

# Package ‘scuttle’

March 30, 2021

**Type** Package

**Version** 1.0.4

**Date** 2020-12-16

**License** GPL-3

**Title** Single-Cell RNA-Seq Analysis Utilities

**Description** Provides basic utility functions for performing single-cell analyses, focusing on simple normalization, quality control and data transformations. Also provides some helper functions to assist development of other packages.

**Depends** SingleCellExperiment

**Imports** methods, utils, stats, Matrix, Rcpp, BiocGenerics, S4Vectors, BiocParallel, GenomicRanges, SummarizedExperiment, DelayedArray, DelayedMatrixStats, beachmat

**Suggests** BiocStyle, knitr, scRNAseq, rmarkdown, testthat

**VignetteBuilder** knitr

**biocViews** ImmunoOncology, SingleCell, RNASeq, QualityControl, Preprocessing, Normalization, Transcriptomics, GeneExpression, Sequencing, Software, DataImport

**LinkingTo** Rcpp, beachmat

**SystemRequirements** C++11

**RoxygenNote** 7.1.1

**NeedsCompilation** yes

**git\_url** <https://git.bioconductor.org/packages/scuttle>

**git\_branch** RELEASE\_3\_12

**git\_last\_commit** a827e27

**git\_last\_commit\_date** 2020-12-17

**Date/Publication** 2021-03-29

**Author** Aaron Lun [aut, cre],  
Davis McCarthy [aut]

**Maintainer** Aaron Lun <[infinite.monkeys.with.keyboards@gmail.com](mailto:infinite.monkeys.with.keyboards@gmail.com)>

**R topics documented:**

addPerCellQC . . . . .	2
aggregateAcrossCells . . . . .	3
aggregateAcrossFeatures . . . . .	7
calculateAverage . . . . .	8
calculateCPM . . . . .	9
calculateFPKM . . . . .	11
calculateTPM . . . . .	12
downsampleBatches . . . . .	13
downsampleMatrix . . . . .	15
fitLinearModel . . . . .	17
geometricSizeFactors . . . . .	18
isOutlier . . . . .	19
librarySizeFactors . . . . .	22
logNormCounts . . . . .	23
makePerCellDF . . . . .	26
makePerFeatureDF . . . . .	28
medianSizeFactors . . . . .	30
mockSCE . . . . .	31
normalizeCounts . . . . .	32
numDetectedAcrossCells . . . . .	35
numDetectedAcrossFeatures . . . . .	37
perCellQCMetrics . . . . .	39
perFeatureQCMetrics . . . . .	42
quickPerCellQC . . . . .	44
readSparseCounts . . . . .	46
scuttle-pkg . . . . .	47
scuttle-utils . . . . .	48
sumCountsAcrossCells . . . . .	48
sumCountsAcrossFeatures . . . . .	50
summarizeAssayByGroup . . . . .	52
uniquifyFeatureNames . . . . .	54
whichNonZero . . . . .	55
<b>Index</b>	<b>57</b>

---

 addPerCellQC

*Add QC metrics to a SummarizedExperiment*


---

**Description**

Convenient utilities to compute QC metrics and add them to a [SummarizedExperiment](#)'s row or column metadata.

**Usage**

```
addPerCellQC(x, ...)
```

```
addPerFeatureQC(x, ...)
```

## Arguments

- x            A [SummarizedExperiment](#) object or one of its subclasses.
- ...           For `addPerCellQC`, further arguments to pass to [perCellQCMetrics](#).  
For `addPerFeatureQC`, further arguments to pass to [perFeatureQCMetrics](#).

## Details

These functions are simply wrappers around [perCellQCMetrics](#) and [perFeatureQCMetrics](#), respectively. The computed QC metrics are automatically appended onto the existing `colData` or `rowData`. No protection is provided against duplicated column names.

## Value

x is returned with the QC metrics added to the row or column metadata.

## Author(s)

Aaron Lun

## See Also

[perCellQCMetrics](#) and [perFeatureQCMetrics](#), which do the actual work.

## Examples

```
example_sce <- mockSCE()
example_sce <- addPerCellQC(example_sce)
colData(example_sce)

example_sce <- addPerFeatureQC(example_sce)
rowData(example_sce)
```

---

aggregateAcrossCells    *Aggregate data across groups of cells*

---

## Description

Sum counts or average expression values for each feature across groups of cells, while also aggregating values in the `colData` and other fields in a `SummarizedExperiment`.

## Usage

```
aggregateAcrossCells(x, ...)

## S4 method for signature 'SummarizedExperiment'
aggregateAcrossCells(
  x,
  ids,
  ...,
  statistics = NULL,
  average = NULL,
```

```

suffix = FALSE,
subset.row = NULL,
subset.col = NULL,
store.number = "ncells",
coldata.merge = NULL,
use.assay.type = "counts",
subset_row = NULL,
subset_col = NULL,
store_number = "ncells",
coldata_merge = NULL,
use_exprs_values = NULL
)

## S4 method for signature 'SingleCellExperiment'
aggregateAcrossCells(
  x,
  ids,
  ...,
  subset.row = NULL,
  subset.col = NULL,
  use.altexprs = TRUE,
  use.dimred = TRUE,
  dimred.stats = NULL,
  suffix = FALSE,
  subset_row = NULL,
  subset_col = NULL,
  use_altexprs = NULL,
  use_dimred = NULL
)

```

### Arguments

x	A <a href="#">SingleCellExperiment</a> or <a href="#">SummarizedExperiment</a> containing one or more matrices of expression values to be aggregated; possibly along with <a href="#">colData</a> , <a href="#">reducedDims</a> and <a href="#">altExps</a> elements.
...	For the generic, further arguments to be passed to specific methods. For the <a href="#">SummarizedExperiment</a> method, further arguments to be passed to <a href="#">summarizeAssayByGroup</a> . For the <a href="#">SingleCellExperiment</a> method, arguments to be passed to the <a href="#">SummarizedExperiment</a> method.
ids	A factor (or vector coercible into a factor) specifying the group to which each cell in x belongs. Alternatively, a <a href="#">DataFrame</a> of such vectors or factors, in which case each unique combination of levels defines a group.
statistics	Character vector specifying the type of statistics to be computed, see <a href="#">?summarizeAssayByGroup</a> . If not specified, defaults to "sum".
average	Deprecated, specifies whether to compute the average - use <code>statistics="mean"</code> instead. Only used if <code>statistics=NULL</code> .
suffix	Logical scalar indicating whether to always suffix the assay name with the statistic type.
subset.row	An integer, logical or character vector specifying the features to use. If NULL, defaults to all features. For the <a href="#">SingleCellExperiment</a> method, this argument

	will not affect alternative Experiments, where aggregation is always performed for all features (or not at all, depending on <code>use.altexprs</code> ).
<code>subset.col</code>	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of <code>ids</code> .
<code>store.number</code>	String specifying the field of the output <code>colData</code> to store the number of cells in each group. If <code>NULL</code> , nothing is stored.
<code>coldata.merge</code>	A named list of functions specifying how each column metadata field should be aggregated. Each function should be named according to the name of the column in <code>colData</code> to which it applies. Alternatively, a single function can be supplied, see below for more details.
<code>use.assay.type</code>	A character or integer vector specifying the assay(s) of <code>x</code> containing count matrices.
<code>subset_row</code> , <code>subset_col</code> , <code>store_number</code> , <code>use_exprs_values</code> , <code>use_altexprs</code> , <code>use_dimred</code> , <code>coldata_merge</code>	Soft deprecated equivalents to the arguments described above.
<code>use.altexprs</code>	Logical scalar indicating whether aggregation should be performed for alternative experiments. Alternatively, a character or integer vector specifying the alternative experiments to be aggregated.
<code>use.dimred</code>	Logical scalar indicating whether aggregation should be performed for dimensionality reduction results. Alternatively, a character or integer vector specifying the dimensionality reduction results to be aggregated.
<code>dimred.stats</code>	A character vector specifying how the reduced dimensions should be aggregated by group. This can be one or more of "mean" and "median".

## Details

This function summarizes the assay values in `x` for each group in `ids` using `summarizeAssayByGroup` while also aggregating metadata across cells in a “sensible” manner. This makes it useful for obtaining an aggregated `SummarizedExperiment` during an analysis session; in contrast, `summarizeAssayByGroup` is more lightweight and is better for use inside other functions.

Aggregation of the `colData` is controlled using functions in `coldata.merge`. This can either be:

- A function that takes a subset of entries for any given column metadata field and returns a single value. This can be set to, e.g., `sum` or `median` for numeric covariates, or a function that takes the most abundant level for categorical factors.
- A named list of such functions, where each function is applied to the column metadata field after which it is named. Any field that does not have an entry in `coldata.merge` is “unspecified” and handled as described below. A list element can also be set to `FALSE`, in which case no aggregation is performed for the corresponding field.
- `NULL`, in which case all fields are considered to be unspecified.
- `FALSE`, in which case no aggregation of column metadata is performed.

For any unspecified field, we check if all cells of a group have the same value. If so, that value is reported, otherwise a `NA` is reported for the offending group.

By default, each matrix values is returned with the same name as the original per-cell matrix from which it was derived. If `statistics` is of length greater than 1 or `suffix=TRUE`, the names of all aggregated matrices are suffixed with their type of aggregate statistic.

If `ids` is a `DataFrame`, the combination of levels corresponding to each column is also reported in the column metadata. Otherwise, the level corresponding to each column is reported in the `ids` column metadata field as well as in the column names.

**Value**

A SummarizedExperiment of the same class of `x` is returned containing summed/averaged matrices generated by `summarizeAssayByGroup` on all assays in `use.assay.type`. Column metadata are also aggregated according to the rules in `coldata.merge`, see below.

For the `SingleCellExperiment` method, the output also contains aggregated values for the reduced dimensions and alternative Experiments.

**Dealing with SingleCellExperiments**

If `x` is a `SingleCellExperiment`, aggregation is repeated for each entry of `altExps`. This is done by calling `aggregateAcrossCells` on that entry with the same arguments used for the main Experiment - as such, any column metadata in those entries will also be aggregated following the rules in `coldata.merge`. The exception is `subset.row`, which is not applied to the alternative Experiments as the feature sets are different.

If `x` is a `SingleCellExperiment`, each entry of `reducedDims` is averaged across cells. This assumes that the average of low-dimensional coordinates has some meaning for a group of cells but the sum does not. We can explicitly specify computation of the "mean" or "median" (or both) with `dimred.stats`. If `dimred.stats` is of length greater than 1 or `suffix=TRUE`, the name of each matrix in the output `reducedDims` is suffixed with the type of average.

Users can tune the behavior of the function for these additional fields with `use.altexps` and `use.dimred`. Note that if the alternative experiments themselves are `SingleCellExperiments`, any further nested alternative experiment or reduced dimensions will always be aggregated regardless of the value of `use.altexps` or `use.dimred`.

**Author(s)**

Aaron Lun

**See Also**

`summarizeAssayByGroup`, which does the heavy lifting at the assay level.

**Examples**

```
example_sce <- mockSCE()
ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)
out <- aggregateAcrossCells(example_sce, ids)
out

batches <- sample(1:3, ncol(example_sce), replace=TRUE)
out2 <- aggregateAcrossCells(example_sce,
  DataFrame(label=ids, batch=batches))
out2

# Using another column metadata merge strategy.
example_sce$stuff <- runif(ncol(example_sce))
out3 <- aggregateAcrossCells(example_sce, ids,
  coldata_merge=list(stuff=sum))
out3
```

---

`aggregateAcrossFeatures`*Aggregate feature sets in a SummarizedExperiment*

---

### Description

Sum together expression values (by default, counts) for each feature set in each cell of a [SummarizedExperiment](#) object.

### Usage

```
aggregateAcrossFeatures(  
  x,  
  ids,  
  ...,  
  use.assay.type = "counts",  
  use_exprs_values = NULL  
)
```

### Arguments

<code>x</code>	A <a href="#">SummarizedExperiment</a> containing an expression matrix.
<code>ids</code>	A factor of length <code>nrow(x)</code> , specifying the set to which each feature in <code>x</code> belongs. Alternatively, a list of integer or character vectors, where each vector specifies the indices or names of features in a set. Logical vectors are also supported.
<code>...</code>	Further arguments to be passed to <code>sumCountsAcrossFeatures</code> .
<code>use.assay.type</code>	A character or integer vector specifying the assay(s) of <code>x</code> containing expression matrices.
<code>use_exprs_values</code>	Soft-deprecated equivalent of <code>use.assay.type</code> .

### Value

A [SummarizedExperiment](#) of the same class as `x` is returned, containing summed matrices generated by `sumCountsAcrossFeatures` on all assays in `use.assay.type`.

If `ids` is a factor, row metadata is retained for the first instance of a feature from each set in `ids`. This behavior assumes that `ids` specifies duplicates of the same gene, such that the first instance is a reasonable choice.

If `ids` is a list, row metadata is simply discarded. This behavior assumes that `ids` specifies gene sets such that any existing gene-level metadata is meaningless.

### Author(s)

Aaron Lun

### See Also

[sumCountsAcrossFeatures](#), which does the heavy lifting.

**Examples**

```
example_sce <- mockSCE()
ids <- sample(LETTERS, nrow(example_sce), replace=TRUE)
aggr <- aggregateAcrossFeatures(example_sce, ids)
aggr
```

---

calculateAverage	<i>Calculate per-feature average counts</i>
------------------	---

---

**Description**

Calculate the average count for each feature after normalizing observations using per-cell size factors.

**Usage**

```
calculateAverage(x, ...)

## S4 method for signature 'ANY'
calculateAverage(
  x,
  size.factors = NULL,
  subset.row = NULL,
  BPPARAM = SerialParam(),
  size_factors = NULL,
  subset_row = NULL
)

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

## S4 method for signature 'SingleCellExperiment'
calculateAverage(x, size.factors = NULL, ...)
```

**Arguments**

x	A numeric matrix of counts where features are rows and columns are cells. Alternatively, a <a href="#">SummarizedExperiment</a> or a <a href="#">SingleCellExperiment</a> containing such counts.
...	For the generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
size.factors	A numeric vector containing size factors. If NULL, these are calculated or extracted from x.
subset.row	A vector specifying the subset of rows of object for which to return a result.



**BPPARAM** A [BiocParallelParam](#) object specifying whether the calculations should be parallelized. Only relevant for parallelized `rowSums(x)`, e.g., for [DelayedMatrix](#) inputs.

`size_factors`, `subset_row`, `exprs_values`  
Soft-deprecated counterparts to the arguments above.

`assay.type` A string specifying the assay of `x` containing the count matrix.

## Details

The size factor-adjusted average count is defined by dividing each count by the size factor and taking the average across cells. All size factors are scaled so that the mean is 1 across all cells, to ensure that the averages are interpretable on the same scale of the raw counts.

If no size factors are supplied, they are determined automatically:

- For count matrices and [SummarizedExperiment](#) inputs, the sum of counts for each cell is used to compute a size factor via the [librarySizeFactors](#) function.
- For [SingleCellExperiment](#) instances, the function searches for `sizeFactors` from `x`. If none are available, it defaults to library size-derived size factors.

If `size_factors` are supplied, they will override any size factors present in `x`.

## Value

A numeric vector of average count values with same length as number of features (or the number of features in `subset_row` if supplied).

## Author(s)

Aaron Lun

## See Also

[librarySizeFactors](#), for the default calculation of size factors.

[logNormCounts](#), for the calculation of normalized expression values.

## Examples

```
example_sce <- mockSCE()
ave_counts <- calculateAverage(example_sce)
summary(ave_counts)
```

---

calculateCPM

*Calculate CPMs*

---

## Description

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

**Usage**

```

calculateCPM(x, ...)

## S4 method for signature 'ANY'
calculateCPM(
  x,
  size.factors = NULL,
  subset.row = NULL,
  size_factors = NULL,
  subset_row = NULL
)

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

## S4 method for signature 'SingleCellExperiment'
calculateCPM(x, size.factors = NULL, ...)

```

**Arguments**

<code>x</code>	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a <a href="#">SummarizedExperiment</a> or a <a href="#">SingleCellExperiment</a> containing such counts.
<code>...</code>	For the generic, arguments to pass to specific methods. For the <code>SummarizedExperiment</code> method, further arguments to pass to the ANY method. For the <code>SingleCellExperiment</code> method, further arguments to pass to the <code>SummarizedExperiment</code> method.
<code>size.factors</code>	A numeric vector containing size factors to adjust the library sizes. If <code>NULL</code> , the library sizes are used directly.
<code>subset.row</code>	A vector specifying the subset of rows of <code>x</code> for which to return a result.
<code>size_factors</code> , <code>subset_row</code> , <code>exprs_values</code>	Soft-deprecated counterparts to the arguments above.
<code>assay.type</code>	A string or integer scalar specifying the assay of <code>x</code> containing the count matrix.

**Details**

If `size.factors` are provided or available in `x`, they are used to define the effective library sizes. This is done by scaling all size factors such that the mean factor is equal to the mean sum of counts across all features. The effective library sizes are then used as the denominator of the CPM calculation.

**Value**

A numeric matrix of CPM values with the same dimensions as `x` (unless `subset.row` is specified).

**Author(s)**

Aaron Lun

**See Also**

[normalizeCounts](#), on which this function is based.

**Examples**

```
example_sce <- mockSCE()
cpm(example_sce) <- calculateCPM(example_sce)
str(cpm(example_sce))
```

---

calculateFPKM	<i>Calculate FPKMs</i>
---------------	------------------------

---

**Description**

Calculate fragments per kilobase of exon per million reads mapped (FPKM) values from the feature-level counts.

**Usage**

```
calculateFPKM(x, lengths, ..., subset.row = NULL, subset_row = NULL)
```

**Arguments**

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a <a href="#">SummarizedExperiment</a> or a <a href="#">SingleCellExperiment</a> containing such counts.
lengths	Numeric vector providing the effective length for each feature in x.
...	Further arguments to pass to <a href="#">calculateCPM</a> .
subset.row	A vector specifying the subset of rows of x for which to return a result.
subset_row	Soft-deprecated equivalent to the argument above.

**Details**

FPKMs are computed by dividing the CPMs by the effective length of each gene in kilobases. For RNA-seq datasets, the effective length is best set to the sum of lengths of all exons; for nucleus sequencing datasets, the effective length may instead be the entire width of the gene body.

**Value**

A numeric matrix of FPKM values with the same dimensions as x (unless `subset.row` is specified).

**Author(s)**

Aaron Lun, based on code by Davis McCarthy

**See Also**

[calculateCPM](#), for the initial calculation of CPM values.

**Examples**

```
example_sce <- mockSCE()
eff_len <- runif(nrow(example_sce), 500, 2000)
fout <- calculateFPKM(example_sce, eff_len)
str(fout)
```

---

 calculateTPM

*Calculate TPMs*


---

**Description**

Calculate transcripts-per-million (TPM) values for expression from feature-level counts.

**Usage**

```
calculateTPM(x, ...)

## S4 method for signature 'ANY'
calculateTPM(x, lengths = NULL, ...)

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

## S4 method for signature 'SingleCellExperiment'
calculateTPM(x, lengths = NULL, size.factors = NULL, ...)
```

**Arguments**

x	A numeric matrix of counts where features are rows and cells are columns. Alternatively, a <a href="#">SummarizedExperiment</a> or a <a href="#">SingleCellExperiment</a> containing such counts.
...	For the generic, arguments to pass to specific methods. For the ANY method, further arguments to pass to <a href="#">calculateCPM</a> . For the SummarizedExperiment method, further arguments to pass to the ANY method. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
lengths	Numeric vector providing the effective length for each feature in x. Alternatively NULL, see Details.
assay.type	A string specifying the assay of x containing the count matrix.
exprs_values	Soft-deprecated equivalents to the arguments above.
size.factors	A numeric vector containing size factors to adjust the library sizes. If NULL, the library sizes are used directly.

## Details

For read count data, this function assumes uniform coverage along the (effective) length of the transcript. Thus, the number of transcripts for a gene is proportional to the read count divided by the transcript length. Here, the division is done before calculation of the library size to compute per-million values, where `calculateFPKM` will only divide by the length after library size normalization.

For UMI count data, this function should be run with `lengths=NULL`, i.e., no division by the effective length. This is because the number of UMIs is a direct (albeit biased) estimate of the number of transcripts.

## Value

A numeric matrix of TPM values with the same dimensions as `x` (unless `subset.row` is specified).

## Author(s)

Aaron Lun, based on code by Davis McCarthy

## See Also

`calculateCPM`, on which this function is based.

## Examples

```
example_sce <- mockSCE()
eff_len <- runif(nrow(example_sce), 500, 2000)
tout <- calculateTPM(example_sce, lengths = eff_len)
str(tout)
```

---

downsampleBatches	<i>Downsample batches to equal coverage</i>
-------------------	---

---

## Description

A convenience function to downsample all batches so that the average per-cell total count is the same across batches. This mimics the downsampling functionality of `cellranger aggr`.

## Usage

```
downsampleBatches(
  ...,
  batch = NULL,
  block = NULL,
  method = c("median", "mean", "geomean"),
  bycol = TRUE,
  assay.type = 1
)
```

**Arguments**

...	<p>Two or more count matrices, where each matrix has the same set of genes (rows) and contains cells (columns) from a separate batch.</p> <p>Alternatively, one or more entries may be a <a href="#">SummarizedExperiment</a>, in which case the count matrix is extracted from the assays according to <code>assay.type</code>.</p> <p>A list containing two or more of these matrices or <code>SummarizedExperiments</code> can also be supplied.</p> <p>Alternatively, a single count matrix or <code>SummarizedExperiment</code> can be supplied. This is assumed to contain cells from all batches, in which case <code>batch</code> should also be specified.</p>
batch	A factor of length equal to the number of columns in the sole entry of ..., specifying the batch of origin for each column of the matrix. Ignored if there are multiple entries in ...
block	<p>If ... contains multiple matrices or <code>SummarizedExperiments</code>, this should be a character vector of length equal to the number of objects in ..., specifying the blocking level for each object (see <a href="#">Details</a>).</p> <p>Alternatively, if ... contains a single object, this should be a character vector or factor of length specifying the blocking level for each cell in that object.</p>
method	String indicating how the average total should be computed. The geometric mean is computed with a pseudo-count of 1.
bycol	A logical scalar indicating whether downsampling should be performed on a column-by-column basis, see <a href="#">?downsampleMatrix</a> for more details.
assay.type	String or integer scalar specifying the assay of the <code>SummarizedExperiment</code> containing the count matrix, if any <code>SummarizedExperiments</code> are present in ...

**Details**

Downsampling batches with strong differences in sequencing coverage can make it easier to compare them to each other, reducing the burden on the normalization and batch correction steps. This is especially true when the number of cells cannot be easily controlled across batches, resulting in large differences in per-cell coverage even when the total sequencing depth is the same.

Generally speaking, the matrices in ... should be filtered so that only libraries with cells are present. This is most relevant to high-throughput scRNA-seq experiments (e.g., using droplet-based protocols) where the majority of libraries do not actually contain cells. If these are not filtered out, downsampling will equalize coverage among the majority of empty libraries rather than among cell-containing libraries.

In more complex experiments, batches can be organized into blocks where downsampling is performed to the lowest-coverage batch within each block. This is most useful for larger datasets involving technical replicates for the same biological sample. By setting `block=` to the biological sample, we can equalize coverage across replicates within each sample without forcing all samples to have the same coverage (e.g., to avoid loss of information if they are to be analyzed separately anyway).

**Value**

If ... contains two or more matrices, a [List](#) of downsampled matrices is returned.

Otherwise, if ... contains only one matrix, the downsampled matrix is returned directly.

**Author(s)**

Aaron Lun

**See Also**[downsampleMatrix](#), which is called by this function under the hood.**Examples**

```
sce1 <- mockSCE()
sce2 <- mockSCE()

# Downsampling for multiple batches in a single matrix:
combined <- cbind(sce1, sce2)
batches <- rep(1:2, c(ncol(sce1), ncol(sce2)))
downsampled <- downsampleBatches(counts(combined), batch=batches)
downsampled[1:10,1:10]

# Downsampling for multiple matrices:
downsampled2 <- downsampleBatches(counts(sce1), counts(sce2))
downsampled2
```

---

downsampleMatrix	<i>Downsample a count matrix</i>
------------------	----------------------------------

---

**Description**

Downsample a count matrix to a desired proportion, either on a whole-matrix or per-cell basis.

**Usage**

```
downsampleMatrix(x, prop, bycol = TRUE, sink = NULL)
```

**Arguments**

x	An integer or numeric matrix-like object containing counts.
prop	A numeric scalar or, if bycol=TRUE, a vector of length ncol(x). All values should lie in [0, 1] specifying the downsampling proportion for the matrix or for each cell.
bycol	A logical scalar indicating whether downsampling should be performed on a column-by-column basis.
sink	A <a href="#">RealizationSink</a> object specifying the format of the downsampled matrix should be returned.

## Details

Given multiple 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.

If `bycol=TRUE`, sampling without replacement is performed on the count vector for each cell. This yields a new count vector where the total is equal to `prop` times the original total count. Each count in the returned matrix is guaranteed to be smaller than the original value in `x`. Different proportions can be specified for different cells by setting `prop` to a vector; in this manner, downsampling can be used as an alternative to scaling for per-cell normalization.

If `bycol=FALSE`, downsampling without replacement is performed on the entire matrix. This yields a new matrix where the total count across all cells is equal to `prop` times the original total. The new total count for each cell may not be exactly equal to `prop` times the original value, which may or may not be more appropriate than `bycol=TRUE` for particular applications.

By default, the output format will be a dense matrix for an ordinary matrix `x`, and a sparse matrix for any sparse format (i.e., if `is_sparse(x)` returns `TRUE`). This can be overridden to specify custom formats with `sink`, e.g., for HDF5-backed matrices, which may be helpful for dealing with large matrices where the downsampled result may not fit in memory.

Note that this function was originally implemented in the **scater** package as `downsampleCounts`, was moved to the **DropletUtils** package as `downsampleMatrix`, and finally found a home here.

## Value

An numeric matrix-like object of downsampled counts. This is a [dgCMatrix](#) unless `sink` is set, in which case it is a [DelayedMatrix](#).

## Author(s)

Aaron Lun

## See Also

[downsampleReads](#) in the **DropletUtils** package, which downsamples reads rather than observed counts.

[normalizeCounts](#), where downsampling can be used as an alternative to scaling normalization.

## Examples

```
sce <- mockSCE()
sum(counts(sce))

downsampled <- downsampleMatrix(counts(sce), prop = 0.5, bycol=FALSE)
sum(downsampled)

downsampled2 <- downsampleMatrix(counts(sce), prop = 0.5, bycol=TRUE)
sum(downsampled2)
```



---

fitLinearModel	<i>Fit a linear model</i>
----------------	---------------------------

---

## Description

No-frills fitting of a linear model to the rows of any matrix-like object with numeric values.

## Usage

```
fitLinearModel(
  x,
  design,
  get.coefs = TRUE,
  subset.row = NULL,
  BPPARAM = SerialParam(),
  rank.error = TRUE
)
```

## Arguments

<code>x</code>	A numeric matrix-like object where columns are samples (e.g., cells) and rows are usually features (e.g., genes).
<code>design</code>	A numeric design matrix with number of rows equal to <code>ncol(x)</code> . This should be of full column rank.
<code>get.coefs</code>	A logical scalar indicating whether the coefficients should be returned.
<code>subset.row</code>	An integer, character or logical vector indicating the rows of <code>x</code> to use for model fitting.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying the parallelization backend to use.
<code>rank.error</code>	Logical scalar indicating whether to throw an error when <code>design</code> is not of full rank.

## Details

This function is basically a stripped-down version of `lm.fit`, made to operate on any matrix representation (ordinary, sparse, whatever). It is generally intended for use inside other functions that require robust and efficient linear model fitting.

If `design` is not of full rank and `rank.error=TRUE`, an error is raised. If `rank.error=FALSE`, NA values are returned for all entries of the output list.

## Value

If `get.coefs=TRUE`, a list is returned containing:

- `coefficients`, a numeric matrix of coefficient estimates, with one row per row of `x` (or a subset thereof specified by `subset.row`) and one column per column of `design`.
- `mean`, a numeric vector of row means of `x`. Computed as a courtesy to avoid iterating over the matrix twice.
- `variance`, a numeric vector of residual variances per row of `x`. Computed by summing the residual effects from the fitted model.
- `residual.df`, an integer scalar containing the residual degrees of freedom for `design`.

Otherwise, if `get.coefs=FALSE`, the same list is returned without coefficients.

**Author(s)**

Aaron Lun

**Examples**

```

y <- Matrix::rsparsematrix(1000, 1000, 0.1)
design <- model.matrix(~runif(1000))

output <- fitLinearModel(y, design)
head(output$coefficients)
head(output$variance)

```

---

geometricSizeFactors *Compute geometric size factors*

---

**Description**

Define per-cell size factors from the geometric mean of counts per cell.

**Usage**

```

geometricSizeFactors(x, ...)

## S4 method for signature 'ANY'
geometricSizeFactors(
  x,
  subset.row = NULL,
  pseudo.count = 1,
  BPPARAM = SerialParam()
)

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

computeGeometricFactors(x, ...)

```

**Arguments**

x	For <code>geometricSizeFactors</code> , a numeric matrix of counts with one row per feature and column per cell. Alternatively, a <a href="#">SummarizedExperiment</a> or <a href="#">SingleCellExperiment</a> containing such counts. For <code>computeGeometricFactors</code> , only a <a href="#">SingleCellExperiment</a> containing a count matrix is accepted.
...	For the <code>geometricSizeFactors</code> generic, arguments to pass to specific methods. For the <code>SummarizedExperiment</code> method, further arguments to pass to the ANY method. For <code>computeGeometricFactors</code> , further arguments to pass to <code>geometricSizeFactors</code> .
subset.row	A vector specifying whether the size factors should be computed from a subset of rows of x.

<code>pseudo.count</code>	Numeric scalar specifying the pseudo-count to add during log-transformation.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object indicating how calculations are to be parallelized. Only relevant when <code>x</code> is a <a href="#">DelayedArray</a> object.
<code>assay.type</code>	String or integer scalar indicating the assay of <code>x</code> containing the counts.

### Details

The geometric mean provides an alternative measure of the average coverage per cell, in contrast to the library size factors (i.e., the arithmetic mean) computed by [librarySizeFactors](#). The main advantage of the geometric mean is that it is more robust to composition biases from a few strongly upregulated genes.

On the other hand, the geometric mean is a poor estimator of the relative bias at low or zero counts. This is because the scaling of the coverage applies to the expectation of the raw counts, so the geometric mean only becomes an accurate estimator if the mean of the logs approaches the log of the mean (usually at high counts). The arbitrary pseudo-count also has a bigger influence at low counts.

As such, the geometric mean is only well-suited for deeply sequenced features, e.g., antibody-derived tags.

### Value

For `geometricSizeFactors`, a numeric vector of size factors is returned for all methods.

For `computeGeometricFactors`, `x` is returned containing the size factors in `sizeFactors(x)`.

### Author(s)

Aaron Lun

### See Also

[normalizeCounts](#) and [logNormCounts](#), where these size factors are used by default.

[geometricSizeFactors](#) and [medianSizeFactors](#), for two other simple methods of computing size factors.

### Examples

```
example_sce <- mockSCE()
summary(geometricSizeFactors(example_sce))
```

---

isOutlier

*Identify outlier values*

---

### Description

Convenience function to determine which values in a numeric vector are outliers based on the median absolute deviation (MAD).

**Usage**

```
isOutlier(
  metric,
  nmads = 3,
  type = c("both", "lower", "higher"),
  log = FALSE,
  subset = NULL,
  batch = NULL,
  share.medians = FALSE,
  share.mads = FALSE,
  share.missing = TRUE,
  min.diff = NA,
  share_medians = NULL,
  share_mads = NULL,
  share_missing = NULL,
  min_diff = NULL
)
```

**Arguments**

<code>metric</code>	Numeric vector of values.
<code>nmads</code>	A numeric scalar, specifying the minimum number of MADs away from median required for a value to be called an outlier.
<code>type</code>	String indicating whether outliers should be looked for at both tails ("both"), only at the lower tail ("lower") or the upper tail ("higher").
<code>log</code>	Logical scalar, should the values of the metric be transformed to the log <sub>2</sub> scale before computing MADs?
<code>subset</code>	Logical or integer vector, which subset of values should be used to calculate the median/MAD? If NULL, all values are used.
<code>batch</code>	Factor of length equal to <code>metric</code> , specifying the batch to which each observation belongs. A median/MAD is calculated for each batch, and outliers are then identified within each batch.
<code>share.medians</code>	Logical scalar indicating whether the median calculation should be shared across batches. Only used if <code>batch</code> is specified.
<code>share.mads</code>	Logical scalar indicating whether the MAD calculation should be shared across batches. Only used if <code>batch</code> is specified.
<code>share.missing</code>	Logical scalar indicating whether a common MAD/median should be used for any batch that has no values left after subsetting. Only relevant when both <code>batch</code> and <code>subset</code> are specified.
<code>min.diff</code>	A numeric scalar indicating the minimum difference from the median to consider as an outlier. Ignored if NA.
<code>share_medians</code> , <code>share_mads</code> , <code>share_missing</code> , <code>min_diff</code>	Soft-deprecated equivalents of the arguments above.

**Details**

Lower and upper thresholds are stored in the "threshold" attribute of the returned vector. By default, this is a numeric vector of length 2 for the threshold on each side. If `type="lower"`, the higher limit is Inf, while if `type="higher"`, the lower limit is -Inf.

If `min.diff` is not NA, the minimum distance from the median required to define an outlier is set as the larger of `nmads` MADs and `min.diff`. This aims to avoid calling many outliers when the MAD is very small, e.g., due to discreteness of the metric. If `log=TRUE`, this difference is defined on the  $\log_2$  scale.

If `subset` is specified, the median and MAD are computed from a subset of cells and the values are used to define the outlier threshold that is applied to all cells. In a quality control context, this can be handy for excluding groups of cells that are known to be low quality (e.g., failed plates) so that they do not distort the outlier definitions for the rest of the dataset.

Missing values trigger a warning and are automatically ignored during estimation of the median and MAD. The corresponding entries of the output vector are also set to NA values.

### Value

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

### Handling batches

If `batch` is specified, outliers are defined within each batch separately using batch-specific median and MAD values. This gives the same results as if the input metrics were subsetted by batch and `isOutlier` was run on each subset, and is often useful when batches are known *a priori* to have technical differences (e.g., in sequencing depth).

If `share.medians=TRUE`, a shared median is computed across all cells. If `share.mads=TRUE`, a shared MAD is computed using all cells (based on either a batch-specific or shared median, depending on `share.medians`). These settings are useful to enforce a common location or spread across batches, e.g., we might set `share.mads=TRUE` for log-library sizes if coverage varies across batches but the variance across cells is expected to be consistent across batches.

If a batch does not have sufficient cells to compute the median or MAD (e.g., after applying `subset`), the default setting of `share.missing=TRUE` will set these values to the shared median and MAD. This allows us to define thresholds for low-quality batches based on information in the rest of the dataset. (Note that the use of shared values only affects this batch and not others unless `share.medians` and `share.mads` are also set.) Otherwise, if `share.missing=FALSE`, all cells in that batch will have NA in the output.

If `batch` is specified, the "threshold" attribute in the returned vector is a matrix with one named column per level of batch and two rows (one per threshold).

### Author(s)

Aaron Lun

### See Also

[quickPerCellQC](#), a convenience wrapper to perform outlier-based quality control.

[perCellQCMetrics](#), to compute potential QC metrics.

### Examples

```
example_sce <- mockSCE()
stats <- perCellQCMetrics(example_sce)

str(isOutlier(stats$sum))
str(isOutlier(stats$sum, type="lower"))
```

```
str(isOutlier(stats$sum, type="higher"))

str(isOutlier(stats$sum, log=TRUE))

b <- sample(LETTERS[1:3], ncol(example_sce), replace=TRUE)
str(isOutlier(stats$sum, log=TRUE, batch=b))
```

---

librarySizeFactors      *Compute library size factors*

---

## Description

Define per-cell size factors from the library sizes (i.e., total sum of counts per cell).

## Usage

```
librarySizeFactors(x, ...)

## S4 method for signature 'ANY'
librarySizeFactors(
  x,
  subset.row = NULL,
  geometric = FALSE,
  BPPARAM = SerialParam(),
  subset_row = NULL,
  pseudo_count = 1
)

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

computeLibraryFactors(x, ...)
```

## Arguments

x	For librarySizeFactors, a numeric matrix of counts with one row per feature and column per cell. Alternatively, a <a href="#">SummarizedExperiment</a> or <a href="#">SingleCellExperiment</a> containing such counts. For computeLibraryFactors, only a <a href="#">SingleCellExperiment</a> containing a count matrix is accepted.
...	For the librarySizeFactors generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For computeLibraryFactors, further arguments to pass to librarySizeFactors.
subset.row	A vector specifying whether the size factors should be computed from a subset of rows of x.
geometric	Deprecated, logical scalar indicating whether the size factor should be defined using the geometric mean.
BPPARAM	A <a href="#">BiocParallelParam</a> object indicating how calculations are to be parallelized. Only relevant when x is a <a href="#">DelayedArray</a> object.

subset\_row, exprs\_values  
Soft-deprecated equivalents to the arguments above.

pseudo\_count    Deprecated, numeric scalar specifying the pseudo-count to add when `geometric=TRUE`.

assay.type      String or integer scalar indicating the assay of `x` containing the counts.

### Details

Library sizes are converted into size factors by scaling them so that their mean across cells is unity. This ensures that the normalized values are still on the same scale as the raw counts. Preserving the scale is useful for interpretation of operations on the normalized values, e.g., the pseudo-count used in `logNormCounts` can actually be considered an additional read/UMI. This is important for ensuring that the effect of the pseudo-count decreases with increasing sequencing depth, see `?normalizeCounts` for a discussion of this effect.

With library size-derived size factors, we implicitly assume that sequencing coverage is the only difference between cells. This is reasonable for homogeneous cell populations but is compromised by composition biases from DE between cell types. In such cases, the library size factors will not be correct though any effects on downstream conclusions will vary, e.g., clustering is usually unaffected by composition biases but log-fold change estimates will be less accurate.

### Value

For `librarySizeFactors`, a numeric vector of size factors is returned for all methods.

For `computeLibraryFactors`, `x` is returned containing the size factors in `sizeFactors(x)`.

### Author(s)

Aaron Lun

### See Also

`normalizeCounts` and `logNormCounts`, where these size factors are used by default.

`geometricSizeFactors` and `medianSizeFactors`, for two other simple methods of computing size factors.

### Examples

```
example_sce <- mockSCE()
summary(librarySizeFactors(example_sce))
```

---

logNormCounts

*Compute log-normalized expression values*

---

### Description

Compute log-transformed normalized expression values from a count matrix in a `SingleCellExperiment` object.

**Usage**

```
logNormCounts(x, ...)

## S4 method for signature 'SummarizedExperiment'
logNormCounts(
  x,
  size.factors = NULL,
  log = TRUE,
  pseudo.count = 1,
  center.size.factors = TRUE,
  ...,
  assay.type = "counts",
  name = NULL,
  BPPARAM = SerialParam(),
  size_factors = NULL,
  pseudo_count = NULL,
  center_size_factors = NULL,
  exprs_values = NULL
)

## S4 method for signature 'SingleCellExperiment'
logNormCounts(
  x,
  size.factors = NULL,
  log = TRUE,
  pseudo.count = 1,
  center.size.factors = TRUE,
  ...,
  assay.type = "counts",
  use.altexprs = FALSE,
  name = NULL,
  BPPARAM = SerialParam(),
  size_factors = NULL,
  pseudo_count = NULL,
  center_size_factors = NULL,
  exprs_values = NULL,
  use_altexprs = NULL
)
```

**Arguments**

x	A <a href="#">SingleCellExperiment</a> or <a href="#">SummarizedExperiment</a> object containing a count matrix.
...	For the generic, additional arguments passed to specific methods. For the methods, additional arguments passed to <a href="#">normalizeCounts</a> .
size.factors	A numeric vector of cell-specific size factors. Alternatively NULL, in which case the size factors are extracted or computed from x.
log	Logical scalar indicating whether normalized values should be log2-transformed.
pseudo.count	Numeric scalar specifying the pseudo-count to add when log-transforming expression values.



<code>center.size.factors</code>	Logical scalar indicating whether size factors should be centered at unity before being used.
<code>assay.type</code>	A string or integer scalar specifying the assay of <code>x</code> containing the count matrix.
<code>name</code>	String containing an assay name for storing the output normalized values. Defaults to "logcounts" when <code>log=TRUE</code> and "normcounts" otherwise.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying how library size factor calculations should be parallelized. Only used if <code>size.factors</code> is not specified.
<code>size_factors</code>	Soft-deprecated equivalents to the arguments described previously.
<code>pseudo_count</code>	Soft-deprecated equivalents to the arguments described previously.
<code>center_size_factors</code>	Soft-deprecated equivalents to the arguments described previously.
<code>exprs_values</code>	Soft-deprecated equivalents to the arguments described previously.
<code>use.altexps</code>	Logical scalar indicating whether normalization should be performed for alternative experiments in <code>x</code> . Alternatively, a character vector specifying the names of the alternative experiments to be normalized. Alternatively NULL, in which case no calculations are performed on the alternative experiments.
<code>use_altexps</code>	Soft-deprecated equivalent to the argument above.

## Details

This function is a convenience wrapper around [normalizeCounts](#). It returns a [SingleCellExperiment](#) or [SummarizedExperiment](#) containing the normalized values in a separate assay. This makes it easier to perform normalization by avoiding book-keeping errors during a long analysis workflow.

If `x` is a [SingleCellExperiment](#) that contains alternative Experiments, normalized values can be computed and stored within each alternative experiment by setting `use.altexps` appropriately. By default, `use.altexps=FALSE` to avoid problems from attempting to library size-normalize alternative experiments that have zero total counts for some cells.

If `size.factors=NULL`, size factors are obtained following the rules in [normalizeCounts](#). This is done independently for the main and alternative Experiments when `use.altexps` is specified, i.e. no information is shared between Experiments by default. However, if `size.factors` is supplied, it will override any size factors available in any Experiment.

## Value

`x` is returned containing the (log-)normalized expression values in an additional assay named as `name`.

If `x` is a [SingleCellExperiment](#), the size factors used for normalization are stored in `sizeFactors`. These are centered if `center.size.factors=TRUE`.

If `x` contains alternative experiments and `use.altexps=TRUE`, each of the alternative experiments in `x` will also contain an additional assay. This can be limited to particular `altExps` entries by specifying them in `use.altexps`.

## Author(s)

Aaron Lun, based on code by Davis McCarthy

**See Also**

[normalizeCounts](#), which is used to compute the normalized expression values.

**Examples**

```
example_sce <- mockSCE()

# Standard library size normalization:
example_sce2 <- logNormCounts(example_sce)
assayNames(example_sce2)
logcounts(example_sce2)[1:5,1:5]

# Without logging, the assay is 'normcounts':
example_sce2 <- logNormCounts(example_sce, log=FALSE)
assayNames(example_sce2)
normcounts(example_sce2)[1:5,1:5]

# Pre-loading with size factors:
example_sce2 <- computeMedianFactors(example_sce)
example_sce2 <- logNormCounts(example_sce2)
logcounts(example_sce2)[1:5,1:5]

# Also normalizing the alternative experiments:
example_sce2 <- logNormCounts(example_sce, use.altexps="Spikes")
logcounts(altExp(example_sce2))[1:5,1:5]
```

---

makePerCellDF

*Create a per-cell data.frame*


---

**Description**

Create a per-cell data.frame (i.e., where each row represents a cell) from a [SingleCellExperiment](#), most typically for creating custom **ggplot2** plots.

**Usage**

```
makePerCellDF(
  x,
  features = NULL,
  assay.type = "logcounts",
  use.coldata = TRUE,
  use.dimred = TRUE,
  use.altexps = TRUE,
  prefix.altexps = FALSE,
  check.names = FALSE,
  exprs_values = NULL,
  use_dimred = NULL,
  use_altexps = NULL,
  prefix_altexps = NULL,
  check_names = NULL
)
```

**Arguments**

<code>x</code>	A <a href="#">SingleCellExperiment</a> object. This is expected to have non-NULL row names.
<code>features</code>	Character vector specifying the features for which to extract expression profiles across cells. May also include features in alternative Experiments if permitted by <code>use_altexps</code> .
<code>assay.type</code>	String or integer scalar indicating the assay to use to obtain expression values. Must refer to a matrix-like object with integer or numeric values.
<code>use.coldata</code>	Logical scalar indicating whether column metadata of <code>x</code> should be included. Alternatively, a character or integer vector specifying the column metadata fields to use.
<code>use.dimred</code>	Logical scalar indicating whether data should be extracted for dimensionality reduction results in <code>x</code> . Alternatively, a character or integer vector specifying the dimensionality reduction results to use.
<code>use.altexps</code>	Logical scalar indicating whether (meta)data should be extracted for alternative experiments in <code>x</code> . Alternatively, a character or integer vector specifying the alternative experiments to use.
<code>prefix.altexps</code>	Logical scalar indicating whether <code>altExp</code> -derived fields should be prefixed with the name of the alternative Experiment.
<code>check.names</code>	Logical scalar indicating whether column names of the output should be made syntactically valid and unique.
<code>exprs_values, use_dimred, use_altexps, prefix_altexps, check_names</code>	Soft-deprecated equivalents of the arguments described above.

**Details**

This function enables us to conveniently create a per-feature `data.frame` from a [SingleCellExperiment](#). Each row of the returned `data.frame` corresponds to a column in `x`, while each column of the `data.frame` corresponds to one aspect of the (meta)data in `x`.

Columns are provided in the following order:

1. Columns named according to the entries of `features` represent the expression values across cells for the specified feature in the `assay.type` assay.
2. Columns named according to the columns of `colData(x)` represent column metadata variables. This consists of all variables if `use.coldata=TRUE`, no variables if `use.coldata=FALSE`, and only the specified variables if `use.coldata` is set to an integer or character vector.
3. Columns named in the format of `<DIM>.<NUM>` represent the `<NUM>`th dimension of the dimensionality reduction result `<DIM>`. This is generated for all dimensionality reduction results if `use.dimred=TRUE`, none if `use.dimred=FALSE`, and only the specified results if `use.dimred` is set to an integer or character vector.
4. Columns named according to the row names of successive alternative Experiments, representing the assay data in these objects. These columns are only included if they are specified in `features` and if `use.altexps` is set. Column names are prefixed with the name of the alternative Experiment if `prefix.altexps=TRUE`.

By default, nothing is done to resolve syntactically invalid or duplicated column names. `check_names=TRUE`, this is resolved by passing the column names through `make.names`. Of course, as a result, some columns may not have the same names as the original fields in `x`.

**Value**

A data.frame containing one field per aspect of data in `x` - see Details. Each row corresponds to a cell (i.e., column) of `x`.

**Author(s)**

Aaron Lun

**See Also**

[makePerFeatureDF](#), for the feature-level equivalent.

**Examples**

```
sce <- mockSCE()
sce <- logNormCounts(sce)
reducedDim(sce, "PCA") <- matrix(rnorm(ncol(sce)*10), ncol=10) # made-up PCA.

df <- makePerCellDF(sce, features="Gene_0001")
head(df)
```

---

makePerFeatureDF	<i>Create a per-feature data.frame</i>
------------------	--

---

**Description**

Create a per-feature data.frame (i.e., where each row represents a feature) from a [SingleCellExperiment](#), most typically for creating custom **ggplot2** plots.

**Usage**

```
makePerFeatureDF(
  x,
  cells = NULL,
  assay.type = "logcounts",
  use.rowdata = TRUE,
  check.names = FALSE,
  exprs_values = NULL,
  check_names = NULL
)
```

**Arguments**

<code>x</code>	A <a href="#">SingleCellExperiment</a> object. This is expected to have non-NULL row names.
<code>cells</code>	Character vector specifying the features for which to extract expression profiles across cells.
<code>assay.type</code>	String or integer scalar indicating the assay to use to obtain expression values. Must refer to a matrix-like object with integer or numeric values.

<code>use.rowdata</code>	Logical scalar indicating whether row metadata of <code>x</code> should be included. Alternatively, a character or integer vector specifying the row metadata fields to use.
<code>check.names</code>	Logical scalar indicating whether column names of the output should be made syntactically valid and unique.
<code>exprs_values</code> , <code>check_names</code>	Soft-deprecated equivalents to the arguments above.

## Details

This function enables us to conveniently create a per-feature data.frame from a [SingleCellExperiment](#). Each row of the returned data.frame corresponds to a row in `x`, while each column of the data.frame corresponds to one aspect of the (meta)data in `x`.

Columns are provided in the following order:

1. Columns named according to the entries of `cells` represent the expression values across features for the specified cell in the assay. `type assay`.
2. Columns named according to the columns of `rowData(x)` represent the row metadata variables. This consists of all variables if `use.rowdata=TRUE`, no variables if `use.rowdata=FALSE`, and only the specified variables if `use.rowdata` is set to an integer or character vector.

By default, nothing is done to resolve syntactically invalid or duplicated column names. `check_names=TRUE`, this is resolved by passing the column names through [make.names](#). Of course, as a result, some columns may not have the same names as the original fields in `x`.

## Value

A data.frame containing one field per aspect of data in `x` - see Details. Each row corresponds to a feature (i.e., row) of `x`.

## Author(s)

Aaron Lun

## See Also

[makePerCellDF](#), for the cell-level equivalent.

## Examples

```
example_sce <- mockSCE()
example_sce <- logNormCounts(example_sce)
rowData(example_sce)$Length <- runif(nrow(example_sce))

df <- makePerFeatureDF(example_sce, cells="Cell_001")
head(df)
```

---

medianSizeFactors      *Compute median-based size factors*

---

### Description

Define per-cell size factors by taking the median of ratios to a reference expression profile (a la **DESeq**).

### Usage

```
medianSizeFactors(x, ...)

## S4 method for signature 'ANY'
medianSizeFactors(x, subset.row = NULL, reference = NULL, subset_row = NULL)

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

computeMedianFactors(x, ...)
```

### Arguments

x	For medianSizeFactors, a numeric matrix of counts with one row per feature and column per cell. Alternatively, a <a href="#">SummarizedExperiment</a> or <a href="#">SingleCellExperiment</a> containing such counts. For computeMedianFactors, only a <a href="#">SingleCellExperiment</a> is accepted.
...	For the medianSizeFactors generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For computeMedianFactors, further arguments to pass to medianSizeFactors.
subset.row	A vector specifying whether the size factors should be computed from a subset of rows of x.
reference	A numeric vector of length equal to nrow(x), containing the reference expression profile. Defaults to <a href="#">rowMeans</a> (x).
subset_row, exprs_values	Soft-deprecated equivalent to the arguments above.
assay.type	String or integer scalar indicating the assay of x containing the counts.

### Details

This function implements a modified version of the **DESeq2** size factor calculation. For each cell, the size factor is proportional to the median of the ratios of that cell's counts to reference. The assumption is that most genes are not DE between the cell and the reference, such that the median captures any systematic increase due to technical biases.

The modification stems from the fact that we use the arithmetic mean instead of the geometric mean to compute the default reference, as the former is more robust to the many zeros in single-cell RNA sequencing data. We also ignore all genes with values of zero in reference, as this usually results in undefined ratios when reference is itself computed from x.

**Value**

For `medianSizeFactors`, a numeric vector of size factors is returned for all methods.

For `computeMedianFactors`, `x` is returned containing the size factors in `sizeFactors(x)`.

**Caveats**

For typical scRNA-seq datasets, the median-based approach tends to perform poorly, for various reasons:

- The high number of zeroes in the count matrix means that the median ratio for each cell is often zero. If this method must be used, we recommend subsetting to only the highest-abundance genes to avoid problems with zeroes. (Of course, the smaller the subset, the more sensitive the results are to noise or violations of the non-DE majority.)
- The default reference effectively requires a non-DE majority of genes between *any* pair of cells in the dataset. This is a strong assumption for heterogeneous populations containing many cell types; most genes are likely to exhibit DE between at least one pair of cell types.

For these reasons, the simpler `librarySizeFactors` is usually preferred, which is no less inaccurate but is at least guaranteed to return a positive size factor for any cell with non-zero counts.

One valid application of this method lies in the normalization of antibody-derived tag counts for quantifying surface proteins. These counts are usually large enough to avoid zeroes yet are also susceptible to strong composition biases that preclude the use of `librarySizeFactors`. In such cases, we would also set reference to some estimate of the ambient profile. This assumes that most proteins are not expressed in each cell; thus, counts for most tags for any given cell can be attributed to background contamination that should not be DE between cells.

**Author(s)**

Aaron Lun

**See Also**

`normalizeCounts` and `logNormCounts`, where these size factors can be used.

`librarySizeFactors` and `geometricSizeFactors` for other simple methods for computing size factors.

**Examples**

```
example_sce <- mockSCE()
summary(medianSizeFactors(example_sce))
```

---

mockSCE

*Mock up a SingleCellExperiment*

---

**Description**

Mock up a `SingleCellExperiment` containing simulated data, for use in documentation examples.

**Usage**

```
mockSCE(ncells = 200, ngenes = 2000, nspikes = 100)
```

**Arguments**

nCells	Integer scalar, number of cells to simulate.
nGenes	Integer scalar, number of genes to simulate.
nSpikes	Integer scalar, number of spike-in transcripts to simulate.

**Details**

Users should set a seed to obtain reproducible results from this function.

**Value**

A SingleCellExperiment object containing a count matrix in the "counts" assay, a set of simulated `colData` fields, and spike-in data in the "Spikes" field of `altExps`.

**Author(s)**

Aaron Lun

**See Also**

[SingleCellExperiment](#), for the constructor.

**Examples**

```
set.seed(1000)
sce <- mockSCE()
sce
```

---

normalizeCounts

*Compute normalized expression values*

---

**Description**

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

**Usage**

```
normalizeCounts(x, ...)

## S4 method for signature 'ANY'
normalizeCounts(
  x,
  size.factors = NULL,
  log = TRUE,
  pseudo.count = 1,
  center.size.factors = TRUE,
  subset.row = NULL,
  downsample = FALSE,
  down.target = NULL,
```



```

    down.prop = 0.01,
    BPPARAM = SerialParam(),
    size_factors = NULL,
    pseudo_count = NULL,
    center_size_factors = NULL,
    subset_row = NULL,
    down_target = NULL,
    down_prop = NULL
)

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

## S4 method for signature 'SingleCellExperiment'
normalizeCounts(x, size.factors = NULL, ...)

```

## Arguments

x	A numeric matrix-like object containing counts for cells in the columns and features in the rows. Alternatively, a <a href="#">SingleCellExperiment</a> or <a href="#">SummarizedExperiment</a> object containing such a count matrix.
...	For the generic, arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY or <a href="#">DelayedMatrix</a> methods. For the SingleCellExperiment method, further arguments to pass to the SummarizedExperiment method.
size.factors	A numeric vector of cell-specific size factors. Alternatively NULL, in which case the size factors are extracted or computed from x.
log	Logical scalar indicating whether normalized values should be log2-transformed.
pseudo.count	Numeric scalar specifying the pseudo-count to add when log-transforming expression values.
center.size.factors	Logical scalar indicating whether size factors should be centered at unity before being used.
subset.row	A vector specifying the subset of rows of x for which to return a result.
downsample	Logical scalar indicating whether downsampling should be performed prior to scaling and log-transformation.
down.target	Numeric scalar specifying the downsampling target when downsample=TRUE. If NULL, this is defined by down.prop and a warning is emitted.
down.prop	Numeric scalar between 0 and 1 indicating the quantile to use to define the downsampling target. Only used when downsample=TRUE.
BPPARAM	A <a href="#">BiocParallelParam</a> object specifying how library size factor calculations should be parallelized. Only used if size.factors is not specified.
assay.type	A string or integer scalar specifying the assay of x containing the count matrix.
exprs_values, size_factors, pseudo_count, center_size_factors, subset_row, down_target, down_prop	Soft-deprecated equivalents to the arguments described previously.

## Details

Normalized expression values are computed by dividing the counts for each cell by the size factor for that cell. This removes cell-specific scaling biases due to differences in sequencing coverage, capture efficiency or total RNA content. If `log=TRUE`, log-normalized values are calculated by adding `pseudo.count` to the normalized count and performing a log<sub>2</sub>-transformation.

If no size factors are supplied, they are determined automatically from `x`:

- For count matrices and [SummarizedExperiment](#) inputs, the sum of counts for each cell is used to compute a size factor via the `librarySizeFactors` function.
- For [SingleCellExperiment](#) instances, the function searches for `sizeFactors` from `x`. If none are available, it defaults to library size-derived size factors.

If `size.factors` are supplied, they will override any size factors present in `x`.

## Value

A numeric matrix-like object containing (log-)normalized expression values. This has the same dimensions as `x` (unless `subset.row` is specified) and is of the same class as the original count matrix.

## Centering the size factors

If `center.size.factors=TRUE`, size factors are centred at unity prior to calculation of normalized expression values. This ensures that the computed expression values can be interpreted as being on the same scale as original counts. We can then compare abundances between features normalized with different sets of size factors; the most common use of this fact is in the comparison between spike-in and endogenous abundances when modelling technical noise (see [modelGeneVarWithSpikes](#) package for an example).

More generally, when `log=TRUE`, centering of the size factors ensures that the value of `pseudo.count` can be interpreted as being on the same scale as the counts, i.e., the pseudo-count can actually be thought of as a *count*. This is important as it implies that the pseudo-count's impact will diminish as sequencing coverage improves. Thus, if the size factors are centered, differences between log-normalized expression values will more closely approximate the true log-fold change with increasing coverage, whereas this would not be true of other metrics like log-CPMs with a fixed offset.

The disadvantage of using centered size factors is that the expression values are not directly comparable across different calls to `normalizeCounts`, typically for multiple batches. In theory, this is not a problem for metrics like the CPM, but in practice, we have to apply batch correction methods anyway to perform any joint analysis - see [multiBatchNorm](#) for more details.

## Downsampling instead of scaling

If `downsample=TRUE`, counts for each cell are randomly downsampled instead of being scaled. This is occasionally useful for avoiding artifacts caused by scaling count data with a strong mean-variance relationship. Each cell is downsampled according to the ratio between `down.target` and that cell's size factor. (Cells with size factors below the target are not downsampled and are directly scaled by this ratio.) If `log=TRUE`, a log-transformation is also performed after adding `pseudo.count` to the downsampled counts.

We automatically set `down.target` to the 1st percentile of size factors across all cells involved in the analysis, but this is only appropriate if the resulting expression values are not compared across different `normalizeCounts` calls. To obtain expression values that are comparable across different `normalizeCounts` calls (e.g., in [modelGeneVarWithSpikes](#) or [multiBatchNorm](#)), `down.target`

should be manually set to a constant target value that can be considered a low size factor in every call.

**Author(s)**

Aaron Lun

**See Also**

[logNormCounts](#), which wraps this function for convenient use with `SingleCellExperiment` instances.  
[librarySizeFactors](#), to compute the default size factors.  
[downsampleMatrix](#), to perform the downsampling.

**Examples**

```
example_sce <- mockSCE()

# Standard scaling + log-transformation:
normed <- normalizeCounts(example_sce)
normed[1:5,1:5]

# Scaling without transformation:
normed <- normalizeCounts(example_sce, log=FALSE)
normed[1:5,1:5]

# Downscaling with transformation:
normed <- normalizeCounts(example_sce, downsample=TRUE)
normed[1:5,1:5]

# Using custom size factors:
with.meds <- computeMedianFactors(example_sce)
normed <- normalizeCounts(with.meds)
normed[1:5,1:5]
```

---

numDetectedAcrossCells

*Number of detected expression values per group of cells*

---

**Description**

Computes the number of detected expression values (by default, defined as non-zero counts) for each feature in each group of cells. This function is deprecated: use [summarizeAssayByGroup](#) instead.

**Usage**

```
numDetectedAcrossCells(x, ...)

## S4 method for signature 'ANY'
numDetectedAcrossCells(
  x,
```

```

  ids,
  subset.row = NULL,
  subset.col = NULL,
  store.number = "ncells",
  average = FALSE,
  threshold = 0,
  BPPARAM = SerialParam(),
  subset_row = NULL,
  subset_col = NULL,
  store_number = NULL,
  detection_limit = NULL
)

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

```

### Arguments

x	A numeric matrix of counts containing features in rows and cells in columns. Alternatively, a <a href="#">SummarizedExperiment</a> object containing such a count matrix.
...	For the generic, further arguments to pass to specific methods. For the <a href="#">SummarizedExperiment</a> method, further arguments to pass to the ANY method.
ids	A factor specifying the group to which each cell in x belongs. Alternatively, a <a href="#">DataFrame</a> of such vectors or factors, in which case each unique combination of levels defines a group.
subset.row	An integer, logical or character vector specifying the features to use. If NULL, defaults to all features. For the <a href="#">SingleCellExperiment</a> method, this argument will not affect alternative Experiments, where aggregation is always performed for all features (or not at all, depending on <code>use_alt_exps</code> ).
subset.col	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of ids.
store.number	String specifying the field of the output <a href="#">colData</a> to store the number of cells in each group. If NULL, nothing is stored.
average	Logical scalar indicating whether the proportion of non-zero counts in each group should be computed instead.
threshold	A numeric scalar specifying the threshold above which a gene is considered to be detected.
BPPARAM	A <a href="#">BiocParallelParam</a> object specifying whether summation should be parallelized.
subset_row	Soft-deprecated equivalents to the arguments described above.
subset_col	Soft-deprecated equivalents to the arguments described above.
store_number	Soft-deprecated equivalents to the arguments described above.
assay.type	A string or integer scalar specifying the assay of x containing the matrix of counts (or any other expression quantity that can be meaningfully summed).
exprs_values, detection_limit	Soft-deprecated equivalents of the arguments above.

**Value**

A SummarizedExperiment is returned containing a count matrix in the first assay. Each column corresponds to group as defined by a unique level or combination of levels in `ids`. Each entry of the matrix contains the number of cells with detected expression for a feature and group. The identities of the levels for each column are reported in the `colData`. If `average=TRUE`, the assay is instead a numeric matrix containing the proportion of detected values.

**Author(s)**

Aaron Lun

**See Also**

[sumCountsAcrossCells](#), which computes the sum of counts within a group.

**Examples**

```
example_sce <- mockSCE()

ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)
bycol <- numDetectedAcrossCells(example_sce, ids)
head(bycol)
```

---

numDetectedAcrossFeatures

*Number of detected expression values per group of features*

---

**Description**

Computes the number of detected expression values (by default, defined as non-zero counts) for each group of features in each cell.

**Usage**

```
numDetectedAcrossFeatures(x, ...)

## S4 method for signature 'ANY'
numDetectedAcrossFeatures(
  x,
  ids,
  subset.row = NULL,
  subset.col = NULL,
  average = FALSE,
  threshold = 0,
  BPPARAM = SerialParam(),
  subset_row = NULL,
  subset_col = NULL,
  detection_limit = NULL
)

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

**Arguments**

x	A numeric matrix of counts containing features in rows and cells in columns. Alternatively, a <a href="#">SummarizedExperiment</a> object containing such a count matrix.
...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
ids	A factor of length nrow(x), specifying the set to which each feature in x belongs. Alternatively, a list of integer or character vectors, where each vector specifies the indices or names of features in a set. Logical vectors are also supported.
subset.row	An integer, logical or character vector specifying the features to use. Defaults to all features.
subset.col	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of ids.
average	Logical scalar indicating whether the proportion of non-zero counts in each group should be computed instead.
threshold	A numeric scalar specifying the threshold above which a gene is considered to be detected.
BPPARAM	A <a href="#">BiocParallelParam</a> object specifying whether summation should be parallelized.
subset_row	Soft-deprecated equivalents of the arguments described above.
subset_col	Soft-deprecated equivalents of the arguments described above.
assay.type	A string or integer scalar specifying the assay of x containing the matrix of counts (or any other expression quantity that can be meaningfully summed).
exprs_values, detection_limit	Soft-deprecated equivalents of the arguments above.

**Value**

An integer matrix containing the number of detected expression values in each group of features (row) and cell (column). If average=TRUE, this is instead a numeric matrix containing the proportion of detected values.

**Author(s)**

Aaron Lun

**See Also**

[sumCountsAcrossFeatures](#), on which this function is based.

**Examples**

```
example_sce <- mockSCE()

ids <- sample(paste0("GENE_", 1:100), nrow(example_sce), replace=TRUE)
byrow <- numDetectedAcrossFeatures(example_sce, ids)
head(byrow[,1:10])
```

---

perCellQCMetrics      *Per-cell quality control metrics*


---

### Description

Compute per-cell quality control metrics for a count matrix or a [SingleCellExperiment](#).

### Usage

```
perCellQCMetrics(x, ...)

## S4 method for signature 'ANY'
perCellQCMetrics(
  x,
  subsets = NULL,
  percent.top = integer(0),
  threshold = 0,
  BPPARAM = SerialParam(),
  flatten = TRUE,
  percent_top = NULL,
  detection_limit = NULL
)

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

## S4 method for signature 'SingleCellExperiment'
perCellQCMetrics(
  x,
  subsets = NULL,
  percent.top = integer(0),
  ...,
  flatten = TRUE,
  assay.type = "counts",
  use.altexps = TRUE,
  percent_top = NULL,
  exprs_values = NULL,
  use_altexps = NULL
)
```

### Arguments

x	A numeric matrix of counts with cells in columns and features in rows. Alternatively, a <a href="#">SummarizedExperiment</a> or <a href="#">SingleCellExperiment</a> object containing such a matrix.
...	For the generic, further arguments to pass to specific methods. For the <a href="#">SummarizedExperiment</a> and <a href="#">SingleCellExperiment</a> methods, further arguments to pass to the ANY method.
subsets	A named list containing one or more vectors (a character vector of feature names, a logical vector, or a numeric vector of indices), used to identify interesting feature subsets such as ERCC spike-in transcripts or mitochondrial genes.

<code>percent.top</code>	An integer vector specifying the size(s) of the top set of high-abundance genes. Used to compute the percentage of library size occupied by the most highly expressed genes in each cell.
<code>threshold</code>	A numeric scalar specifying the threshold above which a gene is considered to be detected.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying how parallelization should be performed.
<code>flatten</code>	Logical scalar indicating whether the nested <a href="#">DataFrames</a> in the output should be flattened.
<code>percent_top, detection_limit, exprs_values, use_altexprs</code>	Soft deprecated equivalents to the arguments described above.
<code>assay.type</code>	A string or integer scalar indicating which assays in the <code>x</code> contains the count matrix.
<code>use.altexprs</code>	Logical scalar indicating whether QC statistics should be computed for alternative Experiments in <code>x</code> . If <code>TRUE</code> , statistics are computed for all alternative experiments.  Alternatively, an integer or character vector specifying the alternative Experiments to use to compute QC statistics.  Alternatively <code>NULL</code> , in which case alternative experiments are not used.

## Details

This function calculates useful QC metrics for identification and removal of potentially problematic cells. Obvious per-cell metrics are the sum of counts (i.e., the library size) and the number of detected features. The percentage of counts in the top features also provides a measure of library complexity.

If `subsets` is specified, these statistics are also computed for each subset of features. This is useful for investigating gene sets of interest, e.g., mitochondrial genes, Y chromosome genes. These statistics are stored as nested [DataFrames](#) in the `subsets` field of the output. For example, if the input `subsets` contained "Mito" and "Sex", the output would look like:

```
output
|-- sum
|-- detected
|-- percent.top
+-- subsets
  |-- Mito
  |   |-- sum
  |   |-- detected
  |   +-- percent
  +-- Sex
      |-- sum
      |-- detected
      +-- percent
```

Here, the `percent` field contains the percentage of each cell's count sum assigned to each subset.

If `use.altexprs=TRUE`, the same statistics are computed for each alternative experiment in `x`. This can also be an integer or character vector specifying the alternative Experiments to use. These statistics are also stored as nested [DataFrames](#), this time in the `altexprs` field of the output. For example, if `x` contained the alternative Experiments "Spike" and "Ab", the output would look like:



```

output
|-- sum
|-- detected
|-- percent.top
+-- altexps
|   |-- Spike
|   |   |-- sum
|   |   |-- detected
|   |   +-- percent.total
|   +-- Ab
|       |-- sum
|       |-- detected
|       +-- percent.total
+-- total

```

The total field contains the total sum of counts for each cell across the main and alternative Experiments. The percent field contains the percentage of the total count in each alternative Experiment for each cell.

If `flatten=TRUE`, the nested DataFrames are flattened by concatenating the column names with underscores. This means that, say, the `subsets$Mito$sum` nested field becomes the top-level `subsets_Mito_sum` field. A flattened structure is more convenient for end-users performing interactive analyses, but less convenient for programmatic access as artificial construction of strings is required.

## Value

A [DataFrame](#) of QC statistics where each row corresponds to a column in `x`. This contains the following fields:

- `sum`: numeric, the sum of counts for each cell.
- `detected`: numeric, the number of observations above threshold.

If `flatten=FALSE`, the DataFrame will contain the additional columns:

- `percent.top`: numeric matrix, the percentage of counts assigned to the top most highly expressed genes. Each column of the matrix corresponds to an entry of `percent.top`, sorted in increasing order.
- `subsets`: A nested DataFrame containing statistics for each subset, see [Details](#).
- `altexps`: A nested DataFrame containing statistics for each alternative experiment, see [Details](#). This is only returned for the `SingleCellExperiment` method.
- `total`: numeric, the total sum of counts for each cell across main and alternative Experiments. This is only returned for the `SingleCellExperiment` method.

If `flatten=TRUE`, nested matrices and DataFrames are flattened to remove the hierarchical structure from the output DataFrame.

## Author(s)

Aaron Lun

## See Also

[addPerCellQC](#), to add the QC metrics to the column metadata.

**Examples**

```

example_sce <- mockSCE()
stats <- perCellQCMetrics(example_sce)
stats

# With subsets.
stats2 <- perCellQCMetrics(example_sce, subsets=list(Mito=1:10),
  flatten=FALSE)
stats2$subsets

# With alternative Experiments.
pretend.spike <- ifelse(seq_len(nrow(example_sce)) < 10, "Spike", "Gene")
alt_sce <- splitAltExps(example_sce, pretend.spike)
stats3 <- perCellQCMetrics(alt_sce, flatten=FALSE)
stats3$altexps

```

---

perFeatureQCMetrics    *Per-feature quality control metrics*

---

**Description**

Compute per-feature quality control metrics for a count matrix or a [SummarizedExperiment](#).

**Usage**

```

perFeatureQCMetrics(x, ...)

## S4 method for signature 'ANY'
perFeatureQCMetrics(
  x,
  subsets = NULL,
  threshold = 0,
  BPPARAM = SerialParam(),
  flatten = TRUE,
  detection_limit = NULL
)

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

```

**Arguments**

**x**                    A numeric matrix of counts with cells in columns and features in rows. Alternatively, a [SummarizedExperiment](#) or [SingleCellExperiment](#) object containing such a matrix.

**...**                For the generic, further arguments to pass to specific methods. For the [SummarizedExperiment](#) and [SingleCellExperiment](#) methods, further arguments to pass to the ANY method.

subsets	A named list containing one or more vectors (a character vector of cell names, a logical vector, or a numeric vector of indices), used to identify interesting sample subsets such as negative control wells.
threshold	A numeric scalar specifying the threshold above which a gene is considered to be detected.
BPPARAM	A <a href="#">BiocParallelParam</a> object specifying how parallelization should be performed.
flatten	Logical scalar indicating whether the nested <a href="#">DataFrames</a> in the output should be flattened.
detection_limit, exprs_values	Soft deprecated equivalents to the arguments described above.
assay.type	A string or integer scalar indicating which assays in the x contains the count matrix.

## Details

This function calculates useful QC metrics for features, including the mean across all cells and the number of expressed features (i.e., counts above the detection limit).

If `subsets` is specified, the same statistics are computed for each subset of cells. This is useful for obtaining statistics for cell sets of interest, e.g., negative control wells. These statistics are stored as nested [DataFrames](#) in the output. For example, if `subsets` contained "empty" and "cellpool", the output would look like:

```
output
|-- mean
|-- detected
+-- subsets
  |-- empty
  |   |-- mean
  |   |-- detected
  |   +-- ratio
  +-- cellpool
      |-- mean
      |-- detected
      +-- ratio
```

The `ratio` field contains the ratio of the mean within each subset to the mean across all cells.

If `flatten=TRUE`, the nested [DataFrames](#) are flattened by concatenating the column names with underscores. This means that, say, the `subsets$empty$mean` nested field becomes the top-level `subsets_empty_mean` field. A flattened structure is more convenient for end-users performing interactive analyses, but less convenient for programmatic access as artificial construction of strings is required.

## Value

A [DataFrame](#) of QC statistics where each row corresponds to a row in `x`. This contains the following fields:

- `mean`: numeric, the mean counts for each feature.
- `detected`: numeric, the percentage of observations above threshold.

If `flatten=FALSE`, the output [DataFrame](#) also contains the `subsets` field. This a nested [DataFrame](#) containing per-feature QC statistics for each subset of columns.

If `flatten=TRUE`, `subsets` is flattened to remove the hierarchical structure.

**Author(s)**

Aaron Lun

**See Also**[addPerFeatureQC](#), to add the QC metrics to the row metadata.**Examples**

```
example_sce <- mockSCE()
stats <- perFeatureQCMetrics(example_sce)
stats

# With subsets.
stats2 <- perFeatureQCMetrics(example_sce, subsets=list(Empty=1:10))
stats2
```

---

`quickPerCellQC`*Quick cell-level QC*

---

**Description**

A convenient utility that identifies low-quality cells based on frequently used QC metrics.

**Usage**

```
quickPerCellQC(x, ...)

## S4 method for signature 'ANY'
quickPerCellQC(
  x,
  sum.field = "sum",
  detected.field = "detected",
  sub.fields = NULL,
  ...,
  lib_size = NULL,
  n_features = NULL,
  percent_subsets = NULL
)

## S4 method for signature 'SummarizedExperiment'
quickPerCellQC(
  x,
  ...,
  subsets = NULL,
  assay.type = "counts",
  other.args = list()
)
```

**Arguments**

<code>x</code>	A <a href="#">DataFrame</a> containing per-cell QC statistics, as computed by <a href="#">perCellQCMetrics</a> . Alternatively, a <a href="#">SummarizedExperiment</a> object that can be used to create such a DataFrame via <a href="#">perCellQCMetrics</a> .
<code>...</code>	For the generic, further arguments to pass to specific methods. For the ANY method, further arguments to pass to <a href="#">isOutlier</a> . For the SummarizedExperiment method, further arguments to pass to the ANY method.
<code>sum.field</code>	String specifying the column of <code>x</code> containing the library size for each cell.
<code>detected.field</code>	String specifying the column of <code>x</code> containing the number of detected features per cell.
<code>sub.fields</code>	Character vector specifying the column(s) of <code>x</code> containing the percentage of counts in subsets of “control features”, usually mitochondrial genes or spike-in transcripts. If set to TRUE, this will default to all columns in <code>x</code> with names following the patterns “subsets.*_percent” and “altexps.*_percent”.
<code>lib_size, n_features, percent_subsets</code>	Soft-deprecated equivalents of the arguments above.
<code>subsets, assay.type</code>	Arguments to pass to the <a href="#">perCellQCMetrics</a> function, exposed here for convenience.
<code>other.args</code>	A named list containing other arguments to pass to the <a href="#">perCellQCMetrics</a> function.

**Details**

The ANY method simply calls [isOutlier](#) on the various QC metrics in `x`.

- For `sum.field`, small outliers are detected. These are considered to represent low-quality cells that have not been insufficiently sequenced. Detection is performed on the log-scale to adjust for a heavy right tail and to improve resolution at zero.
- For `detected.field`, small outliers are detected. These are considered to represent low-quality cells with low-complexity libraries. Detection is performed on the log-scale to adjust for a heavy right tail. This is done on the log-scale to adjust for a heavy right tail and to improve resolution at zero.
- For each column specified by `sub.fields`, large outliers are detected. This aims to remove cells with high spike-in or mitochondrial content, usually corresponding to damaged cells. While these distributions often have heavy right tails, the putative low-quality cells are often present in this tail; thus, transformation is not performed to ensure maintain resolution of the filter.

Users can control the outlier detection (e.g., change the number of MADs, specify batches) by passing appropriate arguments to `...`

The SummarizedExperiment method calls [perCellQCMetrics](#) before running outlier detection. This is simply a convenient wrapper that avoids the need for two separate function calls in routine applications.

**Value**

A [DataFrame](#) with one row per cell and containing columns of logical vectors. Each column specifies a reason for why a cell was considered to be low quality, with the final discard column indicating whether the cell should be discarded.

**Author(s)**

Aaron Lun

**See Also**

[perCellQCMetrics](#), for calculation of these metrics.  
[isOutlier](#), to identify outliers with a MAD-based approach.

**Examples**

```
example_sce <- mockSCE()
x <- perCellQCMetrics(example_sce, subsets=list(Mito=1:100))

discarded <- quickPerCellQC(x, percent_subsets=c(
  "subsets_Mito_percent", "altexps_Spikes_percent"))
colSums(as.data.frame(discarded))
```

---

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

---

**Description**

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

**Usage**

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

**Arguments**

<code>file</code>	A string containing a file path to a count table, or a connection object opened in read-only text mode.
<code>sep</code>	A string specifying the delimiter between fields in <code>file</code> .
<code>quote</code>	A string specifying the quote character, e.g., in column or row names.
<code>comment.char</code>	A string specifying the comment character after which values are ignored.
<code>row.names</code>	A logical scalar specifying whether row names are present.
<code>col.names</code>	A logical scalar specifying whether column names are present.

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

### Details

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

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

### Value

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

### Author(s)

Aaron Lun

### See Also

[read.table](#), [readMM](#)

### Examples

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

readSparseCounts(outfile)
```

**Description**

The **scuttle** package provides some utility functions for single-cell 'omics data analysis. This includes some simple methods for computing and filtering on quality control; basic data transformations involving various types of scaling normalization; and flexible aggregation across cells or features.

**scuttle** also implements wrapper functions that simplify boilerplate for developers of client packages. This includes packages such as **scran**, **scater** and **DropletUtils**, to name a few. Note that much of the code here was inherited from the **scater** package.

**Author(s)**

Aaron Lun

---

scuttle-utils      *Developer utilities*

---

**Description**

Various utilities for re-use in packages that happen to depend on **scuttle**. These are exported simply to avoid re-writing them in downstream packages, and should not be touched by end-users.

**Author(s)**

Aaron Lun

---

sumCountsAcrossCells      *Sum expression across groups of cells*

---

**Description**

Sum counts or average expression values for each feature across groups of cells. This function is deprecated; use [summarizeAssayByGroup](#) instead.

**Usage**

```
sumCountsAcrossCells(x, ...)

## S4 method for signature 'ANY'
sumCountsAcrossCells(
  x,
  ids,
  subset.row = NULL,
  subset.col = NULL,
  store.number = "ncells",
  average = FALSE,
  BPPARAM = SerialParam(),
  subset_row = NULL,
  subset_col = NULL,
  store_number = NULL
```



```
)

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

### Arguments

<code>x</code>	A numeric matrix of expression values (usually counts) containing features in rows and cells in columns. Alternatively, a <a href="#">SummarizedExperiment</a> object containing such a matrix.
<code>...</code>	For the generics, further arguments to be passed to specific methods. For the <a href="#">SummarizedExperiment</a> method, further arguments to be passed to the ANY method.
<code>ids</code>	A factor specifying the group to which each cell in <code>x</code> belongs. Alternatively, a <a href="#">DataFrame</a> of such vectors or factors, in which case each unique combination of levels defines a group.
<code>subset.row</code>	An integer, logical or character vector specifying the features to use. If NULL, defaults to all features. For the <a href="#">SingleCellExperiment</a> method, this argument will not affect alternative Experiments, where aggregation is always performed for all features (or not at all, depending on <code>use_alt_exps</code> ).
<code>subset.col</code>	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of <code>ids</code> .
<code>store.number</code>	String specifying the field of the output <a href="#">colData</a> to store the number of cells in each group. If NULL, nothing is stored.
<code>average</code>	Logical scalar indicating whether the average should be computed instead of the sum. Alternatively, a string containing "mean", "median" or "none", specifying the type of average. ("none" is equivalent to FALSE.)
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying whether summation should be parallelized.
<code>subset_row, subset_col, exprs_values, store_number</code>	Soft-deprecated equivalents to the arguments described above.
<code>assay.type</code>	A string or integer scalar specifying the assay of <code>x</code> containing the matrix of counts (or any other expression quantity that can be meaningfully summed).

### Details

These functions provide a convenient method for summing or averaging expression values across multiple columns for each feature. A typical application would be to sum counts across all cells in each cluster to obtain “pseudo-bulk” samples for further analyses, e.g., differential expression analyses between conditions.

The behaviour of `sumCountsAcrossCells` is equivalent to that of `colsum`. However, this function can operate on any matrix representation in object; can do so in a parallelized manner for large matrices without resorting to block processing; and can natively support combinations of multiple factors in `ids`.

Any NA values in `ids` are implicitly ignored and will not be considered during summation. This may be useful for removing undesirable cells by setting their entries in `ids` to NA. Alternatively, we can explicitly select the cells of interest with `subset_col`.

Setting `average=TRUE` will compute the average in each set rather than the sum. This is particularly useful if `x` contains expression values that have already been normalized in some manner, as computing the average avoids another round of normalization to account for differences in the size of

each set. The same effect is obtained by setting `average="mean"`, while setting `average="median"` will instead compute the median across all cells.

### Value

A `SummarizedExperiment` is returned with one column per level of `ids`. Each entry of the assay contains the sum or average across all cells in a given group (column) for a given feature (row). Columns are ordered by `levels(ids)` and the number of cells per level is reported in the `"ncells"` column metadata. For `DataFrame` `ids`, each column corresponds to a unique combination of levels (recorded in the `colData`).

### Author(s)

Aaron Lun

### See Also

[aggregateAcrossCells](#), which also combines information in the `colData`.

[numDetectedAcrossCells](#), which computes the number of cells with detected expression in each group.

### Examples

```
example_sce <- mockSCE()
ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)

out <- sumCountsAcrossCells(example_sce, ids)
head(out)

batches <- sample(1:3, ncol(example_sce), replace=TRUE)
out2 <- sumCountsAcrossCells(example_sce,
  DataFrame(label=ids, batch=batches))
head(out2)
```

---

sumCountsAcrossFeatures

*Sum counts across feature sets*

---

### Description

Sum together expression values (by default, counts) for each feature set in each cell.

### Usage

```
sumCountsAcrossFeatures(x, ...)

## S4 method for signature 'ANY'
sumCountsAcrossFeatures(
  x,
  ids,
  subset.row = NULL,
  subset.col = NULL,
```

```

    average = FALSE,
    BPPARAM = SerialParam(),
    subset_row = NULL,
    subset_col = NULL
  )

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

```

## Arguments

<code>x</code>	A numeric matrix of counts containing features in rows and cells in columns. Alternatively, a <a href="#">SummarizedExperiment</a> object containing such a count matrix.
<code>...</code>	For the <code>sumCountsAcrossFeatures</code> generic, further arguments to be passed to specific methods. For the <code>SummarizedExperiment</code> method, further arguments to be passed to the ANY method.
<code>ids</code>	A factor of length <code>nrow(x)</code> , specifying the set to which each feature in <code>x</code> belongs. Alternatively, a list of integer or character vectors, where each vector specifies the indices or names of features in a set. Logical vectors are also supported.
<code>subset_row</code>	An integer, logical or character vector specifying the features to use. Defaults to all features.
<code>subset_col</code>	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of <code>ids</code> .
<code>average</code>	Logical scalar indicating whether the average should be computed instead of the sum.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying whether summation should be parallelized.
<code>subset_row</code> , <code>subset_col</code> , <code>exprs_values</code>	Soft-deprecated equivalents of the arguments described above.
<code>assay.type</code>	A string or integer scalar specifying the assay of <code>x</code> containing the matrix of counts (or any other expression quantity that can be meaningfully summed).

## Details

This function provides a convenient method for aggregating counts across multiple rows for each cell. Several possible applications are listed below:

- Using a list of genes in `ids`, we can obtain a summary expression value for all genes in one or more gene sets. This allows the activity of various pathways to be compared across cells.
- Genes with multiple mapping locations in the reference will often manifest as multiple rows with distinct Ensembl/Entrez IDs. These counts can be aggregated into a single feature by setting the shared identifier (usually the gene symbol) as `ids`.
- It is theoretically possible to aggregate transcript-level counts to gene-level counts with this function. However, it is often better to do so with dedicated functions (e.g., from the **tximport** or **tximeta** packages) that account for differences in length across isoforms.

The behaviour of this function is equivalent to that of `rowsum`. However, this function can operate on any matrix representation in object, and can do so in a parallelized manner for large matrices without resorting to block processing.

If `ids` is a factor, any NA values are implicitly ignored and will not be considered or reported. This may be useful, e.g., to remove undesirable feature sets by setting their entries in `ids` to NA.

Setting `average=TRUE` will compute the average in each set rather than the sum. This is particularly useful if `x` contains expression values that have already been normalized in some manner, as computing the average avoids another round of normalization to account for differences in the size of each set.

### Value

A count matrix is returned with one row per level of `ids`. In each cell, counts for all features in the same set are summed together (or averaged, if `average=TRUE`). Rows are ordered according to `levels(ids)`.

### Author(s)

Aaron Lun

### See Also

[aggregateAcrossFeatures](#), to perform additional aggregation of row-level metadata.

[numDetectedAcrossFeatures](#), to compute the number of detected features per cell.

### Examples

```
example_sce <- mockSCE()
ids <- sample(LETTERS, nrow(example_sce), replace=TRUE)
out <- sumCountsAcrossFeatures(example_sce, ids)
str(out)
```

---

summarizeAssayByGroup *Summarize an assay by group*

---

### Description

From an assay matrix, compute summary statistics for groups of cells. A typical example would be to compute various summary statistics for clusters.

### Usage

```
summarizeAssayByGroup(x, ...)

## S4 method for signature 'ANY'
summarizeAssayByGroup(
  x,
  ids,
  subset.row = NULL,
  subset.col = NULL,
  statistics = c("mean", "sum", "num.detected", "prop.detected", "median"),
  store.number = "ncells",
  threshold = 0,
  BPPARAM = SerialParam())
```

```
)

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

## Arguments

<code>x</code>	A numeric matrix containing features in rows and cells in columns. Alternatively, a <a href="#">SummarizedExperiment</a> object containing such a matrix.
<code>...</code>	For the generics, further arguments to be passed to specific methods. For the <code>SummarizedExperiment</code> method, further arguments to be passed to the ANY method.
<code>ids</code>	A factor (or vector coercible into a factor) specifying the group to which each cell in <code>x</code> belongs. Alternatively, a <a href="#">DataFrame</a> of such vectors or factors, in which case each unique combination of levels defines a group.
<code>subset.row</code>	An integer, logical or character vector specifying the features to use. If NULL, defaults to all features.
<code>subset.col</code>	An integer, logical or character vector specifying the cells to use. Defaults to all cells with non-NA entries of <code>ids</code> .
<code>statistics</code>	Character vector specifying the type of statistics to be computed, see Details.
<code>store.number</code>	String specifying the field of the output <code>colData</code> to store the number of cells in each group. If NULL, nothing is stored.
<code>threshold</code>	A numeric scalar specifying the threshold above which a gene is considered to be detected.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object specifying whether summation should be parallelized.
<code>assay.type</code>	A string or integer scalar specifying the assay of <code>x</code> containing the assay to be summarized.

## Details

These functions provide a convenient method for summing or averaging expression values across multiple columns for each feature. A typical application would be to sum counts across all cells in each cluster to obtain “pseudo-bulk” samples for further analyses, e.g., differential expression analyses between conditions.

For each feature, the chosen assay can be aggregated by:

- “sum”, the sum of all values in each group. This makes the most sense for raw counts, to allow models to account for the mean-variance relationship.
- “mean”, the mean of all values in each group. This makes the most sense for normalized and/or transformed assays.
- “median”, the median of all values in each group. This makes the most sense for normalized and/or transformed assays, usually generated from large counts where discreteness is less of an issue.
- “num.detected” and “prop.detected”, the number and proportion of values in each group that are non-zero. This makes the most sense for raw counts or sparsity-preserving transformations.

Any NA values in `ids` are implicitly ignored and will not be considered during summation. This may be useful for removing undesirable cells by setting their entries in `ids` to NA. Alternatively, we can explicitly select the cells of interest with `subset_col`.

If `ids` is a factor and contains unused levels, they will not be represented as columns in the output.

### Value

A `SummarizedExperiment` is returned with one column per level of `ids`. Each entry of the assay contains the sum or average across all cells in a given group (column) for a given feature (row). Columns are ordered by `levels(ids)` and the number of cells per level is reported in the "ncells" column metadata. For `DataFrame` `ids`, each column corresponds to a unique combination of levels (recorded in the `colData`).

### Author(s)

Aaron Lun

### See Also

[aggregateAcrossCells](#), which also combines information in the `colData` of `x`.

### Examples

```
example_sce <- mockSCE()
ids <- sample(LETTERS[1:5], ncol(example_sce), replace=TRUE)

out <- summarizeAssayByGroup(example_sce, ids)
out

batches <- sample(1:3, ncol(example_sce), replace=TRUE)
out2 <- summarizeAssayByGroup(example_sce,
  DataFrame(label=ids, batch=batches))
head(out2)
```

---

uniquifyFeatureNames *Make feature names unique*

---

### Description

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

### Usage

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

### Arguments

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

**Details**

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

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

**Value**

A character vector of unique-ified feature names.

**Author(s)**

Aaron Lun

**Examples**

```
uniquifyFeatureNames(
  ID=paste0("ENSG000000", 1:5),
  names=c("A", NA, "B", "C", "A")
)
```

---

whichNonZero

*Find non-zero entries of a matrix*

---

**Description**

Finds the non-zero entries of a matrix in the most efficient manner. Not sure there's much more to say here.

**Usage**

```
whichNonZero(x, ...)

## S4 method for signature 'dgTMatrix'
whichNonZero(x, ...)

## S4 method for signature 'dgCMatrix'
whichNonZero(x, ...)

## S4 method for signature 'ANY'
whichNonZero(x, BPPARAM = SerialParam(), ...)
```

**Arguments**

<code>x</code>	A numeric matrix-like object, usually sparse in content if not in representation.
<code>...</code>	For the generic, additional arguments to pass to the specific methods. For the methods, additional arguments that are currently ignored.
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> object controlling how parallelization should be performed. only used when <code>x</code> is a <a href="#">DelayedMatrix</a> object.

**Value**

A list containing *i*, an integer vector of the row indices of all non-zero entries; *j*, an integer vector of the column indices of all non-zero entries; and *x*, a numeric vector of the values of the non-zero entries.

**Author(s)**

Aaron Lun

**See Also**

[which](#), obviously.

**Examples**

```
x <- Matrix::rsparsematrix(1e6, 1e6, 0.000001)
out <- whichNonZero(x)
str(out)
```



# Index

- .assignIndicesToWorkers (scuttle-utils), 48
- .bpNotSharedOrUp (scuttle-utils), 48
- .checkCountMatrix (scuttle-utils), 48
- .ranksafeQR (scuttle-utils), 48
- .splitColsByWorkers (scuttle-utils), 48
- .splitRowsByWorkers (scuttle-utils), 48
- .splitVectorByWorkers (scuttle-utils), 48
- .subset2index (scuttle-utils), 48
- .unpackLists (scuttle-utils), 48
  
- addPerCellQC, 2, 41
- addPerFeatureQC, 44
- addPerFeatureQC (addPerCellQC), 2
- aggregateAcrossCells, 3, 50, 54
- aggregateAcrossCells, SingleCellExperiment-method (aggregateAcrossCells), 3
- aggregateAcrossCells, SummarizedExperiment-method (aggregateAcrossCells), 3
- aggregateAcrossFeatures, 7, 52
- altExp, 27
- altExps, 4, 6, 25, 32
  
- BiocParallelParam, 9, 17, 19, 22, 25, 33, 36, 38, 40, 43, 49, 51, 53, 55
  
- calculateAverage, 8
- calculateAverage, ANY-method (calculateAverage), 8
- calculateAverage, SingleCellExperiment-method (calculateAverage), 8
- calculateAverage, SummarizedExperiment-method (calculateAverage), 8
- calculateCPM, 9, 11–13
- calculateCPM, ANY-method (calculateCPM), 9
- calculateCPM, SingleCellExperiment-method (calculateCPM), 9
- calculateCPM, SummarizedExperiment-method (calculateCPM), 9
- calculateFPKM, 11, 13
- calculateTPM, 12
- calculateTPM, ANY-method (calculateTPM), 12
- calculateTPM, SingleCellExperiment-method (calculateTPM), 12
- calculateTPM, SummarizedExperiment-method (calculateTPM), 12
- colData, 3–5, 32, 36, 37, 49, 50, 53, 54
- colsum, 49
- computeGeometricFactors (geometricSizeFactors), 18
- computeLibraryFactors (librarySizeFactors), 22
- computeMedianFactors (medianSizeFactors), 30
  
- DataFrame, 4, 5, 36, 40, 41, 43, 45, 49, 53
- DelayedArray, 19, 22
- DelayedMatrix, 9, 16, 33, 55
- dgCMatix, 16
- downsampleBatches, 13
- downsampleMatrix, 14, 15, 15, 35
- downsampleReads, 16
  
- fitLinearModel, 17
  
- geometricSizeFactors, 18, 19, 23, 31
- geometricSizeFactors, ANY-method (geometricSizeFactors), 18
- geometricSizeFactors, SummarizedExperiment-method (geometricSizeFactors), 18
  
- isOutlier, 19, 45, 46
- librarySizeFactors, 9, 19, 22, 31, 34, 35
- librarySizeFactors, ANY-method (librarySizeFactors), 22
- librarySizeFactors, SummarizedExperiment-method (librarySizeFactors), 22
  
- List, 14
- lm. fit, 17
- logNormCounts, 9, 19, 23, 23, 31, 35
- logNormCounts, SingleCellExperiment-method (logNormCounts), 23
- logNormCounts, SummarizedExperiment-method (logNormCounts), 23

- make.names, [27, 29](#)
- makePerCellDF, [26, 29](#)
- makePerFeatureDF, [28, 28](#)
- median, [5](#)
- medianSizeFactors, [19, 23, 30](#)
- medianSizeFactors, ANY-method  
(medianSizeFactors), [30](#)
- medianSizeFactors, SummarizedExperiment-method  
(medianSizeFactors), [30](#)
- mockSCE, [31](#)
- modelGeneVarWithSpikes, [34](#)
- multiBatchNorm, [34](#)
  
- normalizeCounts, [11, 16, 19, 23–26, 31, 32, 34](#)
- normalizeCounts, ANY-method  
(normalizeCounts), [32](#)
- normalizeCounts, SingleCellExperiment-method  
(normalizeCounts), [32](#)
- normalizeCounts, SummarizedExperiment-method  
(normalizeCounts), [32](#)
- numDetectedAcrossCells, [35, 50](#)
- numDetectedAcrossCells, ANY-method  
(numDetectedAcrossCells), [35](#)
- numDetectedAcrossCells, SummarizedExperiment-method  
(numDetectedAcrossCells), [35](#)
- numDetectedAcrossFeatures, [37, 52](#)
- numDetectedAcrossFeatures, ANY-method  
(numDetectedAcrossFeatures), [37](#)
- numDetectedAcrossFeatures, SummarizedExperiment-method  
(numDetectedAcrossFeatures), [37](#)
  
- perCellQCMetrics, [3, 21, 39, 45, 46](#)
- perCellQCMetrics, ANY-method  
(perCellQCMetrics), [39](#)
- perCellQCMetrics, SingleCellExperiment-method  
(perCellQCMetrics), [39](#)
- perCellQCMetrics, SummarizedExperiment-method  
(perCellQCMetrics), [39](#)
- perFeatureQCMetrics, [3, 42](#)
- perFeatureQCMetrics, ANY-method  
(perFeatureQCMetrics), [42](#)
- perFeatureQCMetrics, SummarizedExperiment-method  
(perFeatureQCMetrics), [42](#)
  
- quickPerCellQC, [21, 44](#)
- quickPerCellQC, ANY-method  
(quickPerCellQC), [44](#)
- quickPerCellQC, SummarizedExperiment-method  
(quickPerCellQC), [44](#)
  
- read.table, [47](#)
- readMM, [47](#)
- readSparseCounts, [46](#)
- RealizationSink, [15](#)
- reducedDims, [4, 6](#)
- rowData, [3](#)
- rowMeans, [30](#)
- rowsum, [51](#)
- rowSums, [9](#)
- scuttle-pkg, [47](#)
- scuttle-utils, [48](#)
- SingleCellExperiment, [4, 6, 8–12, 18, 22–34, 36, 39, 42, 49](#)
- sizeFactors, [9, 19, 23, 25, 31, 34](#)
- sum, [5](#)
- sumCountsAcrossCells, [37, 48](#)
- sumCountsAcrossCells, ANY-method  
(sumCountsAcrossCells), [48](#)
- sumCountsAcrossCells, SummarizedExperiment-method  
(sumCountsAcrossCells), [48](#)
- sumCountsAcrossFeatures, [7, 38, 50](#)
- sumCountsAcrossFeatures, ANY-method  
(sumCountsAcrossFeatures), [50](#)
- sumCountsAcrossFeatures, SummarizedExperiment-method  
(sumCountsAcrossFeatures), [50](#)
- summarizeAssayByGroup, [4–6, 35, 48, 52](#)
- summarizeAssayByGroup, ANY-method  
(summarizeAssayByGroup), [52](#)
- summarizeAssayByGroup, SummarizedExperiment-method  
(summarizeAssayByGroup), [52](#)
- SummarizedExperiment, [2–5, 7–12, 14, 18, 22, 24, 25, 30, 33, 34, 36, 38, 39, 42, 45, 49, 51, 53](#)
- uniquifyFeatureNames, [54](#)
- which, [56](#)
- whichNonZero, [55](#)
- whichNonZero, ANY-method (whichNonZero),  
[55](#)
- whichNonZero, dgCMatrx-method  
(whichNonZero), [55](#)
- whichNonZero, dgTMatrix-method  
(whichNonZero), [55](#)