Package 'scater'

April 16, 2019

Type Package

Version 1.10.1

Date 2018-10-26

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 (>= 3.5), SingleCellExperiment, ggplot2, BiocParallel

Imports BiocGenerics, SummarizedExperiment, Matrix, dplyr, ggbeeswarm, grid, DelayedMatrixStats, methods, plyr, reshape2, S4Vectors, stats, utils, viridis, Rcpp (>= 0.12.14), DelayedArray

Suggests BiocStyle, biomaRt, beachmat, cowplot, cluster, destiny, knitr, monocle, mvoutlier, rmarkdown, Rtsne, umap, testthat, magrittr, pheatmap, irlba, tximport, tximportData, tximeta, Biobase, limma

VignetteBuilder knitr

LazyData true

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

LinkingTo Rhdf5lib, Rcpp, beachmat

SystemRequirements C++11

RoxygenNote 6.1.0

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_8

git_last_commit 2e6694a

git_last_commit_date 2019-01-04

Date/Publication 2019-04-15

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` *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*

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)`.

Value

An SingleCellExperiment 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 <- arrange(example_sce, Cell_Cycle)
```

---

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

Description

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

Usage

```
bootstraps(object)

bootstraps(object) <- value

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

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

Arguments

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

Value

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

Author(s)

Davis McCarthy

Examples

```
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)
```

`calcIsExprs` *Calculate which features are expressed in which cells using a threshold on observed counts, transcripts-per-million, counts-per-million, FPKM, or defined expression levels.*

Description

Calculate which features are expressed in which cells using a threshold on observed counts, transcripts-per-million, counts-per-million, FPKM, or defined expression levels.

Usage

```
calcIsExprs(object, detection_limit = 0, exprs_values = "counts")
```

Arguments

`object`	a `SingleCellExperiment` object with expression and/or count data.
`detection_limit`	numeric scalar giving the minimum expression level for an expression observation in a cell for it to qualify as expressed.
`exprs_values`	character scalar indicating whether the count data (`"counts"`), the log-transformed count data (`"logcounts"`), transcript-per-million (`"tpm"`), counts-per-million (`"cpm"`) or FPKM (`"fpkm"`) should be used to define if an observation is expressed or not. Defaults to the first available value of those options in the order shown.

Value

a logical matrix indicating whether or not a feature in a particular cell is expressed.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
assay(example_sce, "is_exprs") <- calcIsExprs(example_sce,
detection_limit = 1, exprs_values = "counts")
```

`calculateAverage` *Calculate average counts, adjusting for size factors or library size*

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 whetherthe 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` *Calculate counts per million (CPM)*

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

`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 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 `sizeFactor(object)`.
`subset_row`	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 *Calculate fragments per kilobase of exon per million reads mapped (FPKM)*

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

object	A SingleCellExperiment object or a numeric matrix of counts.
effective_length	Numeric vector providing the effective length for each feature in object.
...	Further arguments to pass to calculateCPM.
subset_row	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` *Calculate QC metrics*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Cell-level QC metrics

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

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

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

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

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

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

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

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

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

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

Feature-level QC metrics

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

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

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

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

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

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

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

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

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

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

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

Compacted output

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

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

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

The nesting in scater_qc in the rowData is:

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

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

Renamed metrics

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

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

The old metric names have been removed in version 1.9.10.

Author(s)

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

Examples

```
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` *Calculate transcripts-per-million (TPM)*

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 comaprisons between features normalized with different size factors (e.g., spike-ins).

Value

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

Author(s)

Aaron Lun

See Also

`normalizeSCE`

Examples

```
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` *Return* `SingleCellExperiment` *with cells matching conditions.*

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

Value

An SingleCellExperiment 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_treat1 <- filter(example_sce, Treatment == "treat1")
```

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 gene.*

Description

Estimate the percentage of variance explained for each gene.

Usage

```
getExplanatoryPCs(object, variables = NULL, use_dimred = "PCA",
  chunk = 1000, ...)
```

Arguments

`object`	A SingleCellExperiment object containing expression values and per-cell experimental information.
`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.
`use_dimred`	String specifying the field in `reducedDims(object)` that contains the PCA results.
`chunk`	Argument passed to `getVarianceExplained`.
`...`	Arguments passed to `runPCA`.

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 echnical factors (e.g., batch effects, library size) can also be detected and removed prior to further analysis.

The function will attempt to extract existing PCA results in `object` via the `use_dimred` argument. If these are not available, it 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

`object`	A count matrix or SingleCellExperiment object containing counts.
`exprs_values`	A string indicating the assay of `object` containing the counts, if `object` is a SingleCellExperiment.
`subset_row`	A vector specifying whether the rows of `object` 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

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

Details

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

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

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

Value

A ggplot object.

Examples

```
library(ggplot2)

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

## Third plot
p3 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, colour = Diet)) +
   geom_density() +
```

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

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

---

mutate *Add new variables to* colData(object).

Description

Add new variables to colData(object).

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).

Value

An SingleCellExperiment 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 <- mutate(example_sce, is_quiescent = Cell_Cycle == "G0")
```

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

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

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

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]
```

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

Description

Compute normalised 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)

normalise(...)
```

Arguments

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

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).

Note that normalize is exactly the same as normalise.

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

x	A count matrix, with cells in the columns and genes in the rows.
size_factors	A numeric vector of size factors for all cells.
return_log	Logical scalar, should normalized values be returned on the log2 scale?
log_exprs_offset	Numeric scalar specifying the offset to add when log-transforming expression values.
centre_size_factors	Logical scalar indicating whether size fators should be centred.
subset_row	A vector specifying the subset of rows of x for which to return a result.

Details

This function is more memory-efficient than `t(t(x)/size_factors)`, and more generally applicable to different matrix classes than `sweep(x, 2, size_factors, "*")`.

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 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 expressiond 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, ...)

plotPhenoData(...)

plotCellData(...)
```

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, ...)

findImportantPCs(...)
```

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.

findImportantPCs is deprecated - it will simply pass all of its arguments to plotExplanatoryPCs.

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 metadata.
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 facet_wrap when multiple facets are generated.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

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

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

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

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
## 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

object	A SingleCellExperiment object.
freq_exprs	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 NULL, this defaults to "n_cells_by_counts" or equivalent for compacted data.
mean_exprs	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 NULL, this defaults to "mean_counts" or equivalent for compacted data.
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.
exprs_values	String specifying the assay used for the default freq_exprs and mean_exprs. This can be set to, e.g., "logcounts" so that freq_exprs defaults to "n_cells_by_logcounts".
by_show_single	Logical scalar specifying whether a single-level factor for controls should be used for colouring, see ?"scater-vis-var" for details.
show_smooth	Logical scalar, should a smoothed fit (through feature controls if available; all features otherwise) be shown on the plot? See geom_smooth for details.
show_se	Logical scalar, should the standard error be shown for a smoothed fit?
...	Further arguments passed to plotRowData.

Details

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

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 ?"scater-vis-var" for feature-level metadata; data in assays(object) will *not* 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 ?"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.
show_exprs_sd	Logical scalar indicating whether the standard deviation of expression values for each feature should be plotted.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

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

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
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

object	A SingleCellExperiment object.
features	A character vector of row names, a logical vector of integer vector of indices specifying rows of object to show in the heatmap.
columns	A vector specifying the subset of columns in object to show as columns in the heatmp. By default, all columns are used in their original order.
exprs_values	A string or integer scalar indicating which assay of object should be used as expression values for colouring in the heatmap.
center	A logical scalar indicating whether each row should have its mean expression centered at zero prior to plotting.
zlim	A numeric vector of length 2, specifying the upper and lower bounds for the expression values. This winsorizes the expression matrix prior to plotting (but after centering, if center=TRUE). If NULL, it defaults to the range of the expression matrix.
symmetric	A logical scalar specifying whether the default zlim should be symmetric around zero. If TRUE, the maximum absolute value of zlim will be computed and multiplied by c(-1, 1) to redefine zlim.
color	A vector of colours specifying the palette to use for mapping expression values to colours. This defaults to the default setting in pheatmap.
colour_columns_by	A list of values specifying how the columns should be annotated with colours. Each entry of the list can be of the form described by ?"scater-vis-var". A character vector can also be supplied and will be treated as a list of strings.
by_exprs_values	A string or integer scalar specifying which assay to obtain expression values from, for colouring of column-level data - see ?"scater-vis-var" for details.
by_show_single	Logical scalar specifying whether single-level factors should be used for column-level colouring, see ?"scater-vis-var" for details.
show_colnames	Logical scalar specifying whether column names should be shown, if available in object.
...	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 charcter format where a letter indicates the row on the plate and a numeric value indicates the column. Each cell has a plate position such as "A01", "B12", "K24" and so on. From these plate positions, the row is extracted as the letter, and the column as the numeric part. Alternatively, the row and column identities can be directly supplied by setting `plate_position` as a list of two factors.

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

```
## 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")
```

---

plotQC *Produce QC diagnostic plots*

Description

Produce QC diagnostic plots

Usage

```
plotQC(object, type = c("highest-expression", "find-pcs",
  "explanatory-variables", "exprs-freq-vs-mean"), ...)
```

Arguments

object	A SingleCellExperiment object.
type	String specifying the type of QC plot to compute.
...	Arguments passed to further plotting functions.

Details

This is a wrapper function to call a variety of different plotting functions. It has been deprecated in favour of calling the target functions specifically, which involves less typing anyway!

Value

A ggplot plot 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)

example_sce <- calculateQCMetrics(example_sce)
plotQC(example_sce, type="high", colour_cells_by="Mutation_Status")
```

---

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, ...,
  add_ticks = NULL)
```

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 ?"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.
add_ticks	Deprecated; logical scalar indicating whether ticks should be drawn on the axes corresponding to the location of each point.

Details

If ncomponents is a scalar and 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.

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 geom_boxplot when style="full".

Details

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

If style is "full", the usual **ggplot2** boxplot is created for each cell. Here, the box shows the inter-quartile range and whiskers extend no more than 1.5 * 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, ...)

plotFeatureData(...)
```

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 ?"scater-vis-var" for possible values.
shape_by	Specification of a row metadata field or a cell to shape by, see ?"scater-vis-var" for possible values.
size_by	Specification of a row metadata field or a cell 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 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 *Read sparse count matrix from file*

Description

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

Usage

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

Arguments

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

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

Details

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

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

Value

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

Author(s)

Aaron Lun

See Also

read.table, readMM

Examples

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

readSparseCounts(outfile)
```

---

readTxResults *Read transcript quantification data*

Description

Create a SingleCellExperiment object from pseudo-aligner results via **tximport**.

Usage

```
readTxResults(..., full_length = TRUE)

readKallistoResults(..., full_length = TRUE)

readSalmonResults(..., full_length = TRUE)
```

Arguments

...	Arguments to be passed to txiimport.
full_length	Logical scalar indicating whether the sequencing data is full-length (e.g., Smart-seq2) or end-biased, e.g., UMI-based protocols.

Details

If full_length=TRUE, counts are computed from the length-scaled TPMs. Otherwise, counts are not computed from the abundances.

readKallistoResults and readSalmonResults are simply wrappers around readTxResults with type pre-specified.

This function has now been deprecated in favour of txiimeta. The latter produces a SummarizedExperiment that is easily coerced into a SingleCellExperiment.

Value

A SingleCellExperiment containing the abundance, count and feature length information from the supplied samples.

Author(s)

Davis McCarthy and Aaron Lun

References

Soneson C, Love MI, Robinson MD (2015). Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Res.* 4, 1521.

Examples

```
library(tximportData)
dir <- system.file("extdata", package="tximportData")
samples <- read.table(file.path(dir,"samples.txt"), header=TRUE)
files <- file.path(dir,"salmon", samples$run, "quant.sf.gz")
names(files) <- paste0("sample",1:6)

# tx2gene links transcript IDs to gene IDs for summarization
tx2gene <- read.csv(file.path(dir, "tx2gene.gencode.v27.csv"))

sce <- readTxResults(files, type="salmon", tx2gene=tx2gene)
```

`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).

Description

Rename variables of colData(object).

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).

Value

An SingleCellExperiment 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 <- rename(example_sce, Cell_Phase = Cell_Cycle)
```

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

Description

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

Usage

```
runDiffusionMap(object, ncomponents = 2, ntop = 500,
  feature_set = NULL, exprs_values = "logcounts",
  scale_features = TRUE, use_dimred = NULL, n_dimred = NULL,
  rand_seed = 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.
rand_seed	Deprecated, numeric scalar that can be passed to set.seed to make the results reproducible.
...	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 *Perform PCA on cell-level data*

Description

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

Usage

```
runPCA(object, ncomponents = 2, method = c("prcomp", "irlba"),
  ntop = 500, exprs_values = "logcounts", feature_set = NULL,
  scale_features = TRUE, use_coldata = FALSE,
  selected_variables = NULL, detect_outliers = FALSE,
  rand_seed = NULL, ...)
```

Arguments

object	A SingleCellExperiment object.
ncomponents	Numeric scalar indicating the number of principal components to obtain.
method	String specifying how the PCA should be performed.
ntop	Numeric scalar specifying the number of most variable features to use for PCA.
exprs_values	Integer scalar or string indicating which assay of object should be used to obtain the expression values for the calculations.
feature_set	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 ntop argument if specified.
scale_features	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.
use_coldata	Logical scalar specifying whether the column data should be used instead of expression values to perform PCA.
selected_variables	List of strings or a character vector indicating which variables in colData(object) to use for PCA when use_coldata=TRUE. If a list, each entry can take the form described in ?"scater-vis-var".
detect_outliers	Logical scalar, should outliers be detected based on PCA coordinates generated from column-level metadata?
rand_seed	Deprecated, numeric scalar specifying the random seed when using method="irlba".
...	Additional arguments to pass to prcomp_irlba when method="irlba".

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 outliers 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, rand_seed = NULL,
  perplexity = min(50, floor(ncol(object)/5)), pca = TRUE,
  initial_dims = 50, ...)
```

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.
rand_seed	Deprecated, numeric scalar that can be passed to set.seed to make the results reproducible.
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.
...	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.

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, ...)
```

Arguments

object	A SingleCellExperiment object.
ncomponents	Numeric scalar indicating the number of UMAP dimensions to obtain.
ntop	Numeric scalar specifying the number of most variable features to use for UMAP.
feature_set	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 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.
...	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.

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 aavailable 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 aavailable a matrix of count data

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

`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")
```

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.

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 functions like `readTxResults` that account for differences in transcript lengths between 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)
```

`summariseExprsAcrossFeatures` *Summarise expression values across feature*

Description

Create a new `SingleCellExperiment` with counts summarised at a different feature level. A typical use would be to summarise transcript-level counts at gene level.

Usage

```
summariseExprsAcrossFeatures(object, exprs_values = "tpm",
  summarise_by = "feature_id", scaled_tpm_counts = TRUE,
  lib_size = NULL)
```

Arguments

`object`	an `SingleCellExperiment` object.
`exprs_values`	character string indicating which slot of the assayData from the `SingleCellExperiment` object should be used as expression values. Valid options are `'counts'` the counts slot, `'tpm'` the transcripts-per-million slot or `'fpkm'` the FPKM slot.
`summarise_by`	character string giving the column of `rowData(object)` that will be used as the features for which summarised expression levels are to be produced. Default is `'feature_id'`.
`scaled_tpm_counts`	logical, should feature-summarised counts be computed from summed TPM values scaled by total library size? This approach is recommended (see https://f1000research.com/articles/4-1521/v2), so the default is `TRUE` and it is applied if TPM values are available in the object.
`lib_size`	optional vector of numeric values of same length as the number of columns in the `SingleCellExperiment` object providing the total library size (e.g. "count of mapped reads") for each cell/sample.

Details

Only transcripts-per-million (TPM) and fragments per kilobase of exon per million reads mapped (FPKM) expression values should be aggregated across features. Since counts are not scaled by the length of the feature, expression in counts units are not comparable within a sample without adjusting for feature length. Thus, we cannot sum counts over a set of features to get the expression

of that set (for example, we cannot sum counts over transcripts to get accurate expression estimates for a gene). See the following link for a discussion of RNA-seq expression units by Harold Pimentel: https://haroldpimentel.wordpress.com/2014/05/08/what-the-fpkm-a-review-rna-seq-expression-units For more details about the effects of summarising transcript expression values at the gene level see Sonesen et al, 2016 (https://f1000research.com/articles/4-1521/v2).

Value

an SingleCellExperiment 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)

rd <- data.frame(gene_id = rownames(example_sce),
feature_id = paste("feature", rep(1:500, each = 4), sep = "_"))
rownames(rd) <- rownames(example_sce)
rowData(example_sce) <- rd

effective_length <- rep(c(1000, 2000), times = 1000)
tpm(example_sce) <- calculateTPM(example_sce, effective_length)

example_sceset_summarised <-
summariseExprsAcrossFeatures(example_sce, exprs_values = "tpm")
example_sceset_summarised <-
summariseExprsAcrossFeatures(example_sce, exprs_values = "counts")
```

---

uniquifyFeatureNames *Make feature names unique*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

A character vector of unique-ified feature names.

Author(s)

Aaron Lun

Examples

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

`updateSCESet` *Convert an SCESet object to a SingleCellExperiment object*

Description

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

Usage

```
updateSCESet(object)

toSingleCellExperiment(object)
```

Arguments

`object`	an `SCESet` object to be updated

Value

a `SingleCellExperiment` object

Examples

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

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

## End(Not run)
```

Index