedDims` slot.

Author(s)

Aaron Lun, based on code by Davis McCarthy

References

Haghverdi L, Buettner F, Theis FJ. Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics*. 2015; doi:10.1093/bioinformatics/btv325

See Also

`destiny`, `plotDiffusionMap`

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(example_sce)

example_sce <- runDiffusionMap(example_sce)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

runMDS

Perform MDS on cell-level data

Description

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

Usage

```
runMDS(object, ncomponents = 2, ntop = 500, feature_set = NULL,
        exprs_values = "logcounts", scale_features = TRUE,
        use_dimred = NULL, n_dimred = NULL, method = "euclidean")
```

Arguments

object	A SingleCellExperiment object.
ncomponents	Numeric scalar indicating the number of MDS dimensions to obtain.
ntop	Numeric scalar specifying the number of most variable features to use for MDS.
feature_set	Character vector of row names, a logical vector or a numeric vector of indices indicating a set of features to use for MDS. This will override any ntop argument if specified.
exprs_values	Integer scalar or string indicating which assay of object should be used to obtain the expression values for the calculations.
scale_features	Logical scalar, should the expression values be standardised so that each feature has unit variance?
use_dimred	String or integer scalar specifying the entry of reducedDims(object) to use as input to cmdscale . Default is to not use existing reduced dimension results.
n_dimred	Integer scalar, number of dimensions of the reduced dimension slot to use when use_dimred is supplied. Defaults to all available dimensions.
method	String specifying the type of distance to be computed between cells.

Details

The function [cmdscale](#) is used internally to compute the multidimensional scaling components to plot.

Setting use_dimred allows users to easily perform MDS on low-rank approximations of the original expression matrix (e.g., after PCA). In such cases, arguments such as ntop, feature_set, exprs_values and scale_features will be ignored.

Value

A SingleCellExperiment object containing the coordinates of the first ncomponent MDS dimensions for each cell. This is stored in the "MDS" entry of the reducedDims slot.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

[cmdscale](#), [plotMDS](#)

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
```

```

      colData = sc_example_cell_info
    )
    example_sce <- normalize(example_sce)

    example_sce <- runMDS(example_sce)
    reducedDimNames(example_sce)
    head(reducedDim(example_sce))

```

runPCA,SingleCellExperiment-method

Perform PCA on cell-level data

Description

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

Usage

```

## S4 method for signature 'SingleCellExperiment'
runPCA(x, ncomponents = 2,
       method = NULL, ntop = 500, exprs_values = "logcounts",
       feature_set = NULL, scale_features = TRUE, use_coldata = FALSE,
       selected_variables = NULL, detect_outliers = FALSE,
       BSPARAM = ExactParam(), BPPARAM = SerialParam())

```

Arguments

<code>x</code>	A SingleCellExperiment object.
<code>ncomponents</code>	Numeric scalar indicating the number of principal components to obtain.
<code>method</code>	Deprecated, string specifying how the PCA should be performed.
<code>ntop</code>	Numeric scalar specifying the number of most variable features to use for PCA.
<code>exprs_values</code>	Integer scalar or string indicating which assay of object should be used to obtain the expression values for the calculations.
<code>feature_set</code>	Character vector of row names, a logical vector or a numeric vector of indices indicating a set of features to use for PCA. This will override any <code>ntop</code> argument if specified.
<code>scale_features</code>	Logical scalar, should the expression values be standardised so that each feature has unit variance? This will also remove features with standard deviations below $1e-8$.
<code>use_coldata</code>	Logical scalar specifying whether the column data should be used instead of expression values to perform PCA.
<code>selected_variables</code>	List of strings or a character vector indicating which variables in <code>colData(object)</code> to use for PCA when <code>use_coldata=TRUE</code> . If a list, each entry can take the form described in ?"scatter-vis-var" .
<code>detect_outliers</code>	Logical scalar, should outliers be detected based on PCA coordinates generated from column-level metadata?

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.

Details

The function `prcomp` is used internally to do the PCA when `method="prcomp"`. Alternatively, the `irlba` package can be used, which performs a fast approximation of PCA through the `prcomp_irlba` function. This is especially useful for large, sparse matrices.

Note that `prcomp_irlba` involves a random initialization, after which it converges 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 `method="irlba"`.

If `use_coldata=TRUE`, PCA will be performed on column-level metadata instead of the gene expression matrix. The `selected_variables` defaults to a vector containing:

- "pct_counts_top_100_features"
- "total_features_by_counts"
- "pct_counts_feature_control"
- "total_features_feature_control"
- "log10_total_counts_endogenous"
- "log10_total_counts_feature_control"

This can be useful for identifying outlier cells based on QC metrics, especially when combined with `detect_outliers=TRUE`. If outlier identification is enabled, the `outlier` field of the output `colData` will contain the identified outliers.

Value

A `SingleCellExperiment` object containing the first `ncomponent` principal coordinates for each cell. If `use_coldata=FALSE`, this is stored in the "PCA" entry of the `reducedDims` slot. Otherwise, it is stored in the "PCA_coldata" entry.

The proportion of variance explained by each PC is stored as a numeric vector in the "percentVar" attribute of the reduced dimension matrix. Note that this will only be of length equal to `ncomponents` when `method` is not "prcomp". This is because approximate PCA methods do not compute singular values for all components.

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

[prcomp](#), [plotPCA](#)

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
```

```

)
example_sce <- normalize(example_sce)

example_sce <- runPCA(example_sce)
reducedDimNames(example_sce)
head(reducedDim(example_sce))

```

runTSNE *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

```

runTSNE(object, ncomponents = 2, ntop = 500, feature_set = NULL,
  exprs_values = "logcounts", scale_features = TRUE,
  use_dimred = NULL, n_dimred = NULL, perplexity = min(50,
  floor(ncol(object)/5)), pca = TRUE, initial_dims = 50,
  normalize = TRUE, theta = 0.5, external_neighbors = FALSE,
  BNPARAM = KmknParam(), BPPARAM = SerialParam(), ...)

```

Arguments

object	A SingleCellExperiment object.
ncomponents	Numeric scalar indicating the number of t-SNE dimensions to obtain.
ntop	Numeric scalar specifying the number of most variable features to use for t-SNE.
feature_set	Character vector of row names, a logical vector or a numeric vector of indices indicating a set of features to use for t-SNE. This will override any ntop argument if specified.
exprs_values	Integer scalar or string indicating which assay of object should be used to obtain the expression values for the calculations.
scale_features	Logical scalar, should the expression values be standardised so that each feature has unit variance?
use_dimred	String or integer scalar specifying the entry of reducedDims(object) to use as input to Rtsne . Default is to not use existing reduced dimension results.
n_dimred	Integer scalar, number of dimensions of the reduced dimension slot to use when use_dimred is supplied. Defaults to all available dimensions.
perplexity	Numeric scalar defining the perplexity parameter, see ?Rtsne for more details.
pca	Logical scalar passed to Rtsne , indicating whether an initial PCA step should be performed. This is ignored if use_dimred is specified.
initial_dims	Integer scalar passed to Rtsne , specifying the number of principal components to be retained if pca=TRUE.
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.
...	Additional arguments to pass to Rtsne .

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 try to provide a reasonable setting, by scaling the perplexity with the number of cells until it reaches a maximum of 50. However, it is often worthwhile to manually try multiple values to ensure that the conclusions are robust.

Setting `use_dimred` allows users to easily perform t-SNE on low-rank approximations of the original expression matrix (e.g., after PCA). In such cases, arguments such as `ntop`, `feature_set`, `exprs_values` and `scale_features` will be ignored.

If `external_neighbors=TRUE`, the nearest neighbor search step is conducted using 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.

Value

A `SingleCellExperiment` object containing the coordinates of the first `ncomponent` t-SNE dimensions for each cell. This is stored in the "TSNE" entry of the `reducedDims` slot.

Author(s)

Aaron Lun, based on code by Davis McCarthy

References

L.J.P. van der Maaten. Barnes-Hut-SNE. In Proceedings of the International Conference on Learning Representations, 2013.

See Also

[Rtsne](#), [plotTSNE](#)

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
```



```

    assays = list(counts = sc_example_counts),
    colData = sc_example_cell_info
  )
example_sce <- normalize(example_sce)

example_sce <- runTSNE(example_sce)
reducedDimNames(example_sce)
head(reducedDim(example_sce))

```

runUMAP

*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

```

runUMAP(object, ncomponents = 2, ntop = 500, feature_set = NULL,
  exprs_values = "logcounts", scale_features = TRUE,
  use_dimred = NULL, n_dimred = NULL, pca = 50, n_neighbors = 15,
  external_neighbors = FALSE, BNPARAM = KmknnParam(),
  BPPARAM = SerialParam(), ...)

```

Arguments

<code>object</code>	A SingleCellExperiment object.
<code>ncomponents</code>	Numeric scalar indicating the number of UMAP dimensions to obtain.
<code>ntop</code>	Numeric scalar specifying the number of most variable features to use for UMAP.
<code>feature_set</code>	Character vector of row names, a logical vector or a numeric vector of indices indicating a set of features to use for UMAP. This will override any <code>ntop</code> argument if specified.
<code>exprs_values</code>	Integer scalar or string indicating which assay of <code>object</code> should be used to obtain the expression values for the calculations.
<code>scale_features</code>	Logical scalar, should the expression values be standardised so that each feature has unit variance?
<code>use_dimred</code>	String or integer scalar specifying the entry of <code>reducedDims(object)</code> to use as input to Rtsne . Default is to not use existing reduced dimension results.
<code>n_dimred</code>	Integer scalar, number of dimensions of the reduced dimension slot to use when <code>use_dimred</code> is supplied. Defaults to all available dimensions.
<code>pca</code>	Integer scalar specifying how many PCs should be used as input into UMAP, if the PCA is to be recomputed on the subsetted expression matrix. Only used when <code>codeuse_dimred=NULL</code> , and if <code>pca=NULL</code> , no PCA is performed at all.
<code>n_neighbors</code>	Integer scalar, number of nearest neighbors to identify when constructing the initial graph.
<code>external_neighbors</code>	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 <code>external_neighbors=TRUE</code> .
BPPARAM	A BiocParallelParam object specifying how the neighbor search should be parallelized when <code>external_neighbors=TRUE</code> .
...	Additional arguments to pass to umap .

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.

Setting `use_dimred` allows users to easily perform UMAP on low-rank approximations of the original expression matrix (e.g., after PCA). In such cases, arguments such as `ntop`, `feature_set`, `exprs_values` and `scale_features` will be ignored.

If `external_neighbors=TRUE`, the nearest neighbor search step 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

A `SingleCellExperiment` object containing the coordinates of the first `ncomponent` UMAP dimensions for each cell. This is stored in the "UMAP" entry of the `reducedDims` slot.

Author(s)

Aaron Lun

References

McInnes L, Healy J (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. [arXiv](#).

See Also

[umap](#), [plotUMAP](#)

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info
)
example_sce <- normalize(example_sce)

example_sce <- runUMAP(example_sce)
reducedDimNames(example_sce)
head(reducedDim(example_sce))
```

`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.

See Also

[plotColData](#), [plotRowData](#), [plotReducedDim](#), [plotExpression](#), [plotPlatePosition](#), and most other plotting functions.

`scater-vis-var`*Variable selection for visualization*

Description

A number of **scater** functions accept a `SingleCellExperiment` object and extract (meta)data from it for use in a plot. These values are then used on the x- or y-axes (e.g., [plotColData](#)) or for tuning visual parameters, e.g., `colour_by`, `shape_by`, `size_by`. This page describes how the selection of these values can be controlled by the user, by passing appropriate values to the arguments of the desired plotting function.

When plotting by cells

Here, we assume that each visual feature of interest (e.g., point or line) corresponds to a cell in the `SingleCellExperiment` object `sce`. We will also assume that the user wants to change the colour of each feature according to the cell (meta)data. To do so, the user can pass as an argument:

- An unnamed character vector of length 1, i.e., a string. This is initially assumed to be the name of a column-level metadata field. The function will first search the column names of `colData(sce)`, and extract metadata for all cells if a matching field is found. If no match is found, the function will assume that the string represents a gene name. It will search `rownames(sce)` and extract gene expression values for any matching row across all cells. Otherwise, an error is raised.
- A named character vector of length 1, where the name is either `"exprs"` or `"metadata"`. This forces the function to only search for the string in `rownames(sce)` or `colnames(colData(sce))`, respectively. Adding an explicit name is useful when the same field exists in both the row names and column metadata names.
- A character vector of length greater than 1. This will search for nested fields in `colData(sce)`. For example, supplying a character vector `c("A", "B", "C")` will retrieve `colData(sce)AB$C`, where both A and B contain nested DataFrames. See [calculateQCMetrics](#) with `compact=TRUE` for an example of how these can be constructed. The concatenated name `"A:B:C"` will be used in the legend.
- A character vector of length greater than 1 and the first element set to `NA`. This will search for nested fields in the internal column data of a [SingleCellExperiment](#), i.e., in `int_colData`. For example, `c(NA, "size_factor")` would retrieve the values corresponding to `sizeFactors(object)`. The concatenated name without the `NA` is used in the legend. Note that internal fields are *only* searched when `NA` is the first element.
- A data frame with one column and number of rows equal to the number of cells. This should contain values to use for visualization, e.g., for plotting on the x-/y-axis, or for colouring by. In this manner, the user can use new information without manually adding it to the `SingleCellExperiment` object. The column name of the data frame will be used in the legend.

The same logic applies for other visualization parameters such as `shape_by` and `size_by`. Other arguments may also use the same scheme, but this depends on the context; see the documentation for each function for details. In particular, if an argument explicitly refers to a metadata field, any names for the character string will be ignored. Similarly, a character vector of length > 1 is not allowed for an argument that explicitly refers to expression values.

When plotting by features

Here, we assume that each visual feature of interest (e.g., point or line) corresponds to a feature in the `SingleCellExperiment` object `sce`. The scheme is mostly the same as described above, with a few differences:

- `rowData` is searched instead of `colData`, as we are extracting metadata for each feature.
- When extracting expression values, the name of a single cell must be specified. Visualization will then use the expression profile for all features in that cell. (This tends to be a rather unusual choice for colouring.)
- Character strings named with `"exprs"` will search for the string in `colnames(sce)`.
- A data frame input should have number of rows equal to the number of features.

Miscellaneous details

Most functions will have a `by_exprs_values` parameter. This defines the assay of the `SingleCellExperiment` object from which expression values are extracted for use in colouring, shaping or sizing the points. The setting of `by_exprs_values` will usually default to "logcounts", or to the value of `exprs_values` in functions such as [plotExpression](#). However, it can be specified separately from `exprs_values`, which is useful for visualizing two different types of expression values on the same plot.

Most functions will also have a `by_show_single` parameter. If `FALSE`, variables with only one level are not used for visualization, i.e., the visual aspect (colour or shape or size) is set to the default for all points. No guide is created for this aspect, avoiding clutter in the legend when that aspect provides no information. If `TRUE`, all supplied variables are used for visualization, regardless of how many levels they have.

See Also

[plotColData](#), [plotRowData](#), [plotReducedDim](#), [plotExpression](#), [plotPlatePosition](#), and most other plotting functions.

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.

sc_example_cell_info *Cell information for the small example single-cell counts dataset to demonstrate capabilities of scater*

Description

This data.frame contains cell metadata information for the 40 cells included in the example counts dataset included in the package.

Usage

```
sc_example_cell_info
```

Format

a data.frame instance, 1 row per cell.

Value

NULL, but makes available a data frame with cell metadata

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

sc_example_counts *A small example of single-cell counts dataset to demonstrate capabilities of scater*

Description

This data set contains counts for 2000 genes for 40 cells. They are from a real experiment, but details have been anonymised.

Usage

```
sc_example_counts
```

Format

a matrix instance, 1 row per gene.

Value

NULL, but makes available a matrix of count data

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

sumCountsAcrossCells *Sum counts across a set of cells*

Description

Create a count matrix where counts for all cells in a set are summed together.

Usage

```
sumCountsAcrossCells(object, ids, exprs_values = "counts",  
  BPPARAM = SerialParam())
```

Arguments

object	A SingleCellExperiment object or a count matrix.
ids	A factor specifying the set to which each cell in object belongs.
exprs_values	A string or integer scalar specifying the assay of object containing counts, if object is a SingleCellExperiment .
BPPARAM	A BiocParallelParam object specifying how summation should be parallelized.

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.

Any NA values in ids are implicitly ignored and will not be considered or reported. This may be useful, e.g., to remove undesirable cells by setting their entries in ids to NA.

Value

A count matrix where counts for all cells in the same set are summed together for each feature.

Author(s)

Aaron Lun

Examples

```

data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)

ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)
out <- sumCountsAcrossCells(example_sce, ids)
dimnames(out)

```

sumCountsAcrossFeatures

Sum counts across a feature set

Description

Create a count matrix where counts for all features in a set are summed together.

Usage

```

sumCountsAcrossFeatures(object, ids, exprs_values = "counts",
  BPPARAM = SerialParam())

```

Arguments

object	A SingleCellExperiment object or a count matrix.
ids	A factor specifying the set to which each feature in object belongs.
exprs_values	A string or integer scalar specifying the assay of object containing counts, if object is a SingleCellExperiment .
BPPARAM	A BiocParallelParam object specifying whether summation should be parallelized.

Details

This function provides a convenient method for aggregating counts across multiple rows for each cell. For example, 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.

Any NA values in ids 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.

Value

A count matrix where counts for all features in the same set are summed together within each cell.

Author(s)

Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
  assays = list(counts = sc_example_counts),
  colData = sc_example_cell_info)

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("ENSG000000", 1:5),
  names=c("A", NA, "B", "C", "A")
)
```

updateSCESet	<i>Convert an SCESet object to a SingleCellExperiment object</i>
--------------	--

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

arrange, [3](#)
arrange, SingleCellExperiment-method
(arrange), [3](#)

BiocNeighborParam, [56, 58](#)
BiocParallelParam, [54, 56, 58, 63, 64](#)
BiocSingularParam, [54](#)
bootstraps, [4](#)
bootstraps, SingleCellExperiment-method
(bootstraps), [4](#)
bootstraps<- (bootstraps), [4](#)
bootstraps<-, SingleCellExperiment, array-method
(bootstraps), [4](#)

calcAverage (calculateAverage), [5](#)
calculateAverage, [5](#)
calculateCPM, [6, 7](#)
calculateFPKM, [7](#)
calculateQCMetrics, [8, 60](#)
calculateTPM, [12](#)
centreSizeFactors, [13](#)
cmdscales, [52](#)

DelayedMatrix, [24, 25](#)
destiny, [51](#)
DiffusionMap, [50, 51](#)

exprs (norm_exprs), [25](#)
exprs, SingleCellExperiment-method,
(norm_exprs), [25](#)
exprs<-, SingleCellExperiment, ANY-method
(norm_exprs), [25](#)

facet_wrap, [31, 45](#)
filter, [14](#)
filter, SingleCellExperiment-method
(filter), [14](#)
findKNN, [56, 57](#)
fpkm (norm_exprs), [25](#)
fpkm<- (norm_exprs), [25](#)

geom_boxplot, [42](#)
geom_smooth, [32](#)
getBM, [15](#)
getBMFeatureAnnos, [15](#)
getExplanatoryPCs, [16, 28](#)
getVarianceExplained, [16, 17, 29](#)
ggplot, [20](#)

int_colData, [60](#)
isOutlier, [18](#)
isSpike, [8](#)

librarySizeFactors, [5, 6, 19](#)

multiplot, [20](#)
mutate, [21](#)
mutate, SingleCellExperiment-method
(mutate), [21](#)

nexprs, [8, 21](#)
norm_exprs, [25](#)
norm_exprs<- (norm_exprs), [25](#)
normalize, [22](#)
normalize, SingleCellExperiment-method
(normalize), [22](#)
normalize_input, [55](#)
normalizeCounts, [24](#)
normalizeSCE, [5, 6, 13, 25](#)
normalizeSCE (normalize), [22](#)

pheatmap, [36](#)
plotColData, [26, 59, 61](#)
plotDiffusionMap, [51](#)
plotDiffusionMap (Reduced dimension
plots), [47](#)
plotExplanatoryPCs, [16, 28](#)
plotExplanatoryVariables, [17, 29](#)
plotExpression, [30, 59, 61](#)
plotExprsFreqVsMean, [32](#)
plotExprsVsTxLength, [33](#)
plotHeatmap, [35](#)
plotHighestExprs, [37](#)
plotMDS, [52](#)
plotMDS (Reduced dimension plots), [47](#)
plotPCA, [40, 54](#)
plotPCA (Reduced dimension plots), [47](#)
plotPCA, SingleCellExperiment-method
(Reduced dimension plots), [47](#)
plotPCASCE (Reduced dimension plots), [47](#)

plotPlatePosition, [38](#), [59](#), [61](#)
plotReducedDim, [40](#), [48](#), [49](#), [59](#), [61](#)
plotRLE, [42](#)
plotRLE, SingleCellExperiment-method
 (plotRLE), [42](#)
plotRowData, [32](#), [33](#), [43](#), [59](#), [61](#)
plotScater, [45](#)
plotTSNE, [56](#)
plotTSNE (Reduced dimension plots), [47](#)
plotUMAP, [58](#)
plotUMAP (Reduced dimension plots), [47](#)
prcomp, [54](#)
prcomp_irlba, [54](#)

read.table, [47](#)
readMM, [47](#)
readSparseCounts, [46](#)
Reduced dimension plots, [47](#)
reducedDims, [48](#)
rename, [49](#)
rename, SingleCellExperiment-method
 (rename), [49](#)
rowMeans, [9](#)
Rtsne, [55–57](#)
Rtsne_neighbors, [56](#)
runDiffusionMap, [48](#), [49](#), [50](#)
runMDS, [48](#), [49](#), [51](#)
runPCA, [16](#), [48](#), [49](#)
runPCA, SingleCellExperiment-method, [53](#)
runTSNE, [48](#), [49](#), [55](#)
runUMAP, [57](#)

sc_example_cell_info, [62](#)
sc_example_counts, [62](#)
scater-package, [3](#)
scater-plot-args, [59](#)
scater-vis-var, [59](#)
SCESet, [61](#), [66](#)
SCESet-class (SCESet), [61](#)
set.seed, [51](#), [54](#), [56](#), [58](#)
SingleCellExperiment, [4](#), [15](#), [25](#), [53](#), [57](#), [60](#),
 [63](#), [64](#), [66](#)
stand_exprs (norm_exprs), [25](#)
stand_exprs<- (norm_exprs), [25](#)
sumCountsAcrossCells, [63](#)
sumCountsAcrossFeatures, [64](#)

toSingleCellExperiment (updateSCESet),
 [66](#)

umap, [58](#)
uniquifyFeatureNames, [65](#)
updateSCESet, [66](#)
useMart, [15](#)