Package 'scater'

November 7, 2025

Type Package

Version 1.39.0

Date 2025-03-07

License GPL-3

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

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

Depends SingleCellExperiment, scuttle, ggplot2

Imports stats, utils, methods, Matrix, BiocGenerics, S4Vectors, SummarizedExperiment, MatrixGenerics, SparseArray, DelayedArray, beachmat, BiocNeighbors, BiocSingular, BiocParallel, rlang, ggbeeswarm, viridis, Rtsne, RColorBrewer, RcppML, uwot, pheatmap, ggrepel, ggrastr

Suggests BiocStyle, DelayedMatrixStats, snifter, densvis, cowplot, biomaRt, knitr, scRNAseq, robustbase, rmarkdown, testthat, Biobase, scattermore

VignetteBuilder knitr

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

Encoding UTF-8

RoxygenNote 7.3.2

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

BugReports https://support.bioconductor.org/

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

git_branch devel

git_last_commit 4ba4420

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-06

Author Davis McCarthy [aut],
Kieran Campbell [aut],
Aaron Lun [aut, ctb],
Quin Wills [aut],
Vladimir Kiselev [ctb],
Felix G.M. Ernst [ctb],
Alan O'Callaghan [ctb, cre],
Yun Peng [ctb],
Leo Lahti [ctb] (ORCID: <https://orcid.org/0000-0001-5537-637X>),
Tuomas Borman [ctb] (ORCID: <https://orcid.org/0000-0002-8563-8884>)

Maintainer Alan O'Callaghan <alan.ocallaghan@outlook.com>

Contents

`annotateBMFeatures` *Get feature annotation information from Biomart*

Description

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

Usage

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

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

Arguments

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

Details

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

Value

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

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

Author(s)

Aaron Lun, based on code by Davis McCarthy

Examples

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

## End(Not run)
```

---

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

Description

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

Usage

```
bootstraps(object)

bootstraps(object) <- value

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

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

Arguments

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

Value

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

Author(s)

Davis McCarthy

Examples

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

---

calculateMDS *Perform MDS on cell-level data*

Description

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

Usage

```
calculateMDS(x, ...)

## S4 method for signature 'ANY'
calculateMDS(
  x,
  FUN = dist,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  scale = FALSE,
  transposed = FALSE,
  keep_dist = FALSE,
  ...
)

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

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

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

Arguments

x	For calculateMDS, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For runMDS, a SingleCellExperiment object.
...	For the calculateMDS generic, additional arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runMDS, additional arguments to pass to calculateMDS.
FUN	A function that accepts a numeric matrix as its first argument, where rows are samples and columns are features; and returns a distance structure such as that returned by dist or a full symmetric matrix containing the dissimilarities.
ncomponents	Numeric scalar indicating the number of MDS?g dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
keep_dist	Logical scalar indicating whether the dist object calculated by FUN should be stored as 'dist' attribute of the matrix returned/stored by calculateMDS or runMDS.
exprs_values	Alias to assay.type.
assay.type	Integer scalar or string indicating which assay of x contains the expression values.
dimred	String or integer scalar specifying the existing dimensionality reduction results to use.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified.
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function cmdscale is used internally to compute the MDS components with eig = TRUE. The eig and GOF fields of the object returned by cmdscale are stored as attributes "eig" and "GOF" of the MDS matrix calculated.

Value

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

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

Feature selection

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

The subset_row argument specifies the features to use for dimensionality reduction. The aim is to allow users to specify highly variable features to improve the signal/noise ratio, or to specify genes in a pathway of interest to focus on particular aspects of heterogeneity.

If subset_row=NULL, the ntop features with the largest variances are used instead. We literally compute the variances from the expression values without considering any mean-variance trend, so often a more considered choice of genes is possible, e.g., with **scran** functions. Note that the value of ntop is ignored if subset_row is specified.

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

Using reduced dimensions

If x is a SingleCellExperiment, the method can be applied on existing dimensionality reduction results in x by setting the dimred argument. This is typically used to run slower non-linear algorithms (t-SNE, UMAP) on the results of fast linear decompositions (PCA). We might also use this with existing reduced dimensions computed from *a priori* knowledge (e.g., gene set scores), where further dimensionality reduction could be applied to compress the data.

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

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

If x is a numeric matrix, setting transposed=TRUE will treat the rows as cells and the columns as the variables/diemnsions. This allows users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., ntop, subset_row and scale are ignored.

Using alternative Experiments

This section is relevant if `x` is a SingleCellExperiment and `altexp` is not `NULL`. In such cases, the method is run on data from an alternative SummarizedExperiment nested within `x`. This is useful for performing dimensionality reduction on other features stored in `altExp(x, altexp)`, e.g., antibody tags.

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

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

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

`cmdscale`, to perform the underlying calculations.

`dist` for the function used as default to calculate the `dist` object.

`plotMDS`, to quickly visualize the results.

Examples

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

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

`calculateNMF` *Perform NMF on cell-level data*

Description

Perform non-negative matrix factorization (NMF) for the cells, based on the data in a SingleCellExperiment object.

Usage

```
calculateNMF(x, ...)

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

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

## S4 method for signature 'SingleCellExperiment'
calculateNMF(
  x,
  ...,
  exprs_values = "logcounts",
  dimred = NULL,
  n_dimred = NULL,
  assay.type = exprs_values
)

runNMF(x, ..., altexp = NULL, name = "NMF")
```

Arguments

x	For `calculateNMF`, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For `runNMF`, a SingleCellExperiment object.
...	For the `calculateNMF` generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to `nmf`. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For `runNMF`, additional arguments to pass to `calculateNMF`.
ncomponents	Numeric scalar indicating the number of NMF dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?

transposed	Logical scalar, is x transposed with cells in rows?
exprs_values	Alias to assay.type.
assay.type	Integer scalar or string indicating which assay of x contains the expression values.
dimred	String or integer scalar specifying the existing dimensionality reduction results to use.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified.
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

The function nmf is used internally to compute the NMF. 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.

Value

For calculateNMF, a numeric matrix is returned containing the NMF coordinates for each cell (row) and dimension (column).

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

In both cases, the matrix will have the attribute "basis" containing the gene-by-factor basis matrix.

Feature selection

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

The subset_row argument specifies the features to use for dimensionality reduction. The aim is to allow users to specify highly variable features to improve the signal/noise ratio, or to specify genes in a pathway of interest to focus on particular aspects of heterogeneity.

If subset_row=NULL, the ntop features with the largest variances are used instead. We literally compute the variances from the expression values without considering any mean-variance trend, so often a more considered choice of genes is possible, e.g., with **scran** functions. Note that the value of ntop is ignored if subset_row is specified.

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

Using reduced dimensions

If x is a SingleCellExperiment, the method can be applied on existing dimensionality reduction results in x by setting the dimred argument. This is typically used to run slower non-linear algorithms (t-SNE, UMAP) on the results of fast linear decompositions (PCA). We might also use this

with existing reduced dimensions computed from *a priori* knowledge (e.g., gene set scores), where further dimensionality reduction could be applied to compress the data.

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

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

If `x` is a numeric matrix, setting `transposed=TRUE` will treat the rows as cells and the columns as the variables/diemnsions. This allows users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., `ntop`, `subset_row` and `scale` are ignored.

Using alternative Experiments

This section is relevant if `x` is a SingleCellExperiment and `altexp` is not `NULL`. In such cases, the method is run on data from an alternative SummarizedExperiment nested within `x`. This is useful for performing dimensionality reduction on other features stored in `altExp(x, altexp)`, e.g., antibody tags.

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

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

Author(s)

Aaron Lun

See Also

`nmf`, for the underlying calculations.

`plotNMF`, to quickly visualize the results.

Examples

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

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

calculatePCA *Perform PCA on expression data*

Description

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

Usage

```
calculatePCA(x, ...)

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

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

## S4 method for signature 'SingleCellExperiment'
calculatePCA(
  x,
  ...,
  exprs_values = "logcounts",
  dimred = NULL,
  n_dimred = NULL,
  assay.type = exprs_values
)

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

Arguments

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

...	For the calculatePCA generic, additional arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For runPCA, additional arguments to pass to calculatePCA.
ncomponents	Numeric scalar indicating the number of principal components to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
BSPARAM	A BiocSingularParam object specifying which algorithm should be used to perform the PCA.
BPPARAM	A BiocParallelParam object specifying whether the PCA should be parallelized.
exprs_values	Alias to assay.type.
assay.type	Integer scalar or string indicating which assay of x contains the expression values.
dimred	String or integer scalar specifying the existing dimensionality reduction results to use.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified.
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

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

Value

For calculatePCA, a numeric matrix of coordinates for each cell (row) in each of ncomponents PCs (column).

For runPCA, a SingleCellExperiment object is returned containing this matrix in reducedDims(..., name).

In both cases, the attributes of the PC coordinate matrix contain the following elements:

- "percentVar", the percentage of variance explained by each PC. This may not sum to 100 if not all PCs are reported.

- "varExplained", the actual variance explained by each PC.
- "rotation", the rotation matrix containing loadings for all genes used in the analysis and for each PC.

Feature selection

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

The subset_row argument specifies the features to use for dimensionality reduction. The aim is to allow users to specify highly variable features to improve the signal/noise ratio, or to specify genes in a pathway of interest to focus on particular aspects of heterogeneity.

If subset_row=NULL, the ntop features with the largest variances are used instead. We literally compute the variances from the expression values without considering any mean-variance trend, so often a more considered choice of genes is possible, e.g., with **scran** functions. Note that the value of ntop is ignored if subset_row is specified.

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

Using reduced dimensions

If x is a SingleCellExperiment, the method can be applied on existing dimensionality reduction results in x by setting the dimred argument. This is typically used to run slower non-linear algorithms (t-SNE, UMAP) on the results of fast linear decompositions (PCA). We might also use this with existing reduced dimensions computed from *a priori* knowledge (e.g., gene set scores), where further dimensionality reduction could be applied to compress the data.

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

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

If x is a numeric matrix, setting transposed=TRUE will treat the rows as cells and the columns as the variables/diemnsions. This allows users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., ntop, subset_row and scale are ignored.

Using alternative Experiments

This section is relevant if x is a SingleCellExperiment and altexp is not NULL. In such cases, the method is run on data from an alternative SummarizedExperiment nested within x. This is useful for performing dimensionality reduction on other features stored in altExp(x, altexp), e.g., antibody tags.

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

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

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

runPCA, for the underlying calculations.

plotPCA, to conveniently visualize the results.

Examples

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

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

---

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

Description

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

Usage

```
calculateTSNE(x, ...)

## S4 method for signature 'ANY'
calculateTSNE(
  x,
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  scale = FALSE,
  transposed = FALSE,
  perplexity = NULL,
  normalize = TRUE,
  theta = 0.5,
  num_threads = NULL,
  ...,
  external_neighbors = FALSE,
```

```
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam(),
  use_fitsne = FALSE,
  use_densvis = FALSE,
  dens_frac = 0.3,
  dens_lambda = 0.1
)

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

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

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

Arguments

x	For `calculateTSNE`, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For `runTSNE`, a SingleCellExperiment object.
...	For the `calculateTSNE` generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to `Rtsne`. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For `runTSNE`, additional arguments to pass to `calculateTSNE`.
ncomponents	Numeric scalar indicating the number of t-SNE dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
perplexity	Numeric scalar defining the perplexity parameter, see ?`Rtsne` for more details.
normalize	Logical scalar indicating if input values should be scaled for numerical precision, see `normalize_input`.

theta — Numeric scalar specifying the approximation accuracy of the Barnes-Hut algorithm, see Rtsne for details.

num_threads — Integer scalar specifying the number of threads to use in Rtsne. If NULL and BPPARAM is a MulticoreParam, it is set to the number of workers in BPPARAM; otherwise, the Rtsne defaults are used.

external_neighbors — Logical scalar indicating whether a nearest neighbors search should be computed externally with findKNN.

BNPARAM — A BiocNeighborParam object specifying the neighbor search algorithm to use when external_neighbors=TRUE.

BPPARAM — A BiocParallelParam object specifying how the neighbor search should be parallelized when external_neighbors=TRUE.

use_fitsne — Logical scalar indicating whether fitsne should be used to perform t-SNE.

use_densvis — Logical scalar indicating whether densne should be used to perform density-preserving t-SNE.

dens_frac, dens_lambda — See densne

exprs_values — Alias to assay.type.

assay.type — Integer scalar or string indicating which assay of x contains the expression values.

pca — Logical scalar indicating whether a PCA step should be performed inside Rtsne.

dimred — String or integer scalar specifying the existing dimensionality reduction results to use.

n_dimred — Integer scalar or vector specifying the dimensions to use if dimred is specified.

altexp — String or integer scalar specifying an alternative experiment containing the input data.

name — String specifying the name to be used to store the result in the reducedDims of the output.

Details

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

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

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

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

Value

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

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

Feature selection

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

The `subset_row` argument specifies the features to use for dimensionality reduction. The aim is to allow users to specify highly variable features to improve the signal/noise ratio, or to specify genes in a pathway of interest to focus on particular aspects of heterogeneity.

If `subset_row=NULL`, the `ntop` features with the largest variances are used instead. We literally compute the variances from the expression values without considering any mean-variance trend, so often a more considered choice of genes is possible, e.g., with **scran** functions. Note that the value of `ntop` is ignored if `subset_row` is specified.

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

Using reduced dimensions

If `x` is a SingleCellExperiment, the method can be applied on existing dimensionality reduction results in `x` by setting the `dimred` argument. This is typically used to run slower non-linear algorithms (t-SNE, UMAP) on the results of fast linear decompositions (PCA). We might also use this with existing reduced dimensions computed from *a priori* knowledge (e.g., gene set scores), where further dimensionality reduction could be applied to compress the data.

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

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

If `x` is a numeric matrix, setting `transposed=TRUE` will treat the rows as cells and the columns as the variables/diemnsions. This allows users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., `ntop`, `subset_row` and `scale` are ignored.

Using alternative Experiments

This section is relevant if `x` is a SingleCellExperiment and `altexp` is not `NULL`. In such cases, the method is run on data from an alternative SummarizedExperiment nested within `x`. This is useful for performing dimensionality reduction on other features stored in `altExp(x, altexp)`, e.g., antibody tags.

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

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

Author(s)

Aaron Lun, based on code by Davis McCarthy

References

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

See Also

`Rtsne`, for the underlying calculations.

`plotTSNE`, to quickly visualize the results.

Examples

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

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

`calculateUMAP` *Perform UMAP on cell-level data*

Description

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

Usage

```
calculateUMAP(x, ...)

## S4 method for signature 'ANY'
calculateUMAP(
  x,
  ncomponents = 2,
  ntop = 500,
```

```
  subset_row = NULL,
  scale = FALSE,
  transposed = FALSE,
  pca = if (transposed) NULL else 50,
  n_neighbors = 15,
  n_threads = bpnworkers(BPPARAM),
  ...,
  external_neighbors = FALSE,
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam(),
  use_densvis = FALSE,
  dens_frac = 0.3,
  dens_lambda = 0.1
)

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

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

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

Arguments

x	For `calculateUMAP`, a numeric matrix of log-expression values where rows are features and columns are cells. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix. For `runTSNE`, a SingleCellExperiment object containing such a matrix.
...	For the `calculateUMAP` generic, additional arguments to pass to specific methods. For the ANY method, additional arguments to pass to `umap`. For the SummarizedExperiment and SingleCellExperiment methods, additional arguments to pass to the ANY method. For `runUMAP`, additional arguments to pass to `calculateUMAP`.
ncomponents	Numeric scalar indicating the number of UMAP dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.

scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
pca	Integer scalar specifying how many PCs should be used as input into the UMAP algorithm. By default, no PCA is performed if the input is a dimensionality reduction result.
n_neighbors	Integer scalar, number of nearest neighbors to identify when constructing the initial graph.
n_threads	Integer scalar specifying the number of threads to use in umap. If NULL and BPPARAM is a MulticoreParam, it is set to the number of workers in BPPARAM; otherwise, the umap defaults are used.
external_neighbors	Logical scalar indicating whether a nearest neighbors search should be computed externally with findKNN.
BNPARAM	A BiocNeighborParam object specifying the neighbor search algorithm to use when external_neighbors=TRUE.
BPPARAM	A BiocParallelParam object specifying whether the PCA should be parallelized.
use_densvis	Logical scalar indicating whether densne should be used to perform density-preserving t-SNE.
dens_frac, dens_lambda	See densne
exprs_values	Alias to assay.type.
assay.type	Integer scalar or string indicating which assay of x contains the expression values.
dimred	String or integer scalar specifying the existing dimensionality reduction results to use.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified.
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

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

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

Value

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

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

Feature selection

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

The subset_row argument specifies the features to use for dimensionality reduction. The aim is to allow users to specify highly variable features to improve the signal/noise ratio, or to specify genes in a pathway of interest to focus on particular aspects of heterogeneity.

If subset_row=NULL, the ntop features with the largest variances are used instead. We literally compute the variances from the expression values without considering any mean-variance trend, so often a more considered choice of genes is possible, e.g., with **scran** functions. Note that the value of ntop is ignored if subset_row is specified.

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

Using reduced dimensions

If x is a SingleCellExperiment, the method can be applied on existing dimensionality reduction results in x by setting the dimred argument. This is typically used to run slower non-linear algorithms (t-SNE, UMAP) on the results of fast linear decompositions (PCA). We might also use this with existing reduced dimensions computed from *a priori* knowledge (e.g., gene set scores), where further dimensionality reduction could be applied to compress the data.

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

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

If x is a numeric matrix, setting transposed=TRUE will treat the rows as cells and the columns as the variables/diemnsions. This allows users to manually pass in dimensionality reduction results without needing to wrap them in a SingleCellExperiment. As such, no feature selection or standardization is performed, i.e., ntop, subset_row and scale are ignored.

Using alternative Experiments

This section is relevant if x is a SingleCellExperiment and altexp is not NULL. In such cases, the method is run on data from an alternative SummarizedExperiment nested within x. This is useful for performing dimensionality reduction on other features stored in altExp(x, altexp), e.g., antibody tags.

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

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

Author(s)

Aaron Lun

References

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

See Also

umap, for the underlying calculations.

plotUMAP, to quickly visualize the results.

Examples

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

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

---

defunct *Defunct functions*

Description

Functions that have passed on to the function afterlife. Their successors are also listed.

Usage

```
calculateQCMetrics(...)

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

centreSizeFactors(...)

calculateDiffusionMap(x, ...)

## S4 method for signature 'ANY'
calculateDiffusionMap(x, ...)

runDiffusionMap(...)

multiplot(...)
```

Arguments

object, x, ...	Ignored arguments.

Details

calculateQCMetrics is succeeded by perCellQCMetrics and perFeatureQCMetrics.

normalize is succeeded by logNormCounts.

centreSizeFactors has no replacement - the **SingleCellExperiment** is removing support for multiple size factors, so this function is now trivial.

runDiffusionMap and calculateDiffusionMap have no replacement. **destiny** is no longer on Bioconductor. You can calculate a diffusion map yourself, and add it to a reducedDim field, if you so wish.

Value

All functions error out with a defunct message pointing towards its descendent (if available).

Author(s)

Aaron Lun

Examples

```
try(calculateQCMetrics())
```

---

getExplanatoryPCs *Per-PC variance explained by a variable*

Description

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

Usage

```
getExplanatoryPCs(x, dimred = "PCA", n_dimred = 10, ...)
```

Arguments

x	A SingleCellExperiment object containing dimensionality reduction results.
dimred	String or integer scalar specifying the field in reducedDims(x) that contains the PCA results.
n_dimred	Integer scalar specifying the number of the top principal components to use.
...	Additional arguments passed to getVarianceExplained.

Details

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

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

Value

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

Author(s)

Aaron Lun

See Also

`plotExplanatoryPCs`, to plot the results.

`getVarianceExplained`, to compute the variance explained.

Examples

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

r2mat <- getExplanatoryPCs(example_sce)
```

---

getVarianceExplained *Per-gene variance explained by a variable*

Description

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

Usage

```
getVarianceExplained(x, ...)

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

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

Arguments

x	A numeric matrix of expression values, usually log-transformed and normalized. Alternatively, a SummarizedExperiment containing such a matrix.
...	For the generic, arguments to be passed to specific methods. For the SummarizedExperiment method, arguments to be passed to the ANY method.
variables	A DataFrame or data.frame containing one or more variables of interest. This should have number of rows equal to the number of columns in x. For the SummarizedExperiment method, this can also be a character vector specifying column names of colData(x) to use; or NULL, in which case all columns in colData(x) are used.
subset_row	A vector specifying the subset of rows of x for which to return a result.
BPPARAM	A BiocParallelParam object specifying whether the calculations should be parallelized.
exprs_values	Alias for assay.type.
assay.type	String or integer scalar specifying the expression values for which to compute the variance (also an alias exprs_value is accepted).

Details

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

Value

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

Author(s)

Aaron Lun

See Also

getExplanatoryPCs, which calls this function.

plotExplanatoryVariables, to plot the results.

Examples

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

r2mat <- getVarianceExplained(example_sce)
```

---

ggcells *Create a ggplot from a SingleCellExperiment*

Description

Create a base ggplot object from a SingleCellExperiment, the contents of which can be directly referenced in subsequent layers without prior specification.

Usage

```
ggcells(
  x,
  mapping = aes(),
  features = NULL,
  exprs_values = "logcounts",
  use_dimred = TRUE,
  use_altexps = FALSE,
  prefix_altexps = FALSE,
  check_names = TRUE,
  extract_mapping = TRUE,
  assay.type = exprs_values,
  ...
)

ggfeatures(
  x,
  mapping = aes(),
  cells = NULL,
  exprs_values = "logcounts",
  check_names = TRUE,
  extract_mapping = TRUE,
  assay.type = exprs_values,
  ...
)
```

Arguments

x	A SingleCellExperiment object. This is expected to have row names for ggcells and column names for ggfeatures.
mapping	A list containing aesthetic mappings, usually the output of aes or related functions.
features	Character vector specifying the features for which to extract expression profiles across cells. May also include features in alternative Experiments if permitted by use.altexps.
exprs_values, use_dimred, use_altexps, prefix_altexps, check_names	Soft-deprecated equivalents of the arguments described above.
extract_mapping	Logical scalar indicating whether features or cells should be automatically expanded to include variables referenced in mapping.
assay.type	String or integer scalar specifying the expression values for which to compute the variance (also an alias exprs_value is accepted).
...	Further arguments to pass to ggplot.
cells	Character vector specifying the features for which to extract expression profiles across cells.

Details

These functions generate a data.frame from the contents of a SingleCellExperiment and pass it to ggplot. Rows, columns or metadata fields in the x can then be referenced in subsequent **ggplot2** commands.

ggcells treats cells as the data values so users can reference row names of x (if provided in features), column metadata variables and dimensionality reduction results. They can also reference row names and metadata variables for alternative Experiments.

ggfeatures treats features as the data values so users can reference column names of x (if provided in cells) and row metadata variables.

If mapping is supplied, the function will automatically expand features or cells for any features or cells requested in the mapping. This is convenient as features/cells do not have to specified twice (once in data.frame construction and again in later geom or stat layers). Developers may wish to turn this off with extract_mapping=FALSE for greater control.

Value

A ggplot object containing the specified contents of x.

Author(s)

Aaron Lun

See Also

makePerCellDF and makePerFeatureDF, for the construction of the data.frame.

Examples

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

ggcells(example_sce, aes(x=PCA.1, y=PCA.2, colour=Gene_0001)) +
    geom_point()

ggcells(example_sce, aes(x=Mutation_Status, y=Gene_0001)) +
    geom_violin() +
    facet_wrap(~Cell_Cycle)

rowData(example_sce)$GC <- runif(nrow(example_sce))
ggfeatures(example_sce, aes(x=GC, y=Cell_001)) +
    geom_point() +
    stat_smooth()
```

---

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

Description

Counting the number of non-zero counts in each row (per feature) or column (per cell).

Usage

```
nexprs(x, ...)

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

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

Arguments

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

...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
byrow	Logical scalar indicating whether to count the number of detected cells per feature. If FALSE, the function will count the number of detected features per cell.
detection_limit	Numeric scalar providing the value above which observations are deemed to be expressed.
subset_row	Logical, integer or character vector indicating which rows (i.e. features) to use.
subset_col	Logical, integer or character vector indicating which columns (i.e., cells) to use.
BPPARAM	A BiocParallelParam object specifying whether the calculations should be parallelized. Only relevant when x is a DelayedMatrix.
exprs_values	Alias for assay.type.
assay.type	String or integer specifying the assay of x to obtain the count matrix from (also the alias exprs_values is accepted for this argument).

Value

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

Author(s)

Aaron Lun

See Also

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

Examples

```
example_sce <- mockSCE()

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

---

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

Description

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

Usage

```
norm_exprs(object)

norm_exprs(object) <- value

stand_exprs(object)

stand_exprs(object) <- value

fpkm(object)

fpkm(object) <- value
```

Arguments

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

Value

a matrix of normalised expression data

a matrix of standardised expressiond data

a matrix of FPKM values

A matrix of numeric, integer or logical values.

Author(s)

Davis McCarthy

Examples

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

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

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

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

cpm(example_sce) <- calculateCPM(example_sce)

fpkm(example_sce)
```

plotColData *Plot column metadata*

Description

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

Usage

```
plotColData(
  object,
  y,
  x = NULL,
  colour_by = color_by,
  shape_by = NULL,
  size_by = NULL,
  order_by = NULL,
  by_exprs_values = "logcounts",
  other_fields = list(),
  swap_rownames = NULL,
  color_by = NULL,
  point_fun = NULL,
  scattermore = FALSE,
  bins = NULL,
  summary_fun = "sum",
  hex = FALSE,
  by.assay.type = by_exprs_values,
  ...
)
```

Arguments

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

order_by	Specification of a column metadata field or a feature to order points by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	Alias for by.assay.type.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
color_by	Alias to colour_by.
point_fun	Function used to create a geom that shows individual cells. Should take ... args and return a ggplot2 geom. For example, point_fun=function(...) geom_quasirandom(...).
scattermore	Logical, whether to use the scattermore package to greatly speed up plotting a large number of cells. Use point_size = 0 for the most performance gain.
bins	Number of bins, can be different in x and y, to bin and summarize the points and their values, to avoid overplotting. If NULL (default), then the points are plotted without binning. Only used when both x and y are numeric.
summary_fun	Function to summarize the feature value of each point (e.g. gene expression of each cell) when the points binned, defaults to sum. Can be either the name of the function or the function itself.
hex	Logical, whether to use geom_hex.
by.assay.type	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see ?retrieveCellInfo for details (also alias by_exprs_values is accepted for this argument).
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

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

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

Value

A ggplot object.

Note

Arguments shape_by and size_by are ignored when scattermore = TRUE. Using scattermore is only recommended for very large datasets to speed up plotting. Small point size is also recommended. For larger point size, the point shape may be distorted. Also, when scattermore = TRUE, the point_size argument works differently.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

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

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

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

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

plotColData(example_sce, y = "detected",
   x = "Cell_Cycle", colour_by = "Mutation_Status")
# With scattermore
plotColData(example_sce, x = "sum", y = "detected", scattermore = TRUE,
   point_size = 2)
# Bin to show point density
plotColData(example_sce, x = "sum", y = "detected", bins = 10)
# Bin to summarize value (default is sum)
plotColData(example_sce, x = "sum", y = "detected", bins = 10, colour_by = "total")
```

---

plotDots *Create a dot plot of expression values*

Description

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

Usage

```
plotDots(
  object,
  features,
  group = NULL,
  block = NULL,
  exprs_values = "logcounts",
  detection_limit = 0,
  zlim = NULL,
```

```
  colour = color,
  color = NULL,
  max_detected = NULL,
  other_fields = list(),
  by_exprs_values = exprs_values,
  swap_rownames = NULL,
  center = FALSE,
  scale = FALSE,
  assay.type = exprs_values,
  by.assay.type = by_exprs_values
)
```

Arguments

object	A SingleCellExperiment object.
features	A character (or factor) vector of row names, a logical vector, or integer vector of indices specifying rows of object to visualize. When using character or integer vectors, the ordering specified by the user is retained. When using factor vectors, ordering is controlled by the factor levels.
group	String specifying the field of colData(object) containing the grouping factor, e.g., cell types or clusters. Alternatively, any value that can be used in the by argument to retrieveCellInfo.
block	String specifying the field of colData(object) containing a blocking factor (e.g., batch of origin). Alternatively, any value that can be used in the by argument to retrieveCellInfo.
exprs_values	Alias to assay.type.
detection_limit	Numeric scalar providing the value above which observations are deemed to be expressed.
zlim	A numeric vector of length 2, specifying the upper and lower bounds for colour mapping of expression values. Values outside this range are set to the most extreme colour. If NULL, it defaults to the range of the expression matrix. If center=TRUE, this defaults to the range of the centered expression matrix, made symmetric around zero.
colour	A vector of colours specifying the palette to use for increasing expression. This defaults to viridis if center=FALSE, and the the "RdYlBu" colour palette from brewer.pal otherwise.
color	Alias to colour.
max_detected	Numeric value specifying the cap on the proportion of detected expression values.
other_fields	Additional feature-based fields to include in the data.frame, see ?"scater-plot-args" for details. Note that any AsIs vectors or data.frames must be of length equal to nrow(object), not features.
by_exprs_values	Alias for by.assay.type.

swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
center	A logical scalar indicating whether each feature should have its mean expression (specifically, the mean of averages across all groups) centered at zero prior to plotting.
scale	A logical scalar specifying whether each row should have its average expression values scaled to unit variance prior to plotting.
assay.type	A string or integer scalar indicating which assay of object should be used as expression values.
by.assay.type	A string or integer scalar specifying which assay to obtain expression values from, for entries of other_fields. Also alias by_exprs_values is accepted as argument name.

Details

This implements a **Seurat**-style "dot plot" that creates a dot for each feature (row) in each group of cells (column). The proportion of detected expression values and the average expression for each feature in each group of cells is visualized efficiently using the size and colour, respectively, of each dot. If block is specified, batch-corrected averages and proportions for each group are computed with correctGroupSummary.

Some caution is required during interpretation due to the difficulty of simultaneously interpreting both size and colour. For example, if we coloured by z-score on a conventional blue-white-red colour axis, a gene that is downregulated in a group of cells would show up as a small blue dot. If the background colour was also white, this could be easily mistaken for a gene that is not downregulated at all. We suggest choosing a colour scale that remains distinguishable from the background colour at all points. Admittedly, that is easier said than done as many colour scales will approach a lighter colour at some stage, so some magnifying glasses may be required.

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

Value

A ggplot object containing a dot plot.

Author(s)

Aaron Lun

See Also

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

Examples

```
sce <- mockSCE()
sce <- logNormCounts(sce)
```

```
plotDots(sce, features=rownames(sce)[1:10], group="Cell_Cycle")
plotDots(sce, features=rownames(sce)[1:10], group="Cell_Cycle", center=TRUE)
plotDots(sce, features=rownames(sce)[1:10], group="Cell_Cycle", scale=TRUE)
plotDots(sce, features=rownames(sce)[1:10], group="Cell_Cycle", center=TRUE, scale=TRUE)

plotDots(sce, features=rownames(sce)[1:10], group="Treatment", block="Cell_Cycle")
```

`plotExplanatoryPCs` *Plot the explanatory PCs for each variable*

Description

Plot the explanatory PCs for each variable

Usage

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

Arguments

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

Details

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

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

Value

A ggplot object.

Examples

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

plotExplanatoryPCs(example_sce)
```

`plotExplanatoryVariables` *Plot explanatory variables ordered by percentage of variance explained*

Description

Plot explanatory variables ordered by percentage of variance explained

Usage

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

Arguments

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

Details

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

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

Value

A ggplot object.

Examples

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

---

plotExpression *Plot expression values for all cells*

Description

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

Usage

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

```
  color_by = NULL,
  feature_colors = TRUE,
  point_fun = NULL,
  assay.type = exprs_values,
  scattermore = FALSE,
  bins = NULL,
  summary_fun = "sum",
  hex = FALSE,
  by.assay.type = by_exprs_values,
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and other metadata.
features	A character vector or a list specifying the features to plot. If a list is supplied, each entry of the list can be a string, an AsIs-wrapped vector or a data.frame - see ?retrieveCellInfo.
x	Specification of a column metadata field or a feature to show on the x-axis, see the by argument in ?retrieveCellInfo for possible values.
exprs_values	Alias to assay.type.
log2_values	Logical scalar, specifying whether the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes).
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.
size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
order_by	Specification of a column metadata field or a feature to order points by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	Alias to by.assay.type.
xlab	String specifying the label for x-axis. If NULL (default), x will be used as the x-axis label.
feature_colours	Logical scalar indicating whether violins should be coloured by feature when x and colour_by are not specified and one_facet=TRUE.
one_facet	Logical scalar indicating whether grouped violin plots for multiple features should be put onto one facet. Only relevant when x=NULL.
ncol	Integer scalar, specifying the number of columns to be used for the panels of a multi-facet plot.
scales	String indicating whether should multi-facet scales be fixed ("fixed"), free ("free"), or free in one dimension ("free_x", "free_y"). Passed to the scales argument in the facet_wrap when multiple facets are generated.

other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
color_by	Alias to colour_by.
feature_colors	Alias to feature_colours.
point_fun	Function used to create a geom that shows individual cells. Should take ... args and return a ggplot2 geom. For example, point_fun=function(...) geom_quasirandom(...).
assay.type	A string or integer scalar specifying which assay in assays(object) to obtain expression values from. Also the alias assay.type is accepted.
scattermore	Logical, whether to use the scattermore package to greatly speed up plotting a large number of cells. Use point_size = 0 for the most performance gain.
bins	Number of bins, can be different in x and y, to bin and summarize the points and their values, to avoid overplotting. If NULL (default), then the points are plotted without binning. Only used when both x and y are numeric.
summary_fun	Function to summarize the feature value of each point (e.g. gene expression of each cell) when the points binned, defaults to sum. Can be either the name of the function or the function itself.
hex	Logical, whether to use geom_hex.
by.assay.type	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the assay.type argument in ?retrieveCellInfo. Also the alias by.assay.type is accepted.
...	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.

Note

Arguments `shape_by` and `size_by` are ignored when `scattermore = TRUE`. Using `scattermore` is only recommended for very large datasets to speed up plotting. Small point size is also recommended. For larger point size, the point shape may be distorted. Also, when `scattermore = TRUE`, the `point_size` argument works differently.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

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

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

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

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

## use boxplot as well as violin plot
plotExpression(example_sce, rownames(example_sce)[1:6],
    show_boxplot = TRUE, show_violin = FALSE)

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

# Use scattermore
plotExpression(example_sce, "Gene_0001", x = "Gene_0100", scattermore = TRUE,
    point_size = 2)
# Bin to show point density
plotExpression(example_sce, "Gene_0001", x = "Gene_0100", bins = 10)
# Bin to summarize values (default is sum but can be changed with summary_fun)
plotExpression(example_sce, "Gene_0001", x = "Gene_0100", bins = 10,
    colour_by = "Gene_0002", summary_fun = "mean")
```

plotGroupedHeatmap *Plot heatmap of group-level expression averages*

Description

Create a heatmap of average expression values for each group of cells and specified features in a SingleCellExperiment object.

Usage

```
plotGroupedHeatmap(
  object,
  features,
  group,
  block = NULL,
  columns = NULL,
  exprs_values = "logcounts",
  center = FALSE,
  scale = FALSE,
  zlim = NULL,
  colour = color,
  swap_rownames = NULL,
  color = NULL,
  assay.type = exprs_values,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
features	A character (or factor) vector of row names, a logical vector, or integer vector of indices specifying rows of object to visualize. When using character or integer vectors, the ordering specified by the user is retained. When using factor vectors, ordering is controlled by the factor levels.
group	String specifying the field of colData(object) containing the grouping factor, e.g., cell types or clusters. Alternatively, any value that can be used in the by argument to retrieveCellInfo.
block	String specifying the field of colData(object) containing a blocking factor (e.g., batch of origin). Alternatively, any value that can be used in the by argument to retrieveCellInfo.
columns	A vector specifying the subset of columns in object to use when computing averages.
exprs_values	Alias to assay.type.
center	A logical scalar indicating whether each feature should have its mean expression (specifically, the mean of averages across all groups) centered at zero prior to plotting.

scale	A logical scalar specifying whether each row should have its average expression values scaled to unit variance prior to plotting.
zlim	A numeric vector of length 2, specifying the upper and lower bounds for colour mapping of expression values. Values outside this range are set to the most extreme colour. If NULL, it defaults to the range of the expression matrix. If center=TRUE, this defaults to the range of the centered expression matrix, made symmetric around zero.
colour	A vector of colours specifying the palette to use for increasing expression. This defaults to viridis if center=FALSE, and the the "RdYlBu" colour palette from brewer.pal otherwise.
swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
color	Alias to colour.
assay.type	A string or integer scalar indicating which assay of object should be used as expression values.
...	Additional arguments to pass to pheatmap.

Details

This function shows the average expression values for each group of cells on a heatmap, as defined using the group factor. A per-group visualization can be preferable to a per-cell visualization when dealing with large number of cells or groups with different size. If block is also specified, the block effect is regressed out of the averages with correctGroupSummary prior to visualization.

Setting center=TRUE is useful for examining log-fold changes of each group's expression profile from the average across all groups. 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, for the underlying function.

plotHeatmap, for a per-cell heatmap.

Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
example_sce$Group <- paste0(example_sce$Treatment, "+", example_sce$Mutation_Status)

plotGroupedHeatmap(example_sce, features=rownames(example_sce)[1:10],
    group="Group")

plotGroupedHeatmap(example_sce, features=rownames(example_sce)[1:10],
    group="Group", center=TRUE)

plotGroupedHeatmap(example_sce, features=rownames(example_sce)[1:10],
    group="Group", block="Cell_Cycle", center=TRUE)
```

---

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,
  scale = FALSE,
  zlim = NULL,
  colour = color,
  color = NULL,
  colour_columns_by = color_columns_by,
  color_columns_by = NULL,
  column_annotation_colours = column_annotation_colors,
  column_annotation_colors = list(),
  row_annotation_colours = row_annotation_colors,
  row_annotation_colors = list(),
  colour_rows_by = color_rows_by,
  color_rows_by = NULL,
  order_columns_by = NULL,
  by_exprs_values = exprs_values,
  show_colnames = FALSE,
  cluster_cols = is.null(order_columns_by),
```

```
  swap_rownames = NULL,
  assay.type = exprs_values,
  by.assay.type = by_exprs_values,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
features	A character (or factor) vector of row names, a logical vector, or integer vector of indices specifying rows of object to visualize. When using character or integer vectors, the ordering specified by the user is retained. When using factor vectors, ordering is controlled by the factor levels.
columns	A vector specifying the subset of columns in object to show as columns in the heatmap. Also specifies the column order if cluster_cols=FALSE and order_columns_by=NULL. By default, all columns are used.
exprs_values	Alias to assay.type.
center	A logical scalar indicating whether each feature should have its mean expression centered at zero prior to plotting.
scale	A logical scalar specifying whether each feature should have its expression values scaled to have unit variance prior to plotting.
zlim	A numeric vector of length 2, specifying the upper and lower bounds for colour mapping of expression values. Values outside this range are set to the most extreme colour. If NULL, it defaults to the range of the expression matrix. If center=TRUE, this defaults to the range of the centered expression matrix, made symmetric around zero.
colour	A vector of colours specifying the palette to use for increasing expression. This defaults to viridis if center=FALSE, and the the "RdYlBu" colour palette from brewer.pal otherwise.
color, color_columns_by, column_annotation_colors, color_rows_by, row_annotation_colors	Aliases to color, color_columns_by, column_annotation_colors, color_rows_by, row_annotation_colors.
colour_columns_by	A list of values specifying how the columns should be annotated with colours. Each entry of the list can be any acceptable input to the by argument in ?retrieveCellInfo. A character vector can also be supplied and will be treated as a list of strings.
column_annotation_colours	A named list of colour scales to be used for the column annotations specified in colour_columns_by. Names should be character values present in colour_columns_by, If a colour scale is not specified for a particular annotation, a default colour scale is chosen. The full list of colour maps is passed to pheatmap as the annotation_colours argument.
row_annotation_colours	Similar to column_annotation_colours but relating to row annotation rather than column annotation.

colour_rows_by Similar to colour_columns_by but for rows rather than columns. Each entry of the list can be any acceptable input to the by argument in ?retrieveFeatureInfo.

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

by_exprs_values
Alias to by.assay.type.

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

swap_rownames Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.

assay.type A string or integer scalar indicating which assay of object should be used as expression values.

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

Details

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

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

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

Value

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

Author(s)

Aaron Lun

See Also

pheatmap

Examples

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

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

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

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

---

plotHighestExprs *Plot the highest expressing features*

Description

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

Usage

```
plotHighestExprs(
  object,
  n = 50,
  colour_cells_by = color_cells_by,
  drop_features = NULL,
  exprs_values = "counts",
  by_exprs_values = exprs_values,
  feature_names_to_plot = NULL,
  as_percentage = TRUE,
  swap_rownames = NULL,
  color_cells_by = NULL,
  assay.type = exprs_values,
  by.assay.type = by_exprs_values
)
```

Arguments

object	A SingleCellExperiment object.
n	A numeric scalar specifying the number of the most expressed features to show.
colour_cells_by	Specification of a column metadata field or a feature to colour by, see ?retrieveCellInfo for possible values.

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 Alias to assay.type.

by_exprs_values
Alias to by.assay.type.

feature_names_to_plot
String specifying which row-level metadata column contains the feature names. Alternatively, an AsIs-wrapped vector or a data.frame, see ?retrieveFeatureInfo for possible values. Default is NULL, in which case rownames(object) are used.

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

swap_rownames Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.

color_cells_by Alias to colour_cells_by.

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

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

Details

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

Value

A ggplot object.

Examples

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

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

`plotPlatePosition` *Plot cells in plate positions*

Description

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

Usage

```
plotPlatePosition(
  object,
  plate_position = NULL,
  colour_by = color_by,
  size_by = NULL,
  shape_by = NULL,
  order_by = NULL,
  by_exprs_values = "logcounts",
  add_legend = TRUE,
  theme_size = 24,
  point_alpha = 0.6,
  point_size = 24,
  point_shape = 19,
  other_fields = list(),
  swap_rownames = NULL,
  color_by = NULL,
  by.assay.type = by_exprs_values
)
```

Arguments

`object`	A SingleCellExperiment object.
`plate_position`	A character vector specifying the plate position for each cell (e.g., A01, B12, and so on, where letter indicates row and number indicates column). If `NULL`, the function will attempt to extract this from `object$plate_position`. Alternatively, a list of two factors (`"row"` and `"column"`) can be supplied, specifying the row (capital letters) and column (integer) for each cell in `object`.
`colour_by`	Specification of a column metadata field or a feature to colour by, see the by argument in ?`retrieveCellInfo` for possible values.
`size_by`	Specification of a column metadata field or a feature to size by, see the by argument in ?`retrieveCellInfo` for possible values.
`shape_by`	Specification of a column metadata field or a feature to shape by, see the by argument in ?`retrieveCellInfo` for possible values.
`order_by`	Specification of a column metadata field or a feature to order points by, see the by argument in ?`retrieveCellInfo` for possible values.

by_exprs_values
: Alias for by.assay.type.

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.

point_shape
: An integer, or a string specifying the shape of the points. See ?"scater-plot-args" for details.

other_fields
: Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.

swap_rownames
: Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.

color_by
: Alias to colour_by.

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

Details

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

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

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

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

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

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

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

---

plotReducedDim *Plot reduced dimensions*

Description

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

Usage

```
plotReducedDim(
  object,
  dimred,
  ncomponents = 2,
  percentVar = NULL,
  colour_by = color_by,
  shape_by = NULL,
  size_by = NULL,
  order_by = NULL,
  by_exprs_values = "logcounts",
  text_by = NULL,
  text_size = 5,
  text_colour = text_color,
  label_format = c("%s %i", " (%i%%)"),
  other_fields = list(),
  text_color = "black",
  color_by = NULL,
  swap_rownames = NULL,
  point.padding = NA,
  force = 1,
  rasterise = FALSE,
  scattermore = FALSE,
  bins = NULL,
  summary_fun = "sum",
  hex = FALSE,
  by.assay.type = by_exprs_values,
  min.value = NULL,
  max.value = NULL,
  ...
)
```

Arguments

object	A SingleCellExperiment object.
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 the by argument in ?retrieveCellInfo for possible values.
shape_by	Specification of a column metadata field or a feature to shape by, see the by argument in ?retrieveCellInfo for possible values.
size_by	Specification of a column metadata field or a feature to size by, see the by argument in ?retrieveCellInfo for possible values.
order_by	Specification of a column metadata field or a feature to order points by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	Alias for by.assay.type.
text_by	String specifying the column metadata field with which to add text labels on the plot. This must refer to a categorical field, i.e., coercible into a factor. Alternatively, an AsIs vector or data.frame, see ?retrieveCellInfo.
text_size	Numeric scalar specifying the size of added text.
text_colour	String specifying the colour of the added text.
label_format	Character vector of length 2 containing format strings to use for the axis labels. The first string expects a string containing the result type (e.g., "PCA") and an integer containing the component number, while the second string shows the rounded percentage of variance explained and is only relevant when this information is provided in object.
other_fields	Additional cell-based fields to include in the data.frame, see ?"scater-plot-args" for details.
text_color	Alias to text_colour.
color_by	Alias to colour_by.
swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
point.padding, force	See ?ggrepel::geom_text_repel.
rasterise	Whether to rasterise the points in the plot with rasterise. To control the dpi, set options(ggrastr.default.dpi), for example options(ggrastr.default.dpi=300).
scattermore	Logical, whether to use the scattermore package to greatly speed up plotting a large number of cells. Use point_size = 0 for the most performance gain.

bins — Number of bins, can be different in x and y, to bin and summarize the points and their values, to avoid overplotting. If NULL (default), then the points are plotted without binning. Only used when both x and y are numeric.

summary_fun — Function to summarize the feature value of each point (e.g. gene expression of each cell) when the points binned, defaults to sum. Can be either the name of the function or the function itself.

hex — Logical, whether to use geom_hex.

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

min.value, max.value — Minimum and maximum values, beyond which colour_by values (if numeric) are truncated. Can be set to a numeric value to prevent outlying values from skewing the colour scale, or set to quantiles of the colour_by variable by setting to (e.g.) "q10" for the 10th quantile.

... — Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

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

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

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

Value

A ggplot object

Note

Arguments shape_by and size_by are ignored when scattermore = TRUE. Using scattermore is only recommended for very large datasets to speed up plotting. Small point size is also recommended. For larger point size, the point shape may be distorted. Also, when scattermore = TRUE, the point_size argument works differently.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

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

example_sce <- runPCA(example_sce, ncomponents=5)
```

```
plotReducedDim(example_sce, "PCA")
plotReducedDim(example_sce, "PCA", colour_by="Cell_Cycle")
plotReducedDim(example_sce, "PCA", colour_by="Gene_0001")

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

# Use scattermore
plotPCA(example_sce, ncomponents = 4, scattermore = TRUE, point_size = 3)

# Bin to show point density
plotPCA(example_sce, bins = 10)
# Bin to summarize values (default is sum)
plotPCA(example_sce, bins = 10, colour_by = "Gene_0001")
```

`plotRLE` *Plot relative log expression*

Description

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

Usage

```
plotRLE(
  object,
  exprs_values = "logcounts",
  exprs_logged = TRUE,
  style = "minimal",
  legend = TRUE,
  ordering = NULL,
  colour_by = color_by,
  by_exprs_values = exprs_values,
  BPPARAM = BiocParallel::bpparam(),
  color_by = NULL,
  assay.type = exprs_values,
  by.assay.type = by_exprs_values,
  assay_logged = exprs_logged,
  ...
)
```

Arguments

`object`	A SingleCellExperiment object.
`exprs_values`	Alias to `assay.type`.

exprs_logged	A logical scalar indicating whether the expression matrix is already log-transformed. If not, a log2-transformation (+1) will be performed prior to plotting.
style	String defining the boxplot style to use, either "minimal" (default) or "full"; see Details.
legend	Logical scalar specifying whether a legend should be shown.
ordering	A vector specifying the ordering of cells in the RLE plot. This can be useful for arranging cells by experimental conditions or batches.
colour_by	Specification of a column metadata field or a feature to colour by, see the by argument in ?retrieveCellInfo for possible values.
by_exprs_values	Alias to by.assay.type.
BPPARAM	A BiocParallelParam object to be used to parallelise operations using DelayedArray.
color_by	Alias to colour_by.
assay.type	A string or integer scalar specifying the expression matrix in object to use.
by.assay.type	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see the assay.type argument in ?retrieveCellInfo.
assay_logged	Alias to exprs_logged.
...	further arguments passed to geom_boxplot when style="full".

Details

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

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

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

Value

A ggplot object

Author(s)

Davis McCarthy, with modifications by Aaron Lun

References

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

Examples

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

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

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

---

plotRowData *Plot row metadata*

Description

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

Usage

```
plotRowData(
  object,
  y,
  x = NULL,
  colour_by = color_by,
  shape_by = NULL,
  size_by = NULL,
  by_exprs_values = "logcounts",
  other_fields = list(),
  color_by = NULL,
  by.assay.type = by_exprs_values,
  ...
)
```

Arguments

object	A SingleCellExperiment object containing expression values and experimental information.
y	String specifying the column-level metadata field to show on the y-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo.
x	String specifying the column-level metadata to show on the x-axis. Alternatively, an AsIs vector or data.frame, see ?retrieveFeatureInfo. If NULL, nothing is shown on the x-axis.

colour_by	Specification of a row metadata field or a cell to colour by, see ?retrieveFeatureInfo for possible values.
shape_by	Specification of a row metadata field or a cell to shape by, see ?retrieveFeatureInfo for possible values.
size_by	Specification of a row metadata field or a cell to size by, see ?retrieveFeatureInfo for possible values.
by_exprs_values	Alias to by.assay.type.
other_fields	Additional feature-based fields to include in the data.frame, see ?"scater-plot-args" for details.
color_by	Alias to colour_by.
by.assay.type	A string or integer scalar specifying which assay to obtain expression values from, for use in point aesthetics - see ?retrieveFeatureInfo for details.
...	Additional arguments for visualization, see ?"scater-plot-args" for details.

Details

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

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

Value

A ggplot object.

Examples

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

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

---

plotScater *Plot an overview of expression for each cell*

Description

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

Usage

```
plotScater(
  x,
  nfeatures = 500,
  exprs_values = "counts",
  colour_by = color_by,
  by_exprs_values = exprs_values,
  block1 = NULL,
  block2 = NULL,
  ncol = 3,
  line_width = 1.5,
  theme_size = 10,
  color_by = NULL,
  assay.type = exprs_values,
  by.assay.type = by_exprs_values
)
```

Arguments

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

Details

For each cell, the features are ordered from most-expressed to least-expressed. The cumulative proportion of the total expression for the cell is computed across the top nfeatures features. These

plots can flag cells with a very high proportion of the library coming from a small number of features; such cells are likely to be problematic for downstream analyses.

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

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

Examples

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

---

projectReducedDim *Project cells into an arbitrary dimensionality reduction space.*

Description

Projects observations into arbitrary dimensionality reduction space (e.g., t-SNE, UMAP) using a tricube weighted average of the k nearest neighbours.

Usage

```
projectReducedDim(x, ...)

## S4 method for signature 'matrix'
projectReducedDim(x, old.embedding, ...)

## S4 method for signature 'SummarizedExperiment'
projectReducedDim(
  x,
  old.sce,
  dimred.embed = "TSNE",
  dimred.knn = "PCA",
  dimred.name = dimred.embed,
  k = 5
)
```

Arguments

x	A numeric matrix of a dimensionality reduction containing the cells that should be projected into the existing embedding defined in either old.embedding or old.sce. Alternatively, a SummarizedExperiment or SingleCellExperiment containing such a matrix.
...	Passed to methods.
old.embedding	If x is a matrix and old is given, then old.embedding is the existing dimensionality reduction embedding that x should be projected into.
old.sce	The object containing the original dimensionality points. If x is a matrix, then old.points must be supplied as a matrix of
dimred.embed	The name of the target dimensionality reduction that points should be embedded into, if .
dimred.knn	The name of the dimensionality reduction to use to identify the K-nearest neighbours from x in the dimensionality reduction slot of the same name defined in either old or old.sce.
dimred.name	The name of the dimensionality reduction that the projected embedding will be saved as, for the SummarizedExperiment method.
k	The number of nearest neighours to use to project points into the embedding.

Value

When x is a matrix, a matrix is returned. When x is a SummarizedExperiment (or SingleCellExperiment), the return value is of the same class as the input, but the projected dimensionality reduction is added as a reducedDim field.

Examples

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

example_sce_new <- mockSCE()
example_sce_new <- logNormCounts(example_sce_new)
example_sce_new <- runPCA(example_sce_new)

## sce method
projectReducedDim(
    example_sce_new,
    old.sce = example_sce,
    dimred.embed="UMAP",
    dimred.knn="PCA"
)

## matrix method
projectReducedDim(
    reducedDim(example_sce, "PCA"),
    new.points = reducedDim(example_sce_new, "PCA"),
```

```
    old.embedding = reducedDim(example_sce, "UMAP")
)
```

Reduced dimension plots

Plot specific reduced dimensions

Description

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

Usage

```
plotPCASCE(object, ..., ncomponents = 2, dimred = "PCA")

plotTSNE(object, ..., ncomponents = 2, dimred = "TSNE")

plotUMAP(object, ..., ncomponents = 2, dimred = "UMAP")

plotDiffusionMap(object, ..., ncomponents = 2, dimred = "DiffusionMap")

plotMDS(object, ..., ncomponents = 2, dimred = "MDS")

plotNMF(object, ..., ncomponents = 2, dimred = "NMF")

## S4 method for signature 'SingleCellExperiment'
plotPCA(object, ..., ncomponents = 2, dimred = "PCA")
```

Arguments

`object`	A SingleCellExperiment object.
`...`	Additional arguments to pass to `plotReducedDim`.
`ncomponents`	Numeric scalar indicating the number of dimensions components to (calculate and) plot. This can also be a numeric vector, see ?`plotReducedDim` for details.
`dimred`	A string or integer scalar indicating the reduced dimension result in `reducedDims(object)` to plot.

Details

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

- `"PCA"` for `plotPCA`
- `"TSNE"` for `plotTSNE`
- `"DiffusionMap"` for `plotDiffusionMap`

- "MDS" for "plotMDS"
- "NMF" for "plotNMF"
- "UMAP" for "plotUMAP"

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

Value

A ggplot object.

Author(s)

Davis McCarthy, with modifications by Aaron Lun

See Also

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

plotReducedDim, for the underlying plotting function.

Examples

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

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

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

## Same for TSNE:
example_sce <- runTSNE(example_sce)
plotTSNE(example_sce, colour_by="Mutation_Status")

## Not run:
## Same for DiffusionMaps:
example_sce <- runDiffusionMap(example_sce)
plotDiffusionMap(example_sce)

## End(Not run)

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

reexports *Objects exported from other packages*

Description

These objects are imported from other packages. Follow the links below to see their documentation.

scuttle addPerCellQC, addPerFeatureQC, aggregateAcrossCells, aggregateAcrossFeatures, calculateAverage, calculateCPM, calculateFPKM, calculateTPM, computeLibraryFactors, computeMedianFactors, isOutlier, librarySizeFactors, logNormCounts, makePerCellDF, makePerFeatureDF, medianSizeFactors, mockSCE, normalizeCounts, numDetectedAcrossCells, numDetectedAcrossFeatures, perCellQCMetrics, perFeatureQCMetrics, quickPerCellQC, readSparseCounts, sumCountsAcrossCells, sumCountsAcrossFeatures, uniquifyFeatureNames

retrieveCellInfo *Cell-based data retrieval*

Description

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

Usage

```
retrieveCellInfo(
  x,
  by,
  search = c("colData", "assays", "altExps"),
  exprs_values = "logcounts",
  swap_rownames = NULL,
  assay.type = exprs_values
)
```

Arguments

x	A SingleCellExperiment object.
by	A string specifying the field to extract (see Details). Alternatively, a data.frame, DataFrame or an AsIs vector.
search	Character vector specifying the types of data or metadata to use.
exprs_values	Alias to assay.type.
swap_rownames	Column name of rowData(object) to be used to identify features instead of rownames(object) when labelling plot elements.
assay.type	String or integer scalar specifying the assay from which expression values should be extracted.

Details

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

Given a character string in by, this function will:

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

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

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

Value

A list containing name, a string with the name of the extracted field (usually identically to by); and value, a vector of length equal to ncol(x) containing per-cell (meta)data values. If by=NULL, both name and value are set to NULL.

Author(s)

Aaron Lun

See Also

makePerCellDF, which provides a more user-friendly interface to this function.

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

Examples

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

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

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

`retrieveFeatureInfo` *Feature-based data retrieval*

Description

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

Usage

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

Arguments

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

Details

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

Given a character string in `by`, this function will:

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

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

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

Value

A list containing `name`, a string with the name of the extracted field (usually identically to `by`); and `value`, a vector of length equal to `ncol(x)` containing per-feature (meta)data values. If `by=NULL`, both `name` and `value` are set to `NULL`.

Author(s)

Aaron Lun

See Also

`makePerFeatureDF`, which provides a more user-friendly interface to this function.

`plotRowData` and other feature-based plotting functions.

Examples

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

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

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

---

runColDataPCA *Perform PCA on column metadata*

Description

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

Usage

```
runColDataPCA(
  x,
  ncomponents = 2,
  variables = NULL,
  scale = TRUE,
  outliers = FALSE,
  BSPARAM = ExactParam(),
  BPPARAM = SerialParam(),
  name = "PCA_coldata"
)
```

Arguments

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

Details

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

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

Value

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

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

Author(s)

Aaron Lun, based on code by Davis McCarthy

See Also

runPCA, for the corresponding method operating on expression data.

Examples

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

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

---

runMultiUMAP *Multi-modal UMAP*

Description

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

Usage

```
calculateMultiUMAP(x, ...)

## S4 method for signature 'ANY'
calculateMultiUMAP(x, ..., metric = "euclidean")

## S4 method for signature 'SummarizedExperiment'
calculateMultiUMAP(
  x,
  exprs_values,
  metric = "euclidean",
  assay.type = exprs_values,
  ...
)

## S4 method for signature 'SingleCellExperiment'
calculateMultiUMAP(
  x,
  exprs_values,
  dimred,
  altexp,
  altexp_exprs_values = "logcounts",
  assay.type = exprs_values,
  altexp.assay.type = altexp_exprs_values,
  ...
```

```
)

runMultiUMAP(x, ..., name = "MultiUMAP")
```

Arguments

x	For calculateMultiUMAP, a list of numeric matrices where each row is a cell and each column is some dimension/variable. For gene expression data, this is usually the matrix of PC coordinates. Alternatively, a SummarizedExperiment containing relevant matrices in its assays. Alternatively, a SingleCellExperiment containing relevant matrices in its assays, reducedDims or altExps. This is also the only permissible argument for runMultiUMAP.
...	For the generic, further arguments to pass to specific methods. For the ANY method, further arguments to pass to umap. For the SummarizedExperiment and SingleCellExperiment methods, and for runMultiUMAP, further arguments to pass to the ANY method.
metric	Character vector specifying the type of distance to use for each matrix in x. This is recycled to the same number of matrices supplied in x.
exprs_values	Alias to assay.type.
assay.type	A character or integer vector of assays to extract and transpose for use in the UMAP. For the SingleCellExperiment, this argument can be missing, in which case no assays are used.
dimred	A character or integer vector of reducedDims to extract for use in the UMAP. This argument can be missing, in which case no assays are used.
altexp	A character or integer vector of altExps to extract and transpose for use in the UMAP. This argument can be missing, in which case no alternative experiments are used.
altexp_exprs_values	Alias to altexp.assay.type.
altexp.assay.type	A character or integer vector specifying the assay to extract from alternative experiments, when altexp is specified. This is recycled to the same length as altexp.
name	String specifying the name of the reducedDims in which to store the UMAP.

Details

These functions serve as convenience wrappers around umap for multi-modal analysis. The idea is that each input matrix in x corresponds to data for a different mode. A typical example would consist of the PC coordinates generated from gene expression counts, plus the log-abundance matrix for ADT counts from CITE-seq experiments; one might also include matrices of transformed intensities from indexed FACS, to name some more possibilities.

Roughly speaking, the idea is to identify nearest neighbors *within* each mode to construct the simplicial sets. Integration of multiple modes is performed by intersecting the sets to obtain a single

graph, which is used in the rest of the UMAP algorithm. By performing an intersection, we focus on relationships between cells that are consistently neighboring across all the modes, thus providing greater resolution of differences at any mode. The neighbor search within each mode also avoids difficulties with quantitative comparisons of distances between modes.

The most obvious use of this function is to generate a low-dimensional embedding for visualization. However, users can also set `n_components` to a higher value (e.g., 10-20) to retain more information for downstream steps like clustering. This Do, however, remember to set the seed appropriately.

By default, all modes use the distance metric of `metric` to construct the simplicial sets *within* each mode. However, it is possible to vary this by supplying a vector of metrics, e.g., `"euclidean"` for the first matrix, `"manhattan"` for the second. For the SingleCellExperiment method, matrices are extracted in the order of assays, reduced dimensions and alternative experiments, so any variation in `metrics` is also assumed to follow this order.

Value

For `calculateMultiUMAP`, a numeric matrix containing the low-dimensional UMAP embedding.

For `runMultiUMAP`, `x` is returned with a `MultiUMAP` field in its `reducedDims`.

Author(s)

Aaron Lun

See Also

`runUMAP`, for the more straightforward application of UMAP.

Examples

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

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

# Running a multimodal analysis using PCs for expression
# and log-counts for the ADTs:
exprs_sce <- runMultiUMAP(exprs_sce, dimred="PCA", altexp="ADT")
plotReducedDim(exprs_sce, "MultiUMAP")
```

scater-pkg *The* **scater** *package*

Description

Provides functions for convenient visualization of single-cell data, mostly via **ggplot2**. It also used to provide utilities for data transformation and quality control, but these have largely been moved to the **scuttle** package.

Author(s)

Davis McCarthy, Aaron Lun

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`.

`point_shape`**:** An integer, or a string specifying the shape of the points. Details see `vignette("ggplot2-specs")`. Defaults to `19`.

`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`.

`show_boxplot`**:** Logical, should a box plot be shown? Defaults to `FALSE`.

Miscellaneous fields

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

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

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

See Also

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

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.

`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