Package 'scater'

April 12, 2018

Type Package

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

Version 1.6.3

Date 2018-02-13

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 (>= 3.4), Biobase, ggplot2, SingleCellExperiment, SummarizedExperiment

Imports biomaRt, BiocGenerics, data.table, dplyr, edgeR, ggbeeswarm, grid, limma, Matrix, matrixStats, methods, parallel, plyr, reshape2, rhdf5, rjson, S4Vectors, shiny, shinydashboard, stats, tximport, utils, viridis, Rcpp

Suggests BiocStyle, beachmat, 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, Coverage

LinkingTo Rhdf5lib, Rcpp, beachmat

SystemRequirements C++11

RoxygenNote 6.0.1

NeedsCompilation yes

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

BugReports https://support.bioconductor.org/

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

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.

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

Description

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

Usage

```
norm_exprs(object)

norm_exprs(object) <- value

stand_exprs(object)

stand_exprs(object) <- value

fpkm(object)

fpkm(object) <- value
```

Arguments

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

Value

a matrix of normalised expression data

a matrix of standardised expressiond data

a matrix of FPKM values

A matrix of numeric, integer or logical values.

Author(s)

Davis McCarthy

Examples

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

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

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

counts(example_sce) <- sc_example_counts
norm_exprs(example_sce) <- log2(calculateCPM(example_sce, use.size.factors = FALSE) + 1)

stand_exprs(example_sce) <- log2(calculateCPM(example_sce, use.size.factors = FALSE) + 1)

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

cpm(example_sce) <- calculateCPM(example_sce, use.size.factors = FALSE)

fpkm(example_sce)
```

---

areSizeFactorsCentred *Check if the size factors are centred at unity*

Description

Checks if each set of size factors is centred at unity, such that abundances can be reasonably compared between features normalized with different sets of size factors.

Usage

```
areSizeFactorsCentred(object, centre = 1, tol = 1e-06)
```

Arguments

object	an SingleCellExperiment object containing multiple sets of size factors.
centre	a numeric scalar, the value around which all sets of size factors should be centred.
tol	a numeric scalar, the tolerance for testing equality of the mean of each size factor set to centre.

Value

a SingleCellExperiment object with centred size factors

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
keep_gene <- rowSums(counts(example_sce)) > 0
example_sce <- example_sce[keep_gene,]

sizeFactors(example_sce) <- runif(ncol(example_sce))
areSizeFactorsCentred(example_sce)
example_sce <- normalize(example_sce, centre = TRUE)
areSizeFactorsCentred(example_sce)
```

---

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

Description

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

Usage

```
arrange(object, ...)

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

Arguments

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

Value

An SingleCellExperiment object.

Examples

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

---

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

Description

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

Usage

```
bootstraps(object)

bootstraps(object) <- value

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

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

Arguments

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

Value

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

Author(s)

Davis McCarthy

Examples

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

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

Description

Calculate average counts per feature, adjusting them as appropriate to take into account for size factors for normalization or library sizes (total counts).

Usage

```
calcAverage(object, size.factors = NULL)
```

Arguments

`object`	a `SingleCellExperiment` object or a matrix of counts
`size.factors`	numeric(), vector of size factors to use to scale library size in computation of counts-per-million. Extracted from the object if it is a `SingleCellExperiment` object; if object is a matrix, then if non-NULL, the provided size factors are used. Default is `NULL`, in which case size factors are all set to 1 (i.e. library size adjustment only).

Value

Vector of average count values with same length as number of features.

Examples

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

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

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

Description

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

Usage

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

Arguments

`object`	a `SingleCellExperiment` 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_values`	character scalar indicating whether the count data (`"counts"`), the log-transformed count data (`"logcounts"`), transcript-per-million (`"tpm"`), counts-per-million (`"cpm"`) or FPKM (`"fpkm"`) should be used to define if an observation is expressed or not. Defaults to the first available value of those options in the order shown.

Value

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

Examples

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

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

Description

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

Usage

```
calculateCPM(object, use.size.factors = TRUE)
```

Arguments

object	an SingleCellExperiment object
use.size.factors	a logical scalar specifying whether the size factors should be used to construct effective library sizes, or if the library size should be directly defined as the sum of counts for each cell.

Value

Matrix of CPM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
cpm(example_sce) <- calculateCPM(example_sce, use.size.factors = FALSE)
```

---

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

Description

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

Usage

```
calculateFPKM(object, effective_length, use.size.factors = TRUE)
```

Arguments

object	an SingleCellExperiment object
effective_length	vector of class "numeric" providing the effective length for each feature in the SCESet object
use.size.factors	a logical scalar, see calculateCPM

Value

Matrix of FPKM values.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
effective_length <- rep(1000, 2000)
fpkm(example_sce) <- calculateFPKM(example_sce, effective_length,
use.size.factors = FALSE)
```

`calculateQCMetrics` *Calculate QC metrics*

Description

Calculate QC metrics

Usage

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

Arguments

`object`	an SingleCellExperiment object containing expression values and experimental information. Must have been appropriately prepared.
`exprs_values`	character(1), indicating slot of the `assays` of the `object` should be used to define expression? Valid options are "counts" [default; recommended], "tpm", "fpkm" and "logcounts", or anything else in the object added manually by the user.
`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 `SingleCellExperiment` 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 `SingleCellExperiment` 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_counts: 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 `SingleCellExperiment` 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 `SingleCellExperiment` object), with the appropriate term replacing "counts" in the name.

Value

an SingleCellExperiment object

Examples

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

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

## with a named set of feature controls defined
```

```
example_sce <- calculateQCMetrics(example_sce,
                                  feature_controls = list(ERCC = 1:40))
```

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

Description

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

Usage

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

Arguments

`object`	a `SingleCellExperiment` object
`effective_length`	vector of class `"numeric"` providing the effective length for each feature in the `SingleCellExperiment` object
`calc_from`	character string indicating whether to compute TPM from `"counts"`, `"normcounts"` or `"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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
tpm(example_sce) <- calculateTPM(example_sce, effective_length = 5e04,
    calc_from = "counts")

## calculate from FPKM
fpkm(example_sce) <- calculateFPKM(example_sce, effective_length = 5e04,
use.size.factors = FALSE)
tpm(example_sce) <- calculateTPM(example_sce, effective_length = 5e04,
                                 calc_from = "fpkm")
```

deprecated *Retrieve a representation of gene expression*

Description

Deprecated from scater version 1.3.29.

Deprecated from scater version 1.5.2.

Deprecated from scater version 1.5.2.

Usage

```
getExprs(object)

toCellDataSet(sce, exprs_values = "exprs")

fromCellDataSet(cds, exprs_values = "tpm", logged = FALSE,
  logExprsOffset = 1)
```

Arguments

object	An object of type SCESet
sce	An SCESet object
exprs_values	What should exprs(cds) be mapped from in the SCESet? Should be one of "exprs", "tpm", "fpkm", "counts"
cds	A CellDataSet from the monocle package
logged	logical, if exprs_values="exprs", are the expression values already on the log2 scale, or not?
logExprsOffset	numeric, value to add prior to log-transformation.

Value

A matrix representation of expression values.

An object of class CellDataSet

An object of class SCESet

Examples

```
## Not run: "Deprecated"
## Not run: "Deprecated"
```

downsampleCounts *Downsample a count matrix*

Description

Downsample a count matrix to a desired proportion.

Usage

```
downsampleCounts(x, prop)
```

Arguments

x	matrix of counts
prop	numeric scalar or vector of length ncol(x) in [0, 1] indicating the downsampling proportion

Details

Given multiple 10X batches of very different sequencing depths, it can be beneficial to downsample the deepest batches to match the coverage of the shallowest batches. This avoids differences in technical noise that can drive clustering by batch.

Downsampling without replacement is performed on the counts in each cell to generate the output matrix. Each count in the returned matrix is guaranteed to be smaller than the original value in x. This provides an alternative to downsampling in the CellRanger aggr function.

Value

an integer matrix of downsampled counts

Examples

```
sce10x <- read10xResults(system.file("extdata", package="scater"))
downsampled <- downsampleCounts(counts(sce10x), prop = 0.5)
```

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

Description

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

Usage

```
filter(object, ...)

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

Arguments

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

Value

An SingleCellExperiment object.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts),
colData = sc_example_cell_info)
example_sce_treat1 <- filter(example_sce, Treatment == "treat1")
```

---

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 = "logcounts", 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 colData(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", "tpm", "fpkm" and "logcounts" (default), 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 rownames(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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]
example_sce <- calculateQCMetrics(example_sce)
findImportantPCs(example_sce, variable="total_features")
```

---

getBMFeatureAnnos *Get feature annotation information from Biomart*

Description

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

Usage

```
getBMFeatureAnnos(object, filters = "ensembl_transcript_id",
  attributes = c("ensembl_transcript_id", "ensembl_gene_id", feature_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 SingleCellExperiment 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 rowData(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 rowData(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

a SingleCellExperiment object

Examples

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

## End(Not run)
```

---

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, subset = NULL, batch = NULL, min.diff = NA)
```

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?
subset	logical or integer vector, which subset of values should be used to calculate the median/MAD? If NULL, all values are used. Missing values will trigger a warning and will be automatically ignored.
batch	factor of length equal to metric, specifying the batch to which each observation belongs. A median/MAD is calculated for each batch, and outliers are then identified within each batch.
min.diff	numeric scalar indicating the minimum difference from the median to consider as an outlier. The outlier threshold is defined from the larger of nmads MADs and min.diff, to avoid calling many outliers when the MAD is very small. If NA, it is ignored.

Value

a logical vector of the same length as the metric argument

Examples

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

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

---

kallisto-wrapper *kallisto wrapper functions*

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.

Read kallisto results for a single sample into a list

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

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")

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

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

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.
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_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.
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.

Details

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

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

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.

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.

a SingleCellExperiment object

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

`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* `colData(object)`.

Description

Adds ne

Usage

```
mutate(object, ...)

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

Arguments

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

Value

An SingleCellExperiment object.

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts),
colData = sc_example_cell_info)
example_sce <- mutate(example_sce, is_quiescent = Cell_Cycle == "G0")
```

newSCESet *Create a new SCESet object*

Description

Deprecated from scater version 1.3.29; the package now uses the SingleCellExperiment class. To convert an SCESet object to SingleCellExperiment see the toSingleCellExperiment function. This function is retained for backwards compatibility.

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 = NULL,
  logExprsOffset = NULL)
```

Arguments

exprsData	expression data matrix for an experiment (features x cells)
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.

Details

This function now returns a SingleCellExperiment object, whereas earlier versions produced an SCESet object. The scater package now uses SingleCellExperiment as its data structure instead of SCESet.

Value

a SingleCellExperiment object

Examples

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

## End(Not run)
```

---

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, lowerDetectionLimit = 0, exprs_values = "counts",
  byrow = FALSE, subset_row = NULL, subset_col = NULL)
```

Arguments

object	a SingleCellExperiment object
lowerDetectionLimit	numeric scalar providing the value above which observations are deemed to be expressed. Defaults to object@lowerDetectionLimit.
exprs_values	character scalar indicating whether the count data ("counts"), the log-transformed count data ("logcounts"), transcript-per-million ("tpm"), counts-per-million ("cpm") or FPKM ("fpkm") should be used to define if an observation is expressed or not. Defaults to the first available value of those options in the order shown. However, if is_exprs(object) is present, it will be used directly; exprs_values and lowerDetectionLimit are ignored.
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.
subset_row	logical, integeror character vector indicating which rows (i.e. features/genes) to use when calculating the number of expressed features in each cell, when byrow=FALSE.
subset_col	logical, integer or character vector indicating which columns (i.e., cells) to use to calculate the number of cells expressing each gene when byrow=TRUE.

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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
nexprs(example_sce)[1:10]
nexprs(example_sce, byrow = TRUE)[1:10]
```

---

normaliseExprs *Normalise expression expression levels for an SingleCellExperiment object*

Description

Compute normalised expression values from an SingleCellExperiment 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, return_log = TRUE,
  ...)

normalizeExprs(...)
```

Arguments

object	an SingleCellExperiment object.
method	character string specified the method of calculating 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 calculating normalisation factors. 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 SingleCellExperiment object should be used for the calculations. Valid options are 'counts', 'tpm', 'cpm', 'fpkm' and 'exprs'. Defaults to the first available value of these options in in order shown.

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.

return_log
: logical(1), should normalized values be returned on the log scale? Default is TRUE. If TRUE and return_norm_as_exprs is TRUE then normalised output is stored as "logcounts" in the returned object; if TRUE and return_norm_as_exprs is FALSE then normalised output is stored as "norm_exprs"; if FALSE output is stored as "normcounts"

...
: 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 SingleCellExperiment object. The 'raw' values used can be the values in the 'counts' (default), or another specified assay slot of the SingleCellExperiment. Normalised expression values are computed through normalizeSCE and are on the log2-scale by default (if return_log is TRUE), with an offset defined by the metadata(object)$log.exprs.offset value in the SingleCellExperiment object. These are added to the 'norm_exprs' slot of the returned object. If 'exprs_values' argument is 'counts' and return_log is FALSE a 'normcounts' slot is added, containing normalised counts-per-million values.

If the raw values are counts, this function will compute size factors using methods in calcNormFactors. Library sizes are multiplied by size factors to obtain an "effective library size" before calculation of the aforementioned normalized expression values. If feature_set is specified, only the specified features will be used to calculate the size factors.

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 values in 'norm_exprs' are 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 a SingleCellExperiment object.

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

Value

an SingleCellExperiment object

Author(s)

Davis McCarthy

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
```

```
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
keep_gene <- rowSums(counts(example_sce)) > 0
example_sce <- example_sce[keep_gene,]

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

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

Description

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

Usage

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

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

normalise(...)
```

Arguments

`object`	a `SingleCellExperiment` object.
`exprs_values`	character string indicating which slot of the assayData from the `SingleCellExperiment` object should be used to compute log-transformed expression values. Valid options are `'counts'`, `'tpm'`, `'cpm'` and `'fpkm'`. Defaults to the first available value of the options in the order shown.
`return_log`	logical(1), should normalized values be returned on the log scale? Default is `TRUE`. If `TRUE`, output is stored as `"logcounts"` in the returned object; if `FALSE` output is stored as `"normcounts"`
`log_exprs_offset`	scalar numeric value giving the offset to add when taking log2 of normalised values to return as expression values. If `NULL`, value is taken from `metadata(object)$log.exprs.offs` if defined, otherwise 1.
`centre_size_factors`	logical, should size factors centred at unity be stored in the returned object if `exprs_values="counts"`? Defaults to TRUE. Regardless, centred size factors will always be used to calculate `exprs` from count data. This argument is ignored for other `exprs_values`, where no size factors are used/modified.

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 SingleCellExperiment object

Warning about centred size factors

Centring the size factors ensures that the computed exprs can be interpreted as being on the same scale as log-counts. This does not affect relative comparisons between cells in the same object, as all size factors are scaled by the same amount. However, if two different SingleCellExperiment objects are run separately through normalize, the size factors in each object will be rescaled differently. This means that the size factors and exprs will *not* be comparable between objects.

This lack of comparability is not always obvious. For example, if we subsetted an existing SingleCellExperiment, and ran normalize separately on each subset, the resulting exprs in each subsetted object would *not* be comparable to each other. This is despite the fact that all cells were originally derived from a single SingleCellExperiment object.

In general, it is advisable to only compare size factors and exprs between cells in one SingleCellExperiment object. If objects are to be combined, new size factors should be computed using all cells in the combined object, followed by running normalize.

Author(s)

Davis McCarthy and Aaron Lun

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
keep_gene <- rowSums(counts(example_sce)) > 0
example_sce <- example_sce[keep_gene,]

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

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

`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 = "logcounts", nvars_to_plot = 10, min_marginal_r2 = 0,
  variables = NULL, return_object = FALSE, theme_size = 10, ...)
```

Arguments

`object`	an SingleCellExperiment 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 "logcounts" (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 `colData(object)` are considered and the `nvars_to_plot` variables with the highest median marginal R-squared are plotted.
`return_object`	logical, should an `SingleCellExperiment` 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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]
example_sce <- calculateQCMetrics(example_sce)
vars <- names(colData(example_sce))[c(2:3, 5:14)]
plotExplanatoryVariables(example_sce, 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, features, x = NULL, exprs_values = "logcounts",
  log2_values = FALSE, colour_by = NULL, shape_by = NULL,
  size_by = NULL, ncol = 2, xlab = NULL, show_median = FALSE,
  show_violin = TRUE, theme_size = 10, ...)

plotExpressionDefault(object, aesth, ncol = 2, xlab = NULL, ylab = NULL,
  show_median = FALSE, show_violin = TRUE, show_smooth = FALSE,
  theme_size = 10, alpha = 0.6, size = NULL, scales = "fixed",
  one_facet = FALSE, se = TRUE, jitter = "swarm")
```

Arguments

`object`	an `SingleCellExperiment` 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.
`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" (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), "logcounts" (log-transformed count data; 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.
log2_values	should the expression values be transformed to the log2-scale for plotting (with an offset of 1 to avoid logging zeroes)?
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. Alternatively, a data frame with one column, containing a value for each cell that will be mapped to a colour.
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. Alternatively, a data frame with one column containing values to map to shapes.
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. Alternatively, a data frame with one column containing values to map to sizes.
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
theme_size	numeric scalar giving default font size for plotting theme (default is 10)
...	optional arguments (from those listed above) passed to plotExpressionDefault
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.
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.
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.
one_facet	logical, should expression values for features be plotted in one facet instead of mutiple facets, one per feature? Default if x = NULL.
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.

Details

Plot expression values (default log2(counts-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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- calculateQCMetrics(example_sce)
sizeFactors(example_sce) <- colSums(counts(example_sce))
example_sce <- normalize(example_sce)

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

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

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

plotExpression(example_sce, "Gene_0001", x = "Mutation_Status")
plotExpression(example_sce, c("Gene_0001", "Gene_0004"), x="Mutation_Status")
plotExpression(example_sce, "Gene_0001", x = "Gene_0002")
plotExpression(example_sce, c("Gene_0001", "Gene_0004"), x="Gene_0002")
## plot expression against expression values for Gene_0004
plotExpression(example_sce, 1:4, "Gene_0004")
plotExpression(example_sce, 1:4, "Gene_0004", show_smooth = TRUE)
plotExpression(example_sce, 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 SingleCellExperiment object.
feature_set	character, numeric or logical vector indicating a set of features to plot. If character, entries must all be in rownames(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 rownames(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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts),
colData = sc_example_cell_info)
example_sce <- normalize(example_sce)

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

example_sce <- calculateQCMetrics(
example_sce, feature_controls = list(controls1 = 1:20,
```

```
                                    controls2 = 500:1000),
                                    cell_controls = list(set_1 = 1:5,
                                    set_2 = 31:40))
plotExprsFreqVsMean(example_sce)
```

---

plotExprsVsTxLength *Plot expression against transcript length*

Description

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

Usage

```
plotExprsVsTxLength(object, tx_length = "median_feat_eff_len",
  exprs_values = "logcounts", 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 SingleCellExperiment object
tx_length	transcript lengths to plot on the x-axis. Can be one of: (1) the name of a column of rowData(object) containing the transcript length values, or (2) the name of an element of assays(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" (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), "logcounts" (log-transformed count data; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other slots that have been added to the "assays" slot by the user.
colour_by	optional character string supplying name of a column of rowData(object) which will be used as a variable by which to colour expression values on the plot. Alternatively, a data frame with one column, containing a value for each feature to map to a colour.
shape_by	optional character string supplying name of a column of rowData(object) which will be used as a variable to define the shape of points for expression values on the plot. Alternatively, a data frame with one column containing values to map to shapes.
size_by	optional character string supplying name of a column of rowData(object) which will be used as a variable to define the size of points for expression values on the plot. Alternatively, a data frame with one column containing values to map to sizes.

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")
rd <- DataFrame(gene_id = rownames(sc_example_counts),
        feature_id = paste("feature", rep(1:500, each = 4), sep = "_"),
     median_tx_length = rnorm(2000, mean = 5000, sd = 500))
rownames(rd) <- rownames(sc_example_counts)
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts),
colData = sc_example_cell_info, rowData = rd)
example_sce <- normalize(example_sce)

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

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

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

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

plotFeatureData *Plot feature (gene) data from a SingleCellExperiment object*

Description

plotFeatureData and plotRowData are synonymous.

Usage

```
plotFeatureData(object, aesth = aes_string(x = "n_cells_counts", y =
  "log10_total_counts"), ...)

plotRowData(...)
```

Arguments

object	an SingleCellExperiment 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 SingleCellExperiment object prior).
...	arguments passed to plotMetadata, e.g. theme_size, size, alpha, shape.

Details

Plot feature (gene) data from an SingleCellExperiment 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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- calculateQCMetrics(example_sce)
plotFeatureData(example_sce, aesth = aes(x = n_cells_counts, y = log10_total_counts))

plotRowData(example_sce, aesth = aes(x = n_cells_counts, y = log10_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 colData(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 "logcounts".
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 rownames(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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- calculateQCMetrics(example_sce,
feature_controls = list(set1 = 1:500))
plotHighestExprs(example_sce, col_by_variable="total_features")
```

```
plotHighestExprs(example_sce, col_by_variable="Mutation_Status")
plotQC(example_sce, type = "highest-express")
```

---

plotMDS *Produce a multidimensional scaling plot for a SingleCellExperiment object*

Description

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

Usage

```
plotMDS(object, ncomponents = 2, colour_by = NULL, shape_by = NULL,
  size_by = NULL, return_SCE = FALSE, rerun = FALSE, draw_plot = TRUE,
  exprs_values = "logcounts", theme_size = 10, legend = "auto", ...)
```

Arguments

object	an SingleCellExperiment object
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. Alternatively, a data frame with one column containing values to map to colours for all cells.
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_SCE	logical, should the function return an SingleCellExperiment object with principal component values for cells in the reducedDims slot. Default is FALSE, in which case a ggplot object is returned.
rerun	logical, should PCA be recomputed even if object contains a "PCA" element in the reducedDims slot?
draw_plot	logical, should the plot be drawn on the current graphics device? Only used if return_SCE is TRUE, otherwise the plot is always produced.
exprs_values	a string specifying the expression values to use for colouring the points, if colour_by or size_by are set as feature names.
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).
...	arguments passed to S4 plotMDS method

Details

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

Value

If return_SCE is TRUE, then the function returns an SingleCellExperiment object, otherwise it returns a ggplot object.

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

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

## define cell-cell distances differently
plotMDS(example_sce, colour_by = "Cell_Cycle",
shape_by = "Treatment", size_by = "Mutation_Status", method = "canberra")
```

---

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

---

plotPhenoData *Plot cell phenotype data from an SingleCellExperiment object*

Description

plotPhenoData, plotColData and plotCellData are synonymous.

Usage

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

plotColData(...)

plotCellData(...)
```

Arguments

object	an SingleCellExperiment 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).
...	arguments passed to plotPhenoData (if plotColData or plotCellData) or to plotMetadata, e.g.theme_size, size, alpha, shape.

Details

Plot phenotype data from a SingleCellExperiment 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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- calculateQCMetrics(example_sce)
plotPhenoData(example_sce, aesth = aes_string(x = "log10(total_counts)",
y = "total_features", colour = "Mutation_Status"))

plotColData(example_sce, aesth = aes_string(x = "log10(total_counts)",
y = "total_features", colour = "Mutation_Status"))

plotCellData(example_sce, 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, exprs_values = "logcounts",
  theme_size = 24, legend = "auto")
```

Arguments

object	an SingleCellExperiment 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 SingleCellExperiment 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. Alternatively, a data frame with one column containing values to map to colours for all cells.
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)
exprs_values	a string specifying the expression values to use for colouring the points, if colour_by is set as a feature name.
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.

Value

A ggplot object.

Examples

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

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

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

## Must have exprs slot defined in object
plotPlatePosition(example_sce, colour_by = "Gene_0004")
```

plotQC *Produce QC diagnostic plots*

Description

Produce QC diagnostic plots

Usage

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

Arguments

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

drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]
example_sce <- calculateQCMetrics(example_sce)
plotQC(example_sce, type="high", col_by_variable="Mutation_Status")
plotQC(example_sce, type="find", variable="total_features")
vars <- names(colData(example_sce))[c(2:3, 5:14)]
plotQC(example_sce, type="expl", variables=vars)
```

plotReducedDim *Plot reduced dimension representation of cells*

Description

Plot reduced dimension representation of cells

Usage

```
plotReducedDimDefault(df_to_plot, ncomponents = 2, percentVar = NULL,
  colour_by = NULL, shape_by = NULL, size_by = NULL, theme_size = 10,
  legend = "auto")

plotReducedDim(object, use_dimred, ncomponents = 2, colour_by = NULL,
  shape_by = NULL, size_by = NULL, exprs_values = "logcounts",
  percentVar = NULL, ...)
```

Arguments

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.
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.
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. Alternatively, a data frame with one column containing values to map to colours for all cells.
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.
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).
object	an SingleCellExperiment object with a non-NULL reducedDimension slot.
use_dimred	character, name of reduced dimension representation of cells stored in SingleCellExperiment object to plot (e.g. "PCA", "TSNE", etc).
exprs_values	a string specifying the expression values to use for colouring the points, if colour_by or size_by are set as feature names.
...	optional arguments (from those listed above) passed to plotReducedDimDefault

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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

reducedDim(example_sce, "PCA") <- prcomp(t(exprs(example_sce)), scale. = TRUE)$x
plotReducedDim(example_sce, "PCA")
plotReducedDim(example_sce, "PCA", colour_by="Cell_Cycle")
plotReducedDim(example_sce, "PCA", colour_by="Cell_Cycle", shape_by="Treatment")
plotReducedDim(example_sce, "PCA", colour_by="Cell_Cycle", size_by="Treatment")
plotReducedDim(example_sce, "PCA", ncomponents=5)
plotReducedDim(example_sce, "PCA", ncomponents=5, colour_by="Cell_Cycle", shape_by="Treatment")
plotReducedDim(example_sce, "PCA", colour_by="Gene_0001")
```

---

plotRLE *Plot a relative log expression (RLE) plot*

Description

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

Usage

```
plotRLE(object, exprs_mats = list(logcounts = "logcounts"),
  exprs_logged = c(TRUE), colour_by = NULL, style = "minimal",
  legend = "auto", order_by_colour = TRUE, ncol = 1, ...)
```

Arguments

`object`	an `SingleCellExperiment` object
`exprs_mats`	named list of expression matrices. Entries can either be a character string, in which case the corresponding expression matrix will be extracted from the SingleCellExperiment `object`, or a matrix of expression values.
`exprs_logged`	logical vector of same length as `exprs_mats` indicating whether the corresponding entry in `exprs_mats` contains logged expression values (`TRUE`) or not (`FALSE`).

colour_by	character string defining the column of colData(object) to be used as a factor by which to colour the points in the plot. Alternatively, a data frame with one column, containing values to map to colours for all cells.
style	character(1), either "minimal" (default) or "full", defining the boxplot style to use. "minimal" uses Tufte-style boxplots and is fast for large numbers of cells. "full" uses the usual ggplot2 and is more detailed and flexible, but can take a long time to plot for large datasets.
legend	character, specifying how the legend(s) be shown? Default is "auto", which hides legends that have only one level and shows others. Alternative is "none" (hide all legends).
order_by_colour	logical, should cells be ordered (grouped) by the colour_by variable? Default is TRUE. Useful for visualising differences between batches or experimental conditions.
ncol	integer, number of columns for the facetting of the plot. Default is 1.
...	further arguments passed to geom_boxplot.

Details

Unwanted variation can be highly problematic and so its detection is often crucial. Relative log expression (RLE) plots are a powerful tool for visualising such variation in high dimensional data. RLE plots are particularly useful for assessing whether a procedure aimed at removing unwanted variation, i.e. a normalisation procedure, has been successful. These plots, while originally devised for gene expression data from microarrays, can also be used to reveal unwanted variation in single-cell expression data, where such variation can be problematic.

If style is "full", as usual with boxplots, the box shows the inter-quartile range and whiskers extend no more than 1.5 * IQR from the hinge (the 25th or 75th percentile). Data beyond the whiskers are called outliers and are plotted individually. The median (50th percentile) is shown with a white bar.

If style is "minimal", then median is shown with a circle, the IQR in a grey line, and "whiskers" (as defined above) for the plots are shown with coloured lines. No outliers are shown for this plot style.

Value

a ggplot plot object

Author(s)

Davis McCarthy

References

Gandolfo LC, Speed TP. RLE Plots: Visualising Unwanted Variation in High Dimensional Data. arXiv [stat.ME]. 2017. Available: http://arxiv.org/abs/1704.03590

Examples

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

```
drop_genes <- apply(logcounts(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

plotRLE(example_sce, list(logcounts= "logcounts", counts = "counts"), c(TRUE, FALSE),
       colour_by = "Mutation_Status", style = "minimal")

plotRLE(example_sce, list(logcounts = "logcounts", counts = "counts"), c(TRUE, FALSE),
       colour_by = "Mutation_Status", style = "full",
       outlier.alpha = 0.1, outlier.shape = 3, outlier.size = 0)
```

`plotScater` *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 `SingleCellExperiment` dataset.

Usage

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

Arguments

`x`	an `SingleCellExperiment` object
`block1`	character string defining the column of `colData(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 `colData(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 `colData(object)` to be used as a factor by which to colour the points in the plot. Alternatively, a data frame with one column containing a value for each cell, which will be mapped to a corresponding colour.
`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"` (transcripts per million), `"counts"` (raw counts) [default], `"cpm"` (counts per million), or `"fpkm"` (FPKM values).
`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
`...`	arguments passed to `plotSCE`

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

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

cpm(example_sce) <- calculateCPM(example_sce, use.size.factors = FALSE)
plotScater(example_sce, exprs_values = "cpm", block1 = "Treatment",
block2 = "Mutation_Status", colour_by = "Cell_Cycle")
# Error is thrown if chosen expression values are not available
```

`read10xResults` *Load in data from 10x experiment*

Description

Creates a full or sparse matrix from a sparse data matrix provided by 10X genomics.

Usage

```
read10xResults(data_dir, min_total_cell_counts = NULL,
  min_mean_gene_counts = NULL)

read10XResults(...)
```

Arguments

`data_dir`	Directory containing the matrix.mtx, genes.tsv, and barcodes.tsv files provided by 10x. A vector or named vector can be given in order to load several data directories. If a named vector is given, the cell barcode names will be prefixed with the name.

min_total_cell_counts
: integer(1) threshold such that cells (barcodes) with total counts below the threshold are filtered out

min_mean_gene_counts
: numeric(1) threshold such that genes with mean counts below the threshold are filtered out.

...
: passed arguments

Details

This function was developed from the `Read10X` function from the **Seurat** package.

Value

Returns an SingleCellExperiment object with counts data stored as a sparse matrix with rows and columns labeled.

Examples

```
sce10x <- read10xResults(system.file("extdata", package="scater"))
```

---

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)
```

---

rename *Rename variables of* `colData(object)`.

Description

Rename variables of `colData(object)`.

Usage

```
rename(object, ...)

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

Arguments

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

Value

An SingleCellExperiment object.

Examples

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

---

runDiffusionMap *Plot a diffusion map for a SingleCellExperiment object*

Description

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

Usage

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

plotDiffusionMap(object, colour_by = NULL, shape_by = NULL,
  size_by = NULL, return_SCE = FALSE, draw_plot = TRUE, theme_size = 10,
  legend = "auto", rerun = FALSE, ncomponents = 2, ...)
```

Arguments

object	an SingleCellExperiment object
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.
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).

exprs_values	character string indicating which values should be used as the expression values for this plot. Valid arguments are "tpm" (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), "logcounts" (log-transformed count data; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other named element of the assayData slot of the SingleCellExperiment object that can be accessed with the assay function.
scale_features	logical, should the expression values be standardised so that each feature has unit variance? Default is TRUE.
use_dimred	character(1), use named reduced dimension representation of cells stored in SingleCellExperiment object instead of recomputing (e.g. "PCA"). Default is NULL, no reduced dimension values are provided to Rtsne.
n_dimred	integer(1), number of components of the reduced dimension slot to use. Default is NULL, in which case (if use_dimred is not NULL) all components of the reduced dimension slot are used.
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
...	further arguments passed to DiffusionMap
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. Alternatively, a data frame with one column containing values to map to colours for all cells.
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_SCE	logical, should the function return an SingleCellExperiment object with principal component values for cells in the reducedDims 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_SCE 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).
rerun	logical, should PCA be recomputed even if object contains a "PCA" element in the reducedDims slot?

Details

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

Value

If return_SCE is TRUE, then the function returns an SingleCellExperiment 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 SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

## Not run:
## Examples plotting diffusion maps
plotDiffusionMap(example_sce)
plotDiffusionMap(example_sce, colour_by = "Cell_Cycle")
plotDiffusionMap(example_sce, colour_by = "Cell_Cycle",
shape_by = "Treatment")
plotDiffusionMap(example_sce, colour_by = "Cell_Cycle",
shape_by = "Treatment", size_by = "Mutation_Status")
plotDiffusionMap(example_sce, shape_by = "Treatment",
size_by = "Mutation_Status")
plotDiffusionMap(example_sce, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status")

plotDiffusionMap(example_sce, shape_by = "Treatment",
return_SCE = TRUE)

## End(Not run)
```

---

runPCA *Plot PCA for a SingleCellExperiment object*

Description

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

Usage

```
runPCA(object, ntop = 500, ncomponents = 2, exprs_values = "logcounts",
  feature_set = NULL, scale_features = TRUE, pca_data_input = "logcounts",
  selected_variables = NULL, detect_outliers = FALSE)

plotPCASCE(object, colour_by = NULL, shape_by = NULL, size_by = NULL,
```

```
  return_SCE = FALSE, draw_plot = TRUE, theme_size = 10,
  legend = "auto", rerun = FALSE, ncomponents = 2,
  detect_outliers = FALSE, ...)

## S4 method for signature 'SingleCellExperiment'
plotPCA(object, colour_by = NULL,
  shape_by = NULL, size_by = NULL, return_SCE = FALSE, draw_plot = TRUE,
  theme_size = 10, legend = "auto", rerun = FALSE, ncomponents = 2,
  detect_outliers = FALSE, ...)
```

Arguments

object	an SingleCellExperiment 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" (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), "logcounts" (log-transformed count data; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values) or any other named element of the assays slot of the SingleCellExperiment object that can be accessed with the assay function.
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.
pca_data_input	character argument defining which data should be used as input for the PCA. Possible options are "logcounts" (default), which uses log-count data to produce a PCA at the cell level; "coldata" or "pdata" (for backwards compatibility) which uses numeric variables from colData(object) to do PCA at the cell level; and "rowdata" which uses numeric variables from rowData(object) to do PCA at the feature level.
selected_variables	character vector indicating which variables in colData(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.
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. Alternatively, a data frame with one column, containing values to map to colours for all cells.

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. Alternatively, a data frame with one column containing values to map to shapes.
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. Alternatively, a data frame with one column containing values to map to sizes.
return_SCE	logical, should the function return an SingleCellExperiment object with principal component values for cells in the reducedDim 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_SCE 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).
rerun	logical, should PCA be recomputed even if object contains a "PCA" element in the reducedDims slot?
...	further arguments passed to plotPCASCE

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

Value

either a ggplot plot object or an SingleCellExperiment object

Examples

```
## Set up an example SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

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

```
size_by = "Mutation_Status")
plotPCA(example_sce, shape_by = "Treatment", size_by = "Mutation_Status")
plotPCA(example_sce, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status")

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

plotPCA(example_sce, shape_by = "Treatment", return_SCE = TRUE)

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

---

runTSNE *Plot t-SNE for an SingleCellExperiment object*

Description

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

Usage

```
runTSNE(object, ntop = 500, ncomponents = 2, exprs_values = "logcounts",
  feature_set = NULL, use_dimred = NULL, n_dimred = NULL,
  scale_features = TRUE, rand_seed = NULL,
  perplexity = floor(ncol(object)/5), ...)

plotTSNE(object, colour_by = NULL, shape_by = NULL, size_by = NULL,
  return_SCE = FALSE, draw_plot = TRUE, theme_size = 10,
  legend = "auto", rerun = FALSE, ncomponents = 2, ...)
```

Arguments

object	an SingleCellExperiment object
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" (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), "logcounts" (log-transformed count data; default), "norm_exprs" (normalised expression values) or "stand_exprs" (standardised expression values), or any other named element of the assayData slot of the SingleCellExperiment object that can be accessed with the assay function.
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).
use_dimred	character(1), use named reduced dimension representation of cells stored in SingleCellExperiment object instead of recomputing (e.g. "PCA"). Default is NULL, no reduced dimension values are provided to Rtsne.
n_dimred	integer(1), number of components of the reduced dimension slot to use. Default is NULL, in which case (if use_dimred is not NULL) all components of the reduced dimension slot are used.
scale_features	logical, should the expression values be standardised so that each feature has unit variance? Default is TRUE.
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.
...	further arguments passed to Rtsne
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. Alternatively, a data frame with one column containing values to map to colours for all cells.
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. Alternatively, a data frame with one column containing values to map to shapes.
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. Alternatively, a data frame with one column containing values to map to sizes.
return_SCE	logical, should the function return an SingleCellExperiment object with principal component values for cells in the reducedDims 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_SCE 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).
rerun	logical, should PCA be recomputed even if object contains a "PCA" element in the reducedDims slot?

Details

The function Rtsne is used internally to compute the t-SNE. Note that the algorithm is not deterministic, so different runs of the function will produce differing plots (see set.seed to set a random seed for replicable results). The value of the perplexity parameter can have a large effect on the resulting plot, so it can often be worthwhile to try multiple values to find the most appealing visualisation.

Value

If return_SCE is TRUE, then the function returns a SingleCellExperiment 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 SingleCellExperiment
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]

## Examples plotting t-SNE
plotTSNE(example_sce, perplexity = 10)
plotTSNE(example_sce, colour_by = "Cell_Cycle", perplexity = 10)
plotTSNE(example_sce, colour_by = "Cell_Cycle", shape_by = "Treatment",
size_by = "Mutation_Status", perplexity = 10)
plotTSNE(example_sce, shape_by = "Treatment", size_by = "Mutation_Status",
perplexity = 5)
plotTSNE(example_sce, feature_set = 1:100, colour_by = "Treatment",
shape_by = "Mutation_Status", perplexity = 5)

plotTSNE(example_sce, shape_by = "Treatment", return_SCE = TRUE,
perplexity = 10)
```

---

salmon-wrapper *Salmon wrapper functions*

Description

Salmon wrapper functions

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

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

```
readSalmonResultsOneSample(directory)

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

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

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

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.

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

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 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.

an SingleCellExperiment object

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 Salmon results are in the directory "output", then call:
readSalmonResultsOneSample("output")

## End(Not run)
## 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)

## 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(object)
```

Arguments

object	SinglCellExperiment 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")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
example_sce <- normalize(example_sce)
drop_genes <- apply(exprs(example_sce), 1, function(x) {var(x) == 0})
example_sce <- example_sce[!drop_genes, ]
example_sce <- calculateQCMetrics(example_sce,
feature_controls = list(set1 = 1:40))
## Not run:
scater_gui(example_sce)

## 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

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

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

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

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

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

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

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

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

References

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

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

Description

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

Usage

```
sc_example_cell_info
```

Format

a data.frame instance, 1 row per cell.

Value

NULL, but makes aavailable a data frame with cell metadata

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

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

Description

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

Usage

```
sc_example_counts
```

Format

a matrix instance, 1 row per gene.

Value

NULL, but makes aavailable a matrix of count data

Author(s)

Davis McCarthy, 2015-03-05

Source

Wellcome Trust Centre for Human Genetics, Oxford

`summariseExprsAcrossFeatures` *Summarise expression values across feature*

Description

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

Usage

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

Arguments

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

Details

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

Value

an SingleCellExperiment object

Examples

```
data("sc_example_counts")
data("sc_example_cell_info")
example_sce <- SingleCellExperiment(
assays = list(counts = sc_example_counts), colData = sc_example_cell_info)
rd <- data.frame(gene_id = rownames(example_sce),
feature_id = paste("feature", rep(1:500, each = 4), sep = "_"))
rownames(rd) <- rownames(example_sce)
rowData(example_sce) <- rd
effective_length <- rep(c(1000, 2000), times = 1000)
tpm(example_sce) <- calculateTPM(example_sce, effective_length, calc_from = "counts")

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

---

updateSCESet *Convert an SCESet object to a SingleCellExperiment object*

Description

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

Usage

```
updateSCESet(object)

toSingleCellExperiment(object)
```

Arguments

object	an SCESet object to be updated

Value

a SingleCellExperiment object

Examples

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

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

## End(Not run)
```

Index