Package 'DropletUtils'

November 11, 2026

Version 1.32.1

Date 2026-09-25

Title Utilities for Handling Single-Cell Droplet Data

Depends SingleCellExperiment

Imports utils, stats, methods, Matrix, Rcpp, BiocGenerics, S4Vectors, IRanges, GenomicRanges, SummarizedExperiment, BiocParallel, SparseArray (>= 1.5.18), DelayedArray (>= 0.31.9), DelayedMatrixStats, HDF5Array, rhdf5, edgeR, R.utils, dqrng, beachmat, scuttle (>= 1.21.4)

Suggests testthat, knitr, BiocStyle, rmarkdown, jsonlite, DropletTestFiles

biocViews ImmunoOncology, SingleCell, Sequencing, RNASeq, GeneExpression, Transcriptomics, DataImport, Coverage

Description Provides a number of utility functions for handling single-cell (RNA-seq) data from droplet technologies such as 10X Genomics. This includes data loading from count matrices or molecule information files, identification of cells from empty droplets, removal of barcode-swapped pseudo-cells, and downsampling of the count matrix.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

LinkingTo Rcpp, beachmat, assorthead, Rhdf5lib, BH, dqrng, scuttle

SystemRequirements C++17, GNU make

RoxygenNote 7.3.2

Encoding UTF-8

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

git_branch RELEASE_3_23

git_last_commit ce41edd

git_last_commit_date 2026-09-24

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Aaron Lun [aut],
Jonathan Griffiths [ctb, cre],
Davis McCarthy [ctb],
Dongze He [ctb],
Rob Patro [ctb]

Maintainer Jonathan Griffiths <jonathan.griffiths.94@gmail.com>

Contents

ambientContribMaximum . . . 2
ambientContribNegative . . . 5
ambientContribSparse . . . 8
ambientProfileBimodal . . . 10
ambientProfileEmpty . . . 11
barcodeRanks . . . 14
chimericDrops . . . 17
cleanTagCounts . . . 19
defaultDrops . . . 21
downsampleReads . . . 23
emptyDrops . . . 25
emptyDropsCellRanger . . . 30
encodeSequences . . . 33
get10xMolInfoStats . . . 34
hashedDrops . . . 35
makeCountMatrix . . . 40
read10xCounts . . . 42
read10xMolInfo . . . 45
reexports . . . 47
removeAmbience . . . 48
swappedDrops . . . 50
write10xCounts . . . 53

Index **56**

`ambientContribMaximum` *Ambient contribution by maximum scaling*

Description

Compute the maximum contribution of the ambient solution to an expression profile for a group of droplets, by scaling the ambient profile and testing for significant deviations in the count profile.

Usage

```
maximumAmbience(...)

ambientContribMaximum(y, ...)

## S4 method for signature 'ANY'
ambientContribMaximum(
  y,
  ambient,
  threshold = 0.1,
  dispersion = 0,
  num.points = 100,
  num.iter = 5,
  mode = c("scale", "profile", "proportion"),
  BPPARAM = SerialParam()
)

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

Arguments

`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `controlAmbience`, arguments to pass to `ambientContribMaximum`.
`y`	A numeric matrix-like object containing counts, where each row represents a gene and each column represents a cluster of cells (see Caveats). Alternatively, a SummarizedExperiment object containing such a matrix. `y` can also be a numeric vector of counts; this is coerced into a one-column matrix.
`ambient`	A numeric vector of length equal to `nrow(y)`, containing the proportions of transcripts for each gene in the ambient solution. Alternatively, a matrix where each row corresponds to a row of `y` and each column contains a specific ambient profile for the corresponding column of `y`.
`threshold`	Numeric scalar specifying the p-value threshold to use, see Details.
`dispersion`	Numeric scalar specifying the dispersion to use in the negative binomial model. Defaults to zero, i.e., a Poisson model.
`num.points`	Integer scalar specifying the number of points to use for the grid search.
`num.iter`	Integer scalar specifying the number of iterations to use for the grid search.
`mode`	String indicating the output to return, see Value.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

On occasion, it is useful to estimate the maximum possible contribution of the ambient solution to a count profile. This represents the most pessimistic explanation of a particular expression pattern and can be used to identify and discard suspect genes or clusters prior to downstream analyses.

This function implements the following algorithm:

1. We compute the mean ambient contribution for each gene by scaling `ambient` by some factor. `ambient` itself is usually derived by summing counts across barcodes with low total counts, see the output of `emptyDrops` for an example.
2. We compute a p-value for each gene based on the probability of observing a count equal to or below that in `y`, using the lower tail of a negative binomial (or Poisson) distribution with mean set to the ambient contribution. The per-gene null hypothesis is that the expected count in `y` is equal to the sum of the scaled ambient proportion and some (non-negative) contribution from actual intra-cellular transcripts.
3. We combine p-values across all genes using Simes' method. This represents the evidence against the joint null hypothesis (that all of the per-gene nulls are true).
4. We find the largest scaling factor that fails to reject this joint null at the specified `threshold`. If `sum(ambient)` is equal to unity, this scaling factor can be interpreted as the maximum number of transcript molecules contributed to `y` by the ambient solution.

The process of going from a scaling factor to a combined p-value has no clean analytical solution, so we use an iterative grid search to identify to largest possible scaling factor at a decent resolution. `num.points` and `num.iter` control the resolution of the grid search, and generally do not need to be changed.

`maximumAmbience` is soft-deprecated; use `ambientContribMaximum` instead.

Value

If `mode="scale"`, a numeric vector is returned quantifying the maximum "contribution" of the ambient solution to each column of `y`. Scaling `ambient` by each entry yields the maximum ambient profile for the corresponding column of `y`.

If `mode="profile"`, a numeric matrix is returned containing the maximum ambient profile for each column of `y`. This is computed by scaling as described above; if `ambient` is a matrix, each column is scaled by the corresponding entry of the scaling vector.

If `mode="proportion"`, a numeric matrix is returned containing the maximum proportion of counts in `y` that are attributable to ambient contamination. This is computed by simply dividing the output of `mode="profile"` by `y` and capping all values at 1.

Caveats

The above algorithm is rather *ad hoc* and offers little in the way of theoretical guarantees. The p-value is used as a score rather than providing any meaningful error control. Empirically, increasing `threshold` will return a higher scaling factor by making the estimation more robust to drop-outs in `y`, at the cost of increasing the risk of over-estimation of the ambient contribution.

Our abuse of the p-value machinery means that the reported scaling often exceeds the actual contribution, especially at low counts where the reduced power fails to penalize overly large scaling factors. Hence, the function works best when `y` contains aggregated counts for one or more groups

of droplets with the same expected expression profile, e.g., clusters of related cells. Higher counts provide more power to detect deviations, hopefully leading to a more accurate estimate of the scaling factor. (On a practical note, this function is rather slow so it is more feasible to calculate on cluster-level profiles rather than per cell.)

Note that this function returns the *maximum possible* contribution of the ambient solution to y, not the actual contribution. In the most extreme case, if the ambient profile is similar to the expectation of y (e.g., due to sequencing a relatively homogeneous cell population), the maximum possible contribution of the ambient solution would be 100% of y, and subtraction would yield an empty count vector!

Author(s)

Aaron Lun

See Also

ambientProfileEmpty and ambientProfileBimodal, to estimate the ambient profile.

ambientContribSparse and ambientContribNegative, for other methods to estimate the ambient contribution.

emptyDrops, which uses the ambient profile to call cells.

ambientProfileEmpty or ambientProfileBimodal, to obtain an estimate to use in ambient.

ambientContribNegative or ambientContribSparse, for other methods of estimating the contribution.

Examples

```
# Making up some data for, e.g., a single cluster.
ambient <- c(runif(900, 0, 0.1), runif(100))
y <- rpois(1000, ambient * 100)
y[1:100] <- y[1:100] + rpois(100, 20) # actual biology.

# Estimating the maximum possible scaling factor:
scaling <- ambientContribMaximum(y, ambient)
scaling

# Estimating the maximum contribution to 'y' by 'ambient'.
contribution <- ambientContribMaximum(y, ambient, mode="profile")
DataFrame(ambient=drop(contribution), total=y)
```

---

ambientContribNegative

Ambient contribution from negative controls

Description

Estimate the contribution of the ambient solution to a particular expression profile, based on the abundance of negative control features that should not be expressed in the latter.

Usage

```
controlAmbience(...)

ambientContribNegative(y, ...)

## S4 method for signature 'ANY'
ambientContribNegative(
  y,
  ambient,
  features,
  mode = c("scale", "profile", "proportion")
)

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

Arguments

`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `controlAmbience`, arguments to pass to `ambientContribNegative`.
`y`	A numeric matrix-like object containing counts, where each row represents a feature (e.g., a gene or a conjugated tag) and each column represents either a cell or group of cells. Alternatively, a SummarizedExperiment object containing such a matrix. `y` can also be a numeric vector of counts; this is coerced into a one-column matrix.
`ambient`	A numeric vector of length equal to `nrow(y)`, containing the proportions of transcripts for each gene in the ambient solution. Alternatively, a matrix where each row corresponds to a row of `y` and each column contains a specific ambient profile for the corresponding column of `y`.
`features`	A logical, integer or character vector specifying the negative control features in `y` and `ambient`. Alternatively, a list of vectors specifying mutually exclusive sets of features.
`mode`	String indicating the output to return, see Value.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

Negative control features should be those that cannot be expressed and thus fully attributable to ambient contamination. This is most commonly determined *a priori* from the biological context and experimental system. For example, if spike-ins were introduced into the solution prior to cell capture, these would serve as a gold standard for ambient contamination in `y`. For single-nuclei sequencing, mitochondrial transcripts can serve a similar role under the assumption that all high-quality libraries are stripped nuclei.

If `features` is a list, it is expected to contain multiple sets of mutually exclusive features. Each cell should only express features in at most one set; no cell should express features in different sets. The expression of multiple sets can thus be attributed to ambient contamination. For this mode, an archetypal pairing is that of hemoglobins with immunoglobulins (Young and Behjati, 2018), which should not be co-expressed in any (known) cell type.

`controlAmbience` is soft-deprecated; use `ambientContribNegative` instead.

Value

If `mode="scale"`, a numeric vector is returned quantifying the estimated "contribution" of the ambient solution to each column of `y`. Scaling `ambient` by each entry yields the maximum ambient profile for the corresponding column of `y`.

If `mode="profile"`, a numeric matrix is returned containing the estimated ambient profile for each column of `y`. This is computed by scaling as described above; if `ambient` is a matrix, each column is scaled by the corresponding entry of the scaling vector.

If `mode="proportion"`, a numeric matrix is returned containing the estimated proportion of counts in `y` that are attributable to ambient contamination. This is computed by simply dividing the output of `mode="profile"` by `y` and capping all values at 1.

Author(s)

Aaron Lun

References

Young MD and Behjati S (2018). SoupX removes ambient RNA contamination from droplet based single-cell RNA sequencing data. *biorXiv*.

See Also

`ambientProfileEmpty` or `ambientProfileBimodal`, to obtain a profile estimate to use in `ambient`.

`ambientContribMaximum` or `ambientContribSparse`, for other methods of estimating contribution when negative control features are not available.

Examples

```
# Making up some data.
ambient <- c(runif(900, 0, 0.1), runif(100))
y <- rpois(1000, ambient * 50)
y <- y + c(integer(100), rpois(900, 5)) # actual biology, but first 100 genes silent.

# Using the first 100 genes as negative controls:
scaling <- ambientContribNegative(y, ambient, features=1:100)
scaling

# Estimating the negative control contribution to 'y' by 'ambient'.
contribution <- ambientContribNegative(y, ambient, features=1:100, mode="profile")
DataFrame(ambient=drop(contribution), total=y)
```

`ambientContribSparse` *Ambient contribution by assuming sparsity*

Description

Estimate the contribution of the ambient solution to each droplet by assuming that no more than a certain percentage of features are actually present/expressed in the cell.

Usage

```
ambientContribSparse(y, ...)

## S4 method for signature 'ANY'
ambientContribSparse(
  y,
  ambient,
  prop = 0.5,
  mode = c("scale", "profile", "proportion"),
  BPPARAM = SerialParam()
)

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

Arguments

`y`	A numeric matrix-like object containing counts, where each row represents a feature (usually a conjugated tag) and each column represents a cell or group of cells. Alternatively, a SummarizedExperiment object containing such a matrix. `y` can also be a numeric vector of counts; this is coerced into a one-column matrix.
`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
`ambient`	A numeric vector of length equal to `nrow(y)`, containing the proportions of transcripts for each feature in the ambient solution.
`prop`	Numeric scalar specifying the maximum proportion of features that are expected to be present for any cell.
`mode`	String indicating the output to return, see Value.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

The assumption here is that of sparsity, i.e., no more than `prop * nrow(y)` features should be actually present in each cell with a non-zero number of molecules. This is reasonable for most tag-based applications where we would expect only 1-2 tags (for cell hashing) or a minority of tags (for general CITE-seq) to be present per cell. Thus, counts for all other features must be driven by ambient contamination, allowing us to estimate a scaling factor for each cell based on the ratio to the ambient profile.

For gene expression, the sparsity assumption is less justifiable. Each cell could feasibly express a majority of the transcriptome (once we ignore constitutively silent features in the annotation, like pseudogenes). The sparsity of gene expression data also yields less precise scale estimates, reducing their utility in downstream applications. See `ambientContribNegative` or `ambientContribMaximum` instead, which operate from different assumptions.

Value

If `mode="scale"`, a numeric vector is returned quantifying the estimated "contribution" of the ambient solution to each column of `y`. Scaling `ambient` by each entry yields the estimated ambient profile for the corresponding column of `y`.

If `mode="profile"`, a numeric matrix is returned containing the estimated ambient profile for each column of `y`. This is computed by scaling as described above; if `ambient` is a matrix, each column is scaled by the corresponding entry of the scaling vector.

If `mode="proportion"`, a numeric matrix is returned containing the proportion of counts in `y` that are attributable to ambient contamination. This is computed by simply dividing the output of `mode="profile"` by `y` and capping all values at 1.

Author(s)

Aaron Lun

See Also

`ambientProfileBimodal`, to estimate the ambient profile for use in `ambient`.

`cleanTagCounts`, where this function is used to estimate ambient scaling factors.

Examples

```
amb <- 1:10
y <- matrix(rpois(10000, lambda=amb), nrow=10)
y[sample(length(y), 1000, replace=TRUE)] <- 1000

scaling <- ambientContribSparse(y, ambient=amb)
hist(scaling)
```

`ambientProfileBimodal` *Ambient profile from bimodality*

Description

Estimate the concentration of each feature in the ambient solution from a filtered count matrix containing only counts for cells, by assuming that each feature has a bimodal abundance distribution with ambient and high-expressing components.

Usage

```
inferAmbience(...)

ambientProfileBimodal(x, ...)

## S4 method for signature 'ANY'
ambientProfileBimodal(x, min.prop = 0.05)

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

Arguments

`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `inferAmbience`, arguments to pass to `ambientProfileBimodal`.
`x`	A numeric matrix-like object containing counts for each feature (row) and cell (column). Alternatively, a SummarizedExperiment object containing such a matrix.
`min.prop`	Numeric scalar in (0, 1) specifying the expected minimum proportion of barcodes contributed by each sample.
`assay.type`	Integer or scalar specifying the assay containing the count matrix.

Details

In some cases, we want to know the ambient profile but we only have the count matrix for the cell-containing libraries. This can be useful in functions such as `hashedDrops` or as a reference profile in `medianSizeFactors`. However, as we only have the cell-containing libraries, we cannot use `ambientProfileEmpty`.

This function estimates the ambient profile by assuming that each feature only labels a minority of the cells. Under this assumption, each feature's log-count distribution has a bimodal distribution where the lower mode represents ambient contamination. This is generally reasonable for tag-based applications like cell hashing or CITE-seq where data is usually binary, i.e., the marker is either present or not. We fit a two-component mixture model and identify all barcodes assigned to the lower component; and the mean of those counts is used as an estimate of the ambient contribution.

The initialization of the mixture model is controlled by `min.prop`, which starts the means of the lower and upper components at the `min.prop` and `1-min.prop` quantiles, respectively. This means that each sample is expected to contribute somewhere between `[min.prop, 1-min.prop]` barcodes. Larger values improve convergence but require stronger assumptions about the relative proportions of multiplexed samples.

`inferAmbience` is soft-deprecated; use `ambientProfileBimodal` instead.

Value

A numeric vector of length equal to `nrow(x)`, containing the estimated ambient proportions for each feature.

Author(s)

Aaron Lun

See Also

`hashedDrops`, where this function is used in the absence of an ambient profile.

`ambientProfileEmpty`, which should be used when the raw matrix (prior to filtering for cells) is available.

`ambientContribSparse` and related functions, to estimate the contribution of ambient contamination in each library.

Examples

```
x <- rbind(
    rpois(1000, rep(c(100, 1), c(100, 900))),
    rpois(1000, rep(c(2, 100, 2), c(100, 100, 800))),
    rpois(1000, rep(c(3, 100, 3), c(200, 700, 100)))
)

# Should be close to 1, 2, 3
ambientProfileBimodal(x)
```

---

ambientProfileEmpty *Estimate the ambient profile from empty droplets*

Description

Estimate the transcript proportions in the ambient solution from an unfiltered count matrix, assuming that low-count barcodes correspond to "known empty" droplets. Zeroes are filled in using the Good-Turing method.

Usage

```
estimateAmbience(...)

ambientProfileEmpty(m, ...)

## S4 method for signature 'ANY'
ambientProfileEmpty(
  m,
  lower = 100,
  by.rank = NULL,
  known.empty = NULL,
  round = TRUE,
  good.turing = TRUE,
  BPPARAM = SerialParam()
)

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

Arguments

`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For `estimateAmbience`, arguments to pass to `ambientProfileEmpty`.
`m`	A numeric matrix-like object - usually a dgTMatrix or dgCMatrix - containing droplet data *prior to any filtering or cell calling*. Columns represent barcoded droplets, rows represent genes. For `emptyDrops`, this may also be a SummarizedExperiment object containing such a matrix.
`lower`	A numeric scalar specifying the lower bound on the total UMI count, at or below which all barcodes are assumed to correspond to empty droplets.
`by.rank`	An optional integer scalar, used as an alternative to `lower` to identifying assumed empty droplets - see Details.
`known.empty`	an optional integer vector indexing barcodes that will be assumed to be empty, over-riding `lower` and `by.rank`.
`round`	Logical scalar indicating whether to check for non-integer values in `m` and, if present, round them for ambient profile estimation (see ?`ambientProfileEmpty`) and the multinomial simulations.
`good.turing`	Logical scalar indicating whether to perform Good-Turing estimation of the proportions.
`BPPARAM`	A BiocParallelParam object indicating whether parallelization should be used.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

This function obtains an estimate of the composition of the ambient pool of RNA based on the barcodes with total UMI counts less than or equal to `lower`. For each gene, counts are summed across all low-count barcodes and the resulting count vector is used for Good-Turing estimation of the proportions for each transcript. The aim here is to obtain reasonable proportions for genes with counts of zero in low-count barcodes but non-zero counts for other barcodes (thus avoiding likelihoods of zero when modelling the latter with the proportions).

This function will also attempt to detect whether `m` contains non-integer values by seeing if the column and row sums are discrete. If such values are present, `m` is first `round`ed to the nearest integer value before proceeding. This may be relevant when the count matrix is generated from pseudo-alignment methods like Alevin (see the **tximeta** package for details). Rounding is performed by default as discrete count values are necessary for the Good-Turing algorithm, but if `m` is known to be discrete, setting `round=FALSE` can provide a small efficiency improvement.

Setting `good.turing=FALSE` may be convenient to obtain raw counts for use in further modelling.

`estimateAmbience` is soft-deprecated; use `ambientProfileEmpty` instead.

Value

A numeric vector of length equal to `nrow(m)`, containing the estimated proportion of each transcript in the ambient solution.

If `good.turing=FALSE`, the vector instead contains the sum of counts for each gene across all low-count barcodes.

Behavior at zero counts

Good-Turing returns zero probabilities for zero counts if none of the summed counts are equal to 1. This is technically correct but not helpful, so we protect against this by adding a single "pseudo-feature" with a count of 1 to the profile. The modified profile is used to calculate a Good-Turing estimate of observing any feature that has zero counts, which is then divided to get the per-feature probability. We scale down all the other probabilities to make space for this new pseudo-probability, which has some properties of unclear utility (see https://github.com/MarioniLab/DropletUtils/issues/39).

Note that genes with counts of zero across all barcodes in `m` automatically have proportions of zero. This ensures that the estimation is not affected by the presence/absence of non-expressed genes in different annotations. In any case, such genes are likely to be completely irrelevant to downstream steps and can be safely ignored.

Finding the assumed empty droplets

The default approach is to assume that all barcodes with total counts less than or equal to `lower` are empty. This is generally effective but may not be adequate for datasets with unusually low or high sequencing depth, such that all or none of the barcodes are detected as empty respectively. For example, there is no obvious choice for `lower` in CITE-seq data given that the coverage can be highly variable.

In such cases, an alternative approach can be used by passing an integer to the `by.rank` argument. This specifies the number of barcodes with the highest total counts to ignore; the remaining barcodes are assumed to be ambient. The idea is that, even if the exact threshold is unknown, we

can be certain that a given experiment does not contain more than a particular number of genuine cell-containing barcodes based on the number of cells that were loaded into the machine. By setting `by.rank` to something greater than this *a priori* known number, we exclude the most likely candidates and use the remaining barcodes to compute the ambient profile.

Another alternative when working with some multimodal data, such as CITE-seq, could be to use statistics from one modality (e.g. mRNA counts) to define empty droplets in the other modality (e.g. CITE-seq) or combining CITE-seq with mRNA-counts. In this case, one may set `known.empty` to an integer vector indexing barcodes in columns of 'm' to mark cells for the ambient pool. For the purpose of retaining cells, if `lower` is set, it will be used to define the pool of ambient RNA in `barcodeRanks`. Otherwise the median of total counts of barcodes that have `known.empty` set will be used in its place.

Author(s)

Aaron Lun

See Also

`emptyDrops` and `hashedDrops`, where the ambient profile estimates are used for testing.

`ambientContribMaximum` and related functions, to estimate the contribution of ambient contamination in each library.

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts()

ambience <- ambientProfileEmpty(my.counts)
head(ambience)
```

---

barcodeRanks *Calculate barcode ranks*

Description

Compute barcode rank statistics and identify the knee and inflection points on the total count curve.

Usage

```
barcodeRanks(m, ...)

## S4 method for signature 'ANY'
barcodeRanks(
  m,
  lower = 100,
  exclude.from = 50,
```

```
  window = 1,
  gradient.threshold = -1,
  fit.bounds = NULL,
  df = 20,
  ...,
  BPPARAM = SerialParam()
)

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

Arguments

`m`	A numeric matrix-like object containing UMI counts, where columns represent barcoded droplets and rows represent genes. Alternatively, a SummarizedExperiment containing such a matrix.
`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
`lower`	A numeric scalar specifying the lower bound on the total UMI count, at or below which all barcodes are assumed to correspond to empty droplets and excluded from knee/inflection point identification.
`exclude.from`	An integer scalar specifying the number of highest ranking barcodes to exclude from knee/inflection point identification.
`window`	Numeric scalar specifying the length of the window (in log10 units) for knee/inflection point identification. Larger values improve stability of estimates at the cost of sensitivity to changes in the curve.
`gradient.threshold`	Numeric scalar specifying the maximum threshold on the gradient for identifying potential elbow points. Lower values increase the stringency of elbow point identification.
`fit.bounds, df`	Deprecated and ignored.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

Analyses of droplet-based scRNA-seq data often show a plot of the log-total count against the log-rank of each barcode where the highest ranks have the largest totals. This is equivalent to a transposed empirical cumulative density plot with log-transformed axes, which focuses on the barcodes with the largest counts. To create this plot, the `barcodeRanks` function will compute these ranks for all barcodes in `m`. Barcodes with the same total count receive the same average rank to avoid problems with discrete runs of the same total.

The function will also identify the inflection and knee points on the curve for downstream use. Both of these points correspond to a sharp transition between two components of the total count

distribution, presumably reflecting the difference between empty droplets with little RNA and cell-containing droplets with much more RNA. Only points with total counts above `lower` will be considered for knee/inflection point identification. Similarly, the first `exclude.from` points will be ignored to avoid instability at the start of the curve.

The actual identification of the knee/inflection points is based on a simple curve-tracing algorithm. We trace a window of fixed length `window` through the curve, and for each window, we consider the straight line connecting its ends:

- To find the knee, we filter for windows where the midpoint of the window lies above the end-connecting line. Of these, we select the window with the shortest end-connecting line, i.e., the strongest curvature. The midpoint of that window is defined as the knee.
- To find the inflection, we pick the window with the lowest (i.e., most negative) gradient of the end-connecting line. The midpoint of that window is defined as the inflection.
- In cases with multiple knee/inflection points, we aim to report the earlier values, i.e., those with higher log-totals. This is achieved by ignoring all windows after the first one that contains an "elbow" point in the curve. A window contains an elbow if its midpoint lies below the end-connecting line and the gradient is less than `gradient.threshold`.

Value

A DataFrame where each row corresponds to a column of `m`, and containing the following fields:

`rank`: Numeric, the rank of each barcode (averaged across ties).

`total`: Numeric, the total counts for each barcode.

The metadata contains `knee`, a numeric scalar containing the total count at the knee point; and `inflection`, a numeric scalar containing the total count at the inflection point.

Author(s)

Aaron Lun

See Also

`emptyDrops`, where this function is used.

Examples

```
# Mocking up some data:
set.seed(2000)
my.counts <- DropletUtils:::simCounts()

# Computing barcode rank statistics:
br.out <- barcodeRanks(my.counts)
names(br.out)

# Making a plot.
plot(br.out$rank, br.out$total, log="xy", xlab="Rank", ylab="Total")
o <- order(br.out$rank)
abline(h=metadata(br.out)$knee, col="dodgerblue", lty=2)
```

```
abline(h=metadata(br.out)$inflection, col="forestgreen", lty=2)
legend("bottomleft", lty=2, col=c("dodgerblue", "forestgreen"),
    legend=c("knee", "inflection"))
```

`chimericDrops` *Remove chimeric molecules*

Description

Remove chimeric molecules within each cell barcode's library in a droplet experiment.

Usage

```
chimericDrops(sample, barcode.length = NULL, use.library = NULL, ...)

removeChimericDrops(
  cells,
  umis,
  genes,
  nreads,
  ref.genes,
  min.frac = 0.8,
  get.chimeric = FALSE,
  get.diagnostics = FALSE
)
```

Arguments

`sample`	String containing paths to the molecule information HDF5 files, produced by CellRanger for 10X Genomics data.
`barcode.length`	An integer scalar specifying the length of the cell barcode, see `read10xMolInfo`.
`use.library`	An integer vector specifying the library indices for which to extract molecules from `sample`. Alternatively, a character string specifying one or more library types, e.g., `"Gene expression"`.
`...`	Further arguments to be passed to `removeChimericDrops`.
`cells`	Character vector containing cell barcodes, where each entry corresponds to one molecule.
`umis`	Integer vector containing encoded UMI sequences, see `?encodeSequences` for details.
`genes`	Integer vector specifying the gene indices. Each index should refer to an element of `ref.genes`.
`nreads`	Integer vector containing the number of reads per molecule.
`ref.genes`	A character vector containing the names or symbols of all genes.

`min.frac` A numeric scalar specifying the minimum fraction of reads required for a chimeric molecule to be retained.

`get.chimeric` A logical scalar indicating whether the UMI counts corresponding to chimeric molecules should be returned.

`get.diagnostics` A logical scalar indicating whether to return statistics for each molecule grouping.

Details

Chimeric molecules are occasionally generated during library preparation for highly multiplexed droplet experiments. Here, incomplete PCR products from one molecule hybridise to another molecule for extension using shared sequences like the poly-A tail for 3' protocols. This produces an amplicon where the UMI and cell barcode originate from one transcript molecule but the gene sequence is from another. If the second template is from another cell, this effect results in contamination of one cell's profile by another, similar to the contamination between samples discussed in `?swappedDrops`.

Chimerism manifests as molecules that have the same UMI sequence and cell barcode but are assigned to different genes. To remove them, this function will simply discard all molecules within the same cell that share UMI sequences. Of course, this may also remove non-chimeric molecules that have the same UMI by chance, but for typical UMI lengths (10-12 bp for 10X protocols) we expect UMI collisions to be very rare between molecules from the same cell.

Nonetheless, to mitigate losses due to collisions, we retain any molecule that has a much greater number of reads compared to all other molecules with the same UMI in the same cell. This is based on the expectation that the original non-chimeric molecule will have undergone more rounds of PCR amplification compared to its chimeric offspring, and thus will have higher read coverage. For all molecules with the same UMI within a given cell, we compute the proportion of reads assigned to each molecule and we keep the molecule with a proportion above `min.frac`. If no molecule passes this threshold, the entire set is discarded.

The `use.library` argument can be used to only check for chimeras within a given feature type, e.g., CRISPR tags. This is most relevant in situations where `sample` contains multiple libraries that involve different sets of shared sequences, such that chimeras are unlikely to form between molecules from different libraries. Analysis of just one library can be achieved by setting `use.library` to the name or index of the desired feature set.

Value

A list is returned with the `cleaned` entry, a sparse matrix containing the UMI count for each gene (row) and cell barcode (column) after removing chimeric molecules. All cell barcodes that were originally observed are reported as columns, though note that it is theoretically possible for some barcodes to contain no counts.

If `get.chimeric=TRUE`, a `chimeric` entry is returned in the list. This is a sparse matrix of UMI counts corresponding to the chimeric molecules. Adding the cleaned and chimeric matrices should yield the total UMI count prior to removal of swapped molecules.

If `get.diagnostics=TRUE`, the top-level list will also contain an additional `diagnostics` DataFrame. Each row corresponds to a group of molecules in the same cell with the same UMI. The DataFrame holds the number of molecules in the group, the sum of reads across all molecules in the group, and the proportion of reads assigned to the most sequenced molecule.

Author(s)

Aaron Lun

References

Dixit A. (2016). Correcting chimeric crosstalk in single cell RNA-seq experiments. *biorXiv*, https://doi.org/10.1101/093237

Examples

```
# Mocking up some 10x HDF5-formatted data.
curfile <- DropletUtils:::simBasicMolInfo(tempfile())

out <- chimericDrops(curfile)
dim(out$cleaned)

out2 <- chimericDrops(curfile, get.diagnostics=TRUE)
out2$diagnostics
```

---

cleanTagCounts *Clean a tag-based dataset*

Description

Remove low-quality libraries from a count matrix where each row is a tag and each column corresponds to a cell-containing barcode.

Usage

```
cleanTagCounts(x, ...)

## S4 method for signature 'ANY'
cleanTagCounts(
  x,
  controls,
  ...,
  ambient = NULL,
  exclusive = NULL,
  sparse.prop = 0.5
)

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

Arguments

x	A numeric matrix-like object containing counts for each tag (row) in each cell (column). Alternatively, a SummarizedExperiment containing such a matrix.
...	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment, further arguments to pass to the ANY method. For the ANY method, further arguments to pass to isOutlier. This includes batch to account for multi-batch experiments, and nmads to specify the stringency of the outlier-based filter.
controls	A vector specifying the rows of x corresponding to control tags. These are expected to be isotype controls that should not exhibit any real binding.
ambient	A numeric vector of length equal to nrow(x), containing the relative concentration of each tag in the ambient solution. Defaults to ambientProfileBimodal(x) if not explicitly provided.
exclusive	A character vector of names of mutually exclusive tags that should never be expressed on the same cell. Alternatively, a list of vectors of mutually exclusive sets of tags - see ambientContribNegative for details.
sparse.prop	Numeric scalar specifying the minimum proportion of tags that should be present per cell.
assay.type	Integer or string specifying the assay containing the count matrix.

Details

We remove cells for which there is no detectable ambient contamination. Specifically, we expect non-zero counts for most tags due to the deeply sequenced nature of tag-based data. If sparse.prop or more tags have zero counts, this is indicative of a failure in library preparation for that cell.

We also remove cells for which the total control count is unusually high. The control coverage is used as a proxy for non-specific binding, most notably from contamination of droplets by protein aggregates. High levels of non-specific activity are undesirable as this masks the actual marker profile of affected cells. The upper threshold is defined with isOutlier on the log-total control count.

If controls is missing, we instead compute the ambient scaling factor for each cell. This represents the amount of ambient contamination - see ?ambientContribSparse for more details - and cells with unusually high values are assumed to be affected by protein aggregates. High outliers are again identified and removed based on the log-ambient scale.

If controls is missing and exclusive is specified, the ambient scaling factor is computed by ambientContribNegative instead. This can be helpful for explicitly removing cells with impossible marker combinations, though it is only as comprehensive as the knowledge of mutually exclusive marker sets.

Value

A DataFrame with one row per column of x, containing the following fields:

- zero.ambient, a logical field indicating whether each cell has zero ambient contamination.
- sum.controls, a numeric field containing the sum of counts for all control features. Only present if controls is supplied.

- high.controls, a logical field indicating whether each cell has unusually high control total. Only present if controls is supplied.
- ambient.scale, a numeric field specifying the relative amount of ambient contamination. Only present if controls is *not* supplied.
- high.ambient, a numeric field indicating whether each cell has unusually high ambient contamination. Only present if controls is *not* supplied.
- discard, a logical field indicating whether a column in x should be discarded.

Author(s)

Aaron Lun

See Also

ambientContribSparse, to estimate the ambient contamination for each droplet.

isOutlier, to identify the outliers in a distribution of values.

Examples

```
x <- rbind(
    rpois(1000, rep(c(100, 10), c(100, 900))),
    rpois(1000, rep(c(20, 100, 20), c(100, 100, 800))),
    rpois(1000, rep(c(30, 100, 30), c(200, 700, 100)))
)

# Adding a zero-ambient column plus a high-ambient column.
x <- cbind(0, x, 1000)

df <- cleanTagCounts(x)
df
```

---

defaultDrops *Call cells from their total number of UMIs*

Description

Call cells according to the number of UMIs associated with each barcode, as implemented in CellRanger version 2.

Usage

```
defaultDrops(m, ...)

## S4 method for signature 'ANY'
defaultDrops(m, expected = 3000, upper.quant = 0.99, lower.prop = 0.1)

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

Arguments

m	A numeric matrix-like object containing counts, where columns represent barcoded droplets and rows represent features. The matrix should only contain barcodes for an individual sample, prior to any filtering for cells. Alternatively, a SummarizedExperiment containing such a matrix.
...	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
expected	A numeric scalar specifying the expected number of cells in this sample, as specified in the call to CellRanger.
upper.quant	A numeric scalar between 0 and 1 specifying the quantile of the top expected barcodes to consider for the first step of the algorithm.
lower.prop	A numeric scalar between 0 and 1 specifying the fraction of molecules of the upper.quant quantile result that a barcode must exceed to be called as a cell.
assay.type	Integer or string specifying the assay containing the count matrix.

Details

The defaultDrops function will call cells based on library size similarly to the CellRanger software suite from 10X Genomics. Default arguments correspond to an exact reproduction of CellRanger's algorithm, where the number of expected cells was also left at CellRanger default value.

The method computes the upper.quant quantile of the top expected barcodes, ordered by decreasing number of UMIs. Any barcodes containing more molecules than lower.prop times this quantile is considered to be a cell, and is retained for further analysis.

This method may be vulnerable to calling very well-captured background RNA as cells, or missing real cells with low RNA content. See ?emptyDrops for an alternative approach for cell calling.

Value

A logical vector of length ncol(m), indicating whether each column of m was called as a cell.

Author(s)

Jonathan Griffiths

References

10X Genomics (2017). Cell Ranger Algorithms Overview. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/algorithms/overview

See Also

emptyDrops, for another method for calling cells.

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts()

# Identify likely cell-containing droplets.
called <- defaultDrops(my.counts)
table(called)

# Get matrix of called cells.
cell.counts <- my.counts[, called]
```

`downsampleReads` *Downsample reads in a 10X Genomics dataset*

Description

Generate a UMI count matrix after downsampling reads from the molecule information file produced by CellRanger for 10X Genomics data.

Usage

```
downsampleReads(
  sample,
  prop,
  barcode.length = NULL,
  bycol = FALSE,
  features = NULL,
  use.library = NULL
)
```

Arguments

`sample`	A string containing the path to the molecule information HDF5 file.
`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.
`barcode.length`	An integer scalar specifying the length of the cell barcode, see `read10xMolInfo`.
`bycol`	A logical scalar indicating whether downsampling should be performed on a column-by-column basis.
`features`	A character vector containing the names of the features on which to perform downsampling.
`use.library`	An integer vector specifying the library indices for which to extract molecules from `sample`. Alternatively, a character vector specifying the library type(s), e.g., `"Gene expression"`.

Details

This function downsamples the reads for each molecule by the specified `prop`, using the information in `sample`. It then constructs a UMI count matrix based on the molecules with non-zero read counts. The aim is to eliminate differences in technical noise that can drive clustering by batch, as described in `downsampleMatrix`.

Subsampling the reads with `downsampleReads` recapitulates the effect of differences in sequencing depth per cell. This provides an alternative to downsampling with the CellRanger `aggr` function or subsampling with the 10X Genomics R kit. Note that this differs from directly subsampling the UMI count matrix with `downsampleMatrix`.

If `bycol=FALSE`, downsampling without replacement is performed on all reads from the entire dataset. The total number of reads for each cell after downsampling may not be exactly equal to `prop` times the original value. Note that this is the more natural approach and is the default, which differs from the default used in `downsampleMatrix`.

If `bycol=TRUE`, sampling without replacement is performed on the reads for each cell. The total number of reads for each cell after downsampling is guaranteed to be `prop` times the original total (rounded to the nearest integer). Different proportions can be specified for different cells by setting `prop` to a vector, where each proportion corresponds to a cell/GEM combination in the order returned by `get10xMolInfoStats`.

The `use.library` argument is intended for studies with multiple feature types, e.g., antibody capture or CRISPR tags. As the reads for each feature type are generated in a separate sequencing library, it is generally most appropriate to downsample reads for each feature type separately. This can be achieved by setting `use.library` to the name or index of the desired feature set. The features of interest can also be directly specified with `features`. (This will be intersected with any `use.library` choice if both are specified.)

Value

A numeric sparse matrix containing the downsampled UMI counts for each gene (row) and barcode (column). If `features` is set, only the rows with names in `features` are returned.

Author(s)

Aaron Lun

See Also

`downsampleMatrix`, for more general downsampling of the count matrix.

`read10xMolInfo`, to read the contents of the molecule information file.

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::simBasicMolInfo(tempfile())

# Downsampling by the reads.
downsampleReads(out, barcode.length=4, prop=0.5)
```

`emptyDrops` *Identify empty droplets*

Description

Distinguish between droplets containing cells and ambient RNA in a droplet-based single-cell RNA sequencing experiment.

Usage

```
testEmptyDrops(
  m,
  lower = 100,
  niters = 10000,
  test.ambient = FALSE,
  ignore = NULL,
  alpha = Inf,
  round = TRUE,
  by.rank = NULL,
  known.empty = NULL,
  BPPARAM = SerialParam()
)

emptyDrops(m, ...)

## S4 method for signature 'ANY'
emptyDrops(
  m,
  lower = 100,
  retain = NULL,
  barcode.args = list(),
  round = TRUE,
  test.ambient = FALSE,
  ...,
  BPPARAM = SerialParam()
)

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

Arguments

`m`	A numeric matrix-like object - usually a dgTMatrix or dgCMatrix - containing droplet data *prior to any filtering or cell calling*. Columns represent barcoded droplets, rows represent genes. For `emptyDrops`, this may also be a SummarizedExperiment object containing such a matrix.

lower	A numeric scalar specifying the lower bound on the total UMI count, at or below which all barcodes are assumed to correspond to empty droplets.
niters	An integer scalar specifying the number of iterations to use for the Monte Carlo p-value calculations.
test.ambient	A logical scalar indicating whether results should be returned for barcodes with totals less than or equal to lower.
ignore	A numeric scalar specifying the lower bound on the total UMI count, at or below which barcodes will be ignored (see Details for how this differs from lower).
alpha	A numeric scalar specifying the scaling parameter for the Dirichlet-multinomial sampling scheme.
round	Logical scalar indicating whether to check for non-integer values in m and, if present, round them for ambient profile estimation (see ?ambientProfileEmpty) and the multinomial simulations.
by.rank	An integer scalar parametrizing an alternative method for identifying assumed empty droplets - see ?ambientProfileEmpty for more details. If set, this is used to redefine lower and any specified value for lower is ignored.
known.empty	an optional integer vector indexing barcodes that will be assumed to be empty, over-riding lower and by.rank. See ?ambientProfileEmpty for more details.
	For the SummarizedExperiment method, further arguments to pass to the ANY method.
	For the ANY method, further arguments to pass to testEmptyDrops.
BPPARAM	A BiocParallelParam object indicating whether parallelization should be used.
...	For the generic, further arguments to pass to individual methods.
retain	A numeric scalar specifying the threshold for the total UMI count above which all barcodes are assumed to contain cells.
barcode.args	Further arguments to pass to barcodeRanks.
assay.type	Integer or string specifying the assay containing the count matrix.

Value

testEmptyDrops will return a DataFrame with the following components:

Total: Integer, the total UMI count for each barcode.

LogProb: Numeric, the log-probability of observing the barcode's count vector under the null model.

PValue: Numeric, the Monte Carlo p-value against the null model.

Limited: Logical, indicating whether a lower p-value could be obtained by increasing niters.

emptyDrops will return a DataFrame like testEmptyDrops, with an additional FDR field.

The metadata of the output DataFrame will contains the ambient profile in ambient, the estimated/specified value of alpha, the specified value of lower (possibly altered by use.rank) and the number of iterations in niters. For emptyDrops, the metadata will also contain the retention threshold in retain.

Details about `testEmptyDrops`

The `testEmptyDrops` function first obtains an estimate of the composition of the ambient pool of RNA based on the barcodes with total UMI counts less than or equal to `lower` (see `?ambientProfileEmpty` for details). This assumes that a cell-containing droplet would generally have higher total counts than empty droplets containing RNA from the ambient pool. Counts for the low-count barcodes are pooled together, and an estimate of the proportion vector for the ambient pool is calculated using `goodTuringProportions`. The count vector for each barcode above `lower` is then tested for a significant deviation from these proportions.

Then, `testEmptyDrops` will test each barcode for significant deviations from the ambient profile. The null hypothesis is that transcript molecules are included into droplets by multinomial sampling from the ambient profile. For each barcode, the probability of obtaining its count vector based on the null model is computed. Then, `niters` count vectors are simulated from the null model. The proportion of simulated vectors with probabilities lower than the observed multinomial probability for that barcode is used to calculate the p-value.

We use this Monte Carlo approach as an exact multinomial p-value is difficult to calculate. However, the p-value is lower-bounded by the value of `niters` (Phipson and Smyth, 2010), which can result in loss of power if `niters` is too small. Users can check whether this loss of power has any practical consequence by checking the `Limited` field in the output. If any barcodes have `Limited=TRUE` but does *not* reject the null hypothesis, it suggests that `niters` should be increased.

The stability of the Monte Carlo p-values depends on `niters`, which is only set to a default of 10000 for speed. Larger values improve stability with the only cost being that of time, so users should set `niters` to the largest value they are willing to wait for.

The `ignore` argument can also be set to ignore barcodes with total counts less than or equal to `ignore`. This differs from the `lower` argument in that the ignored barcodes are not necessarily used to compute the ambient profile. Users can interpret `ignore` as the minimum total count required for a barcode to be considered as a potential cell. In contrast, `lower` is the maximum total count below which all barcodes are assumed to be empty droplets.

Details about `emptyDrops`

The `emptyDrops` function identifies droplets that are likely to contain cells by calling `testEmptyDrops`. The Benjamini-Hochberg correction is applied to the Monte Carlo p-values to correct for multiple testing. Cells can then be defined by taking all barcodes with significantly non-ambient profiles, e.g., at a false discovery rate of 0.1%.

Barcodes that contain more than `retain` total counts are always retained. This ensures that large cells with profiles that are very similar to the ambient pool are not inadvertently discarded. If `retain` is not specified, it is set to the total count at the knee point detected by `barcodeRanks`. Manual specification of `retain` may be useful if the knee point was not correctly identified in complex log-rank curves. Users can also set `retain=Inf` to disable automatic retention of barcodes with large totals.

All barcodes with total counts above `retain` are assigned p-values of zero *during correction*, reflecting our assumption that they are true positives. This ensures that their Monte Carlo p-values do not affect the correction of other genes, and also means that they will have FDR values of zero. However, their original Monte Carlo p-values are still reported in the output, as these may be useful for diagnostic purposes.

This effect also means that users will not be able to recover the reported FDR by simply running p.adjust on the reported PValue. Similarly, setting test.ambient=TRUE will also modify the p-values prior to correction, see commentary below.

In general, users should call emptyDrops rather than testEmptyDrops. The latter is a "no frills" version that is largely intended for use within other functions.

Handling overdispersion

By default, alpha=Inf which means that the count vector is modelled with a multinomial distribution. This is appropriate when molecules are independently sampled into each droplet.

If alpha is set to a positive number, sampling is assumed to follow a Dirichlet-multinomial (DM) distribution. The parameter vector of the DM distribution is defined as the estimated ambient profile scaled by alpha. Smaller values of alpha model overdispersion in the counts, due to dependencies in sampling between molecules (e.g., aggregates, PCR duplication). If alpha=NULL, a maximum likelihood estimate is obtained from the count profiles for all barcodes with totals less than or equal to lower.

Users can check whether the model is suitable by extracting the p-values for all barcodes with test.ambient=TRUE. Under the null hypothesis, the p-values for presumed ambient barcodes (i.e., with total counts less than or equal to lower) should be uniformly distributed. Skews in the p-value distribution are indicative of an inaccuracy in the model. For example, an inaccurate alpha or ambient profile will manifest in the overenrichment of low p-values. Conversely, very sparse data will often exhibit in a enrichment of p-values at 1 as the Good-Turing probabilities in the ambient profile cannot be zero.

NA values in the results

We assume that barcodes with total UMI counts less than or equal to lower correspond to empty droplets. These are used to estimate the ambient expression profile against which the remaining barcodes are tested. Under this definition, these low-count barcodes cannot be cell-containing droplets and are excluded from the hypothesis testing. By removing these uninteresting tests, we obtain a modest improvement in detection power for the high-count barcodes.

However, it is still desirable for the number of rows of the output DataFrame to be the same as ncol(m). This allows easy subsetting of m based on a logical vector constructed from the output (e.g., to retain all FDR values below a threshold). To satisfy this requirement, the rows for the excluded barcodes are filled in with NA values for all fields in the output. We suggest using which to pick barcodes below a FDR threshold, see the Examples.

If test.ambient=TRUE, non-NA p-values will be reported for all barcodes with positive total counts, including those not greater than lower. This is occasionally useful for diagnostics to ensure that the p-values are well-calibrated for barcodes corresponding to (presumably) empty droplets. Specifically, if the null hypothesis were true, p-values for low-count barcodes should have a uniform distribution. Any strong peaks in the p-values near zero indicate that emptyDrops is not controlling the FDR correctly.

Note that, when setting test.ambient=TRUE in emptyDrops, barcodes less than or equal to lower will still have NA values in FDR. Such barcodes are still explicitly ignored in the correction as these are considered to be uninteresting. For back-compatibility purposes, setting test.ambient=NA will include these barcodes in the correction.

Non-empty droplets versus cells

Technically speaking, `emptyDrops` is designed to identify barcodes that correspond to non-empty droplets. This is close to but not quite the same as identifying cells, as droplets containing cell fragments, stripped nuclei and damaged cells will still be significantly non-empty. As such, it may often be necessary to perform additional quality control on the significant barcodes; we suggest doing so using methods from the **scater** package.

On occasion, `emptyDrops` may identify many more non-empty droplets than the expected number of cells. This is probably due to the generation of multiple cell fragments when a single cell is extensively damaged. In such cases, it is informative to construct a MA plot comparing the average expression between retained low-count barcodes and discarded barcodes to see which genes are driving the differences (and thus contributing to the larger number of non-empty calls). Mitochondrial and ribosomal genes are typical offenders; the former can be either up or down in the ambient solution, depending on whether the damage was severe enough to dissociate mitochondria from the cell fragments, while the latter is usually down in low-count barcodes due to loss of cytoplasmic RNA in cell fragments.

To mitigate this effect, we can filtering out the problematic genes from the matrix provided to `emptyDrops`. This eliminates their effect on the significance calculations and reduces the number of uninteresting non-empty calls, see https://github.com/MarioniLab/DropletUtils/issues/36 for an example. Of course, the full set of genes can still be retained for downstream analysis.

Author(s)

Aaron Lun

References

Lun A, Riesenfeld S, Andrews T, Dao TP, Gomes T, participants in the 1st Human Cell Atlas Jamboree, Marioni JC (2019). Distinguishing cells from empty droplets in droplet-based single-cell RNA sequencing data. *Genome Biol.* 20, 63.

Phipson B, Smyth GK (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Stat. Appl. Genet. Mol. Biol.* 9:Article 39.

See Also

`barcodeRanks`, for choosing the knee point.

`defaultDrops`, for an implementation of the cell-calling method used by CellRanger version 2.

`ambientProfileEmpty`, for more details on estimation of the ambient profile.

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts()

# Identify likely cell-containing droplets.
out <- emptyDrops(my.counts)
out
```

```
is.cell <- out$FDR <= 0.001
sum(is.cell, na.rm=TRUE)

# Subsetting the matrix to the cell-containing droplets.
# (using 'which()' to handle NAs smoothly).
cell.counts <- my.counts[,which(is.cell),drop=FALSE]
dim(cell.counts)

# Check if p-values are lower-bounded by 'niters'
# (increase 'niters' if any Limited==TRUE and Sig==FALSE)
table(Sig=is.cell, Limited=out$Limited)
```

`emptyDropsCellRanger` *CellRanger's emptyDrops variant*

Description

An approximate implementation of the `--soloCellFilter EmptyDrops_CR` filtering approach, which itself was reverse-engineered from the behavior of CellRanger 3.

Usage

```
emptyDropsCellRanger(m, ...)

## S4 method for signature 'ANY'
emptyDropsCellRanger(
  m,
  n.expected.cells = 3000,
  max.percentile = 0.99,
  max.min.ratio = 10,
  umi.min = 500,
  umi.min.frac.median = 0.01,
  cand.max.n = 20000,
  ind.min = 45000,
  ind.max = 90000,
  round = TRUE,
  niters = 10000,
  BPPARAM = SerialParam()
)

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

Arguments

`m`	A numeric matrix-like object containing counts, where columns represent barcoded droplets and rows represent features. The matrix should only contain bar-

codes for an individual sample, prior to any filtering for barcodes. Alternatively, a SummarizedExperiment containing such an object.

`...` Further arguments to pass to individual methods. Specifically, for the SummarizedExperiment method, further arguments to pass to the ANY method.

`n.expected.cells`
An integer scalar specifying the number of expected cells in a sample. Corresponds to the `nExpectedCells` argument in **STARsolo**.

`max.percentile` A numeric scalar between 0 and 1 used to define the maximum UMI count in the simple filtering algorithm. Corresponds to the `maxPercentile` argument in **STARsolo**.

`max.min.ratio` An integer scalar specifying the ratio of the maximum and minimum UMI count in the simple filtering algorithm. Corresponds to the `maxMinRatio` argument in **STARsolo**.

`umi.min` An integer scalar specifying the minimum UMI count for inclusion of a barcode in the cell candidate pool. Corresponds to the `umiMin` argument in **STARsolo**.

`umi.min.frac.median`
A numeric scalar between 0 and 1 used to define the minimum UMI count for inclusion of a barcode in the cell candidate pool. Specifically, the minimum is defined as `umi.min.frac.median` times the median UMI count of the real cells assigned by the simple filtering algorithm. Corresponds to the `umiMinFracMedian` argument in **STARsolo**.

`cand.max.n` An integer scalar specifying the maximum number of barcodes that can be included in the cell candidate pool. In effect, this applies a minimum threshold that is defined as the `cand.max.n`-th largest UMI count among all cells that are *not* selected by the simple filtering algorithm. Corresponds to the `candMaxN` in **STARsolo**.

`ind.min` An integer scalar specifying the lowest UMI count ranking for inclusion of a barcode in the ambient profile. Corresponds to the `indMin` argument in **STARsolo**.

`ind.max` An integer scalar specifying the highest UMI count ranking for inclusion of a barcode in the ambient profile. Corresponds to the `indMin` argument in **STARsolo**.

`round` A logical scalar indicating whether to check for non-integer values in `m` and, if present, round them for ambient profile estimation (see `?ambientProfileEmpty`) and the multinomial simulations.

`niters` An integer scalar specifying the number of iterations to use for the Monte Carlo p-value calculations.

`BPPARAM` A BiocParallelParam object indicating whether parallelization should be used.

`assay.type` String or integer specifying the assay of interest.

Details

`emptyDropsCellRanger` splits each sample's barcodes into three subsets.

1. The first subset contains barcodes that are selected by the "simple filtering algorithm", which are regarded as high quality cells without any further filtering. The minimum threshold T for this subset is defined by taking the `max.percentile` percentile of the top `n.expected.cells` barcodes, and then dividing by the `max.min.ratio` to obtain a minimum UMI count. (This is closely related to the algorithm used by `defaultDrops`.) All barcodes identified in this manner will have an FDR of zero.
2. The second subset contains the ambient pool and is defined as all barcodes with rankings between `ind.min` and `ind.max`. The barcodes that fall in this category will be used to compute the ambient profile. None of these barcodes are considered to be potential cells.
3. The third subset contains the pool of barcodes that are potential cells, i.e., cell candidates. This is defined as all barcodes with total counts below T and higher than all of the thresholds defined by `umi.min`, `umi.min.frac.median` and `cand.max.n`. Only the barcodes within this subset will be tested for signficant deviations from the ambient profile, i.e., FDR is not `NaN`.

As of time of writing, the arguments in **STARsolo** have a one-to-one correspondence with the arguments in `emptyDropsCellRanger`. All parameter defaults are set as the same as those used in STARsolo 2.7.9a.

The main differences between `emptyDropsCellRanger` and `emptyDrops` are:

- `emptyDropsCellRanger` does not use the knee point to identify "presumed real" cells, instead relying on a threshold based on the expected number of cells.
- `emptyDropsCellRanger` takes barcodes whose total count ranks within a certain range - by default, $(45,000, 90,000]$ - to compute the ambient profile. In contrast, `emptyDrops` only defines the upper bound using `lower` or `by.rank`.
- `emptyDropsCellRanger` defines a cell candidate pool according to three parameters, `umi.min`, `umi.min.frac.median` and `cand.max.n`. In `emptyDrops`, this is only defined by `lower`.

Value

A DataFrame with the same fields as that returned by `emptyDrops`.

Author(s)

Dongze He, Rob Patro

References

Kaminow B, Yunusov D, Dobin A (2021). STARsolo: accurate, fast and versatile mapping/quantification of single-cell and single-nucleus RNA-seq data. https://www.biorxiv.org/content/10.1101/2021.05.05.442755v1

See Also

`emptyDrops`, for the original implementation.

Examples

```
# Mocking up some data:
set.seed(0)
my.counts <- DropletUtils:::simCounts(nempty=100000, nlarge=2000, nsmall=1000)

# Identify likely cell-containing droplets.
out <- emptyDropsCellRanger(my.counts)
out

is.cell <- out$FDR <= 0.01
sum(is.cell, na.rm=TRUE)

# Subsetting the matrix to the cell-containing droplets.
# (using 'which()' to handle NAs smoothly).
cell.counts <- my.counts[,which(is.cell),drop=FALSE]
dim(cell.counts)
```

---

encodeSequences *Encode nucleotide sequences*

Description

Encode short nucleotide sequences into integers with a 2-bit encoding.

Usage

```
encodeSequences(sequences)
```

Arguments

sequences	A character vector of short nucleotide sequences, e.g., UMIs or cell barcodes.

Details

Each pair of bits encodes a nucleotide - 00 is A, 01 is C, 10 is G and 11 is T. The least significant byte contains the 3'-most nucleotides, and the remaining bits are set to zero. Thus, the sequence "CGGACT" is converted to the binary form:

```
01 10 10 00 01 11
```

... which corresponds to the integer 1671.

A consequence of R's use of 32-bit integers means that no element of sequences can be more than 15 nt long. Otherwise, integer overflow will occur.

Value

An integer vector containing the encoded sequences.

Author(s)

Aaron Lun

References

10X Genomics (2017). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/molecule_info

Examples

```
encodeSequences("CGGACT")
```

`get10xMolInfoStats` *Get 10x cell statistics*

Description

Compute some basic per-cell statistics from the 10x molecule information file.

Usage

```
get10xMolInfoStats(sample, barcode.length=NULL, use.library=NULL)
```

Arguments

`sample`	A string containing the path to the molecule information HDF5 file.
`barcode.length`	An integer scalar specifying the length of the cell barcode, see `read10xMolInfo`.
`use.library`	An integer vector specifying the library indices for which to extract molecules from `sample`. Alternatively, a character vector specifying the library type(s), e.g., `"Gene expression"`.

Value

A DataFrame containing one row per cell library, with the fields:

`cell`: Character, the cell barcode.

`gem_group`: Integer, the GEM group.

`num.umis`: Integer, the number of UMIs assigned to this cell barcode/GEM group combination.

`num.reads`: Integer, the number of reads for this combination.

`num.genes`: Integer, the number of detected genes.

Author(s)

Aaron Lun

See Also

read10xMolInfo

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::simBasicMolInfo(tempfile())

get10xMolInfoStats(out)
```

---

hashedDrops *Demultiplex cell hashing data*

Description

Demultiplex cell barcodes into their samples of origin based on the most abundant hash tag oligo (HTO). Also identify potential doublets based on the presence of multiple significant HTOs.

Usage

```
hashedDrops(x, ...)

## S4 method for signature 'ANY'
hashedDrops(
  x,
  ambient = NULL,
  min.prop = 0.05,
  pseudo.count = 5,
  constant.ambient = FALSE,
  doublet.nmads = 3,
  doublet.min = 2,
  doublet.mixture = FALSE,
  confident.nmads = 3,
  confident.min = 2,
  combinations = NULL
)

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

Arguments

x	A numeric/integer matrix-like object containing UMI counts. Rows correspond to HTOs and columns correspond to cell barcodes. Each barcode is assumed to correspond to a cell, i.e., cell calling is assumed to have already been performed. Alternatively, a SummarizedExperiment object containing such a matrix.

`...`	For the generic, further arguments to pass to individual methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
`ambient`	A numeric vector of length equal to `nrow(x)`, specifying the relative abundance of each HTO in the ambient solution - see Details.
`min.prop`	Numeric scalar to be used to infer the ambient profile when `ambient=NULL`, see `ambientProfileBimodal`.
`pseudo.count`	A numeric scalar specifying the minimum pseudo-count when computing log-fold changes.
`constant.ambient`	Logical scalar indicating whether a constant level of ambient contamination should be used to estimate `LogFC2` for all cells.
`doublet.nmads`	A numeric scalar specifying the number of median absolute deviations (MADs) to use to identify doublets.
`doublet.min`	A numeric scalar specifying the minimum threshold on the log-fold change to use to identify doublets.
`doublet.mixture`	Logical scalar indicating whether to use a 2-component mixture model to identify doublets.
`confident.nmads`	A numeric scalar specifying the number of MADs to use to identify confidently assigned singlets.
`confident.min`	A numeric scalar specifying the minimum threshold on the log-fold change to use to identify singlets.
`combinations`	An integer matrix specifying valid *combinations* of HTOs. Each row corresponds to a single sample and specifies the indices of rows in `x` corresponding to the HTOs used to label that sample.
`assay.type`	Integer or string specifying the assay containing the count matrix.

Details

The idea behind cell hashing is that cells from the same sample are stained with reagent conjugated with a single HTO. Cells are mixed across multiple samples and subjected to droplet-based single-cell sequencing. Cell barcode libraries can then be demultiplexed into individual samples based on whether their unique HTO is detected.

We identify the sample of origin for each cell barcode as that corresponding to the most abundant HTO. (See below for more details on exactly how "most abundant" is defined.) The log-fold change between the largest and second-largest abundances is reported for each barcode (`LogFC`), with large log-fold changes representing confident assignment to a single sample. We also report the log-fold change of the second-most abundant HTO over the estimated level of ambient contamination (`LogFC2`), with large log-fold changes indicating that a doublet is present.

To facilitate quality control, we explicitly identify problematic barcodes as outliers on the relevant metrics.

- By default, we identify putative doublets as those with LogFC2 values that are (i) doublet.nmads MADs above the median and (ii) greater than doublet.min. The hard threshold is more-or-less arbitrary and aims to avoid overly aggressive detection of large outliers in a naturally right-skewed distribution (given that the log-fold changes are positive by definition, and most of the distribution is located near zero).
- Alternatively, if doublet.mixture=TRUE, we fit a two-component mixture model to the LogFC2 distribution. Doublets are identified as all members of the component with the larger mean. This avoids the need for the arbitrary parameters mentioned above but only works when there are many doublets, otherwise both components will be fitted to the non-doublet values. (Initialization of the model assumes at least 5% doublets.)

Of the non-doublet libraries, we consider them to be confidently assigned to a single sample if their LogFC values are (i) *not* less than confident.nmads MADs below the median and (ii) greater than confident.min. The hard threshold is again arbitrary, but this time it aims to avoid insufficiently aggressive outlier detection - typically from an inflation of the MAD when the LogFC values are large, positive and highly variable.

Value

A DataFrame with one row per column of x, containing the following fields:

- Total, integer specifying the total count across all HTOs for each barcode.
- Best, integer specifying the HTO with the highest abundance for each barcode.
- Second, integer specifying the HTO with the second-highest abundance.
- LogFC, numeric containing the log-fold change between the abundances of the best and second-best HTO.
- LogFC2, numeric containing the log-fold change in the second-best HTO over the ambient contamination.
- Doublet, logical specifying whether a barcode is a doublet.
- Confident, logical specifying whether a barcode is a confidently assigned singlet.

In addition, the metadata contains ambient, a numeric vector containing the (estimate of the) ambient profile; doublet.threshold, the threshold applied to LogFC2 to identify doublets; and confident.threshold, the threshold applied to non-doublet LogFC values to identify confident singlets.

If combinations is specified, Best instead specifies the sample (i.e., row index of combinations). The interpretation of LogFC and LogFC2 are slightly different, and Second is not reported - see "Resolving combinatorial hashes".

Use only on non-empty droplets

This function assumes that cell calling has already been performed, e.g., with emptyDrops. Specifically, x should only contain columns corresponding to non-empty droplets. If empty droplets are included, their log-fold changes will simply reflect stochastic sampling in the ambient solution and violate the assumptions involved in outlier detection.

If x contains columns for both empty and non-empty droplets, it is straightforward to simply run emptyDrops on the HTO count matrix to identify the latter. Note that some fiddling with the lower= argument may be required, depending on the sequencing depth of the HTO libraries.

Adjusting abundances for ambient contamination

HTO abundances require some care to compute due to the presence of ambient contamination in each library. Ideally, the experiment would be performed in such a manner that the concentration of each HTO is the same. However, if one HTO is present at higher concentration in the ambient solution, this might incorrectly cause us to assign all barcodes to the corresponding sample.

To adjust for ambient contamination, we assume that the ambient contamination in each library follows the same profile as `ambient`. We further assume that a minority of HTOs in a library are actually driven by the presence of cell(s), the rest coming from the ambient solution. We estimate the level of ambient contamination in each barcode by scaling `ambient`, using a **DESeq**-like normalization algorithm to compute the scaling factor. (The requisite assumption of a non-DE majority follows from the two assumptions above.) We then subtract the scaled ambient proportions from the HTO count profile to remove the effect of contamination. Abundances that would otherwise be negative are set to zero.

The scaling factor for each barcode is defined by computing ratios between the HTO counts and `ambient`, and taking the median across all HTOs. However, this strict definition is only used when there are at least 5 HTOs being considered. For experiments with 3-4 HTOs, we assume that higher-order multiplets are negligible and define the scaling factor as the third-largest ratio. For experiments with only 2 HTOs, the second-most abundant HTO is always used to estimate the ambient contamination.

Ideally, `ambient` would be obtained from libraries that do not correspond to cell-containing droplets. For example, we could get this information from the `metadata` of the `emptyDrops` output, had we run `emptyDrops` on the HTO count matrix (see below). Unfortunately, in some cases (e.g., public data), counts are provided for only the cell-containing barcodes. If `ambient=NULL`, the profile is inferred from `x` using `ambientProfileBimodal`.

Computing the log-fold changes

HTO abundances may be set to zero after subtracting the ambient noise. Thus, we need to add a pseudo-count to ensure that we can actually compute the log-fold changes described in "Value".

For each barcode, we define the pseudo-count as the average ambient HTO count, i.e., the average of the scaled `ambient` for that barcode. This is motivated by the assumption that the number of contaminating transcript molecules is roughly the same in each droplet, such that any differences in ambient coverage between libraries reflect barcode-specific biases (capture efficiency, sequencing depth) that would also affect cell-derived HTO counts. By using the average ambient count as the pseudo-count, we ensure that the shrinkage of the log-fold changes is not driven by the sequencing depth, e.g., a constant pseudo-count would inflict greater shrinkage on libraries that have not been sequenced as deeply. This avoids excessive variability in the log-fold change distribution that would otherwise reduce the precision of outlier detection. Another nice aspect of this approach is that it collapses to a no-op if the experiment is well-executed with identical concentrations of all HTOs in the ambient solution. (That said, we still enforce a minimum pseudo-count of `pseudo.count` if the average ambient count is lower than that, simply to avoid highly variable log-fold changes when dealing with very low counts.)

Once the pseudo-count is added to the ambient-subtracted abundances, we compute the log-fold changes as described in "Value". `LogFC` is defined as the log-fold change in the most abundant HTO over the second-most abundant HTO. `LogFC2` is defined as the log-fold change in the second-most abundant HTO over the ambient contamination. By default, the denominator for `LogFC2` is set

to the per-barcode average ambient count, equivalent to the pseudo-count used above. This cancels out any variation in sequencing depth for more precise outlier calls.

Handling 2 or fewer samples

If `x` has no more than two rows, `LogFC2`, `Doublet` and `doublet.threshold` are set to `NA`. Strictly speaking, doublet detection is not possible as the second HTO is always used to estimate the ambient scaling and thus `LogFC2` is always zero. `Confident` calls are still available in the output of this function so assignment to the individual samples can still be performed. In this scenario, the non-confident assignments are probably also doublets, though this cannot be said with much certainty.

To work around this limitation, we can set `constant.ambient=TRUE`, which defines the denominator of each barcode's `LogFC2` as the median of the per-barcode average ambient counts across all barcodes. This is useful in scenarios where `nrow(x)` is too small and we cannot assume that the abundances of most HTOs are driven by ambient contamination. By assuming most barcodes are not doublets, we can obtain a dataset-wide baseline for the ambient contamination to compute `LogFC2`. The cost of this approach is that the log-fold changes will be more variable as sequencing depth is not cancelled out.

If `x` has no more than one row, `Confident`, `LogFC` and `confident.threshold` are set to `NA`. Obviously, if there is only one HTO, the identity of the assigned sample is a foregone conclusion.

Resolving combinatorial hashes

In some applications, samples are labelled with a combination of HTOs to enable achieve greater multiplexing throughput. This is accommodated by passing `combinations` to specify the valid HTO combinations that were used for sample labelling. Each row of `combinations` corresponds to a sample and should contain non-duplicated row indices of `x` corresponding to the HTOs used in that sample.

The calculation for the single-HTO case is then generalized for HTO combinations. The most important differences are that:

- The reported `LogFC` is now the log-fold change between the nth most abundant HTO and the $n + 1$th HTO, where n is the number of HTOs in a valid combination. This captures the drop-off in abundance beyond the expected number of HTOs.
- The reported `LogFC2` is now the log-fold change of the $n + 1$th HTO over the ambient contamination. This captures the high abundance of the more-than-expected number of HTOs when doublets are present.
- `Best` no longer refers to the row index of `x`, but instead to the row index of `combinations`. This may contain `NA` values if a particular combination of HTOs is observed but not present in the expected set.
- `Second` is no longer reported as we cannot conveniently determine the identity of the second sample.

We also generalize the edge-case behavior when there are not enough HTOs to support doublet detection. Consider that an inter-sample doublet may result in up to $2n$ abundant HTOs. Estimation of the scaling factor will attempt to avoid using the top $2n$ ratios. If `nrow(x)` is equal to or less than $2n$, doublet statistics will not be reported at all, i.e., `Doublet` and `LogFC2` are set to `NA`. This can be overcome by setting `constant.ambient=TRUE` as described above.

Author(s)

Aaron Lun

References

Stoeckius M, Zheng S, Houck-Loomis B et al. (2018) Cell Hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics. *Genome Biol.* 19, 1:224

See Also

`emptyDrops`, to identify which barcodes are likely to contain cells.

Examples

```
# Mocking up an example dataset with 10 HTOs and 10% doublets.
ncells <- 1000
nhto <- 10
y <- matrix(rpois(ncells*nhto, 50), nrow=nhto)
true.sample <- sample(nhto, ncells, replace=TRUE)
y[cbind(true.sample, seq_len(ncells))] <- 1000

ndoub <- ncells/10
next.sample <- (true.sample[1:ndoub]  + 1) %% nrow(y)
next.sample[next.sample==0] <- nrow(y)
y[cbind(next.sample, seq_len(ndoub))] <- 500

# Computing hashing statistics.
stats <- hashedDrops(y)

# Doublets show up in the top-left, singlets in the bottom right.
plot(stats$LogFC, stats$LogFC2)

# Most cells should be singlets with low second log-fold changes.
hist(stats$LogFC2, breaks=50)

# Identify confident singlets or doublets at the given threshold.
summary(stats$Confident)
summary(stats$Doublet)

# Checking against the known truth, in this case
# 'Best' contains the putative sample of origin.
table(stats$Best, true.sample)
```

---

makeCountMatrix *Make a count matrix*

Description

Construct a count matrix from per-molecule information, typically the cell and gene of origin.

Usage

```
makeCountMatrix(gene, cell, all.genes=NULL, all.cells=NULL, value=NULL)
```

Arguments

gene	An integer or character vector specifying the gene to which each molecule was assigned.
cell	An integer or character vector specifying the cell to which each molecule was assigned.
all.genes	A character vector containing the names of all genes in the dataset.
all.cells	A character vector containing the names of all cells in the dataset.
value	A numeric vector containing values for each molecule.

Details

Each element of the vectors gene, cell and (if specified) value contain information for a single transcript molecule. Each entry of the output matrix corresponds to a single gene and cell combination. If multiple molecules are present with the same combination, their values in value are summed together, and the sum is used as the entry of the output matrix.

If value=NULL, it will default to a vector of all 1's. Each entry of the output matrix represents the number of molecules with the corresponding combination, i.e., UMI counts. Users can pass other metrics such as the number of reads covering each molecule. This would yield a read count matrix rather than a UMI count matrix.

If all.genes is not specified, it is kept as NULL for integer gene. Otherwise, it is defined as the sorted unique values of character gene. The same occurs for cell and all.cells.

If gene is integer, its values should be positive and no greater than length(all.genes) if all.genes!=NULL. If gene is character, its values should be a subset of those in all.genes. The same requirements apply to cell and all.cells.

Value

A sparse matrix where rows are genes, columns are cells and entries are the sum of value for each gene/cell combination. Rows and columns are named if the gene or cell are character or if all.genes or all.cells are specified.

Author(s)

Aaron Lun

See Also

read10xMolInfo

Examples

```
nmolecules <- 100
gene.id <- sample(LETTERS, nmolecules, replace=TRUE)
cell.id <- sample(20, nmolecules, replace=TRUE)
makeCountMatrix(gene.id, cell.id)
```

`read10xCounts` *Load data from a 10X Genomics experiment*

Description

Creates a SingleCellExperiment from the CellRanger output directories for 10X Genomics data.

Usage

```
read10xCounts(
  samples,
  sample.names = names(samples),
  col.names = FALSE,
  row.names = c("id", "symbol"),
  type = c("auto", "mtx", "hdf5", "prefix", "sparse", "HDF5"),
  delayed = FALSE,
  version = c("auto", "2", "3"),
  genome = NULL,
  compressed = NULL,
  intersect.genes = FALSE,
  mtx.two.pass = FALSE,
  mtx.class = c("CsparseMatrix", "SVT_SparseMatrix"),
  mtx.threads = 1,
  BPPARAM = SerialParam()
)
```

Arguments

`samples`	A character vector containing one or more directory names, each corresponding to a 10X sample. Each directory should contain a matrix file, a gene/feature annotation file, and a barcode annotation file. Alternatively, each string may contain a path to a HDF5 file in the sparse matrix format generated by 10X. These can be mixed with directory names when `type="auto"`. Alternatively, each string may contain a prefix of names for the three-file system described above, where the rest of the name of each file follows the standard 10X output.
`sample.names`	A character vector of length equal to `samples`, containing the sample names to store in the column metadata of the output object. If `NULL`, the file paths in `samples` are used directly.

col.names	A logical scalar indicating whether the columns of the output object should be named with the cell barcodes.
row.names	String specifying whether to use Ensembl IDs ("ID") or gene symbols ("Symbol") as row names. For symbols, the Ensembl ID will be appended to disambiguate rows where the same symbol corresponds to multiple Ensembl IDs.
type	String specifying the type of 10X format to read data from.
delayed	Logical scalar indicating whether sparse matrices should be wrapped in DelayedArrays before combining. Only relevant for multiple samples.
version	String specifying the version of the 10X format to read data from.
genome	String specifying the genome if type="HDF5" and version='2'.
compressed	Logical scalar indicating whether the text files are compressed for type="mtx" or "prefix".
intersect.genes	Logical scalar indicating whether to take the intersection of common genes across all samples. If FALSE, differences in gene information across samples will cause an error to be raised.
mtx.two.pass	Logical scalar indicating whether to use a two-pass approach for loading data from a Matrix Market file. This reduces peak memory usage at the cost of some additional runtime. Only relevant when type="mtx" or type="prefix".
mtx.class	String specifying the class of the output matrix when type="mtx" or type="prefix".
mtx.threads	Integer scalar specifying the number of threads to use for reading Matrix Market files. Only relevant when type="mtx" or type="prefix".
BPPARAM	A BiocParallelParam object specifying how loading should be parallelized for multiple samples.

Details

This function has a long and storied past. It was originally developed as the read10xResults function in **scater**, inspired by the Read10X function from the **Seurat** package. It was then migrated to this package in an effort to consolidate some 10X-related functionality across various packages.

If type="auto", the format of the input file is automatically detected for each samples based on whether it ends with ".h5". If so, type is set to "HDF5"; otherwise it is set to "sparse".

- If type="mtx" (or its older alias "sparse"), count data are assumed to be stored in a directory. This should contain a (possibly Gzipped) MatrixMarket text file ("matrix.mtx") with additional tab-delimited files for barcodes ("barcodes.tsv") and gene annotation ("features.tsv" for version 3 or "genes.tsv" for version 2).
- If type="prefix", count data are assumed to follow same three-file structure for each sample as described for type="mtx". However, each sample is defined by a prefix in the file names rather than by being stored a separate directory. For example, if the samples entry is "xyx_", the files are expected to be "xyz_matrix.mtx", "xyz_barcodes.tsv", etc.
- If type="hdf5" (or its older alias "HDF5"), count data are assumed to follow the 10X sparse HDF5 format for large data sets. It is loaded as a TENxMatrix object, which is a stub object that refers back to the file in samples. Users may need to set genome if it cannot be automatically determined when version="2".

When type="mtx" or "prefix" and compressed=NULL, the function will automatically search for both the unzipped and Gzipped versions of the files. This assumes that the compressed files have an additional ".gz" suffix. We can restrict to only compressed or uncompressed files by setting compressed=TRUE or FALSE, respectively.

CellRanger 3.0 introduced a major change in the format of the output files for both types. If version="auto", the version of the format is automatically detected from the supplied paths. For type="mtx", this is based on whether there is a "features.tsv.gz" file in the directory. For type="HDF5", this is based on whether there is a top-level "matrix" group with a "matrix/features" subgroup in the file.

Matrices are combined by column if multiple samples were specified. This will throw an error if the gene information is not consistent across samples. For type="mtx" or "prefix", users can set delayed=TRUE to save memory during the combining process. This also avoids integer overflow for very large datasets.

If col.names=TRUE and length(sample)==1, each column is named by the cell barcode. For multiple samples, the index of each sample in samples is concatenated to the cell barcode to form the column name. This avoids problems with multiple instances of the same cell barcodes in different samples.

Note that user-level manipulation of sparse matrices requires loading of the **Matrix** package. Otherwise, calculation of rowSums, colSums, etc. will result in errors.

Value

A SingleCellExperiment object containing count data for each gene (row) and cell (column) across all samples.

- Row metadata will contain the fields "ID" and "Symbol". The former is the gene identifier (usually Ensembl), while the latter is the gene name. If version="3", it will also contain the "Type" field specifying the type of feature (e.g., gene or antibody).
- Column metadata will contain the fields "Sample" and "Barcode". The former contains the name of the sample (or if not supplied, the path in samples) from which each column was obtained. The latter contains to the cell barcode sequence and GEM group for each cell library.
- Rows are named with the gene identifier. Columns are named with the cell barcode in certain settings, see Details.
- The assays will contain a single "counts" matrix, containing UMI counts for each gene in each cell. Note that the matrix representation will depend on the format of the samples, see Details.
- The metadata contains a "Samples" field, containing the input samples character vector.

Author(s)

Davis McCarthy, with modifications from Aaron Lun

References

Zheng GX, Terry JM, Belgrader P, and others (2017). Massively parallel digital transcriptional profiling of single cells. *Nat Commun* 8:14049.

10X Genomics (2017). Gene-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/output/matrices

10X Genomics (2018). Feature-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/matrices

10X Genomics (2018). HDF5 Gene-Barcode Matrix Format. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/advanced/h5_matrices

10X Genomics (2018). HDF5 Feature Barcode Matrix Format. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/advanced/h5_matrices

See Also

splitAltExps, to split alternative feature sets (e.g., antibody tags) into their own Experiments.

write10xCounts, to create 10X-formatted file(s) from a count matrix.

Examples

```
# Mocking up some 10X genomics output.
example(write10xCounts, echo=FALSE)

# Reading it in.
sce10x <- read10xCounts(tmpdir)

# Column names are dropped with multiple 'samples'.
sce10x2 <- read10xCounts(c(tmpdir, tmpdir))
```

---

read10xMolInfo *Read the 10X molecule information file*

Description

Extract relevant fields from the molecule information HDF5 file, produced by CellRanger for 10X Genomics data.

Usage

```
read10xMolInfo(
  sample,
  barcode.length = NULL,
  keep.unmapped = FALSE,
  get.cell = TRUE,
  get.umi = TRUE,
  get.gem = TRUE,
  get.gene = TRUE,
  get.reads = TRUE,
  get.library = TRUE,
  extract.library.info = FALSE,
  version = c("auto", "2", "3")
)
```

Arguments

sample	A string containing the path to the molecule information HDF5 file.
barcode.length	An integer scalar specifying the length of the cell barcode. Only relevant when version="2".
keep.unmapped	A logical scalar indicating whether unmapped molecules should be reported.
get.cell, get.umi, get.gem, get.gene, get.reads, get.library	Logical scalar indicating whether the corresponding field should be extracted for each molecule.
extract.library.info	Logical scalar indicating whether the library information should be extracted. Only relevant when version="3".
version	String specifying the version of the 10X molecule information format to read data from.

Details

Molecules that were not assigned to any gene have gene set to length(genes)+1. By default, these are removed when keep.unmapped=FALSE.

CellRanger 3.0 introduced a major change in the format of the molecule information files. When version="auto", the function will attempt to determine the version format of the file. This can also be user-specified by setting version explicitly.

For files produced by version 2.2 of the CellRanger software, the length of the cell barcode is not given. Instead, the barcode length is automatically inferred if barcode.length=NULL and version="2". Currently, version 1 of the 10X chemistry uses 14 nt barcodes, while version 2 uses 16 nt barcodes.

Setting any of the get.* arguments will (generally) avoid extraction of the corresponding field. This can improve efficiency if that field is not necessary for further analysis. Aside from the missing field, the results are guaranteed to be identical, i.e., same order and number of rows.

Value

A named list is returned containing data, a DataFrame where each row corresponds to a single transcript molecule. This contains the following fields:

barcode: Character, the cell barcode for each molecule.

umi: Integer, the processed UMI barcode in 2-bit encoding.

gem_group: Integer, the GEM group.

gene: Integer, the index of the gene to which the molecule was assigned. This refers to an entry in the genes vector, see below.

reads: Integer, the number of reads mapped to this molecule.

reads: Integer, the number of reads mapped to this molecule.

library: Integer, the library index in cases where multiple libraries are present in the same file. Only reported when version="3".

A field will not be present in the DataFrame if the corresponding `get.*` argument is `FALSE`,

The second element of the list is `genes`, a character vector containing the names of all genes in the annotation. This is indexed by the `gene` field in the `data` DataFrame.

If `version="3"`, a `feature.type` entry is added to the list. This is a character vector of the same length as `genes`, containing the feature type for each gene.

If `extract.library.info=TRUE`, an additional element named `library.info` is returned. This is a list of lists containing per-library information such as the `"library_type"`. The `library` field in the `data` DataFrame indexes this list.

Author(s)

Aaron Lun, based on code by Jonathan Griffiths

References

Zheng GX, Terry JM, Belgrader P, and others (2017). Massively parallel digital transcriptional profiling of single cells. *Nat Commun* 8:14049.

10X Genomics (2017). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/output/molecule_info

10X Genomics (2018). Molecule info. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/molecule_info

See Also

`makeCountMatrix`, which creates a count matrix from this information.

Examples

```
# Mocking up some 10X HDF5-formatted data.
out <- DropletUtils:::simBasicMolInfo(tempfile())

# Reading the resulting file.
read10xMolInfo(out)
```

---

reexports *Objects exported from other packages*

Description

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

scuttle `downsampleBatches`, `downsampleMatrix`

`removeAmbience` *Remove the ambient profile*

Description

Estimate and remove the ambient profile from a count matrix, given pre-existing groupings of similar cells. This function is largely intended for plot beautification rather than real analysis.

Usage

```
removeAmbience(y, ...)

## S4 method for signature 'ANY'
removeAmbience(
  y,
  ambient,
  groups,
  features = NULL,
  ...,
  size.factors = librarySizeFactors(y),
  dispersion = 0.1,
  sink = NULL,
  BPPARAM = SerialParam()
)

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

Arguments

`y`	A numeric matrix-like object containing counts for each gene (row) and cell or group of cells (column). Alternatively, a SummarizedExperiment containing such a matrix.
`...`	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method. For the ANY method, Further arguments to pass to `ambientContribMaximum`.
`ambient`	A numeric vector of length equal to `nrow(y)`, containing the proportions of transcripts for each gene in the ambient solution.
`groups`	A vector of length equal to `ncol(y)`, specifying the assigned group for each cell. This can also be a DataFrame, see `?sumCountsAcrossCells`.
`features`	A vector of control features or a list of mutually exclusive feature sets, see `?ambientContribNegative` for more details.
`size.factors`	Numeric scalar specifying the size factors for each column of `y`, defaults to library size-derived size factors.

dispersion	Numeric scalar specifying the dispersion to use in the quantile-quantile mapping.
sink	An optional RealizationSink object of the same dimensions as y.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed.
assay.type	Integer or string specifying the assay containing the count matrix.

Details

This function will aggregate counts from each group of related cells into an average profile. For each group, we estimate the contribution of the ambient profile and subtract it from the average. By default, this is done with ambientContribMaximum, but if enough is known about the biological system, users can specify feaures to use ambientContribNegative instead.

We then perform quantile-quantile mapping of counts in y from the old to new averages. This approach preserves the mean-variance relationship and improves the precision of estimate of the ambient contribution, but relies on a sensible grouping of similar cells, e.g., unsupervised clusters or cell type annotations. As such, this function is best used at the end of the analysis to clean up expression matrices prior to visualization.

Value

A numeric matrix-like object of the same dimensions as y, containing the counts after removing the ambient contamination. The exact representation of the output will depend on the class of y and whether sink was used.

Author(s)

Aaron Lun

See Also

ambientContribMaximum and ambientContribNegative, to estimate the ambient contribution.

estimateAmbience, to estimate the ambient profile.

The **SoupX** package, which provides another implementation of the same general approach.

Examples

```
# Making up some data.
ngenes <- 1000
ambient <- runif(ngenes, 0, 0.1)
cells <- c(runif(100) * 10, integer(900))
y <- matrix(rpois(ngenes * 100, cells + ambient), nrow=ngenes)

# Pretending that all cells are in one group, in this example.
removed <- removeAmbience(y, ambient, groups=rep(1, ncol(y)))
summary(rowMeans(removed[1:100,]))
summary(rowMeans(removed[101:1000,]))
```

swappedDrops *Clean barcode-swapped droplet data*

Description

Remove the effects of barcode swapping on droplet-based single-cell RNA-seq data, specifically 10X Genomics datasets.

Usage

```
swappedDrops(samples, barcode.length = NULL, use.library = NULL, ...)

removeSwappedDrops(
  cells,
  umis,
  genes,
  nreads,
  ref.genes,
  min.frac = 0.8,
  get.swapped = FALSE,
  get.diagnostics = FALSE,
  hdf5.out = FALSE
)
```

Arguments

samples	A character vector containing paths to the molecule information HDF5 files, produced by CellRanger for 10X Genomics data. Each file corresponds to one sample in a multiplexed pool.
barcode.length	An integer scalar specifying the length of the cell barcode, see read10xMolInfo.
use.library	An integer scalar specifying the library index for which to extract molecules from sample. Alternatively, a string specifying the library type, e.g., "Gene expression".
...	Further arguments to be passed to removeSwappedDrops.
cells	A list of character vectors containing cell barcodes. Each vector corresponds to one sample in a multiplexed pool, and each entry of the vector corresponds to one molecule.
umis	A list of integer vectors containing encoded UMI sequences, organized as described for cells. See ?encodeSequences to convert sequences to integers.
genes	A list of integer vectors specifying the gene indices, organized as described for cells. Each index should refer to an element of ref.genes.
nreads	A list of integer vectors containing the number of reads per molecule, organized as described for cells.
ref.genes	A character vector containing the names or symbols of all genes.

`min.frac`	A numeric scalar specifying the minimum fraction of reads required for a swapped molecule to be assigned to a sample.
`get.swapped`	A logical scalar indicating whether the UMI counts corresponding to swapped molecules should be returned.
`get.diagnostics`	A logical scalar indicating whether to return the number of reads for each molecule in each sample.
`hdf5.out`	Deprecated and ignored.

Details

Barcode swapping on the Illumina sequencer occurs when multiplexed samples undergo PCR re-amplification on the flow cell by excess primer with different barcodes. This results in sequencing of the wrong sample barcode and molecules being assigned to incorrect samples after debarcoding. With droplet data, there is the opportunity to remove such effects based on the combination of gene, UMI and cell barcode for each observed transcript molecule. It is very unlikely that the same combination will arise from different molecules in multiple samples. Thus, observation of the same combination across multiple samples is indicative of barcode swapping.

We can remove swapped molecules based on the number of reads assigned to each gene-UMI-barcode combination. From the total number of reads assigned to that combination, the fraction of reads in each sample is calculated. The sample with the largest fraction that is greater than `min.frac` is defined as the putative sample of origin to which the molecule is assigned. This assumes that the swapping rate is low, so the sample of origin for a molecule should contain the majority of the reads. In other all samples, reads for the combination are assumed to derive from swapping and do not contribute to the count matrix. Setting `min.frac=1` will effectively remove all molecules that appear in multiple samples. We do not recommend setting `min.frac` lower than 0.5.

If `diagnostics=TRUE`, a diagnostics matrix is returned containing the number of reads per gene-UMI-barcode combination in each sample. Each row corresponds to a combination and each column corresponds to a sample. This can be useful for examining the level of swapping across samples on a molecule-by-molecule basis, though for the sake of memory, the actual identity of the molecules is not returned. By default, the matrix is returned as a HDF5Matrix, which reduces memory usage and avoids potential issues with integer overflow. If `hdf5.out=FALSE`, a sparse matrix is returned instead, which is faster but uses more memory.

`swappedDrops` is a wrapper around `removeSwappedDrops` that extracts the relevant data from the 10X Genomics molecule information file. For other types of droplet-based data, it may be more convenient to call `removeSwappedDrops` directly.

Value

A list is returned with the `cleaned` entry, itself a list of sparse matrices. Each matrix corresponds to a sample and contains the UMI count for each gene (row) and cell barcode (column) after removing swapped molecules. All cell barcodes that were originally observed are reported as columns, though note that it is possible for some barcodes to contain no counts.

If `get.swapped=TRUE`, a `swapped` entry is returned in the top-level list. This is a list containing sample-specific sparse matrices of UMI counts corresponding to the swapped molecules. Adding the cleaned and swapped matrices for each sample should yield the total UMI count prior to removal of swapped molecules.

If `get.diagnostics=TRUE`, the top-level list will also contain an additional `diagnostics` matrix.

Format of the molecule information file

`swappedDrops` makes a few assumptions about the nature of the data in each molecule information file. These are necessary to simplify downstream processing and are generally acceptable in most cases.

Each molecule information file should contain data from only a single 10X run. Users should *not* combine multiple samples into a single molecule information file. The function will emit a warning upon detecting multiple GEM groups from any molecule information file. Molecules with different GEM groups will not be recognised as coming from a different sample, though they will be recognised as being derived from different cell-level libraries.

In files produced by CellRanger version 3.0, an additional per-molecule field is present indicating the (c)DNA library from which the molecule was derived. Library preparation can be performed separately for different features (e.g., antibodies, CRISPR tags) such that one 10X run can contain data from multiple libraries. This allows for arbitrarily complicated multiplexing schemes - for example, gene expression libraries might be multiplexed together across one set of samples, while the antibody-derived libraries might be multiplexed across another *different* set of samples. For simplicity, we assume that multiplexing was performed across the same set of `samples` for all libraries therein.

If a different multiplexing scheme was applied for each library type, users can set `use.library` to only check for swapping within a given library type(s). For example, if the multiplexed set of samples for the gene expression libraries is different from the multiplexed set for the CRISPR libraries, one could run `swappedDrops` separately on each set of samples with `use.library` set to the corresponding type. This avoids having to take the union of both sets of samples for a single `swappedDrops` run, which could detect spurious swaps between samples that were never multiplexed together for the same library type.

Author(s)

Jonathan Griffiths, with modifications by Aaron Lun

References

Griffiths JA, Lun ATL, Richard AC, Bach K, Marioni JC (2018). Detection and removal of barcode swapping in single-cell RNA-seq data. *Nat. Commun.* 9, 1:2667.

See Also

`read10xMolInfo`, `encodeSequences`

Examples

```
# Mocking up some 10x HDF5-formatted data, with swapping.
curfiles <- DropletUtils:::simSwappedMolInfo(tempfile(), nsamples=3)

# Obtaining count matrices with swapping removed.
out <- swappedDrops(curfiles)
lapply(out$cleaned, dim)
```

```
out <- swappedDrops(curfiles, get.swapped=TRUE, get.diagnostics=TRUE)
names(out)
```

`write10xCounts` *Write count data in the 10x format*

Description

Create a directory containing the count matrix and cell/gene annotation from a sparse matrix of UMI counts, in the format produced by the CellRanger software suite.

Usage

```
write10xCounts(
  path,
  x,
  barcodes = colnames(x),
  gene.id = rownames(x),
  gene.symbol = gene.id,
  gene.type = "Gene Expression",
  overwrite = FALSE,
  type = c("auto", "mtx", "hdf5", "sparse", "HDF5"),
  genome = "unknown",
  version = c("2", "3"),
  chemistry = "Single Cell 3' v3",
  original.gem.groups = 1L,
  library.ids = "custom"
)
```

Arguments

`path`	A string containing the path to the output directory (for `type="mtx"`) or file (for `type="hdf5"`).
`x`	A sparse numeric matrix of UMI counts.
`barcodes`	A character vector of cell barcodes, one per column of `x`.
`gene.id`	A character vector of gene identifiers, one per row of `x`.
`gene.symbol`	A character vector of gene symbols, one per row of `x`.
`gene.type`	A character vector of gene types, expanded to one per row of `x`. Only used when `version="3"`.
`overwrite`	A logical scalar specifying whether `path` should be overwritten if it already exists.

type — String specifying the type of 10X format to save x to. This is either a directory containing a sparse matrix with row/column annotation ("mtx", or its older alias "sparse") or a HDF5 file containing the same information ("hdf5", or its older alias "HDF5").

genome — String specifying the genome for storage when type="hdf5". This can be a character vector with one genome per feature if version="3".

version — String specifying the version of the CellRanger format to produce.

chemistry, original.gem.groups, library.ids — Strings containing metadata attributes to be added to the HDF5 file for type="hdf5". Their interpretation is not formally documented and is left to the user's imagination.

Details

This function will try to automatically detect the desired format based on whether path ends with ".h5". If so, it assumes that path specifies a HDF5 file path and sets type="hdf5". Otherwise it will set type="mtx" under the assumption that path specifies a path to a directory.

Note that there were major changes in the output format for CellRanger version 3.0 to account for non-gene features such as antibody or CRISPR tags. Users can switch to this new format using version="3". See the documentation for "latest" for this new format, otherwise see "2.2" or earlier.

The primary purpose of this function is to create files to use for testing read10xCounts. In principle, it is possible to re-use the HDF5 matrices in cellranger reanalyze. We recommend against doing so routinely due to CellRanger's dependence on undocumented metadata attributes that may change without notice.

Value

For type="mtx", a directory is produced at path. If version="2", this will contain the files "matrix.mtx", "barcodes.tsv" and "genes.tsv". If version="3", it will instead contain "matrix.mtx.gz", "barcodes.tsv.gz" and "features.tsv.gz".

For type="hdf5", a HDF5 file is produced at path containing data in column-sparse format. If version="2", data are stored in the HDF5 group named genome. If version="3", data are stored in the group "matrix".

A TRUE value is invisibly returned.

Author(s)

Aaron Lun

References

10X Genomics (2017). Gene-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/output/matrices

10X Genomics (2018). Feature-Barcode Matrices. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/matrices

10X Genomics (2018). HDF5 Gene-Barcode Matrix Format. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/2.2/advanced/h5_matrices

10X Genomics (2018). HDF5 Feature Barcode Matrix Format. https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/advanced/h5_matrices

See Also

read10xCounts, to read CellRanger matrices into R.

Examples

```
# Mocking up some count data.
library(Matrix)
my.counts <- abs(rsparsematrix(100, 10, 0.2) * 10)
cell.ids <- paste0("BARCODE-", seq_len(ncol(my.counts)))

ngenes <- nrow(my.counts)
gene.ids <- paste0("ENSG0000", seq_len(ngenes))
gene.symb <- paste0("GENE", seq_len(ngenes))

# Writing this to file:
tmpdir <- tempfile()
write10xCounts(tmpdir, my.counts, gene.id=gene.ids,
    gene.symbol=gene.symb, barcodes=cell.ids)
list.files(tmpdir)

# Creating a version 3 HDF5 file:
tmph5 <- tempfile(fileext=".h5")
write10xCounts(tmph5, my.counts, gene.id=gene.ids,
    gene.symbol=gene.symb, barcodes=cell.ids, version='3')
```

Index

∗ **internal**
reexports, 47

ambientContribMaximum, 2, *7*, *9*, *14*, *48, 49*
ambientContribMaximum,ANY-method *(*ambientContribMaximum*)*, 2
ambientContribMaximum,SummarizedExperiment-method *(*ambientContribMaximum*)*, 2
ambientContribNegative, *5*, 5, *9*, *20*, *48, 49*
ambientContribNegative,ANY-method *(*ambientContribNegative*)*, 5
ambientContribNegative,SummarizedExperiment-method *(*ambientContribNegative*)*, 5
ambientContribSparse, *5*, *7*, 8, *11*, *20, 21*
ambientContribSparse,ANY-method *(*ambientContribSparse*)*, 8
ambientContribSparse,SummarizedExperiment-method *(*ambientContribSparse*)*, 8
ambientProfileBimodal, *5*, *7*, *9*, 10, *20*, *36*, *38*
ambientProfileBimodal,ANY-method *(*ambientProfileBimodal*)*, 10
ambientProfileBimodal,SummarizedExperiment-method *(*ambientProfileBimodal*)*, 10
ambientProfileEmpty, *5*, *7*, *10, 11*, 11, *12*, *26, 27*, *29*, *31*
ambientProfileEmpty,ANY-method *(*ambientProfileEmpty*)*, 11
ambientProfileEmpty,SummarizedExperiment-method *(*ambientProfileEmpty*)*, 11

barcodeRanks, 14, *26, 27*, *29*
barcodeRanks,ANY-method *(*barcodeRanks*)*, 14
barcodeRanks,SummarizedExperiment-method *(*barcodeRanks*)*, 14
BiocParallelParam, *3*, *8*, *12*, *15*, *26*, *31*, *43*, *49*

chimericDrops, 17
cleanTagCounts, *9*, 19
cleanTagCounts,ANY-method *(*cleanTagCounts*)*, 19
cleanTagCounts,SummarizedExperiment-method *(*cleanTagCounts*)*, 19
controlAmbience *(*ambientContribNegative*)*, 5

DataFrame, *16*, *18*, *20*, *32*, *37*, *46*, *48*
defaultDrops, 21, *29*, *32*
defaultDrops,ANY-method *(*defaultDrops*)*, 21
defaultDrops,SummarizedExperiment-method *(*defaultDrops*)*, 21
DelayedArray, *43*
dgCMatrix, *12*, *25*
dgTMatrix, *12*, *25*
downsampleBatches, *47*
downsampleBatches *(*reexports*)*, 47
downsampleMatrix, *24*, *47*
downsampleMatrix *(*reexports*)*, 47
downsampleReads, 23

emptyDrops, *4, 5*, *14*, *16*, *22*, 25, *28*, *32*, *37, 38*, *40*
emptyDrops,ANY-method *(*emptyDrops*)*, 25
emptyDrops,SummarizedExperiment-method *(*emptyDrops*)*, 25
emptyDropsCellRanger, 30
emptyDropsCellRanger,ANY-method *(*emptyDropsCellRanger*)*, 30
emptyDropsCellRanger,SummarizedExperiment-method *(*emptyDropsCellRanger*)*, 30
encodeSequences, *17*, 33, *50*, *52*
estimateAmbience, *49*
estimateAmbience *(*ambientProfileEmpty*)*, 11

get10xMolInfoStats, *24*, 34
goodTuringProportions, *27*

hashedDrops, *10, 11*, *14*, 35
hashedDrops,ANY-method *(*hashedDrops*)*, 35
hashedDrops,SummarizedExperiment-method *(*hashedDrops*)*, 35
HDF5Matrix, *51*

inferAmbience *(*ambientProfileBimodal*)*, 10
isOutlier, *20, 21*

makeCountMatrix, 40, *47*
maximumAmbience *(*ambientContribMaximum*)*, 2
medianSizeFactors, *10*
metadata, *38*

p.adjust, *28*

read10xCounts, 42, *54, 55*
read10xMolInfo, *17*, *23, 24*, *34, 35*, *41*, 45, *50*, *52*
RealizationSink, *49*
reexports, 47
removeAmbience, 48
removeAmbience,ANY-method *(*removeAmbience*)*, 48
removeAmbience,SummarizedExperiment-method *(*removeAmbience*)*, 48
removeChimericDrops *(*chimericDrops*)*, 17
removeSwappedDrops *(*swappedDrops*)*, 50
round, *13*

SingleCellExperiment, *42*, *44*
splitAltExps, *45*
sumCountsAcrossCells, *48*
SummarizedExperiment, *3*, *6*, *8*, *10*, *12*, *15*, *20*, *22*, *25*, *31*, *35*, *48*
swappedDrops, *18*, 50

TENxMatrix, *43*
testEmptyDrops *(*emptyDrops*)*, 25

which, *28*
write10xCounts, *45*, 53