Package 'scater'

April 15, 2017

Type Package

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

Author Davis McCarthy

Version 1.2.0

Date 2016-10-14

License GPL (>= 2)

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.

Depends R (>= 3.3), Biobase, ggplot2, methods

Imports biomaRt, BiocGenerics, data.table, dplyr, edgeR, ggbeeswarm, grid, limma, matrixStats, parallel, plyr, reshape2, rhdf5, rjson, shiny, shinydashboard, stats, tximport, viridis

Suggests BiocStyle, cowplot, cluster, destiny, knitr, monocle, mvoutlier, rmarkdown, Rtsne, testthat, magrittr

VignetteBuilder knitr

LazyData true

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

RoxygenNote 5.0.1

NeedsCompilation yes

URL https://github.com/davismcc/scater

BugReports https://github.com/davismcc/scater/issues

R topics documented:

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

Description

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

Details

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

`arrange` *Arrange rows of* `pData(object)` *by variables.*

Description

The `SCESet` returned will have cells ordered by the corresponding variable in `pData(object)`.

Usage

```
arrange(object, ...)

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

arrange.SCESet(object, ...)
```

Arguments

object	A SCESet object.
...	Additional arguments to be passed to dplyr::arrange to act on pData(object).

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- arrange(example_sceset, Cell_Cycle)
```

---

bootstraps *Accessor and replacement for bootstrap results in an SCESet object*

Description

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

Usage

```
bootstraps(object)

bootstraps(object) <- value

bootstraps.SCESet(object)

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

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

Arguments

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

Value

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

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
bootstraps(example_sceset)
```

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

Description

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

Usage

```
calcIsExprs(object, lowerDetectionLimit = NULL, exprs_data = "counts")
```

Arguments

`object`	an SCESet object with expression and/or count data.
`lowerDetectionLimit`	numeric scalar giving the minimum expression level for an expression observation in a cell for it to qualify as expressed.
`exprs_data`	character scalar indicating whether the count data (`"counts"`), the transformed expression data (`"exprs"`), transcript-per-million (`"tpm"`), counts-per-million (`"cpm"`) or FPKM (`"fpkm"`) should be used to define if an observation is expressed or not.

Value

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

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData=sc_example_counts)
is_exprs(example_sceset) <- calcIsExprs(example_sceset, lowerDetectionLimit = 1,
exprs_data = "exprs")
```

calculateFPKM *Calculate fragments per kilobase of exon per million reads mapped (FPKM)*

Description

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

Usage

```
calculateFPKM(object, effective_length)
```

Arguments

object	an SCESet object
effective_length	vector of class "numeric" providing the effective length for each feature in the SCESet object

Value

Matrix of FPKM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
effective_length <- rep(1000, 2000)
fpkm(example_sceset) <- calculateFPKM(example_sceset, effective_length)
```

calculateQCMetrics *Calculate QC metrics*

Description

Calculate QC metrics

Usage

```
calculateQCMetrics(object, feature_controls = NULL, cell_controls = NULL,
  nmads = 5, pct_feature_controls_threshold = 80)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
feature_controls	a named list containing one or more vectors (character vector of feature names, logical vector, or a numeric vector of indices are all acceptable) used to identify feature controls (for example, ERCC spike-in genes, mitochondrial genes, etc).
cell_controls	a character vector of cell (sample) names, or a logical vector, or a numeric vector of indices used to identify cell controls (for example, blank wells or bulk controls).
nmads	numeric scalar giving the number of median absolute deviations to be used to flag potentially problematic cells based on total_counts (total number of counts for the cell, or library size) and total_features (number of features with non-zero expression). For total_features, cells are flagged for filtering only if total_features is nmads below the median. Default value is 5.
pct_feature_controls_threshold	numeric scalar giving a threshold for percentage of expression values accounted for by feature controls. Used as to flag cells that may be filtered based on high percentage of expression from feature controls.

Details

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

The following QC metrics are computed:

total_counts: Total number of counts for the cell (aka "library size")

log10_total_counts: Total counts on the log10-scale

total_features: The number of endogenous features (i.e. not control features) for the cell that have expression above the detection limit (default detection limit is zero)

filter_on_depth: Would this cell be filtered out based on its log10-depth being (by default) more than 5 median absolute deviations from the median log10-depth for the dataset?

filter_on_coverage: Would this cell be filtered out based on its coverage being (by default) more than 5 median absolute deviations from the median coverage for the dataset?

filter_on_pct_counts_feature_controls: Should the cell be filtered out on the basis of having a high percentage of counts assigned to control features? Default threshold is 80 percent (i.e. cells with more than 80 percent of counts assigned to feature controls are flagged).

counts_feature_controls: Total number of counts for the cell that come from (one or more sets of user-defined) control features. Defaults to zero if no control features are indicated. If more than one set of feature controls are defined (for example, ERCC and MT genes are defined as controls), then this metric is produced for all sets, plus the union of all sets (so here, we get columns counts_feature_controls_ERCC, counts_feature_controls_MT and counts_feature_controls).

log10_counts_feature_controls: Just as above, the total number of counts from feature controls, but on the log10-scale. Defaults to zero (i.e.~log10(0 + 1), offset to avoid negative infinite values) if no feature control are indicated.

pct_counts_feature_controls: Just as for the counts described above, but expressed as a percentage of the total counts. Defined for all control sets and their union, just like the raw counts. Defaults to zero if no feature controls are defined.

filter_on_pct_counts_feature_controls: Would this cell be filtered out on the basis that the percentage of counts from feature controls is higher than a defined threhold (default is 80%)? Just as with `counts_feature_controls`, this is defined for all control sets and their union.

pct_counts_top_50_features: What percentage of the total counts is accounted for by the 50 highest-count features? Also computed for the top 100 and top 200 features, with the obvious changes to the column names. Note that the top "X" percentage will not be computed if the total number of genes is less than "X".

pct_dropout: Percentage of features that are not "detectably expressed", i.e. have expression below the `lowerDetectionLimit` threshold.

counts_endogenous_features: Total number of counts for the cell that come from endogenous features (i.e. not control features). Defaults to 'depth' if no control features are indicated.

log10_counts_endogenous_features: Total number of counts from endogenous features on the log10-scale. Defaults to all counts if no control features are indicated.

n_detected_feature_controls: Number of defined feature controls that have expression greater than the threshold defined in the object (that is, they are "detectably expressed"; see `object@lowerDetectionLimit` to check the threshold). As with other metrics for feature controls, defined for all sets of feature controls (set names appended as above) and their union. So we might commonly get columns `n_detected_feature_controls_ERCC`, `n_detected_feature_controls_MT` and `n_detected_feature_controls` (ERCC and MT genes detected).

is_cell_control: Has the cell been defined as a cell control? If more than one set of cell controls are defined (for example, blanks and bulk libraries are defined as cell controls), then this metric is produced for all sets, plus the union of all sets (so we could typically get columns `is_cell_control_Blank`, `is_cell_control_Bulk`, and `is_cell_control`, the latter including both blanks and bulks as cell controls).

These cell-level QC metrics are added as columns to the "phenotypeData" slot of the `SCESet` object so that they can be inspected and are readily available for other functions to use. Furthermore, wherever "counts" appear in the above metrics, the same metrics will also be computed for "exprs", "tpm" and "fpkm" values (if TPM and FPKM values are present in the `SCESet` object), with the appropriate term replacing "counts" in the name. The following feature-level QC metrics are also computed:

mean_exprs: The mean expression level of the gene/feature.

exprs_rank: The rank of the feature's mean expression level in the cell.

n_cells_exprs: The number of cells for which the expression level of the feature is above the detection limit (default detection limit is zero).

total_feature_counts: The total number of counts assigned to that feature across all cells.

log10_total_feature_counts: Total feature counts on the log10-scale.

pct_total_counts: The percentage of all counts that are accounted for by the counts assigned to the feature.

pct_dropout: The percentage of all cells that have no detectable expression (i.e. `is_exprs(object)` is `FALSE`) for the feature.

is_feature_control: Is the feature a control feature? Default is 'FALSE' unless control features are defined by the user. If more than one feature control set is defined (as above), then a column of this type is produced for each control set (e.g. here, `is_feature_control_ERCC` and `is_feature_control_MT`) as well as the column named `is_feature_control`, which indicates if the feature belongs to any of the control sets.

These feature-level QC metrics are added as columns to the "featureData" slot of the SCESet object so that they can be inspected and are readily available for other functions to use. As with the cell-level metrics, wherever "counts" appear in the above, the same metrics will also be computed for "exprs", "tpm" and "fpkm" values (if TPM and FPKM values are present in the SCESet object), with the appropriate term replacing "counts" in the name.

Value

an SCESet object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
example_sceset <- calculateQCMetrics(example_sceset)

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

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

---

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

Description

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

Usage

```
calculateTPM(object, effective_length = NULL, calc_from = "counts")
```

Arguments

object	an SCESet object
effective_length	vector of class "numeric" providing the effective length for each feature in the SCESet object
calc_from	character string indicating whether to compute TPM from "counts", "norm_counts", "fpkm" or "norm_fpkm". Default is to use "counts", in which case the effective_length argument must be supplied.

Value

Matrix of TPM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
effective_length <- rep(1000, 2000)
tpm(example_sceset) <- calculateTPM(example_sceset, effective_length,
    calc_from = "counts")

## calculate from FPKM
fpkm(example_sceset) <- calculateFPKM(example_sceset, effective_length)
tpm(example_sceset) <- calculateTPM(example_sceset, effective_length,
                                    calc_from = "fpkm")
```

---

cellNames<- *Get or set cell names from an SCESet object*

Description

Get or set cell names from an SCESet object

Usage

```
cellNames(object) <- value

cellNames(object)

## S4 replacement method for signature 'SCESet,vector'
cellNames(object)<-value
```

Arguments

object	An SCESet object.
value	a vector of cell names to apply to the SCESet object.

Details

Simply a wrapper to sampleNames.

Value

A vector of cell names.

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
cellNames(example_sceset)

data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
cellNames(example_sceset) <- 1:ncol(example_sceset)
```

`cellPairwiseDistances` *cellPairwiseDistances in an SCESet object*

Description

SCESet objects can contain a matrix of pairwise distances between cells. These functions conveniently access and replace the cell pairwise distances with the value supplied, which must be a matrix of the correct size. The function `cellDist` is simply shorthand for `cellPairwiseDistances`.

Usage

```
cellPairwiseDistances(object)

cellPairwiseDistances(object) <- value

cellDist(object)

cellDist(object) <- value

cellPairwiseDistances.SCESet(object)

## S4 method for signature 'SCESet'
cellPairwiseDistances(object)

cellDistSCESet(object)

## S4 method for signature 'SCESet'
cellDist(object)

## S4 replacement method for signature 'SCESet,matrix'
cellPairwiseDistances(object) <- value

## S4 replacement method for signature 'SCESet,dist'
cellPairwiseDistances(object) <- value

## S4 replacement method for signature 'SCESet,matrix'
cellDist(object) <- value

## S4 replacement method for signature 'SCESet,dist'
cellDist(object) <- value
```

Arguments

object	a SCESet object.
value	a matrix of class "numeric" containing cell pairwise distances

Value

An SCESet object containing new cell pairwise distances matrix.

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
cellPairwiseDistances(example_sceset)
```

---

counts *Accessors for the 'counts' element of an SCESet object.*

Description

The counts element holds the count data as a matrix of non-negative integer count values, one row for each feature (gene, exon, region, etc), and one column for each cell. It is an element of the assayData slot of the SCESet object.

Usage

```
## S4 method for signature 'SCESet'
counts(object)

## S4 replacement method for signature 'SCESet,matrix'
counts(object)<-value

## S4 method for signature 'SCESet'
counts(object)

## S4 replacement method for signature 'SCESet,matrix'
counts(object) <- value
```

Arguments

object	a SCESet object.
value	an integer matrix

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
counts(example_sceset)
```

---

cpm *Accessors for the 'cpm' (counts per million) element of an SCESet object.*

Description

The cpm element of the arrayData slot in an SCESet object holds a matrix containing counts-per-million values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
cpm(object)

cpm(object) <- value

## S4 method for signature 'SCESet'
cpm(object)

## S4 replacement method for signature 'SCESet,matrix'
cpm(object)<-value

## S4 method for signature 'SCESet'
cpm(object)

## S4 replacement method for signature 'SCESet,matrix'
cpm(object) <- value
```

Arguments

object	a SCESet object.
value	a matrix of class "numeric"

Value

a matrix of counts-per-million values

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData=sc_example_counts)
cpm(example_sceset)[1:10, 1:6]
```

`fData<-,SCESet,AnnotatedDataFrame-method`

Replaces featureData in an SCESet object

Description

SCESet objects contain feature information (inherited from the ExpressionSet class). This function conveniently replaces the feature data with the value supplied, which must be an AnnotatedDataFrame.

Usage

```
## S4 replacement method for signature 'SCESet,AnnotatedDataFrame'
fData(object) <- value

## S4 replacement method for signature 'SCESet,data.frame'
fData(object) <- value
```

Arguments

`object`	An SCESet object.
`value`	an AnnotatedDataFrame with updated featureData to replace existing

Value

A matrix of expression count data, where rows correspond to features (e.g. genes) and columns correspond to cells.

Examples

```
## Not run:
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
fData(example_sceset)

## End(Not run)
```

`featurePairwiseDistances` *featurePairwiseDistances in an SCESet object*

Description

SCESet objects can contain a matrix of pairwise distances between features (e.g. genes, transcripts). These functions conveniently access and replace the gene pairwise distances with the value supplied, which must be a matrix of the correct size. The function `featDist` is simply shorthand for `featurePairwiseDistances`.

Usage

```
featurePairwiseDistances(object)

featurePairwiseDistances(object) <- value

featDist(object)

featDist(object) <- value

featurePairwiseDistancesSCESet(object)

## S4 method for signature 'SCESet'
featurePairwiseDistances(object)

featDistSCESet(object)

## S4 method for signature 'SCESet'
featDist(object)

## S4 replacement method for signature 'SCESet,matrix'
featurePairwiseDistances(object) <- value

## S4 replacement method for signature 'SCESet,dist'
featurePairwiseDistances(object) <- value

## S4 replacement method for signature 'SCESet,matrix'
featDist(object) <- value

## S4 replacement method for signature 'SCESet,dist'
featDist(object) <- value
```

Arguments

`object`	a `SCESet` object.
`value`	a matrix of class `"numeric"` containing feature pairwise distances

Value

An SCESet object containing new feature pairwise distances matrix.

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
featurePairwiseDistances(example_sceset)
```

`filter` *Return* `SCESet` *with cells matching conditions.*

Description

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

Usage

```
filter(object, ...)

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

filter.SCESet(object, ...)
```

Arguments

`object`	A `SCESet` object.
`...`	Additional arguments to be passed to `dplyr::filter` to act on `pData(object)`.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset_treat1 <- filter(example_sceset, Treatment == "treat1")
```

`findImportantPCs` *Find most important principal components for a given variable*

Description

Find most important principal components for a given variable

Usage

```
findImportantPCs(object, variable = "total_features",
  plot_type = "pcs-vs-vars", exprs_values = "exprs", ntop = 500,
  feature_set = NULL, scale_features = TRUE, theme_size = 10)
```

Arguments

`object`	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
`variable`	character scalar providing a variable name (column from `pData(object)`) for which to determine the most important PCs.
`plot_type`	character string, indicating which type of plot to produce. Default, `"pairs-pcs"` produces a pairs plot for the top 5 PCs based on their R-squared with the variable of interest. A value of `"pcs-vs-vars"` produces plots of the top PCs against the variable of interest.
`exprs_values`	which slot of the `assayData` in the `object` should be used to define expression? Valid options are "counts" (default), "tpm", "fpkm" and "exprs", or anything else in the object added manually by the user.
`ntop`	numeric scalar indicating the number of most variable features to use for the PCA. Default is `500`, but any `ntop` argument is overrided if the `feature_set` argument is non-NULL.
`feature_set`	character, numeric or logical vector indicating a set of features to use for the PCA. If character, entries must all be in `featureNames(object)`. If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to `nrow(object)`.
`scale_features`	logical, should the expression values be standardised so that each feature has unit variance? Default is `TRUE`.
`theme_size`	numeric scalar providing base font size for ggplot theme.

Details

Plot the top 5 or 6 most important PCs (depending on the `plot_type` argument for a given variable. Importance here is defined as the R-squared value from a linear model regressing each PC onto the variable of interest.

Value

a `ggplot` plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]
example_sceset <- calculateQCMetrics(example_sceset)
findImportantPCs(example_sceset, variable="total_features")
```

`fpkm` *Accessors for the 'fpkm' (fragments per kilobase of exon per million reads mapped) element of an SCESet object.*

Description

The `fpkm` element of the arrayData slot in an SCESet object holds a matrix containing fragments per kilobase of exon per million reads mapped (FPKM) values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
fpkm(object)

fpkm(object) <- value

## S4 method for signature 'SCESet'
fpkm(object)

## S4 replacement method for signature 'SCESet,matrix'
fpkm(object)<-value

## S4 method for signature 'SCESet'
fpkm(object)

## S4 replacement method for signature 'SCESet,matrix'
fpkm(object) <- value
```

Arguments

`object`	a `SCESet` object.
`value`	a matrix of class `"numeric"`

Value

a matrix of FPKM values

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
fpkm(example_sceset)
```

`fromCellDataSet` *Convert a* `CellDataSet` *to an* `SCESet`

Description

Convert a `CellDataSet` to an `SCESet`

Usage

```
fromCellDataSet(cds, exprs_values = "tpm", logged = FALSE)
```

Arguments

`cds`	A `CellDataSet` from the `monocle` package
`exprs_values`	What should `exprs(cds)` be mapped to in the `SCESet`? Should be one of "exprs", "tpm", "fpkm", "counts"
`logged`	logical, if a value is supplied for the exprsData argument, are the expression values already on the log2 scale, or not?

Value

An object of class `SCESet`

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
if ( requireNamespace("monocle") ) {
    # cds <- toCellDataSet(example_sceset) # not run
    # sceset <- fromCellDataSet(cds) # not run
}
```

getBMFeatureAnnos *Get feature annotation information from Biomart*

Description

Use the biomaRt package to add feature annotation information to an SCESet.

Usage

```
getBMFeatureAnnos(object, filters = "ensembl_transcript_id",
  attributes = c("ensembl_transcript_id", "ensembl_gene_id", "mgi_symbol",
  "chromosome_name", "transcript_biotype", "transcript_start", "transcript_end",
  "transcript_count"), feature_symbol = "mgi_symbol",
  feature_id = "ensembl_gene_id", biomart = "ENSEMBL_MART_ENSEMBL",
  dataset = "mmusculus_gene_ensembl", host = "www.ensembl.org")
```

Arguments

object	an SCESet object
filters	character vector defining the "filters" terms to pass to the biomaRt::getBM function.
attributes	character vector defining the biomaRt attributes to pass to the attributes argument of getBM.
feature_symbol	character string defining the biomaRt attribute to be used to define the symbol to be used for each feature (which appears as the feature_symbol in fData(object), subsequently). Default is "mgi_symbol", gene symbols for mouse. This should be changed if the organism is not Mus musculus!
feature_id	character string defining the biomaRt attribute to be used to define the ID to be used for each feature (which appears as the feature_id in fData(object), subsequently). Default is "ensembl_gene_id", Ensembl gene IDs for mouse. This should be changed if the organism is not Mus musculus!
biomart	character string defining the biomaRt to be used. Default is "ENSEMBL_MART_ENSEMBL".
dataset	character string defining the biomaRt dataset to use. Default is "mmusculus_gene_ensembl", which should be changed if the organism is not the mouse!
host	optional character string argument which can be used to select a particular "host" from biomaRt to use. Useful for accessing archived versions of biomaRt data. Default is "www.ensembl.org", in which case the current version of the biomaRt (now hosted by Ensembl) is used.

Details

See the documentation for the biomaRt package, specifically for the functions useMart and getBM, for information on what are permitted values for the filters, attributes, biomart, dataset and host arguments.

Value

an SCESet object

Examples

```
## Not run:
object <- getBMFeatureAnnos(object)

## End(Not run)
```

---

getExprs *Retrieve a representation of gene expression*

Description

Gene expression can be summarised in a variety of ways, e.g. as TPM, FPKM or as raw counts. Many internal methods and external packages rely on accessing a generic representation of expression without worrying about the particulars. Scater allows the user to set `object@useForExprs` to the preferred type (either "exprs", "TPM", "fpkm" or "counts") and that particular representation will be returned by calls to `getExprs`. Note if such representation is not defined, this method returns `NULL`.

Usage

```
getExprs(object)
```

Arguments

`object`	An object of type `SCESet`

Value

A matrix representation of expression corresponding to `object@useForExprs`.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd, useForExprs = "exprs")
all(exprs(example_sceset) == getExprs(example_sceset)) # TRUE
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd, useForExprs = "counts")
all(exprs(example_sceset) == getExprs(example_sceset)) # FALSE
all(counts(example_sceset) == getExprs(example_sceset)) # TRUE
```

get_exprs *Generic accessor for expression data from an SCESet object.*

Description

Access by name a matrix of expression values, one row for each feature (gene, exon, region, etc), and one column for each cell stored an element of the assayData slot of the SCESet object.

Usage

```
get_exprs(object, exprs_values)

## S4 method for signature 'SCESet'
get_exprs(object, exprs_values)

## S4 method for signature 'SCESet'
get_exprs(object, exprs_values = "exprs")
```

Arguments

object	a SCESet object.
exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "norm_tpm" (normalised TPM values), "fpkm" (FPKM values), "norm_fpkm" (normalised FPKM values), "counts" (counts for each feature), "norm_counts", "cpm" (counts-per-million), "norm_cpm" (normalised counts-per-million), "exprs" (whatever is in the 'exprs' slot of the SCESet object; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other slots that have been added to the "assayData" slot by the user.

Value

a matrix of expression values

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
get_exprs(example_sceset, "counts")

## new slots can be defined and accessed
set_exprs(example_sceset, "scaled_counts") <- t(t(counts(example_sceset)) /
colSums(counts(example_sceset)))
get_exprs(example_sceset, "scaled_counts")[1:6, 1:6]
```

isOutlier *Identify if a cell is an outlier based on a metric*

Description

Convenience function to determine which values for a metric are outliers based on median-absolute-deviation (MAD).

Usage

```
isOutlier(metric, nmads = 5, type = c("both", "lower", "higher"),
  log = FALSE, na.rm = FALSE)
```

Arguments

metric	numeric or integer vector of values for a metric
nmads	scalar, number of median-absolute-deviations away from median required for a value to be called an outlier
type	character scalar, choice indicate whether outliers should be looked for at both tails (default: "both") or only at the lower end ("lower") or the higher end ("higher")
log	logical, should the values of the metric be transformed to the log10 scale before computing median-absolute-deviation for outlier detection?
na.rm	logical, should NA (missing) values be removed before computing median and median-absolute-deviation values? If FALSE then return values for median and median-absolute-deviation will be NA if any value is NA.

Value

a logical vector of the same length as the metric argument

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
example_sceset <- calculateQCMetrics(example_sceset)

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

is_exprs *Accessors for the 'is_exprs' element of an SCESet object.*

Description

The is_exprs element holds a logical matrix indicating whether or not each observation is above the defined lowerDetectionLimit in the SCESet object. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
is_exprs(object)

is_exprs(object) <- value

## S4 method for signature 'SCESet'
is_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
is_exprs(object)<-value

## S4 method for signature 'SCESet'
is_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
is_exprs(object) <- value
```

Arguments

object	a SCESet object.
value	an integer matrix

Value

a logical matrix indicating if observations are "expressed" or not

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
is_exprs(example_sceset)
```

mergeSCESet *Merge SCESet objects*

Description

Merge two SCESet objects that have the same features but contain different cells/samples.

Usage

```
mergeSCESet(x, y, fdata_cols_x = 1:ncol(fData(x)),
  fdata_cols_y = fdata_cols_x, pdata_cols_x = NULL, pdata_cols_y = NULL)
```

Arguments

x	an SCESet object
y	an SCESet object
fdata_cols_x	a logical or numeric vector indicating which columns of featureData for x are shared between x and y and should feature in the returned merged SCESet. Default is all columns of fData(x).
fdata_cols_y	a logical or numeric vector indicating which columns of featureData for y are shared between x and y and should feature in the returned merged SCESet. Default is fdata_cols_x.
pdata_cols_x	a logical or numeric vector indicating which columns of phenoData of x should be retained.
pdata_cols_y	a logical or numeric vector indicating which columns of phenoData of y should be retained.

Details

Existing cell-cell pairwise distances and feature-feature pairwise distances will not be valid for a merged SCESet so these are set to NULL in the returned object. Similarly experimentData will need to be added anew to the merged SCESet returned.

Value

a merged SCESet object combining data and metadata from x and y

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
mergeSCESet(example_sceset[, 1:20], example_sceset[, 21:40])

## with specification of columns of fData
example_sceset <- calculateQCMetrics(example_sceset)
mergeSCESet(example_sceset[, 1:20], example_sceset[, 21:40], fdata_cols_x = c(1, 7))

## with specification of columns of pData
mergeSCESet(example_sceset[, 1:20], example_sceset[, 21:40], pdata_cols_x = 1:6)
```

`multiplot` *Multiple plot function for ggplot2 plots*

Description

Place multiple `ggplot` plots on one page.

Usage

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

Arguments

`..., plotlist`	ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
`cols`	numeric scalar giving the number of columns in the layout
`layout`	a matrix specifying the layout. If present, `cols` is ignored.

Details

If the layout is something like `matrix(c(1,2,3,3), nrow=2, byrow=TRUE)`, then plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go all the way across the bottom. There is no way to tweak the relative heights or widths of the plots with this simple function. It was adapted from http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/

Value

a `ggplot` plot object

Examples

```
library(ggplot2)
## This example uses the ChickWeight dataset, which comes with ggplot2
## First plot
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet, group = Chick)) +
   geom_line() +
   ggtitle("Growth curve for individual chicks")
## Second plot
p2 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet)) +
   geom_point(alpha = .3) +
   geom_smooth(alpha = .2, size = 1) +
   ggtitle("Fitted growth curve per diet")
## Third plot
p3 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, colour = Diet)) +
   geom_density() +
   ggtitle("Final weight, by diet")
## Fourth plot
p4 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight, fill = Diet)) +
    geom_histogram(colour = "black", binwidth = 50) +
   facet_grid(Diet ~ .) +
   ggtitle("Final weight, by diet") +
```

```
   theme(legend.position = "none")        # No legend (redundant in this graph)
## Combine plots and display
multiplot(p1, p2, p3, p4, cols = 2)
```

`mutate` *Add new variables to* `pData(object)`.

Description

Adds new columns to `pData(object)` preserving existing variables.

Usage

```
mutate(object, ...)

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

mutate.SCESet(object, ...)
```

Arguments

`object`	A `SCESet` object.
`...`	Additional arguments to be passed to `dplyr::mutate` to act on `pData(object)`.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- mutate(example_sceset, is_quiescent = Cell_Cycle == "G0")
```

`newSCESet` *Create a new SCESet object.*

Description

Scater requires that all data be housed in SCESet objects. SCESet extends Bioconductor's ExpressionSet class, and the same basic interface is supported. newSCESet() expects a matrix of expression values as its first argument, with rows as features (usually genes) and columns as cells. Per-feature and per-cell metadata can be supplied with the featureData and phenoData arguments, respectively. Use of these optional arguments is strongly encouraged. The SCESet also includes a slot 'counts' to store an object containing raw count data.

Usage

```
newSCESet(exprsData = NULL, countData = NULL, tpmData = NULL,
  fpkmData = NULL, cpmData = NULL, phenoData = NULL, featureData = NULL,
  experimentData = NULL, is_exprsData = NULL,
  cellPairwiseDistances = dist(vector()),
  featurePairwiseDistances = dist(vector()), lowerDetectionLimit = 0,
  logExprsOffset = 1, logged = FALSE, useForExprs = "exprs")
```

Arguments

exprsData	expression data matrix for an experiment
countData	data matrix containing raw count expression values
tpmData	matrix of class "numeric" containing transcripts-per-million (TPM) expression values
fpkmData	matrix of class "numeric" containing fragments per kilobase of exon per million reads mapped (FPKM) expression values
cpmData	matrix of class "numeric" containing counts per million (CPM) expression values (optional)
phenoData	data frame containing attributes of individual cells
featureData	data frame containing attributes of features (e.g. genes)
experimentData	MIAME class object containing metadata data and details about the experiment and dataset.
is_exprsData	matrix of class "logical", indicating whether or not each observation is above the lowerDetectionLimit.
cellPairwiseDistances	object of class "dist" (or a class that extends "dist") containing cell-cell distance or dissimilarity values.
featurePairwiseDistances	object of class "dist" (or a class that extends "dist") containing feature-feature distance or dissimilarity values.
lowerDetectionLimit	the minimum expression level that constitutes true expression (defaults to zero and uses count data to determine if an observation is expressed or not).
logExprsOffset	numeric scalar, providing the offset used when doing log2-transformations of expression data to avoid trying to take logs of zero. Default offset value is 1.
logged	logical, if a value is supplied for the exprsData argument, are the expression values already on the log2 scale, or not?
useForExprs	character string, either 'exprs' (default),'tpm','counts' or 'fpkm' indicating which expression representation both internal methods and external packages should use when performing analyses.

Details

SCESet objects store a matrix of expression values. These values are typically transcripts-per-million (tpm), counts-per-million (cpm), fragments per kilobase per million mapped (FPKM) or some other output from a program that calculates expression values from RNA-Seq reads. We recommend that expression values on the log2 scale are used for the 'exprs' slot in the SCESet. For example, you may wish to store raw tpm values in the 'tpm' slot and log2(tpm + 1) values

in the 'exprs' slot. However, expression values could also be values from a single cell qPCR run or some other type of assay. The newSCESet function can also accept raw count values. In this case see calculateTPM and calculateFPKM for computing TPM and FPKM expression values, respectively, from counts. The function cpm from the package edgeR to can be used to compute log2(counts-per-million), if desired.

An SCESet object has to have the 'exprs' slot defined, so if the exprsData argument is NULL, then this function will define 'exprs' with the following order of precedence: log2(TPM + logExprsOffset), if tpmData is defined; log2(FPKM + logExprsOffset) if fpkmData is defined; otherwise log2(counts-per-million + logExprsOffset) are used. The cpm function from the edgeR package is used to compte cpm. Note that for many analyses counts-per-million are not recommended, and if possible transcripts-per-million should be used.

In many downstream functions you will likely find it most convenient if the 'exprs' values are on the log2-scale, so this is recommended.

Value

a new SCESet object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset
```

---

nexprs *Count the number of expressed genes per cell*

Description

An efficient internal function that avoids the need to construct 'is_exprs_mat' by counting the number of expressed genes per cell on the fly.

Usage

```
nexprs(object, threshold = NULL, subset.row = NULL, byrow = FALSE)
```

Arguments

object	an SCESet object
threshold	numeric scalar providing the value above which observations are deemed to be expressed. Defaults to object@lowerDetectionLimit.
subset.row	logical or character vector indicating which rows (i.e. features/genes) to subset and calculate 'is_exprs_mat' for.
byrow	logical scalar indicating if TRUE to count expressing cells per feature (i.e. gene) and if FALSE to count expressing features (i.e. genes) per cell.

Value

a numeric vector of the same length as the number of features if byrow argument is TRUE and the same length as the number of cells if byrow is FALSE

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
nexprs(example_sceset)[1:10]
nexprs(example_sceset, byrow = TRUE)[1:10]
```

`normaliseExprs` *Normalise expression expression levels for an SCESet object*

Description

Compute normalised expression values from an SCESet object and return the object with the normalised expression values added.

Usage

```
normaliseExprs(object, method = "none", design = NULL, feature_set = NULL,
  exprs_values = "counts", return_norm_as_exprs = TRUE, ...)

normalizeExprs(...)
```

Arguments

`object`	an `SCESet` object.
`method`	character string giving method to be used to calculate normalisation factors. Passed to `calcNormFactors`.
`design`	design matrix defining the linear model to be fitted to the normalised expression values. If not `NULL`, then the residuals of this linear model fit are used as the normalised expression values.
`feature_set`	character, numeric or logical vector indicating a set of features to use for the PCA. If character, entries must all be in `featureNames(object)`. If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to `nrow(object)`.
`exprs_values`	character string indicating which slot of the assayData from the `SCESet` object should be used as expression values. Valid options are `'counts'`, the count values, `'exprs'` the expression slot, `'tpm'` the transcripts-per-million slot or `'fpkm'` the FPKM slot.
`return_norm_as_exprs`	logical, should the normalised expression values be returned to the `exprs` slot of the object? Default is TRUE. If FALSE, values in the `exprs` slot will be left untouched. Regardless, normalised expression values will be returned to the `norm_exprs` slot of the object.
`...`	arguments passed to `normaliseExprs` (in the case of `normalizeExprs`) or to `calcNormFactors`.

Details

This function allows the user to compute normalised expression values from an SCESet object. The 'raw' values used can be the values in the 'counts' (default), 'exprs', 'tpm' or 'fpkm' slot of the SCESet. Normalised expression values are added to the 'norm_exprs' slot of the object. Normalised expression values are on the log2-scale, with an offset defined by the logExprsOffset slot of the SCESet object. If the 'exprs_values' argument is one of 'counts', 'tpm' or 'fpkm', then a corresponding slot with normalised values is added: 'norm_counts', 'norm_tpm' or 'norm_fpkm', as appropriate. If 'exprs_values' argument is 'counts' a 'norm_cpm' slot is also added, containing normalised counts-per-million values.

Normalisation is done relative to a defined feature set, if desired, which defines the 'library size' by which expression values are divided. If no feature set is defined, then all features are used. A normalisation size factor can be computed (optionally), which internally uses calcNormFactors. Thus, any of the methods available for calcNormFactors can be used: "TMM", "RLE", "upperquartile" or "none". See that function for further details. Library sizes are multiplied by size factors to obtain a "normalised library size" before normalisation.

If the user wishes to remove the effects of certain explanatory variables, then the 'design' argument can be defined. The design argument must be a valid design matrix, for example as produced by model.matrix, with the relevant variables. A linear model is then fitted using lmFit on expression values after any size-factor and library size normalisation as descrived above. The returned normalised expression values are then the residuals from the linear model fit.

After normalisation, normalised expression values can be accessed with the norm_exprs function (with corresponding accessor functions for counts, tpm, fpkm, cpm). These functions can also be used to assign normalised expression values produced with external tools to an SCESet object.

normalizeExprs is exactly the same as normaliseExprs, provided for those who prefer North American spelling.

Value

an SCESet object

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
keep_gene <- rowSums(counts(example_sceset)) > 0
example_sceset <- example_sceset[keep_gene,]

## Apply TMM normalisation taking into account all genes
example_sceset <- normaliseExprs(example_sceset, method = "TMM")
## Scale counts relative to a set of control features (here the first 100 features)
example_sceset <- normaliseExprs(example_sceset, method = "none",
feature_set = 1:100)
```

normalize *Normalise an SCESet object using pre-computed size factors*

Description

Compute normalised expression values from an SCESet object using the size factors stored in the object. Return the object with the normalised expression values added.

Usage

```
normalize.SCESet(object, exprs_values = "counts", logExprsOffset = NULL,
  recompute_cpm = TRUE, return_norm_as_exprs = TRUE)

## S4 method for signature 'SCESet'
normalize(object, exprs_values = "counts",
  logExprsOffset = NULL, recompute_cpm = TRUE,
  return_norm_as_exprs = TRUE)

normalise(...)
```

Arguments

object	an SCESet object.
exprs_values	character string indicating which slot of the assayData from the SCESet object should be used as expression values. Valid options are 'counts', the count values, 'exprs' the expression slot, 'tpm' the transcripts-per-million slot or 'fpkm' the FPKM slot.
logExprsOffset	scalar numeric value giving the offset to add when taking log2 of normalised values to return as expression values. If NULL (default), then the value from object@logExprsOffset is used.
recompute_cpm	logical, should the counts-per-million values be recomputed after normalising with the stored size factors in the object and stored in cpm(object) in the returned object?
return_norm_as_exprs	logical, should the normalised expression values be returned to the exprs slot of the object? Default is TRUE. If FALSE, values in the exprs slot will be left untouched. Regardless, normalised expression values will be returned in the norm_exprs(object) slot.
...	arguments passed to normalize when calling normalise.

Details

normalize is exactly the same as normalise, the option provided for those who have a preference for North American or British/Australian spelling.

Value

an SCESet object

Author(s)

Davis McCarthy and Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
keep_gene <- rowSums(counts(example_sceset)) > 0
example_sceset <- example_sceset[keep_gene,]

## Apply TMM normalisation taking into account all genes
example_sceset <- normaliseExprs(example_sceset, method = "TMM")
## Scale counts relative to a set of control features (here the first 100 features)
example_sceset <- normaliseExprs(example_sceset, method = "none",
feature_set = 1:100)

## normalize the object using the saved size factors
example_sceset <- normalize(example_sceset)
```

---

norm_counts *Accessors for the 'norm_counts' element of an SCESet object.*

Description

The norm_counts element holds normalised count data as a matrix of non-negative values, one row for each feature (gene, exon, region, etc), and one column for each cell. It is an element of the assayData slot of the SCESet object.

Usage

```
norm_counts(object)

norm_counts(object) <- value

## S4 method for signature 'SCESet'
norm_counts(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_counts(object)<-value

## S4 method for signature 'SCESet'
norm_counts(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_counts(object) <- value
```

Arguments

object	a SCESet object.
value	an integer matrix

Value

a matrix of normalised count data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
norm_counts(example_sceset)
```

`norm_cpm` *Accessors for the 'norm_cpm' (normalised counts per million) element of an SCESet object.*

Description

The `norm_cpm` element of the arrayData slot in an SCESet object holds a matrix containing normalised counts-per-million values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
norm_cpm(object)

norm_cpm(object) <- value

## S4 method for signature 'SCESet'
norm_cpm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_cpm(object)<-value

## S4 method for signature 'SCESet'
norm_cpm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_cpm(object) <- value
```

Arguments

`object`	a `SCESet` object.
`value`	a matrix of class `"numeric"`

Value

a matrix of normalised counts-per-million data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData=sc_example_counts)
norm_cpm(example_sceset)
```

`norm_exprs` *Accessors for the 'norm_exprs' (normalised expression) element of an SCESet object.*

Description

The `norm_exprs` element of the arrayData slot in an SCESet object holds a matrix containing normalised expression values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
norm_exprs(object)

norm_exprs(object) <- value

## S4 method for signature 'SCESet'
norm_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_exprs(object)<-value

## S4 method for signature 'SCESet'
norm_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_exprs(object) <- value
```

Arguments

`object`	a `SCESet` object.
`value`	an integer matrix

Details

The default for normalised expression values is mean-centred and variance-standardised expression data from the `exprs` slot of the `SCESet` object. The function `normaliseExprs` (or `normalizeExprs`) provides more options and functionality for normalising expression data.

Value

a matrix of normalised expression data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
norm_exprs(example_sceset)
```

`norm_fpkm` *Accessors for the 'norm_fpkm' (normalised fragments per kilobase of exon per million reads mapped) element of an SCESet object.*

Description

The `norm_fpkm` element of the arrayData slot in an SCESet object holds a matrix containing normalised fragments per kilobase of exon per million reads mapped (FPKM) values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
norm_fpkm(object)

norm_fpkm(object) <- value

## S4 method for signature 'SCESet'
norm_fpkm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_fpkm(object)<-value

## S4 method for signature 'SCESet'
norm_fpkm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_fpkm(object) <- value
```

Arguments

`object`	a `SCESet` object.
`value`	a matrix of class `"numeric"`

Value

a matrix of normalised FPKM data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
norm_fpkm(example_sceset)
```

---

norm_tpm *Accessors for the 'norm_tpm' (transcripts per million) element of an SCESet object.*

Description

The norm_tpm element of the arrayData slot in an SCESet object holds a matrix containing normalised transcripts-per-million values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
norm_tpm(object)

norm_tpm(object) <- value

## S4 method for signature 'SCESet'
norm_tpm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_tpm(object)<-value

## S4 method for signature 'SCESet'
norm_tpm(object)

## S4 replacement method for signature 'SCESet,matrix'
norm_tpm(object) <- value
```

Arguments

object	a SCESet object.
value	a matrix of class "numeric"

Value

a matrix of normalised transcripts-per-million data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
norm_tpm(example_sceset)
```

`pData<-,SCESet,AnnotatedDataFrame-method`
Replaces phenoData in an SCESet object

Description

SCESet objects contain phenotype information (inherited from the ExpressionSet class). This function conveniently replaces the phenotype data with the value supplied, which must be an AnnotatedDataFrame.

Usage

```
## S4 replacement method for signature 'SCESet,AnnotatedDataFrame'
pData(object) <- value

## S4 replacement method for signature 'SCESet,data.frame'
pData(object) <- value
```

Arguments

`object`	An SCESet object.
`value`	an AnnotatedDataFrame with updated phenoData to replace existing

Value

A matrix of expression count data, where rows correspond to features (e.g. genes) and columns correspond to cells.

Examples

```
## Not run:
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
pData(example_sceset)

## End(Not run)
```

plot *Plot an overview of expression for each cell*

Description

Plot the relative proportion of the library accounted for by the most highly expressed features for each cell for an SCESet dataset.

Usage

```
## S4 method for signature 'SCESet,ANY'
plot(x, y, ...)

plotSCESet(x, block1 = NULL, block2 = NULL, colour_by = NULL,
  nfeatures = 500, exprs_values = "tpm", ncol = 3, linewidth = 1.5,
  theme_size = 10)
```

Arguments

x	an SCESet object
y	optional argument for generic plot functions, not used for plotting an SCESet object
...	arguments passed to plotSCESet
block1	character string defining the column of pData(object) to be used as a factor by which to separate the cells into blocks (separate panels) in the plot. Default is NULL, in which case there is no blocking.
block2	character string defining the column of pData(object) to be used as a factor by which to separate the cells into blocks (separate panels) in the plot. Default is NULL, in which case there is no blocking.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
nfeatures	numeric scalar indicating the number of features to include in the plot.
exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "cpm" (counts per million), "fpkm" (FPKM values), "counts" (counts for each feature) or "exprs" (whatever is in the 'exprs' slot of the SCESet object; if already on the log2 scale, as indicated by the logged slot of the object, then exprs values are set to the power of 2 (so they are back on the raw scale they were on) before making the plot).
ncol	number of columns to use for facet_wrap if only one block is defined.
linewidth	numeric scalar giving the "size" parameter (in ggplot2 parlance) for the lines plotted. Default is 1.5.
theme_size	numeric scalar giving font size to use for the plotting theme

Details

Plots produced by this function are intended to provide an overview of large-scale differences between cells. For each cell, the features are ordered from most-expressed to least-expressed and 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 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.

Value

a ggplot plot object

Examples

```
## Set up an example SCESet
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)

plot(example_sceset, exprs_values = "exprs")
plot(example_sceset, exprs_values = "exprs", colour_by = "Cell_Cycle")
plot(example_sceset, exprs_values = "exprs", block1 = "Treatment",
colour_by = "Cell_Cycle")
plot(example_sceset, exprs_values = "exprs", block1 = "Treatment",
block2 = "Mutation_Status", colour_by = "Cell_Cycle")
# What happens if chosen expression values are not available?
plot(example_sceset, block1 = "Treatment", colour_by = "Cell_Cycle")
```

---

plotDiffusionMap *Plot a diffusion map for an SCESet object*

Description

Produce a diffusion map plot of two components for an `SCESet` dataset.

Usage

```
plotDiffusionMap(object, ...)

plotDiffusionMapSCESet(object, ntop = 500, ncomponents = 2,
  exprs_values = "exprs", colour_by = NULL, shape_by = NULL,
  size_by = NULL, feature_set = NULL, return_SCESet = FALSE,
  scale_features = TRUE, draw_plot = TRUE, theme_size = 10,
  rand_seed = NULL, sigma = NULL, distance = "euclidean",
  legend = "auto", ...)

## S4 method for signature 'SCESet'
plotDiffusionMap(object, ntop = 500, ncomponents = 2,
  exprs_values = "exprs", colour_by = NULL, shape_by = NULL,
```

```
size_by = NULL, feature_set = NULL, return_SCESet = FALSE,
scale_features = FALSE, draw_plot = TRUE, theme_size = 10,
rand_seed = NULL, sigma = NULL, distance = "euclidean",
legend = "auto", ...)
```

Arguments

object	an SCESet object
...	further arguments passed to DiffusionMap
ntop	numeric scalar indicating the number of most variable features to use for the diffusion map. Default is 500, but any ntop argument is overrided if the feature_set argument is non-NULL.
ncomponents	numeric scalar indicating the number of principal components to plot, starting from the first diffusion map component. Default is 2. If ncomponents is 2, then a scatterplot of component 1 vs component 2 is produced. If ncomponents is greater than 2, a pairs plots for the top components is produced. NB: computing many components for the diffusion map can become time consuming.
exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "norm_tpm" (normalised TPM values), "fpkm" (FPKM values), "norm_fpkm" (normalised FPKM values), "counts" (counts for each feature), "norm_counts", "cpm" (counts-per-million), "norm_cpm" (normalised counts-per-million), "exprs" (whatever is in the 'exprs' slot of the SCESet object; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other named element of the assayData slot of the SCESet object that can be accessed with the get_exprs function.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
shape_by	character string defining the column of pData(object) to be used as a factor by which to define the shape of the points in the plot.
size_by	character string defining the column of pData(object) to be used as a factor by which to define the size of points in the plot.
feature_set	character, numeric or logical vector indicating a set of features to use for the diffusion map. If character, entries must all be in featureNames(object). If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to nrow(object).
return_SCESet	logical, should the function return an SCESet object with principal component values for cells in the reducedDimension slot. Default is FALSE, in which case a ggplot object is returned.
scale_features	logical, should the expression values be standardised so that each feature has unit variance? Default is TRUE.
draw_plot	logical, should the plot be drawn on the current graphics device? Only used if return_SCESet is TRUE, otherwise the plot is always produced.
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
rand_seed	(optional) numeric scalar that can be passed to set.seed to make plots reproducible.
sigma	argument passed to DiffusionMap
distance	argument passed to DiffusionMap

legend — character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

Details

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

Value

If return_SCESet is TRUE, then the function returns an SCESet object, otherwise it returns a ggplot object.

References

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

See Also

destiny

Examples

```
## Set up an example SCESet
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]

## Examples plotting diffusion maps
plotDiffusionMap(example_sceset)
plotDiffusionMap(example_sceset, colour_by = "Cell_Cycle")
plotDiffusionMap(example_sceset, colour_by = "Cell_Cycle",
shape_by = "Treatment")
plotDiffusionMap(example_sceset, colour_by = "Cell_Cycle",
shape_by = "Treatment", size_by = "Mutation_Status")
plotDiffusionMap(example_sceset, shape_by = "Treatment",
size_by = "Mutation_Status")
plotDiffusionMap(example_sceset, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status")

plotDiffusionMap(example_sceset, shape_by = "Treatment",
return_SCESet = TRUE)
```

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

Description

Plot explanatory variables ordered by percentage of phenotypic variance explained

Usage

```
plotExplanatoryVariables(object, method = "density", exprs_values = "exprs",
  nvars_to_plot = 10, min_marginal_r2 = 0, variables = NULL,
  return_object = FALSE, theme_size = 10, ...)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
method	character scalar indicating the type of plot to produce. If "density", the function produces a density plot of R-squared values for each variable when fitted as the only explanatory variable in a linear model. If "pairs", then the function produces a pairs plot of the explanatory variables ordered by the percentage of feature expression variance (as measured by R-squared in a marginal linear model) explained.
exprs_values	which slot of the assayData in the object should be used to define expression? Valid options are "exprs" (default), "tpm", "fpkm", "cpm", and "counts".
nvars_to_plot	integer, the number of variables to plot in the pairs plot. Default value is 10.
min_marginal_r2	numeric scalar giving 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.
variables	optional character vector giving the variables to be plotted. Default is NULL, in which case all variables in pData(object) are considered and the nvars_to_plot variables with the highest median marginal R-squared are plotted.
return_object	logical, should an SCESet object with median marginal R-squared values added to varMetadata(object) be returned?
theme_size	numeric scalar giving font size to use for the plotting theme
...	parameters to be passed to pairs.

Details

If the method argument is "pairs", then the function produces a pairs plot of the explanatory variables ordered by the percentage of feature expression variance (as measured by R-squared in a marginal linear model) explained by variable. Median percentage R-squared is reported on the plot for each variable. Discrete variables are coerced to a factor and plotted as integers with jittering. Variables with only one unique value are quietly ignored.

Value

A ggplot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]
example_sceset <- calculateQCMetrics(example_sceset)
vars <- names(pData(example_sceset))[c(2:3, 5:14)]
plotExplanatoryVariables(example_sceset, variables=vars)
```

---

plotExpression *Plot expression values for a set of features (e.g. genes or transcripts)*

Description

Plot expression values for a set of features (e.g. genes or transcripts)

Usage

```
plotExpression(object, ...)

plotExpressionSCESet(object, features, x = NULL, exprs_values = "exprs",
  colour_by = NULL, shape_by = NULL, size_by = NULL, ncol = 2,
  xlab = NULL, show_median = FALSE, show_violin = TRUE,
  show_smooth = FALSE, alpha = 0.6, theme_size = 10,
  log2_values = FALSE, size = NULL, scales = "fixed", se = TRUE,
  jitter = "swarm")

plotExpressionDefault(object, aesth, ncol = 2, xlab = NULL, ylab = NULL,
  show_median = FALSE, show_violin = TRUE, show_smooth = FALSE,
  alpha = 0.6, size = NULL, scales = "fixed", one_facet = FALSE,
  se = TRUE, jitter = "swarm")

## S4 method for signature 'SCESet'
plotExpression(object, ...)

## S4 method for signature 'data.frame'
plotExpression(object, ...)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared. For the plotExpressionDefault method, the object argument is a data.frame in 'long' format providing expression values for a set of features to plot, plus metadata used in the aesth argument, but this is not meant to be a user-level operation.

...	optional arguments (from those listed above) passed to plotExpressionSCESet or plotExpressionDefault
features	a character vector of feature names or Boolean vector or numeric vector of indices indicating which features should have their expression values plotted
x	character string providing a column name of pData(object) or a feature name (i.e. gene or transcript) to plot on the x-axis in the expression plot(s). If a feature name, then expression values for the feature will be plotted on the x-axis for each subplot.
exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "norm_tpm" (normalised TPM values), "fpkm" (FPKM values), "norm_fpkm" (normalised FPKM values), "counts" (counts for each feature), "norm_counts", "cpm" (counts-per-million), "norm_cpm" (normalised counts-per-million), "exprs" (whatever is in the 'exprs' slot of the SCESet object; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other slots that have been added to the "assayData" slot by the user.
colour_by	optional character string supplying name of a column of pData(object) which will be used as a variable by which to colour expression values on the plot.
shape_by	optional character string supplying name of a column of pData(object) which will be used as a variable to define the shape of points for expression values on the plot.
size_by	optional character string supplying name of a column of pData(object) which will be used as a variable to define the size of points for expression values on the plot.
ncol	number of columns to be used for the panels of the plot
xlab	label for x-axis; if NULL (default), then x will be used as the x-axis label
show_median	logical, show the median for each group on the plot
show_violin	logical, show a violin plot for the distribution for each group on the plot
show_smooth	logical, show a smoothed fit through the expression values on the plot
alpha	numeric value between 0 (completely transparent) and 1 (completely solid) defining how transparent plotted points (cells) should be. Points are jittered horizontally if the x-axis value is categorical rather than numeric to avoid overplotting.
theme_size	numeric scalar giving default font size for plotting theme (default is 10)
log2_values	should the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes)?
size	numeric scalar optionally providing size for points if size_by argument is not given. Default is NULL, in which case **ggplot2** default is used.
scales	character scalar, should scales be fixed ("fixed"), free ("free"), or free in one dimension ("free_x"; "free_y", the default). Passed to the scales argument in the facet_wrap function from the ggplot2 package.
se	logical, should standard errors be shown (default TRUE) for the smoothed fit through the cells. (Ignored if show_smooth is FALSE).
jitter	character scalar to define whether points are to be jittered ("jitter") or presented in a "beeswarm" style (if "swarm"; default). "Beeswarm" style usually looks more attractive, but for datasets with a large number of cells, or for dense plots, the jitter option may work better.
aesth	an aes object to use in the call to ggplot.

ylab	character string defining a label for the y-axis (y-axes) of the plot.
one_facet	logical, should expression values for features be plotted in one facet instead of mutiple facets, one per feature? Default if x = NULL.

Details

Plot expression values (default log2(transcripts-per-million + 1), if available) for a set of features.

Value

a ggplot plot object

Examples

```
## prepare data
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset)

## default plot
plotExpression(example_sceset, 1:15)
plotExpression(example_sceset, 1:15, jitter = "jitter")

## plot expression against an x-axis value
plotExpression(example_sceset, 1:6, "Mutation_Status")

## explore options
plotExpression(example_sceset, 1:6, x="Mutation_Status", exprs_values="exprs",
colour_by="Cell_Cycle", show_violin=TRUE, show_median=TRUE)
plotExpression(example_sceset, 1:6, x="Mutation_Status", exprs_values="counts",
colour_by="Cell_Cycle", show_violin=TRUE, show_median=TRUE)

## plot expression against expression values for Gene_0004
plotExpression(example_sceset, 1:4, "Gene_0004")
plotExpression(example_sceset, 1:4, "Gene_0004", show_smooth = TRUE)
plotExpression(example_sceset, 1:4, "Gene_0004", show_smooth = TRUE, se = FALSE)
```

---

plotExprsFreqVsMean *Plot frequency of expression against mean expression level*

Description

Plot frequency of expression against mean expression level

Usage

```
plotExprsFreqVsMean(object, feature_set = NULL, feature_controls = NULL,
  shape = 1, alpha = 0.7, show_smooth = TRUE, se = TRUE, ...)
```

Arguments

object	an SCESet object.
feature_set	character, numeric or logical vector indicating a set of features to plot. If character, entries must all be in featureNames(object). If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to nrow(object). If NULL, then the function checks if feature controls are defined. If so, then only feature controls are plotted, if not, then all features are plotted.
feature_controls	character, numeric or logical vector indicating a set of features to be used as feature controls for computing technical dropout effects. If character, entries must all be in featureNames(object). If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to nrow(object). If NULL, then the function checks if feature controls are defined. If so, then these feature controls are used.
shape	(optional) numeric scalar to define the plotting shape.
alpha	(optional) numeric scalar (in the interval 0 to 1) to define the alpha level (transparency) of plotted points.
show_smooth	logical, should a smoothed fit through feature controls (if available; all features if not) be shown on the plot? Lowess used if a small number of feature controls. For details see geom_smooth.
se	logical, should standard error (confidence interval) be shown for smoothed fit?
...	further arguments passed to plotMetadata (should only be size, if anythin).

Details

This function plots gene expression frequency versus mean expression level, which can be useful to assess the effects of technical dropout in the dataset. We fit a non-linear least squares curve for the relationship between expression frequency and mean expression and use this to define the number of genes above high technical dropout and the numbers of genes that are expressed in at least 50 of genes to be treated as feature controls can be specified, otherwise any feature controls previously defined are used.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
ex_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
ex_sceset <- calculateQCMetrics(ex_sceset)
plotExprsFreqVsMean(ex_sceset)

ex_sceset <- calculateQCMetrics(
ex_sceset, feature_controls = list(controls1 = 1:20,
                                   controls2 = 500:1000),
                                   cell_controls = list(set_1 = 1:5,
                                   set_2 = 31:40))
```

```
plotExprsFreqVsMean(ex_sceset)
```

`plotExprsVsTxLength` *Plot expression against transcript length*

Description

Plot expression values from an SCESet object against transcript length values defined in the SCESet object or supplied as an argument.

Usage

```
plotExprsVsTxLength(object, tx_length = "median_feat_eff_len",
  exprs_values = "exprs", colour_by = NULL, shape_by = NULL,
  size_by = NULL, xlab = NULL, show_exprs_sd = FALSE,
  show_smooth = FALSE, alpha = 0.6, theme_size = 10,
  log2_values = FALSE, size = NULL, se = TRUE)
```

Arguments

`object`	an `SCESet` object
`tx_length`	transcript lengths to plot on the x-axis. Can be one of: (1) the name of a column of `fData(object)` containing the transcript length values, or (2) the name of an element of `assayData(object)` containing a matrix of transcript length values, or (3) a numeric vector of length equal to the number of rows of `object` (number of features).
`exprs_values`	character string indicating which values should be used as the expression values for this plot. Valid arguments are `"tpm"` (default; transcripts per million), `"norm_tpm"` (normalised TPM values), `"fpkm"` (FPKM values), `"norm_fpkm"` (normalised FPKM values), `"counts"` (counts for each feature), `"norm_counts"`, `"cpm"` (counts-per-million), `"norm_cpm"` (normalised counts-per-million), `"exprs"` (whatever is in the `'exprs'` slot of the `SCESet` object; default), `"norm_exprs"` (normalised expression values) or `"stand_exprs"` (standardised expression values) or any other slots that have been added to the `"assayData"` slot by the user.
`colour_by`	optional character string supplying name of a column of `fData(object)` which will be used as a variable by which to colour expression values on the plot.
`shape_by`	optional character string supplying name of a column of `fData(object)` which will be used as a variable to define the shape of points for expression values on the plot.
`size_by`	optional character string supplying name of a column of `fData(object)` which will be used as a variable to define the size of points for expression values on the plot.
`xlab`	label for x-axis; if `NULL` (default), then `x` will be used as the x-axis label
`show_exprs_sd`	logical, show the standard deviation of expression values for each feature on the plot
`show_smooth`	logical, show a smoothed fit through the expression values on the plot

alpha	numeric value between 0 (completely transparent) and 1 (completely solid) defining how transparent plotted points (cells) should be. Points are jittered horizontally if the x-axis value is categorical rather than numeric to avoid overplotting.
theme_size	numeric scalar giving default font size for plotting theme (default is 10)
log2_values	should the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes)?
size	numeric scalar optionally providing size for points if size_by argument is not given. Default is NULL, in which case **ggplot2** default is used.
se	logical, should standard errors be shown (default TRUE) for the smoothed fit through the cells. (Ignored if show_smooth is FALSE).

Value

a ggplot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
fd <- new("AnnotatedDataFrame", data =
data.frame(gene_id = rownames(sc_example_counts),
        feature_id = paste("feature", rep(1:500, each = 4), sep = "_"),
     median_tx_length = rnorm(2000, mean = 5000, sd = 500)))
rownames(fd) <- rownames(sc_example_counts)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd,
featureData = fd)

plotExprsVsTxLength(example_sceset, "median_tx_length")
plotExprsVsTxLength(example_sceset, "median_tx_length", show_smooth = TRUE)
plotExprsVsTxLength(example_sceset, "median_tx_length", show_smooth = TRUE,
show_exprs_sd = TRUE)

## using matrix of tx length values in assayData(object)
set_exprs(example_sceset, "tx_len") <-
matrix(rnorm(ncol(example_sceset) * nrow(example_sceset), mean = 5000, sd = 500),
nrow = nrow(example_sceset))
plotExprsVsTxLength(example_sceset, "tx_len", show_smooth = TRUE,
show_exprs_sd = TRUE)

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

---

plotFeatureData *Plot feature (gene) data from an SCESet object*

Description

Plot feature (gene) data from an SCESet object

Usage

```
plotFeatureData(object, aesth = aes_string(x = "n_cells_exprs", y =
  "prop_total_counts"), theme_size = 10, ...)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
aesth	aesthetics function call to pass to ggplot. This function expects at least x and y variables to be supplied. The default is to produce a density plot of number of cells expressing the feature (requires `calculateQCMetrics` to have been run on the SCESet object prior).
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
...	arguments passed to `plotMetadata`.

Details

Plot feature (gene) data from an SCESet object. If one variable is supplied then a density plot will be returned. If both variables are continuous (numeric) then a scatter plot will be returned. If one variable is discrete and one continuous then a violin plot with jittered points overlaid will be returned. If both variables are discrete then a jitter plot will be produced. The object returned is a ggplot object, so further layers and plotting options (titles, facets, themes etc) can be added.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset)
plotFeatureData(example_sceset, aesth=aes(x=n_cells_exprs, y=pct_total_counts))
```

---

plotHighestExprs *Plot the features with the highest expression values*

Description

Plot the features with the highest expression values

Usage

```
plotHighestExprs(object, col_by_variable = "total_features", n = 50,
  drop_features = NULL, exprs_values = "counts",
  feature_names_to_plot = NULL)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
col_by_variable	variable name (must be a column name of pData(object)) to be used to assign colours to cell-level values.
n	numeric scalar giving the number of the most expressed features to show. Default value is 50.
drop_features	a character, logical or numeric vector indicating which features (e.g. genes, transcripts) to drop when producing the plot. For example, control genes might be dropped to focus attention on contribution from endogenous rather than synthetic genes.
exprs_values	which slot of the assayData in the object should be used to define expression? Valid options are "counts" (default), "tpm", "fpkm" and "exprs".
feature_names_to_plot	character scalar indicating which column of the featureData slot in the object is to be used for the feature names displayed on the plot. Default is NULL, in which case featureNames(object) is used.

Details

Plot the percentage of counts accounted for by the top n most highly expressed features across the dataset.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset, feature_controls = 1:500)
plotHighestExprs(example_sceset, col_by_variable="total_features")
plotHighestExprs(example_sceset, col_by_variable="Mutation_Status")
```

---

plotMDS *Produce a multidimensional scaling plot for an SCESet object*

Description

#' Produce an MDS plot from the cell pairwise distance data in an SCESet dataset.

Usage

```
plotMDS(object, ...)

plotMDSSCESet(object, ncomponents = 2, colour_by = NULL, shape_by = NULL,
  size_by = NULL, return_SCESet = FALSE, draw_plot = TRUE,
  theme_size = 10, legend = "auto")

## S4 method for signature 'SCESet'
plotMDS(object, ncomponents = 2, colour_by = NULL,
  shape_by = NULL, size_by = NULL, return_SCESet = FALSE,
  draw_plot = TRUE, theme_size = 10, legend = "auto")
```

Arguments

object	an SCESet object
...	arguments passed to S4 plotMDS method
ncomponents	numeric scalar indicating the number of principal components to plot, starting from the first principal component. Default is 2. If ncomponents is 2, then a scatterplot of PC2 vs PC1 is produced. If ncomponents is greater than 2, a pairs plots for the top components is produced. NB: computing more than two components for t-SNE can become very time consuming.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
shape_by	character string defining the column of pData(object) to be used as a factor by which to define the shape of the points in the plot.
size_by	character string defining the column of pData(object) to be used as a factor by which to define the size of points in the plot.
return_SCESet	logical, should the function return an SCESet object with principal component values for cells in the reducedDimension slot. Default is FALSE, in which case a ggplot object is returned.
draw_plot	logical, should the plot be drawn on the current graphics device? Only used if return_SCESet is TRUE, otherwise the plot is always produced.
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
legend	character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

Details

The function cmdscale is used internally to compute the multidimensional scaling components to plot.

Value

If return_SCESet is TRUE, then the function returns an SCESet object, otherwise it returns a ggplot object.

Examples

```
## Set up an example SCESet
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]
example_sceset <- calculateQCMetrics(example_sceset)

## define cell-cell distances
cellDist(example_sceset) <- as.matrix(dist(t(exprs(example_sceset))))

## Examples plotting
plotMDS(example_sceset)
plotMDS(example_sceset, colour_by = "Cell_Cycle")
plotMDS(example_sceset, colour_by = "Cell_Cycle",
shape_by = "Treatment")

## define cell-cell distances differently
cellDist(example_sceset) <- as.matrix(dist(t(counts(example_sceset)),
method = "canberra"))
plotMDS(example_sceset, colour_by = "Cell_Cycle",
shape_by = "Treatment", size_by = "Mutation_Status")
```

---

plotMetadata *Plot metadata for cells or features*

Description

Plot metadata for cells or features

Usage

```
plotMetadata(object, aesth = aes_string(x = "log10(total_counts)", y =
  "total_features"), shape = NULL, alpha = NULL, size = NULL,
  theme_size = 10)
```

Arguments

object	a data.frame (or object that can be coerced to such) object containing metadata in columns to plot.
aesth	aesthetics function call to pass to ggplot. This function expects at least x and y variables to be supplied. The default is to plot total_features against log10(total_counts).
shape	numeric scalar to define the plotting shape. Ignored if shape is included in the aesth argument.
alpha	numeric scalar (in the interval 0 to 1) to define the alpha level (transparency) of plotted points. Ignored if alpha is included in the aesth argument.
size	numeric scalar to define the plotting size. Ignored if size is included in the aesth argument.
theme_size	numeric scalar giving default font size for plotting theme (default is 10)

Details

Plot cell or feature metadata from an SCESet object. If one variable is supplied then a density plot will be returned. If both variables are continuous (numeric) then a scatter plot will be returned. If one variable is discrete and one continuous then a violin plot with jittered points overlaid will be returned. If both variables are discrete then a jitter plot will be produced. The object returned is a ggplot object, so further layers and plotting options (titles, facets, themes etc) can be added.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset)
plotMetadata(pData(example_sceset))
```

---

plotPCA *Plot PCA for an SCESet object*

Description

Produce a principal components analysis (PCA) plot of two or more principal components for an SCESet dataset.

Usage

```
plotPCASCESet(object, ntop = 500, ncomponents = 2, exprs_values = "exprs",
  colour_by = NULL, shape_by = NULL, size_by = NULL, feature_set = NULL,
  return_SCESet = FALSE, scale_features = TRUE, draw_plot = TRUE,
  pca_data_input = "exprs", selected_variables = NULL,
  detect_outliers = FALSE, theme_size = 10, legend = "auto")

## S4 method for signature 'SCESet'
plotPCA(object, ntop = 500, ncomponents = 2,
  exprs_values = "exprs", colour_by = NULL, shape_by = NULL,
  size_by = NULL, feature_set = NULL, return_SCESet = FALSE,
  scale_features = TRUE, draw_plot = TRUE, pca_data_input = "exprs",
  selected_variables = NULL, detect_outliers = FALSE, theme_size = 10,
  legend = "auto")
```

Arguments

object	an SCESet object
ntop	numeric scalar indicating the number of most variable features to use for the PCA. Default is 500, but any ntop argument is overrided if the feature_set argument is non-NULL.

ncomponents — numeric scalar indicating the number of principal components to plot, starting from the first principal component. Default is 2. If ncomponents is 2, then a scatterplot of PC2 vs PC1 is produced. If ncomponents is greater than 2, a pairs plots for the top components is produced.

exprs_values — character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "norm_tpm" (normalised TPM values), "fpkm" (FPKM values), "norm_fpkm" (normalised FPKM values), "counts" (counts for each feature), "norm_counts", "cpm" (counts-per-million), "norm_cpm" (normalised counts-per-million), "exprs" (whatever is in the 'exprs' slot of the SCESet object; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other named element of the assayData slot of the SCESet object that can be accessed with the get_exprs function.

colour_by — character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.

shape_by — character string defining the column of pData(object) to be used as a factor by which to define the shape of the points in the plot.

size_by — character string defining the column of pData(object) to be used as a factor by which to define the size of points in the plot.

feature_set — character, numeric or logical vector indicating a set of features to use for the PCA. If character, entries must all be in featureNames(object). If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to nrow(object).

return_SCESet — logical, should the function return an SCESet object with principal component values for cells in the reducedDimension slot. Default is FALSE, in which case a ggplot object is returned.

scale_features — logical, should the expression values be standardised so that each feature has unit variance? Default is TRUE.

draw_plot — logical, should the plot be drawn on the current graphics device? Only used if return_SCESet is TRUE, otherwise the plot is always produced.

pca_data_input — character argument defining which data should be used as input for the PCA. Possible options are "exprs" (default), which uses expression data to produce a PCA at the cell level; "pdata" which uses numeric variables from pData(object) to do PCA at the cell level; and "fdata" which uses numeric variables from fData(object) to do PCA at the feature level.

selected_variables — character vector indicating which variables in pData(object) to use for the phenotype-data based PCA. Ignored if the argument pca_data_input is anything other than "pdata".

detect_outliers — logical, should outliers be detected in the PC plot? Only an option when pca_data_input argument is "pdata". Default is FALSE.

theme_size — numeric scalar giving default font size for plotting theme (default is 10).

legend — character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

... — further arguments passed to plotPCASCESet

Details

The function prcomp is used internally to do the PCA. The function checks whether the object has standardised expression values (by looking at stand_exprs(object)). If yes, the existing standardised expression values are used for the PCA. If not, then standardised expression values are computed using scale (with feature-wise unit variances or not according to the scale_features argument), added to the object and PCA is done using these new standardised expression values.

If the arguments detect_outliers and return_SCESet are both TRUE, then the element $outlier is added to the pData (phenotype data) slot of the SCESet object. This element contains indicator values about whether or not each cell has been designated as an outlier based on the PCA. These values can be accessed for filtering low quality cells with, foe example, example_sceset$outlier.

Value

either a ggplot plot object or an SCESet object

Examples

```
## Set up an example SCESet
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]

## Examples plotting PC1 and PC2
plotPCA(example_sceset)
plotPCA(example_sceset, colour_by = "Cell_Cycle")
plotPCA(example_sceset, colour_by = "Cell_Cycle", shape_by = "Treatment")
plotPCA(example_sceset, colour_by = "Cell_Cycle", shape_by = "Treatment",
size_by = "Mutation_Status")
plotPCA(example_sceset, shape_by = "Treatment", size_by = "Mutation_Status")
plotPCA(example_sceset, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status")

## experiment with legend
example_subset <- example_sceset[, example_sceset$Treatment == "treat1"]
plotPCA(example_subset, colour_by = "Cell_Cycle", shape_by = "Treatment", legend = "all")

plotPCA(example_sceset, shape_by = "Treatment", return_SCESet = TRUE)

## Examples plotting more than 2 PCs
plotPCA(example_sceset, ncomponents = 8)
plotPCA(example_sceset, ncomponents = 4, colour_by = "Treatment",
shape_by = "Mutation_Status")
```

---

plotPhenoData *Plot phenotype data from an SCESet object*

Description

Plot phenotype data from an SCESet object

Usage

```
plotPhenoData(object, aesth = aes_string(x = "log10(total_counts)", y =
  "total_features"), theme_size = 10, ...)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
aesth	aesthetics function call to pass to ggplot. This function expects at least x and y variables to be supplied. The default is to plot total_features against log10(total_counts).
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
...	arguments passed to plotMetadata.

Details

Plot phenotype data from an SCESet object. If one variable is supplied then a density plot will be returned. If both variables are continuous (numeric) then a scatter plot will be returned. If one variable is discrete and one continuous then a violin plot with jittered points overlaid will be returned. If both variables are discrete then a jitter plot will be produced. The object returned is a ggplot object, so further layers and plotting options (titles, facets, themes etc) can be added.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset)
plotPhenoData(example_sceset, aesth = aes_string(x = "log10(total_counts)",
y = "total_features", colour = "Mutation_Status"))
```

---

plotPlatePosition *Plot cells in plate positions*

Description

Plots cells in their position on a plate, coloured by phenotype data or feature expression.

Usage

```
plotPlatePosition(object, plate_position = NULL, colour_by = NULL,
  x_position = NULL, y_position = NULL, theme_size = 24,
  legend = "auto")
```

Arguments

object	an SCESet object. If object$plate_position is not NULL, then this will be used to define each cell's position on the plate, unless the plate_position argument is specified.
plate_position	optional character vector providing a position on the plate for each cell (e.g. A01, B12, etc, where letter indicates row and number indicates column). Specifying this argument overrides any plate position information extracted from the SCESet object.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
x_position	numeric vector providing x-axis positions for the cells (ignored if plate_position is not NULL)
y_position	numeric vector providing y-axis positions for the cells (ignored if plate_position is not NULL)
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
legend	character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

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. So 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. If object$plate_position or the plate_position argument are used to define plate positions, then positions should be provided in this format. Alternatively, numeric values to be used as x- and y-coordinates by supplying both the x_position and y_position arguments to the function.

Examples

```
## prepare data
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- calculateQCMetrics(example_sceset)

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

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

plotPlatePosition(example_sceset, colour_by = "Gene_0004")
```

plotQC *Produce QC diagnostic plots*

Description

Produce QC diagnostic plots

Usage

```
plotQC(object, type = "highest-expression", ...)
```

Arguments

object	an SCESet object containing expression values and experimental information. Must have been appropriately prepared.
type	character scalar providing type of QC plot to compute: "highest-expression" (showing features with highest expression), "find-pcs" (showing the most important principal components for a given variable), "explanatory-variables" (showing a set of explanatory variables plotted against each other, ordered by marginal variance explained), or "exprs-mean-vs-freq" (plotting the mean expression levels against the frequency of expression for a set of features).
...	arguments passed to plotHighestExprs, plotImportantPCs, plotExplanatoryVariables and plotExprsMeanVsFreq as appropriate.

Details

Display useful quality control plots to help with pre-processing of data and identification of potentially problematic features and cells.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]
example_sceset <- calculateQCMetrics(example_sceset)
plotQC(example_sceset, type="high", col_by_variable="Mutation_Status")
plotQC(example_sceset, type="find", variable="total_features")
vars <- names(pData(example_sceset))[c(2:3, 5:14)]
plotQC(example_sceset, type="expl", variables=vars)
```

plotReducedDim *Plot reduced dimension representation of cells*

Description

Plot reduced dimension representation of cells

Usage

```
plotReducedDim(object, ...)

plotReducedDim.default(df_to_plot, ncomponents = 2, colour_by = NULL,
  shape_by = NULL, size_by = NULL, percentVar = NULL, theme_size = 10,
  legend = "auto")

plotReducedDim.SCESet(object, ncomponents = 2, colour_by = NULL,
  shape_by = NULL, size_by = NULL, theme_size = 10, legend = "auto")

## S4 method for signature 'SCESet'
plotReducedDim(object, ncomponents = 2, colour_by = NULL,
  shape_by = NULL, size_by = NULL, theme_size = 10, legend = "auto")

## S4 method for signature 'data.frame'
plotReducedDim(object, ncomponents = 2,
  colour_by = NULL, shape_by = NULL, size_by = NULL, percentVar = NULL,
  legend = "auto")
```

Arguments

object	an SCESet object with a non-NULL reducedDimension slot.
...	optional arguments (from those listed above) passed to plotReducedDim.SCESet or plotReducedDim.default
df_to_plot	data.frame containing a reduced dimension represenation of cells and optional metadata for the plot.
ncomponents	numeric scalar indicating the number of principal components to plot, starting from the first principal component. Default is 2. If ncomponents is 2, then a scatterplot of Dimension 2 vs Dimension 1 is produced. If ncomponents is greater than 2, a pairs plots for the top dimensions is produced.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
shape_by	character string defining the column of pData(object) to be used as a factor by which to define the shape of the points in the plot.
size_by	character string defining the column of pData(object) to be used as a factor by which to define the size of points in the plot.
percentVar	numeric vector giving the proportion of variance in expression explained by each reduced dimension. Only expected to be used internally in the plotPCA function.
theme_size	numeric scalar giving default font size for plotting theme (default is 10).

legend — character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

Details

The function plotReducedDim.default assumes that the first ncomponents columns of df_to_plot contain the reduced dimension components to plot, and that any subsequent columns define factors for colour_by, shape_by and size_by in the plot.

Value

a ggplot plot object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]

reducedDimension(example_sceset) <- prcomp(t(exprs(example_sceset)), scale. = TRUE)$x
plotReducedDim(example_sceset)
plotReducedDim(example_sceset, colour_by="Cell_Cycle")
plotReducedDim(example_sceset, colour_by="Cell_Cycle", shape_by="Treatment")
plotReducedDim(example_sceset, colour_by="Cell_Cycle", size_by="Treatment")
plotReducedDim(example_sceset, ncomponents=5)
plotReducedDim(example_sceset, ncomponents=5, colour_by="Cell_Cycle", shape_by="Treatment")
```

---

plotTSNE *Plot t-SNE for an SCESet object*

Description

Produce a t-distributed stochastic neighbour embedding (t-SNE) plot of two components for an SCESet dataset.

Usage

```
plotTSNE(object, ...)

## S4 method for signature 'SCESet'
plotTSNE(object, ntop = 500, ncomponents = 2,
  exprs_values = "exprs", colour_by = NULL, shape_by = NULL,
  size_by = NULL, feature_set = NULL, return_SCESet = FALSE,
  scale_features = TRUE, draw_plot = TRUE, theme_size = 10,
  rand_seed = NULL, perplexity = floor(ncol(object)/5), legend = "auto",
  ...)
```

Arguments

object	an SCESet object
...	further arguments passed to Rtsne
ntop	numeric scalar indicating the number of most variable features to use for the t-SNE Default is 500, but any ntop argument is overrided if the feature_set argument is non-NULL.
ncomponents	numeric scalar indicating the number of t-SNE components to plot, starting from the first t-SNE component. Default is 2. If ncomponents is 2, then a scatterplot of component 1 vs component 2 is produced. If ncomponents is greater than 2, a pairs plots for the top components is produced. NB: computing more than two components for t-SNE can become very time consuming.
exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (default; transcripts per million), "norm_tpm" (normalised TPM values), "fpkm" (FPKM values), "norm_fpkm" (normalised FPKM values), "counts" (counts for each feature), "norm_counts", "cpm" (counts-per-million), "norm_cpm" (normalised counts-per-million), "exprs" (whatever is in the 'exprs' slot of the SCESet object; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values), or any other named element of the assayData slot of the SCESet object that can be accessed with the get_exprs function.
colour_by	character string defining the column of pData(object) to be used as a factor by which to colour the points in the plot.
shape_by	character string defining the column of pData(object) to be used as a factor by which to define the shape of the points in the plot.
size_by	character string defining the column of pData(object) to be used as a factor by which to define the size of points in the plot.
feature_set	character, numeric or logical vector indicating a set of features to use for the t-SNE calculation. If character, entries must all be in featureNames(object). If numeric, values are taken to be indices for features. If logical, vector is used to index features and should have length equal to nrow(object).
return_SCESet	logical, should the function return an SCESet object with principal component values for cells in the reducedDimension slot. Default is FALSE, in which case a ggplot object is returned.
scale_features	logical, should the expression values be standardised so that each feature has unit variance? Default is TRUE.
draw_plot	logical, should the plot be drawn on the current graphics device? Only used if return_SCESet is TRUE, otherwise the plot is always produced.
theme_size	numeric scalar giving default font size for plotting theme (default is 10).
rand_seed	(optional) numeric scalar that can be passed to set.seed to make plots reproducible.
perplexity	numeric scalar value defining the "perplexity parameter" for the t-SNE plot. Passed to Rtsne - see documentation for that package for more details.
legend	character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternatives are "all" (show all legends) or "none" (hide all legends).

Details

The function Rtsne is used internally to compute the t-SNE.

Value

If return_SCESet is TRUE, then the function returns an SCESet object, otherwise it returns a ggplot object.

References

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

See Also

Rtsne

Examples

```
## Set up an example SCESet
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]

## Examples plotting PC1 and PC2
plotTSNE(example_sceset, perplexity = 10)
plotTSNE(example_sceset, colour_by = "Cell_Cycle", perplexity = 10)
plotTSNE(example_sceset, colour_by = "Cell_Cycle", shape_by = "Treatment",
perplexity = 10)
plotTSNE(example_sceset, colour_by = "Cell_Cycle", shape_by = "Treatment",
size_by = "Mutation_Status", perplexity = 10)
plotTSNE(example_sceset, shape_by = "Treatment", size_by = "Mutation_Status",
perplexity = 5)
plotTSNE(example_sceset, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status", perplexity = 5)

plotTSNE(example_sceset, shape_by = "Treatment", return_SCESet = TRUE,
perplexity = 10)
```

---

readKallistoResults *Read kallisto results from a batch of jobs*

Description

After generating transcript/feature abundance results using kallisto for a batch of samples, read these abundance values into an SCESet object.

Usage

```
readKallistoResults(kallisto_log = NULL, samples = NULL,
  directories = NULL, read_h5 = FALSE, kallisto_version = "current",
  logExprsOffset = 1, verbose = TRUE)
```

Arguments

kallisto_log	list, generated by runKallisto. If provided, then samples and directories arguments are ignored.
samples	character vector providing a set of sample names to use for the abundance results.
directories	character vector providing a set of directories containing kallisto abundance results to be read in.
read_h5	logical, should the bootstrap results be read in from the HDF5 objects produced by kallisto?
kallisto_version	character string indicating whether or not the version of kallisto to be used is "pre-0.42.2" or "current". This is required because the kallisto developers changed the output file extensions and added features in version 0.42.2.
logExprsOffset	numeric scalar, providing the offset used when doing log2-transformations of expression data to avoid trying to take logs of zero. Default offset value is 1.
verbose	logical, should function provide output about progress?

Details

This function expects to find only one set of kallisto abundance results per directory; multiple adundance results in a given directory will be problematic.

Value

an SCESet object

Examples

```
## Not run:
kallisto_log <- runKallisto("targets.txt", "transcripts.idx", single_end=FALSE,
         output_prefix="output", verbose=TRUE, n_bootstrap_samples=10)
sceset <- readKallistoResults(kallisto_log)

## End(Not run)
```

---

readKallistoResultsOneSample

Read kallisto results for a single sample into a list

Description

Read kallisto results for a single sample into a list

Usage

```
readKallistoResultsOneSample(directory, read_h5 = FALSE,
  kallisto_version = "current")
```

Arguments

directory	character string giving the path to the directory containing the kallisto results for the sample.
read_h5	logical, if TRUE then read in bootstrap results from the HDF5 object produced by kallisto.
kallisto_version	character string indicating whether or not the version of kallisto to be used is "pre-0.42.2" or "current". This is required because the kallisto developers changed the output file extensions and added features in version 0.42.2.

Details

The directory is expected to contain results for just a single sample. Putting more than one sample's results in the directory will result in unpredictable behaviour with this function. The function looks for the files (with the default names given by kallisto) 'abundance.txt', 'run_info.json' and (if read_h5=TRUE) 'abundance/h5'. If these files are missing, or if results files have different names, then this function will not find them.

Value

A list with two elements: (1) a data.frame abundance with columns for 'target_id' (feature, transcript, gene etc), 'length' (feature length), 'eff_length' (effective feature length), 'est_counts' (estimated feature counts), 'tpm' (transcripts per million) and possibly many columns containing bootstrap estimated counts; and (2) a list run_info with details about the kallisto run that generated the results.

Examples

```
# If kallisto results are in the directory "output", then call:
# readKallistoResultsOneSample("output")
```

---

readSalmonResults *Read Salmon results from a batch of jobs*

Description

After generating transcript/feature abundance results using Salmon for a batch of samples, read these abundance values into an SCESet object.

Usage

```
readSalmonResults(Salmon_log = NULL, samples = NULL, directories = NULL,
  logExprsOffset = 1, verbose = TRUE)
```

Arguments

Salmon_log	list, generated by runSalmon. If provided, then samples and directories arguments are ignored.
samples	character vector providing a set of sample names to use for the abundance results.

directories	character vector providing a set of directories containing Salmon abundance results to be read in.
logExprsOffset	numeric scalar, providing the offset used when doing log2-transformations of expression data to avoid trying to take logs of zero. Default offset value is 1.
verbose	logical, should function provide output about progress?

Details

This function expects to find only one set of Salmon abundance results per directory; multiple adundance results in a given directory will be problematic.

Value

an SCESet object

Examples

```
## Not run:
## Define output directories in a vector called here "Salmon_dirs"
## and sample names as "Salmon_samples"
sceset <- readSalmonResults(samples = Salmon_samples,
directories = Salmon_dirs)

## End(Not run)
```

---

readSalmonResultsOneSample

Read Salmon results for a single sample into a list

Description

Read Salmon results for a single sample into a list

Usage

```
readSalmonResultsOneSample(directory)
```

Arguments

directory	character string giving the path to the directory containing the Salmon results for the sample.

Details

The directory is expected to contain results for just a single sample. Putting more than one sample's results in the directory will result in unpredictable behaviour with this function. The function looks for the files (with the default names given by Salmon) 'quant.sf', 'stats.tsv', 'libFormatCounts.txt' and the sub-directories 'logs' (which contains a log file) and 'libParams' (which contains a file detailing the fragment length distribution). If these files are missing, or if results files have different names, then this function will not find them.

This function will work for Salmon v0.7.x and greater, as the name of one of the default output directories was changed from "aux" to "aux_info" in Salmon v0.7.

Value

A list with two elements: (1) a data.frame abundance with columns for 'target_id' (feature, transcript, gene etc), 'length' (feature length), 'est_counts' (estimated feature counts), 'tpm' (transcripts per million); (2) a list, run_info, with metadata about the Salmon run that generated the results, including number of reads processed, mapping percentage, the library type used for the RNA-sequencing, including details about number of reads that did not match the given or inferred library type, details about the Salmon command used to generate the results, and so on.

Examples

```
## Not run:
# If Salmon results are in the directory "output", then call:
readSalmonResultsOneSample("output")

## End(Not run)
```

---

readTxResults *Read transcript quantification data with tximport package*

Description

After generating transcript/feature abundance results using kallisto, Salmon, Sailfish or RSEM for a batch of samples, read these abundance values into an SCESet object.

Usage

```
readTxResults(samples = NULL, files = NULL, log = NULL,
  type = "kallisto", txOut = TRUE, logExprsOffset = 1, verbose = TRUE,
  ...)
```

Arguments

samples	character vector providing a set of sample names to use for the abundance results.
files	character vector providing a set of filenames containing kallisto abundance results to be read in.
log	list (optional), generated by runKallisto. If provided, then samples and files arguments are ignored.
type	character, the type of software used to generate the abundances. Options are "kallisto", "salmon", "sailfish", "rsem". This argument is passed to tximport.
txOut	logical, whether the function should just output transcript-level (default TRUE)
logExprsOffset	numeric scalar, providing the offset used when doing log2-transformations of expression data to avoid trying to take logs of zero. Default offset value is 1.
verbose	logical, should function provide output about progress?
...	optional parameters passed to tximport. See documentation for tximport for options and details.

Details

Note: tximport does not import bootstrap estimates from kallisto, Salmon, or Sailfish. If you want bootstrap estimates use the `readKallistoResults` or `readSalmonResults` functions.

Value

an `SCESet` object containing the abundance, count and feature length data from the supplied samples.

References

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

Examples

```
## Not run:
## this example requires installation of the tximportData package from
## Bioconductor
library(tximportData)
dir <- system.file("extdata", package = "tximportData")
list.files(dir)
samples <- read.table(file.path(dir, "samples.txt"), header = TRUE)
samples
directories <- file.path(dir, "kallisto", samples$run)
names(directories) <- paste0("sample", 1:6)
files <- file.path(directories, "abundance.tsv")
sce_example <- readTxResults(samples = names(directories),
files = files, type = "kallisto")

## for faster reading of results use the read_tsv function from the readr pkg
library(readr)
sce_example <- readTxResults(samples = names(directories),
files = files, type = "kallisto", reader = read_tsv)

## End(Not run)
```

`reducedDimension` *Reduced dimension representation for cells in an SCESet object*

Description

SCESet objects can contain a matrix of reduced dimension coordinates for cells. These functions conveniently access and replace the reduced dimension coordinates with the value supplied, which must be a matrix of the correct size. The function `redDim` is simply shorthand for `reducedDimension`.

Usage

```
reducedDimension(object)

reducedDimension(object) <- value
```

```
redDim(object)

redDim(object) <- value

reducedDimension.SCESet(object)

## S4 method for signature 'SCESet'
reducedDimension(object)

redDim.SCESet(object)

## S4 method for signature 'SCESet'
redDim(object)

## S4 replacement method for signature 'SCESet,matrix'
reducedDimension(object) <- value

## S4 replacement method for signature 'SCESet,matrix'
redDim(object) <- value
```

Arguments

object	a SCESet object.
value	a matrix of class "numeric" containing reduced dimension coordinates for cells.

Value

If accessing the reducedDimension slot, then the matrix of reduced dimension coordinates. If replacing the reducedDimension slot then the new matrix is added to the SCESet object.

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
reducedDimension(example_sceset)
```

---

rename *Rename variables of* pData(object).

Description

Rename variables of pData(object).

Usage

```
rename(object, ...)

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

rename.SCESet(object, ...)
```

Arguments

object	A SCESet object.
...	Additional arguments to be passed to dplyr::rename to act on pData(object).

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
example_sceset <- rename(example_sceset, Cell_Phase = Cell_Cycle)
```

---

runKallisto *Run kallisto on FASTQ files to quantify feature abundance*

Description

Run the abundance quantification tool kallisto on a set of FASTQ files. Requires kallisto (http://pachterlab.github.io/kallisto/) to be installed and a kallisto feature index must have been generated prior to using this function. See the kallisto website for installation and basic usage instructions.

Usage

```
runKallisto(targets_file, transcript_index, single_end = TRUE,
  output_prefix = "output", fragment_length = NULL,
  fragment_standard_deviation = NULL, n_cores = 2,
  n_bootstrap_samples = 0, bootstrap_seed = NULL, correct_bias = TRUE,
  plaintext = FALSE, kallisto_version = "current", verbose = TRUE,
  dry_run = FALSE, kallisto_cmd = "kallisto")
```

Arguments

targets_file	character string giving the path to a tab-delimited text file with either 2 columns (single-end reads) or 3 columns (paired-end reads) that gives the sample names (first column) and FastQ file names (column 2 and if applicable 3). The file is assumed to have column headers, although these are not used.
transcript_index	character string giving the path to the kallisto index to be used for the feature abundance quantification.
single_end	logical, are single-end reads used, or paired-end reads?

output_prefix character string giving the prefix for the output folder that will contain the kallisto results. The default is "output" and the sample name (column 1 of targets_file) is appended (preceded by an underscore).

fragment_length scalar integer or numeric giving the estimated average fragment length. Required argument if single_end is TRUE, optional if FALSE (kallisto default for paired-end data is that the value is estimated from the input data).

fragment_standard_deviation scalar numeric giving the estimated standard deviation of read fragment length. Required argument if single_end is TRUE, optional if FALSE (kallisto default for paired-end data is that the value is estimated from the input data).

n_cores integer giving the number of cores (nodes/threads) to use for the kallisto jobs. The package parallel is used. Default is 2 cores.

n_bootstrap_samples integer giving the number of bootstrap samples that kallisto should use (default is 0). With bootstrap samples, uncertainty in abundance can be quantified.

bootstrap_seed scalar integer or numeric giving the seed to use for the bootstrap sampling (default used by kallisto is 42). Optional argument.

correct_bias logical, should kallisto's option to model and correct abundances for sequence specific bias? Requires kallisto version 0.42.2 or higher.

plaintext logical, if TRUE then bootstrapping results are returned in a plain text file rather than an HDF5 https://www.hdfgroup.org/HDF5/ file.

kallisto_version character string indicating whether or not the version of kallisto to be used is "pre-0.42.2" or "current". This is required because the kallisto developers changed the output file extensions and added features in version 0.42.2.

verbose logical, should timings for the run be printed?

dry_run logical, if TRUE then a list containing the kallisto commands that would be run and the output directories is returned. Can be used to read in results if kallisto is run outside an R session or to produce a script to run outside of an R session.

kallisto_cmd (optional) string giving full command to use to call kallisto, if simply typing "kallisto" at the command line does not give the required version of kallisto or does not work. Default is simply "kalliso". If used, this argument should give the full path to the desired kallisto binary.

Details

A kallisto transcript index can be built from a FASTA file: kallisto index [arguments] FASTA-file. See the kallisto documentation for further details.

Value

A list containing three elements for each sample for which feature abundance has been quantified: (1) kallisto_call, the call used for kallisto, (2) kallisto_log the log generated by kallisto, and (3) output_dir the directory in which the kallisto results can be found.

Examples

```
## Not run:
## If in kallisto's 'test' directory, then try these calls:
```

```
## Generate 'targets.txt' file:
write.table(data.frame(Sample="sample1", File1="reads_1.fastq.gz", File2="reads_1.fastq.gz"),
 file="targets.txt", quote=FALSE, row.names=FALSE, sep="\t")
kallisto_log <- runKallisto("targets.txt", "transcripts.idx", single_end=FALSE,
         output_prefix="output", verbose=TRUE, n_bootstrap_samples=10,
         dry_run = FALSE)

## End(Not run)
```

`runSalmon` *Run Salmon on FASTQ files to quantify feature abundance*

Description

Run the abundance quantification tool `Salmon` on a set of FASTQ files. Requires `Salmon` (https://combine-lab.github.io/salmon/) to be installed and a Salmon transcript index must have been generated prior to using this function. See the Salmon website for installation and basic usage instructions.

Usage

```
runSalmon(targets_file, transcript_index, single_end = FALSE,
  output_prefix = "output", lib_type = "A", n_processes = 2,
  n_thread_per_process = 4, n_bootstrap_samples = 0, seqBias = TRUE,
  gcBias = TRUE, posBias = FALSE, allowOrphans = FALSE,
  advanced_opts = NULL, verbose = TRUE, dry_run = FALSE,
  salmon_cmd = "salmon")
```

Arguments

`targets_file`	character string giving the path to a tab-delimited text file with either 2 columns (single-end reads) or 3 columns (paired-end reads) that gives the sample names (first column) and FastQ file names (column 2 and if applicable 3). The file is assumed to have column headers, although these are not used.
`transcript_index`	character string giving the path to the Salmon index to be used for the feature abundance quantification.
`single_end`	logical, are single-end reads used, or paired-end reads?
`output_prefix`	character string giving the prefix for the output folder that will contain the Salmon results. The default is `"output"` and the sample name (column 1 of `targets_file`) is appended (preceded by an underscore).
`lib_type`	scalar, indicating RNA-seq library type. See Salmon documentation for details. Default is "A", for automatic detection.
`n_processes`	integer giving the number of processes to use for parallel Salmon jobs across samples. The package `parallel` is used. Default is 2 concurrent processes.
`n_thread_per_process`	integer giving the number of threads for Salmon to use per process (to parallelize Salmon for a given sample). Default is 4.
`n_bootstrap_samples`	integer giving the number of bootstrap samples that Salmon should use (default is 0). With bootstrap samples, uncertainty in abundance can be quantified.

seqBias	logical, should Salmon's option be used to model and correct abundances for sequence specific bias? Default is TRUE.
gcBias	logical, should Salmon's option be used to model and correct abundances for GC content bias? Requires Salmon version 0.7.2 or higher. Default is TRUE.
posBias	logical, should Salmon's option be used to model and correct abundances for positional biases? Requires Salmon version 0.7.3 or higher. Default is FALSE.
allowOrphans	logical, Consider orphaned reads as valid hits when performing lightweight-alignment. This option will increase sensitivity (allow more reads to map and more transcripts to be detected), but may decrease specificity as orphaned alignments are more likely to be spurious. For more details see Salmon documentation.
advanced_opts	character scalar supplying list of advanced option arguments to apply to each Salmon call. For details see Salmon documentation or type `salmon quant --help-reads` at the command line.
verbose	logical, should timings for the run be printed?
dry_run	logical, if TRUE then a list containing the Salmon commands that would be run and the output directories is returned. Can be used to read in results if Salmon is run outside an R session or to produce a script to run outside of an R session.
salmon_cmd	(optional) string giving full command to use to call Salmon, if simply typing "salmon" at the command line does not give the required version of Salmon or does not work. Default is simply "salmon". If used, this argument should give the full path to the desired Salmon binary.

Details

A Salmon transcript index can be built from a FASTA file: `salmon index [arguments] FASTA-file`. See the Salmon documentation for further details. This simple wrapper does not give access to all nuances of Salmon usage. For finer-grained usage of Salmon please run it at the command line - results can still be read into R with `readSalmonResults`.

Value

A list containing three elements for each sample for which feature abundance has been quantified: (1) `salmon_call`, the call used for Salmon, (2) `salmon_log` the log generated by Salmon, and (3) `output_dir` the directory in which the Salmon results can be found.

Examples

```
## Not run:
## If in Salmon's 'test' directory, then try these calls:
## Generate 'targets.txt' file:
write.table(data.frame(Sample="sample1", File1="reads_1.fastq.gz", File2="reads_1.fastq.gz"),
 file="targets.txt", quote=FALSE, row.names=FALSE, sep="\t")
Salmon_log <- runSalmon("targets.txt", "transcripts.idx", single_end=FALSE,
         output_prefix="output", verbose=TRUE, n_bootstrap_samples=10,
         dry_run = FALSE)

## End(Not run)
```

scater_gui *scater GUI function*

Description

scater shiny app GUI for workflow for less programmatically inclined users or those who would like a quick and easy way to view multiple plots.

Usage

```
scater_gui(sce_set)
```

Arguments

sce_set	SCESet object after running calculateQCMetrics on it

Value

Opens a browser window with an interactive shiny app and visualize all possible plots included in the scater

Author(s)

Davis McCarthy and Vladimir Kiselev

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data=sc_example_cell_info)
rownames(pd) <- pd$Cell
example_sceset <- newSCESet(countData=sc_example_counts, phenoData=pd)
drop_genes <- apply(exprs(example_sceset), 1, function(x) {var(x) == 0})
example_sceset <- example_sceset[!drop_genes, ]
example_sceset <- calculateQCMetrics(example_sceset, feature_controls = 1:40)
## Not run:
scater_gui(example_sceset)

## End(Not run)
```

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

`logged`: Scalar of class `"logical"`, indicating whether or not the expression data in the 'exprs' slot have been log2-transformed or not.

`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.

`useForExprs`: Character string (one of 'exprs','tpm','counts' or 'fpkm') indicating which expression representation both internal methods and external packages should use. Defaults to 'exprs'.

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.

`SCESet-subset` *Subsetting SCESet Objects*

Description

Subset method for SCESet objects, which subsets both the expression data, phenotype data, feature data and other slots in the object.

Usage

```
## S4 method for signature 'SCESet'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	object from which to extract element(s) or in which to replace element(s).
`i, j, ...`	indices specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or `NULL`. Numeric values are coerced to integer as by `as.integer` (and hence truncated towards zero). Character vectors will be matched to the names of the object (or for matrices/arrays, the dimnames): see `Extract` for further details.

For [-indexing only: `i, j, ...` can be logical vectors, indicating elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. `i, j, ...` can also be negative integers, indicating elements/slices to leave out of the selection. When indexing arrays by `[` a single argument i can be a matrix with as many columns as there are dimensions of `x`; the result is then a vector with elements corresponding to the sets of indices in each row of `i`. An index value of `NULL` is treated as if it were `integer(0)`.

`drop` For matrices and arrays. If `TRUE` the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement. See `drop` for further details.

Value

an SCESet object

See Also

`Extract`

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

Description

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

Usage

```
sc_example_cell_info
```

Format

a data.frame instance, 1 row per cell.

Value

NULL, but makes aavailable a data frame with cell metadata

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

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

Description

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

Usage

```
sc_example_counts
```

Format

a matrix instance, 1 row per gene.

Value

NULL, but makes aavailable a matrix of count data

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

set_exprs<- *Assignment method for the new elements of an SCESet object.*

Description

The assayData slot of an SCESet object holds the expression data matrices. This functions makes it convenient to add new transformations of the expression data to the assayData slot.

Usage

```
set_exprs(object, name) <- value

## S4 replacement method for signature 'SCESet,ANY,matrix'
set_exprs(object,name)<-value

## S4 replacement method for signature 'SCESet,ANY,`NULL`'
set_exprs(object, name) <- value
```

Arguments

object	a SCESet object.
name	character string giving the name of the slot to which the data matrix is to be assigned (can already exist or not).
value	a numeric or integer matrix matching the dimensions of the other elements of the assayData slot of the SCESet object.

Value

NULL, but adds expression data to the SCESet object

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
set_exprs(example_sceset, "scaled_counts") <- t(t(counts(example_sceset)) /
colSums(counts(example_sceset)))
get_exprs(example_sceset, "scaled_counts")[1:6, 1:6]

## get rid of scaled counts
set_exprs(example_sceset, "scaled_counts") <- NULL
```

---

sizeFactors *Accessors size factors of an SCESet object.*

Description

For normalisation, library-specific size factors can be defined. Raw values can be divided by the appropriate size factors to obtain normalised counts, TPM, etc.

Usage

```
## S4 method for signature 'SCESet'
sizeFactors(object,type)

## S4 replacement method for signature 'SCESet,numeric'
sizeFactors(object,type)<-value
## S4 replacement method for signature 'SCESet,NULL'
sizeFactors(object,type)<-value

## S4 method for signature 'SCESet'
sizeFactors(object, type = NULL)

## S4 replacement method for signature 'SCESet,numeric'
sizeFactors(object, type = NULL, ...) <- value
```

```
## S4 replacement method for signature 'SCESet,`NULL`'
sizeFactors(object, type = NULL, ...) <- value
```

Arguments

object	a SCESet object.
type	optional character argument providing the type or name of the size factors to be accessed or assigned.
...	further arguments passed to the function
value	a vector of class "numeric" or NULL

Details

The size factors can alternatively be directly accessed from the SCESet object with object$size_factor_type (where "type" in the preceding is replaced by the actual type name).

Author(s)

Davis McCarthy and Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
sizeFactors(example_sceset)
sizeFactors(example_sceset, NULL) <- 2 ^ rnorm(ncol(example_sceset))

example_sceset <- calculateQCMetrics(example_sceset,
                                     feature_controls = list(set1 = 1:40))
sizeFactors(example_sceset, "set1") <- 2 ^ rnorm(ncol(example_sceset))
sizeFactors(example_sceset)
```

---

stand_exprs *Accessors for the 'stand_exprs' (standardised expression) element of an SCESet object.*

Description

The stand_exprs element of the arrayData slot in an SCESet object holds a matrix containing standardised (mean-centred, variance standardised, by feature) expression values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
stand_exprs(object)

stand_exprs(object) <- value

## S4 method for signature 'SCESet'
stand_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
stand_exprs(object)<-value

## S4 method for signature 'SCESet'
stand_exprs(object)

## S4 replacement method for signature 'SCESet,matrix'
stand_exprs(object) <- value
```

Arguments

object	a SCESet object.
value	an integer matrix

Details

The default for normalised expression values is mean-centred and variance-standardised expression data from the exprs slot of the SCESet object. The function normaliseExprs (or normalizeExprs) provides more options and functionality for normalising expression data.

Value

a matrix of standardised expressiond data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
stand_exprs(example_sceset)
```

---

summariseExprsAcrossFeatures

Summarise expression values across feature

Description

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

Usage

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

Arguments

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

Details

Only transcripts-per-million (TPM) and fragments per kilobase of exon per million reads mapped (FPKM) expression values should be aggregated across features. Since counts are not scaled by the length of the feature, expression in counts units are not comparable within a sample without adjusting for feature length. Thus, we cannot sum counts over a set of features to get the expression of that set (for example, we cannot sum counts over transcripts to get accurate expression estimates for a gene). See the following link for a discussion of RNA-seq expression units by Harold Pimentel: https://haroldpimentel.wordpress.com/2014/05/08/what-the-fpkm-a-review-rna-seq-expression-units For more details about the effects of summarising transcript expression values at the gene level see Sonesen et al, 2016 (https://f1000research.com/articles/4-1521/v2).

Value

an SCESet object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
fd <- new("AnnotatedDataFrame", data =
data.frame(gene_id = featureNames(example_sceset),
feature_id = paste("feature", rep(1:500, each = 4), sep = "_")))
rownames(fd) <- featureNames(example_sceset)
fData(example_sceset) <- fd
effective_length <- rep(c(1000, 2000), times = 1000)
tpm(example_sceset) <- calculateTPM(example_sceset, effective_length, calc_from = "counts")

example_sceset_summarised <-
```

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

`toCellDataSet` *Convert an* `SCESet` *to a* `CellDataSet`

Description

Convert an `SCESet` to a `CellDataSet`

Usage

```
toCellDataSet(sce, exprs_values = "exprs")
```

Arguments

`sce`	An `SCESet` object
`exprs_values`	What should `exprs(cds)` be mapped from in the `SCESet`? Should be one of "exprs", "tpm", "fpkm", "counts"

Value

An object of class `SCESet`

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
if ( requireNamespace("monocle") ) {
    toCellDataSet(example_sceset)
}
```

`tpm` *Accessors for the 'tpm' (transcripts per million) element of an SCESet object.*

Description

The `tpm` element of the arrayData slot in an SCESet object holds a matrix containing transcripts-per-million values. It has the same dimensions as the 'exprs' and 'counts' elements, which hold the transformed expression data and count data, respectively.

Usage

```
tpm(object)

tpm(object) <- value

## S4 method for signature 'SCESet'
tpm(object)

## S4 replacement method for signature 'SCESet,matrix'
tpm(object)<-value

## S4 method for signature 'SCESet'
tpm(object)

## S4 replacement method for signature 'SCESet,matrix'
tpm(object) <- value
```

Arguments

object	a SCESet object.
value	a matrix of class "numeric"

Value

a matrix of transcripts-per-million data

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sceset <- newSCESet(countData = sc_example_counts)
tpm(example_sceset)
```

---

updateSCESet *Update an SCESet object to the current version*

Description

It can be necessary to update an SCESet produced with an older version of the package to be compatible with the current version of the package.

Usage

```
updateSCESet(object)
```

Arguments

object	an SCESet object to be updated

Value

an updated SCESet object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)
updateSCESet(example_sceset)
```

---

writeSCESet *Write an SCESet object to an HDF5 file*

Description

Write an SCESet object to an HDF5 file

Usage

```
writeSCESet(object, file_path, type = "HDF5", overwrite_existing = FALSE)
```

Arguments

object	SCESet object to be written to file
file_path	path to written file containing data from SCESet object
type	character string indicating type of output file. Default is "HDF5".
overwrite_existing	logical, if a file of the same name already exists should it be overwritten? Default is FALSE.

Details

Currently writing to HDF5 files is supported. The **rhdf5** package is used to write data to file and can be used to read data from HDF5 files into R. For further details about the HDF5 data format see https://support.hdfgroup.org/HDF5/.

Value

Return is NULL, having written the SCESet object to file.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
pd <- new("AnnotatedDataFrame", data = sc_example_cell_info)
example_sceset <- newSCESet(countData = sc_example_counts, phenoData = pd)

## Not run:
writeSCESet(example_sceset, "test.h5")
file.remove("test.h5")

## End(Not run)
```

Index