Package 'scrapper'

November 26, 2024

Version 1.0.1

Date 2024-11-16

Title Bindings to C++ Libraries for Single-Cell Analysis

Description Implements R bindings to C++ code for analyzing single-cell (expression) data, mostly from various libscran libraries. Each function performs an individual step in the single-cell analysis workflow, ranging from quality control to clustering and marker detection. It is mostly intended for other Bioconductor package developers to build more user-friendly end-to-end workflows.

License MIT + file LICENSE

Imports methods, Rcpp, beachmat (>= 2.21.6), DelayedArray, BiocNeighbors (>= 1.99.0), igraph, parallel

Suggests testthat, knitr, rmarkdown, BiocStyle, MatrixGenerics, sparseMatrixStats, Matrix, scRNAseq

LinkingTo Rcpp, assorthead, beachmat, BiocNeighbors

biocViews Normalization, RNASeq, Software, GeneExpression, Transcriptomics, SingleCell, BatchEffect, QualityControl, DifferentialExpression, FeatureExtraction, PrincipalComponent, Clustering

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

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

git_branch RELEASE_3_20

git_last_commit ad62513

git_last_commit_date 2024-11-16

Repository Bioconductor 3.20

Date/Publication 2024-11-25

Author Aaron Lun [cre, aut]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

adt_quality_control . . . 2
aggregateAcrossCells . . . 4
buildSnnGraph . . . 5
centerSizeFactors . . . 7
chooseHighlyVariableGenes . . . 8
choosePseudoCount . . . 9
clusterGraph . . . 10
clusterKmeans . . . 11
combineFactors . . . 13
computeClrm1Factors . . . 14
correctMnn . . . 14
crispr_quality_control . . . 16
fitVarianceTrend . . . 18
modelGeneVariances . . . 19
normalizeCounts . . . 21
rna_quality_control . . . 22
runAllNeighborSteps . . . 24
runPca . . . 25
runTsne . . . 27
runUmap . . . 28
sanitizeSizeFactors . . . 29
scaleByNeighbors . . . 30
scoreGeneSet . . . 31
scoreMarkers . . . 33
subsampleByNeighbors . . . 35
summarizeEffects . . . 36

Index **38**

`adt_quality_control` *Quality control for ADT count data*

Description

Compute per-cell QC metrics from an initialized matrix of ADT counts, and use the metrics to suggest filter thresholds to retain high-quality cells.

Usage

```
computeAdtQcMetrics(x, subsets, num.threads = 1)

suggestAdtQcThresholds(
  metrics,
  block = NULL,
  min.detected.drop = 0.1,
  num.mads = 3
)

filterAdtQcMetrics(thresholds, metrics, block = NULL)
```

Arguments

x	A matrix-like object where rows are ADTs and columns are cells. Values are expected to be counts.
subsets	List of vectors specifying tag subsets of interest, typically control tags like IgGs. Each vector may be logical (whether to keep each row), integer (row indices) or character (row names).
num.threads	Integer scalar specifying the number of threads to use.
metrics	List with the same structure as produced by computeAdtQcMetrics.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in metrics. Alternatively NULL if all cells are from the same block. For filterAdtQcMetrics, a blocking factor should be provided if block was used to construct thresholds.
min.detected.drop	Minimum drop in the number of detected features from the median, in order to consider a cell to be of low quality.
num.mads	Number of median from the median, to define the threshold for outliers in each metric.
thresholds	List with the same structure as produced by suggestAdtQcThresholds.

Value

For computeAdtQcMetrics, a list is returned containing:

- sum, a numeric vector containing the total ADT count for each cell.
- detected, an integer vector containing the number of detected tags per cell.
- subsets, a list of numeric vectors containing the total count of each control subset.

Each vector is of length equal to the number of cells.

For suggestAdtQcThresholds with block!=NULL, a list is returned containing:

- detected, a numeric vector containing the lower bound on the number of detected tags for each blocking level.
- subsets, a list of numeric vectors containing the upper bound on the sum of counts in each control subset for each blocking level.

Each vector is of length equal to the number of levels in block.

For suggestAdtQcThresholds with block=NULL, a list is returned containing:

- detected, a numeric scalar containing the lower bound on the number of detected tags.
- subsets, a numeric vector containing the upper bound on the sum of counts in each control subset.

For filterAdtQcMetrics, a logical vector of length ncol(x) is returned indicating which cells are of high quality.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_qc/, for the rationale of QC filtering on ADT counts.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))

# Mocking up a control set.
sub <- list(IgG=rbinom(nrow(x), 1, 0.1) > 0)

qc <- computeAdtQcMetrics(x, sub)
str(qc)

filt <- suggestAdtQcThresholds(qc)
str(filt)

keep <- filterAdtQcMetrics(filt, qc)
summary(keep)
```

---

aggregateAcrossCells *Aggregate expression across cells*

Description

Aggregate expression values across cells based on one or more grouping factors. This is primarily used to create pseudo-bulk profiles for each cluster/sample combination.

Usage

```
aggregateAcrossCells(x, factors, num.threads = 1)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. Values are typically expected to be counts.
factors	A list or data frame containing one or more grouping factors, see combineFactors.
num.threads	Integer specifying the number of threads to be used for aggregation.

Value

A list containing:

- sums, a numeric matrix where each row corresponds to a gene and each column corresponds to a unique combination of grouping levels. Each entry contains the summed expression across all cells with that combination.
- detected, an integer matrix where each row corresponds to a gene and each column corresponds to a unique combination of grouping levels. Each entry contains the number of cells with detected expression in that combination.
- combinations, a data frame describing the levels for each unique combination of factors. Rows of this data frame correspond to columns of sums and detected, while columns correspond to the factors in factors.

- counts, the number of cells associated with each combination. Each entry corresponds to a row of combinations.
- index, an integer vector of length equal to the number of cells in x. This specifies the combination in combinations to which each cell was assigned.

Author(s)

Aaron Lun

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))

# Simple aggregation:
clusters <- sample(LETTERS, 100, replace=TRUE)
agg <- aggregateAcrossCells(x, list(cluster=clusters))
str(agg)

# Multi-factor aggregation
samples <- sample(1:5, 100, replace=TRUE)
agg2 <- aggregateAcrossCells(x, list(cluster=clusters, sample=samples))
str(agg2)
```

---

buildSnnGraph *Build a shared nearest neighbor graph*

Description

Build a shared nearest neighbor (SNN) graph where each node is a cell. Edges are formed between cells that share one or more nearest neighbors, weighted by the number or importance of those shared neighbors.

Usage

```
buildSnnGraph(
  x,
  num.neighbors = 10,
  weight.scheme = "ranked",
  num.threads = 1,
  BNPARAM = AnnoyParam()
)
```

Arguments

x	For buildSnnGraph, a numeric matrix where rows are dimensions and columns are cells, typically containing a low-dimensional representation from, e.g., runPca. Alternatively, a named list of nearest-neighbor search results. This should contain index, an integer matrix where rows are neighbors and columns are cells.

Each column contains 1-based indices for the nearest neighbors of the corresponding cell, ordered by increasing distance. The number of neighbors for each cell should be equal to `num.neighbors`, otherwise a warning is raised.

Alternatively, an index constructed by `buildIndex`.

`num.neighbors` Integer scalar specifying the number of neighbors to use to construct the graph.

`weight.scheme` String specifying the weighting scheme to use for constructing the SNN graph. This can be `"ranked"` (default), `"jaccard"` or `"number"`.

`num.threads` Integer scalar specifying the number of threads to use. Only used if `x` is not a list of existing nearest-neighbor search results.

`BNPARAM` A BiocNeighborParam object specifying the algorithm to use. Only used if `x` is not a list of existing nearest-neighbor search results.

Value

If `as.pointer=FALSE`, a list is returned containing:

- `vertices`, an integer scalar specifying the number of vertices in the graph (i.e., cells in `x`).
- `edges`, an integer vector of 1-based indices for graph edges. Pairs of values represent the endpoints of an (undirected) edge, i.e., `edges[1:2]` form the first edge, `edges[3:4]` form the second edge and so on.
- `weights`, a numeric vector of weights for each edge. This has length equal to half the length of `edges`.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_graph_cluster/, for details on the underlying implementation.

Examples

```
data <- matrix(rnorm(10000), ncol=1000)
out <- buildSnnGraph(data)
str(out)

# We can use this to make an igraph::graph.
g <- igraph::make_undirected_graph(out$edges, n = out$vertices)
igraph::E(g)$weight <- out$weight
```

centerSizeFactors *Center size factors*

Description

Scale the size factors so they are centered at unity, which ensures that the scale of the counts are preserved (on average) after normalization.

Usage

```
centerSizeFactors(size.factors, block = NULL, mode = c("lowest", "per-block"))
```

Arguments

size.factors	Numeric vector of size factors across cells.
block	Vector or factor of length equal to size.factors, specifying the block of origin for each cell. Alternatively NULL, in which case all cells are assumed to be in the same block.
mode	String specifying how to scale size factors across blocks. "lowest" will scale all size factors by the the lowest per-block average. "per-block" will center the size factors in each block separately.

Value

Numeric vector of length equal to size.factors, containing the centered size factors.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_norm/, for the rationale behind centering the size factors.

Examples

```
centerSizeFactors(runif(100))

centerSizeFactors(runif(100), block=sample(3, 100, replace=TRUE))
```

`chooseHighlyVariableGenes` *Choose highly variable genes*

Description

Choose highly variable genes based on a variance-related statistic.

Usage

```
chooseHighlyVariableGenes(stats, top = 4000, larger = TRUE, keep.ties = TRUE)
```

Arguments

`stats`	Numeric vector of variances (or a related statistic) across all genes. Typically the residuals from `modelGeneVariances` are used here.
`top`	Integer specifying the number of top genes to retain.
`larger`	Logical scalar indicating whether larger values of `stats` correspond to more variable genes.
`keep.ties`	Logical scalar indicating whether to keep tied values of `stats`, even if `top` may be exceeded.

Value

Integer vector containing the indices of genes in `stats` that are considered to be highly variable.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_variances/, for the underlying implementation.

Examples

```
resids <- rexp(1000)
str(chooseHighlyVariableGenes(resids))
```

`choosePseudoCount` *Choose a suitable pseudo-count*

Description

Choose a suitable pseudo-count to control the bias introduced by log-transformation of normalized counts.

Usage

```
choosePseudoCount(size.factors, quantile = 0.05, max.bias = 1, min.value = 1)
```

Arguments

`size.factors`	Numeric vector of size factors for all cells.
`quantile`	Numeric scalar specifying the quantile to use for defining extreme size factors.
`max.bias`	Numeric scalar specifying the maximum allowed bias.
`min.value`	Numeric scalar specifying the minimum value for the pseudo-count.

Value

A choice of pseudo-count for `normalizeCounts`.

Author(s)

Aaron Lun

See Also

`https://libscran.github.io/scran_norm/`, for the motivation behind calculating a larger pseudo-count.

Examples

```
sf <- runif(100)
choosePseudoCount(sf)
choosePseudoCount(sf, quantile=0.01)
choosePseudoCount(sf, max.bias=0.5)
```

clusterGraph *Graph-based clustering of cells*

Description

Identify clusters of cells using a variety of community detection methods from a graph where similar cells are connected.

Usage

```
clusterGraph(
  x,
  method = c("multilevel", "leiden", "walktrap"),
  multilevel.resolution = 1,
  leiden.resolution = 1,
  leiden.objective = c("modularity", "cpm"),
  walktrap.steps = 4,
  seed = 42
)
```

Arguments

x	List containing graph information or an external pointer to a graph, as returned by buildSnnGraph. Alternatively, an igraph object with edge weights.
method	String specifying the algorithm to use.
multilevel.resolution	Numeric scalar specifying the resolution when method="multilevel".
leiden.resolution	Numeric scalar specifying the resolution when method="leiden".
leiden.objective	String specifying the objective function when method="leiden".
walktrap.steps	Integer scalar specifying the number of steps to use when method="walktrap".
seed	Integer scalar specifying the random seed to use for method="multilevel" or "leiden".

Value

A list containing membership, an integer vector containing the cluster assignment for each cell; and status, an integer scalar indicating whether the algorithm completed successfully (0) or not (non-zero). Additional fields may be present depending on the method:

- For method="multilevel", the levels list contains the clustering result at each level of the algorithm. A modularity numeric vector also contains the modularity at each level, the highest of which corresponds to the reported membership.
- For method="leiden", a quality numeric scalar containg the quality of the partitioning.
- For method="walktrap", a merges matrix specifies the pair of cells or clusters that were merged at each step of the algorithm. A modularity numeric scalar also contains the modularity of the final partitioning.

Author(s)

Aaron Lun

See Also

https://igraph.org, for the underlying implementation of each clustering method.

https://libscran.github.io/scran_graph_cluster/, for wrappers around the **igraph** code.

Examples

```
data <- matrix(rnorm(10000), ncol=1000)
gout <- buildSnnGraph(data)
str(gout)

str(clusterGraph(gout))
str(clusterGraph(gout, method="leiden"))
str(clusterGraph(gout, method="walktrap"))
```

---

clusterKmeans *K-means clustering*

Description

Perform k-means clustering using kmeans++ initialization with the Hartigan-Wong algorithm.

Usage

```
clusterKmeans(
  x,
  k,
  init.method = c("var-part", "kmeans++", "random"),
  refine.method = c("hartigan-wong", "lloyd"),
  var.part.optimize.partition = TRUE,
  var.part.size.adjustment = 1,
  lloyd.iterations = 100,
  hartigan.wong.iterations = 10,
  hartigan.wong.quick.transfer.iterations = 50,
  hartigan.wong.quit.quick.transfer.failure = FALSE,
  seed = 5489L,
  num.threads = 1
)
```

Arguments

x	Numeric matrix where rows are dimensions and columns are cells.
k	Integer scalar specifying the number of clusters.
init.method	String specifying the initialization method: variance partitioning ("var-part"), kmeans++ ("kmeans++") or random initialization ("random").
refine.method	String specifying the refinement method: Lloyd's algorithm ("lloyd") or the Hartigan-Wong algorithm ("hartigan-wong").

`var.part.optimize.partition`
: Logical scalar indicating whether each partition boundary should be optimized to reduce the sum of squares in the child partitions.

`var.part.size.adjustment`
: Numeric scalar between 0 and 1, specifying the adjustment to the cluster size when prioritizing the next cluster to partition. Setting this to 0 will ignore the cluster size while setting this to 1 will generally favor larger clusters.

`lloyd.iterations`
: Integer scalar specifying the maximmum number of iterations for the Lloyd algorithm.

`hartigan.wong.iterations`
: Integer scalar specifying the maximmum number of iterations for the Hartigan-Wong algorithm.

`hartigan.wong.quick.transfer.iterations`
: Integer scalar specifying the maximmum number of quick transfer iterations for the Hartigan-Wong algorithm.

`hartigan.wong.quit.quick.transfer.failure`
: Logical scalar indicating whether to quit the Hartigan-Wong algorithm upon convergence failure during quick transfer iterations.

`seed`
: Integer scalar specifying the seed to use for random or kmeans++ initialization.

`num.threads`
: Integer scalar specifying the number of threads to use.

Value

By default, a list is returned containing:

- `clusters`, a factor containing the cluster assignments for each cell.
- `centers`, a numeric matrix with the coordinates of the cluster centroids (dimensions in rows, centers in columns).
- `iterations`, an integer scalar specifying the number of refinement iterations that were performed
- `status`, an integer scalar specifying the convergence status. Any non-zero value indicates a convergence failure though the exact meaning depends on the choice of `refine.method`.

Author(s)

Aaron Lun

Examples

```
x <- t(as.matrix(iris[,1:4]))
clustering <- clusterKmeans(x, k=3)
table(clustering$clusters, iris[,"Species"])
```

`combineFactors` *Combine multiple factors*

Description

Combine multiple categorical factors based on the unique combinations of levels from each factor.

Usage

```
combineFactors(factors, keep.unused = FALSE)
```

Arguments

`factors`	List of vectors or factors of the same length. Corresponding elements across all vectors/factors represent the combination of levels for a single observation. For factors, any existing levels are respected. For other vectors, the sorted and unique values are used as levels. Alternatively, a data frame where each column is a vector or factor and each row corresponds to an observation.
`keep.unused`	Logical scalar indicating whether to report unused combinations of levels.

Value

List containing `levels`, a data frame containing the sorted and unique combinations of levels from `factors`; and `index`, an integer vector specifying the index into `levels` for each observation.

Author(s)

Aaron Lun

Examples

```
combineFactors(list(
    sample(LETTERS[1:5], 100, replace=TRUE),
    sample(3, 100, replace=TRUE)
))

combineFactors(list(
    factor(sample(LETTERS[1:5], 10, replace=TRUE), LETTERS[1:5]),
    factor(sample(5, 10, replace=TRUE), 1:5)
), keep.unused=TRUE)
```

`computeClrm1Factors` *Compute size factors for ADT counts*

Description

Compute size factors from an ADT count matrix using the CLRm1 method.

Usage

```
computeClrm1Factors(x, num.threads = 1)
```

Arguments

`x`	A matrix-like object containing ADT count data. Rows correspond to tags and columns correspond to cells.
`num.threads`	Number of threads to use.

Value

Numeric vector containing the CLRm1 size factor for each cell.

Author(s)

Aaron Lun

See Also

https://github.com/libscran/clrm1, for a description of the CLRm1 method.

Examples

```
library(Matrix)
x <- abs(rsparsematrix(1000, 100, 0.1) * 10)
head(computeClrm1Factors(x))
```

`correctMnn` *Batch correction with mutual nearest neighbors*

Description

Apply mutual nearest neighbor (MNN) correction to remove batch effects from a low-dimensional matrix.

Usage

```
correctMnn(
  x,
  block,
  num.neighbors = 15,
  num.mads = 3,
  robust.iterations = 2,
  robust.trim = 0.25,
  mass.cap = NULL,
  order = NULL,
  reference.policy = c("max-rss", "max-size", "max-variance", "input"),
  BNPARAM = AnnoyParam(),
  num.threads = 1
)
```

Arguments

x	Numeric matrix where rows are dimensions and columns are cells, typically containing low-dimensional coordinates (e.g., from runPca).
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in x.
num.neighbors	Integer scalar specifying the number of neighbors to use when identifying MNN pairs.
num.mads	Numeric scalar specifying the number of median absolute deviations to use for removing outliers in the center-of-mass calculations.
robust.iterations	Integer scalar specifying the number of iterations for robust calculation of the center of mass.
robust.trim	Numeric scalar in [0, 1) specifying the trimming proportion for robust calculation of the center of mass.
mass.cap	Integer scalar specifying the cap on the number of observations to use for center-of-mass calculations on the reference dataset. A value of 100,000 may be appropriate for speeding up correction of very large datasets. If NULL, no cap is used.
order	Vector containing levels of batch in the desired merge order. If NULL, a suitable merge order is automatically determined.
reference.policy	String specifying the policy to use to choose the first reference batch. This can be based on the largest batch ("max-size"), the most variable batch ("max-variance"), the batch with the largest residual sum of squares ("max-rss"), or the first specified input ("input"). Only used for automatic merges, i.e., when order=NULL.
BNPARAM	A BiocNeighborParam object specifying the nearest-neighbor algorithm to use.
num.threads	Integer scalar specifying the number of threads to use.

Value

List containing:

- corrected, a numeric matrix of the same dimensions as x, containing the corrected values.

- merge.order, character vector containing the unique levels of batch in the automatically determined merge order. The first level in this vector is used as the reference batch; all other batches are iteratively merged to it.
- num.pairs, integer vector of length equal to the number of batches minus 1. This contains the number of MNN pairs at each merge.

Author(s)

Aaron Lun

Examples

```
# Mocking up a dataset with multiple batches.
x <- matrix(rnorm(10000), nrow=10)
b <- sample(3, ncol(x), replace=TRUE)
x[,b==2] <- x[,b==2] + 3
x[,b==3] <- x[,b==3] + 5
lapply(split(colMeans(x), b), mean) # indeed the means differ...

corrected <- correctMnn(x, b)
str(corrected)
lapply(split(colMeans(corrected$corrected), b), mean) # now merged.
```

---

crispr_quality_control

Quality control for CRISPR count data

Description

Compute per-cell QC metrics from an initialized matrix of CRISPR counts, and use the metrics to suggest filter thresholds to retain high-quality cells.

Usage

```
computeCrisprQcMetrics(x, num.threads = 1)

suggestCrisprQcThresholds(
  metrics,
  block = NULL,
  min.detected.drop = 0.1,
  num.mads = 3
)

filterCrisprQcMetrics(thresholds, metrics, block = NULL)
```

Arguments

x	A matrix-like object where rows are CRISPRs and columns are cells. Values are expected to be counts.
num.threads	Integer scalar specifying the number of threads to use.
metrics	List with the same structure as produced by computeCrisprQcMetrics.

block — Factor specifying the block of origin (e.g., batch, sample) for each cell in metrics. Alternatively NULL if all cells are from the same block.
For filterCrisprQcMetrics, a blocking factor should be provided if block was used to construct thresholds.

min.detected.drop — Minimum drop in the number of detected features from the median, in order to consider a cell to be of low quality.

num.mads — Number of median from the median, to define the threshold for outliers in each metric.

thresholds — List with the same structure as produced by suggestCrisprQcThresholds.

Value

For computeCrisprQcMetrics, a list is returned containing:

- sum, a numeric vector containing the total CRISPR count for each cell.
- detected, an integer vector containing the number of detected guides per cell.
- max.value, a numeric vector containing the count for the most abundant guide in cell.
- max.index, an integer vector containing the row index of the most abundant guide in cell.

Each vector is of length equal to the number of cells.

For suggestCrisprQcThresholds with block!=NULL, a list is returned containing:

- max.value, a numeric vector containing the lower bound on the maximum counts for each blocking level.

Each vector is of length equal to the number of levels in block.

For suggestCrisprQcThresholds with block=NULL, a list is returned containing:

- max.value, a numeric scalar containing the lower bound on the maximum counts for each blocking level.

For filterCrisprQcMetrics, a logical vector of length ncol(x) is returned indicating which cells are of high quality.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_qc/, for the rationale of QC filtering on CRISPR counts.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(100, 100, 0.1) * 100))

qc <- computeCrisprQcMetrics(x)
str(qc)

filt <- suggestCrisprQcThresholds(qc)
str(filt)
```

```
keep <- filterCrisprQcMetrics(filt, qc)
summary(keep)
```

`fitVarianceTrend` *Fit a mean-variance trend*

Description

Fit a trend to the per-cell variances with respect to the mean.

Usage

```
fitVarianceTrend(
  means,
  variances,
  mean.filter = TRUE,
  min.mean = 0.1,
  transform = TRUE,
  span = 0.3,
  use.min.width = FALSE,
  min.width = 1,
  min.window.count = 200,
  num.threads = 1
)
```

Arguments

`means`	Numeric vector containing the mean (log-)expression for each gene.
`variances`	Numeric vector containing the variance in the (log-)expression for each gene.
`mean.filter`	Logical scalar indicating whether to filter on the means before trend fitting.
`min.mean`	Numeric scalar specifying the minimum mean of genes to use in trend fitting. Only used if `mean.filter=TRUE`.
`transform`	Logical scalar indicating whether a quarter-root transformation should be applied before trend fitting.
`span`	Numeric scalar specifying the span of the LOWESS smoother. Ignored if `use.min.width=TRUE`.
`use.min.width`	Logical scalar indicating whether a minimum width constraint should be applied to the LOWESS smoother. Useful to avoid overfitting in high-density intervals.
`min.width`	Minimum width of the window to use when `use.min.width=TRUE`.
`min.window.count`	Minimum number of observations in each window. Only used if `use.min.width=TRUE`.
`num.threads`	Number of threads to use.

Value

List containing `fitted`, the fitted values of the trend for each gene; and `residuals`, the residuals from the trend.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_variances/, for the underlying implementation.

Examples

```
x <- runif(1000)
y <- 2^rnorm(1000)
out <- fitVarianceTrend(x, y)

plot(x, y)
o <- order(x)
lines(x[o], out$fitted[o], col="red")
```

---

modelGeneVariances *Model per-gene variances in expression*

Description

Compute the variance in (log-)expression values for each gene, and model the trend in the variances with respect to the mean.

Usage

```
modelGeneVariances(
  x,
  block = NULL,
  block.weight.policy = c("variable", "equal", "none"),
  variable.block.weight = c(0, 1000),
  mean.filter = TRUE,
  min.mean = 0.1,
  transform = TRUE,
  span = 0.3,
  use.min.width = FALSE,
  min.width = 1,
  min.window.count = 200,
  num.threads = 1
)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. It is typically expected to contain log-expression values, e.g., from normalizeCounts.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in x. Alternatively NULL if all cells are from the same block.

`block.weight.policy`
: String specifying the policy to use for weighting different blocks when computing the average for each statistic Only used if `block` is not `NULL`.

`variable.block.weight`
: Numeric vector of length 2, specifying the parameters for variable block weighting. The first and second values are used as the lower and upper bounds, respectively, for the variable weight calculation. Only used if `block` is not `NULL` and `block.weight.policy = "variable"`.

`mean.filter`
: Logical scalar indicating whether to filter on the means before trend fitting.

`min.mean`
: Numeric scalar specifying the minimum mean of genes to use in trend fitting. Only used if `mean.filter=TRUE`.

`transform`
: Logical scalar indicating whether a quarter-root transformation should be applied before trend fitting.

`span`
: Numeric scalar specifying the span of the LOWESS smoother. Ignored if `use.min.width=TRUE`.

`use.min.width`
: Logical scalar indicating whether a minimum width constraint should be applied to the LOWESS smoother. Useful to avoid overfitting in high-density intervals.

`min.width`
: Minimum width of the window to use when `use.min.width=TRUE`.

`min.window.count`
: Minimum number of observations in each window. Only used if `use.min.width=TRUE`.

`num.threads`
: Integer scalar specifying the number of threads to use.

Value

A list containing `statistics`. This is a data frame with the columns `means`, `variances`, `fitted` and `residuals`, each of which is a numeric vector containing the statistic of the same name across all genes.

If `block` is supplied, each of the column vectors described above contains the average across all blocks. The list will also contain `per.block`, a list of data frames containing the equivalent statistics for each block.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_variances/, for the variance modelling.

https://libscran.github.io/scran_blocks/, for details on the blocking.

`fitVarianceTrend`, which fits the mean-variance trend.

Examples

```
library(Matrix)
x <- abs(rsparsematrix(1000, 100, 0.1) * 10)
out <- modelGeneVariances(x)
str(out)

# Throwing in some blocking.
block <- sample(letters[1:4], ncol(x), replace=TRUE)
out <- modelGeneVariances(x, block=block)
str(out)
```

normalizeCounts *Normalize the count matrix*

Description

Apply scaling normalization to a count matrix to obtain log-transformed normalized expression values.

Usage

```
normalizeCounts(
  x,
  size.factors,
  log = TRUE,
  pseudo.count = 1,
  log.base = 2,
  preserve.sparsity = FALSE
)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. Values are typically expected to be counts. Alternatively, an external pointer created by initializeCpp.
size.factors	A numeric vector of length equal to the number of cells in x, containing positive size factors for all cells.
log	Logical scalar indicating whether log-transformation should be performed.
pseudo.count	Numeric scalar specifying the positive pseudo-count to add before any log-transformation. Ignored if log=FALSE.
log.base	Numeric scalar specifying the base of the log-transformation. Ignored if log=FALSE.
preserve.sparsity	Logical scalar indicating whether to preserve sparsity for pseudo.count != 1. If TRUE, users should manually add log(pseudo.count, log.base) to the returned matrix to obtain the desired log-transformed expression values.

Value

If x is a matrix-like object, a DelayedArray is returned containing the (log-transformed) normalized expression matrix.

If x is an external pointer produced by initializeCpp, a new external pointer is returned containing the normalized expression matrix.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_norm/, for the rationale behind normalization.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))
sf <- centerSizeFactors(colSums(x))
normed <- normalizeCounts(x, size.factors=sf)
normed

# Passing a pointer.
ptr <- beachmat::initializeCpp(x)
optr <- normalizeCounts(ptr, sf)
optr
```

---

rna_quality_control *Quality control for RNA count data*

Description

Compute per-cell QC metrics from an initialized matrix of RNA counts, and use the metrics to suggest filter thresholds to retain high-quality cells.

Usage

```
computeRnaQcMetrics(x, subsets, num.threads = 1)

suggestRnaQcThresholds(metrics, block = NULL, num.mads = 3)

filterRnaQcMetrics(thresholds, metrics, block = NULL)
```

Arguments

x	A matrix-like object where rows are genes and columns are cells. Values are expected to be counts.
subsets	List of vectors specifying gene subsets of interest, typically for control-like features like mitochondrial genes or spike-in transcripts. Each vector may be logical (whether to keep each row), integer (row indices) or character (row names).
num.threads	Integer scalar specifying the number of threads to use.
metrics	List with the same structure as produced by computeRnaQcMetrics.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in metrics. Alternatively NULL if all cells are from the same block. For filterRnaQcMetrics, a blocking factor should be provided if block was used to construct thresholds.
num.mads	Number of median from the median, to define the threshold for outliers in each metric.
thresholds	List with the same structure as produced by suggestRnaQcThresholds.

Value

For `computeRnaQcMetrics`, a list is returned containing:

- `sum`, a numeric vector containing the total RNA count for each cell.
- `detected`, an integer vector containing the number of detected genes per cell.
- `subsets`, a list of numeric vectors containing the proportion of counts in each feature subset.

Each vector is of length equal to the number of cells.

For `suggestRnaQcThresholds` with `block!=NULL`, a list is returned containing:

- `sum`, a numeric vector containing the lower bound on the sum for each blocking level.
- `detected`, a numeric vector containing the lower bound on the number of detected genes for each blocking level.
- `subsets`, a list of numeric vectors containing the upper bound on the sum of counts in each feature subset for each blocking level.

Each vector is of length equal to the number of levels in `block`.

For `suggestRnaQcThresholds` with `block=NULL`, a list is returned containing:

- `sum`, a numeric scalar containing the lower bound on the sum.
- `detected`, a numeric scalar containing the lower bound on the number of detected genes.
- `subsets`, a numeric vector containing the upper bound on the sum of counts in each feature subset.

For `filterRnaQcMetrics`, a logical vector of length `ncol(x)` is returned indicating which cells are of high quality.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_qc/, for the rationale of QC filtering on RNA counts.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))

# Mocking up a control set.
sub <- list(mito=rbinom(nrow(x), 1, 0.1) > 0)

qc <- computeRnaQcMetrics(x, sub)
str(qc)

filt <- suggestRnaQcThresholds(qc)
str(filt)

keep <- filterRnaQcMetrics(filt, qc)
summary(keep)
```

`runAllNeighborSteps` *Run all neighbor-related steps*

Description

Run all steps that require a nearest-neighbor search. This includs `runUmap`, `runTsne` and `buildSnnGraph` with `clusterGraph`. The idea is to build the index once, perform the neighbor search, and run each task in parallel to save time.

Usage

```
runAllNeighborSteps(
  x,
  runUmap.args = list(),
  runTsne.args = list(),
  buildSnnGraph.args = list(),
  clusterGraph.args = list(),
  BNPARAM = AnnoyParam(),
  collapse.search = FALSE,
  num.threads = 3
)
```

Arguments

`x`	Numeric matrix where rows are dimensions and columns are cells, typically containing a low-dimensional representation from, e.g., `runPca`. Alternatively, an index constructed by `buildIndex`.
`runUmap.args`	Named list of further arguments to pass to `runUmap`. This can be set to `NULL` to omit the UMAP.
`runTsne.args`	Named list of further arguments to pass to `runTsne`. This can be set to `NULL` to omit the t-SNE.
`buildSnnGraph.args`	Named list of further arguments to pass to `buildSnnGraph`.
`clusterGraph.args`	Named list of further arguments to pass to `clusterGraph`. This can be set to `NULL` to omit the clustering.
`BNPARAM`	A BiocNeighborParam instance specifying the nearest-neighbor search algorithm to use.
`collapse.search`	Logical scalar indicating whether to collapse the nearest-neighbor search for each step into a single search. Steps that need fewer neighbors will take a subset of the neighbors from the collapsed search. This is faster but may not give the same results as separate searches for some algorithms (e.g., approximate searches).
`num.threads`	Integer scalar specifying the number of threads to use. At least one thread should be available for each step.

Value

A named list containing the results of each step. See each individual function for the format of the results.

Author(s)

Aaron Lun

Examples

```
x <- t(as.matrix(iris[,1:4]))
# (Turning down the number of threads so that R CMD check is happy.)
res <- runAllNeighborSteps(x, num.threads=2)
str(res)
```

---

runPca *Principal components analysis*

Description

Run a PCA on the gene-by-cell log-expression matrix to obtain a low-dimensional representation for downstream analyses.

Usage

```
runPca(
  x,
  number = 25,
  scale = FALSE,
  block = NULL,
  block.weight.policy = c("variable", "equal", "none"),
  variable.block.weight = c(0, 1000),
  components.from.residuals = FALSE,
  extra.work = 7,
  iterations = 1000,
  seed = 5489,
  realized = TRUE,
  num.threads = 1
)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. Typically, the matrix is expected to contain log-expression values, and the rows should be filtered to relevant (e.g., highly variable) genes.
number	Integer scalar specifying the number of PCs to retain.
scale	Logical scalar indicating whether to scale all genes to have the same variance.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in x. Alternatively NULL if all cells are from the same block.
block.weight.policy	String specifying the policy to use for weighting different blocks when computing the average for each statistic Only used if block is not NULL.

variable.block.weight
: Numeric vector of length 2, specifying the parameters for variable block weighting. The first and second values are used as the lower and upper bounds, respectively, for the variable weight calculation. Only used if block is not NULL and block.weight.policy = "variable".

components.from.residuals
: Logical scalar indicating whether to compute the PC scores from the residuals in the presence of a blocking factor. By default, the residuals are only used to compute the rotation matrix, and the original expression values of the cells are projected onto this new space. Only used if block is not NULL.

extra.work
: Integer scalar specifying the extra dimensions for the IRLBA workspace.

iterations
: Integer scalar specifying the maximum nunber of restart iterations for IRLBA.

seed
: Integer scalar specifying the seed for the initial random vector in IRLBA.

realized
: Logical scalar indicating whether to realize x into an optimal memory layout for IRLBA. This improves computation time at the cost of increased memory usage.

num.threads
: Number of threads to use.

Value

List containing:

- components, a matrix of PC scores. Rows are dimensions (i.e., PCs) and columns are cells.
- rotation, the rotation matrix. Rows are genes and columns are dimensions.
- variance.explained, the vector of variances explained by each PC.
- total.variance, the total variance in the dataset.
- center, a numeric vector containing the mean for each gene. If block is provided, this is instead a matrix containing the mean for each gene (column) in each block (row).
- scale, a numeric vector containing the scaling for each gene. Only reported if scale=TRUE.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_pca/, for more details on the PCA.

https://libscran.github.io/scran_blocks/, for more details on the block weighting.

Examples

```
library(Matrix)
x <- abs(rsparsematrix(1000, 100, 0.1) * 10)
y <- normalizeCounts(x, size.factors=centerSizeFactors(colSums(x)))

# A simple PCA:
out <- runPca(y)
str(out)

# Blocking on uninteresting factors:
block <- sample(LETTERS[1:3], ncol(y), replace=TRUE)
bout <- runPca(y, block=block)
str(bout)
```

runTsne *t-stochastic neighbor embedding*

Description

Compute t-SNE coordinates to visualize similarities between cells.

Usage

```
runTsne(
  x,
  perplexity = 30,
  num.neighbors = tsnePerplexityToNeighbors(perplexity),
  max.depth = 20,
  leaf.approximation = FALSE,
  max.iterations = 500,
  seed = 42,
  num.threads = 1,
  BNPARAM = AnnoyParam()
)

tsnePerplexityToNeighbors(perplexity)
```

Arguments

x	Numeric matrix where rows are dimensions and columns are cells, typically containing a low-dimensional representation from, e.g., runPca. Alternatively, a named list of nearest-neighbor search results. This should contain index, an integer matrix where rows are neighbors and columns are cells. Each column contains 1-based indices for the nearest neighbors of the corresponding cell, ordered by increasing distance. The number of neighbors should be the same as num.neighbors, otherwise a warning is raised. Alternatively, an index constructed by buildIndex.
perplexity	Numeric scalar specifying the perplexity to use in the t-SNE algorithm.
num.neighbors	Integer scalar specifying the number of neighbors, typically derived from perplexity.
max.depth	Integer scalar specifying the maximum depth of the Barnes-Hut quadtree. Smaller values (7-10) improve speed at the cost of accuracy.
leaf.approximation	Logical scalar indicating whether to use the "leaf approximation" approach, which sacrifices some accuracy for greater speed. Only effective when max.depth is small enough for multiple cells to be assigned to the same leaf node of the quadtree.
max.iterations	Integer scalar specifying the maximum number of iterations to perform.
seed	Integer scalar specifying the seed to use for generating the initial coordinates.
num.threads	Integer scalar specifying the number of threads to use.
BNPARAM	A BiocNeighborParam object specifying the algorithm to use. Only used if x is not a list of existing nearest-neighbor search results.

Value

For `runTsne`, a numeric matrix where rows are cells and columns are the two dimensions of the embedding.

For `tsnePerplexityToNeighbors`, an integer scalar specifying the number of neighbors to use for a given `perplexity`.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/qdtsne/, for an explanation of the approximations.

Examples

```
x <- t(as.matrix(iris[,1:4]))
embedding <- runTsne(x)
plot(embedding[,1], embedding[,2], col=iris[,5])
```

---

runUmap *Uniform manifold approxation and projection*

Description

Compute UMAP coordinates to visualize similarities between cells.

Usage

```
runUmap(
  x,
  num.dim = 2,
  num.neighbors = 15,
  num.epochs = -1,
  min.dist = 0.1,
  seed = 1234567890,
  num.threads = 1,
  parallel.optimization = FALSE,
  BNPARAM = AnnoyParam()
)
```

Arguments

x	Numeric matrix where rows are dimensions and columns are cells, typically containing a low-dimensional representation from, e.g., `runPca`. Alternatively, a named list of nearest-neighbor search results. This should contain `index`, an integer matrix where rows are neighbors and columns are cells. Each column contains 1-based indices for the nearest neighbors of the corresponding cell, ordered by increasing distance. The number of neighbors should be the same as `num.neighbors`, otherwise a warning is raised. Alternatively, an index constructed by `buildIndex`.

num.dim	Integer scalar specifying the number of dimensions of the output embedding.
num.neighbors	Integer scalar specifying the number of neighbors to use in the UMAP algorithm.
num.epochs	Integer scalar specifying the number of epochs to perform. If set to -1, an appropriate number of epochs is chosen based on ncol(x).
min.dist	Numeric scalar specifying the minimum distance between points.
seed	Integer scalar specifying the seed to use.
num.threads	Integer scalar specifying the number of threads to use.
parallel.optimization	Logical scalar specifying whether to parallelize the optimization step.
BNPARAM	A BiocNeighborParam object specifying the algorithm to use. Only used if x is not a list of existing nearest-neighbor search results.

Value

A numeric matrix where rows are cells and columns are the two dimensions of the embedding.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/umappp/, for details on the underlying implementation.

Examples

```
x <- t(as.matrix(iris[,1:4]))
embedding <- runUmap(x)
plot(embedding[,1], embedding[,2], col=iris[,5])
```

---

sanitizeSizeFactors *Sanitize size factors*

Description

Replace invalid size factors, i.e., zero, negative, infinite or NaN values.

Usage

```
sanitizeSizeFactors(
  size.factors,
  replace.zero = TRUE,
  replace.negative = TRUE,
  replace.infinite = TRUE,
  replace.nan = TRUE
)
```

Arguments

size.factors	Numeric vector of size factors across cells.
replace.zero	Logical scalar indicating whether to replace size factors of zero with the lowest positive factor. If FALSE, zeros are retained.
replace.negative	Logical scalar indicating whether to replace negative size factors with the lowest positive factor. If FALSE, negative values are retained.
replace.infinite	Logical scalar indicating whether to replace infinite size factors with the largest positive factor. If FALSE, infinite values are retained.
replace.nan	Logical scalar indicating whether to replace NaN size factors with unity. If FALSE, NaN values are retained.

Value

Numeric vector of length equal to size.factors, containing the sanitized size factors.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_norm/, for more details on the sanitization.

Examples

```
sf <- 2^rnorm(100)
sf[1] <- 0
sf[2] <- -1
sf[3] <- Inf
sf[4] <- NaN
sanitizeSizeFactors(sf)
```

---

scaleByNeighbors *Scale and combine multiple embeddings*

Description

Scale multiple embeddings (usually derived from different modalities across the same set of cells) so that their within-population variances are comparable, and then combine them into a single embedding matrix for combined downstream analysis.

Usage

```
scaleByNeighbors(
  x,
  num.neighbors = 20,
  num.threads = 1,
  weights = NULL,
  BNPARAM = AnnoyParam()
)
```

Arguments

x	List of numeric matrices of principal components or other embeddings, one for each modality. For each entry, rows are dimensions and columns are cells. All entries should have the same number of columns but may have different numbers of rows.
num.neighbors	Integer scalar specifying the number of neighbors to use to define the scaling factor.
num.threads	Integer scalar specifying the number of threads to use.
weights	Numeric vector of length equal to that of x, specifying the weights to apply to each modality. Each value represents a multiplier of the within-population variance of its modality, i.e., larger values increase the contribution of that modality in the combined output matrix. NULL is equivalent to an all-1 vector, i.e., all modalities are scaled to have the same within-population variance.
BNPARAM	A BiocNeighborParam object specifying how to perform the neighbor search.

Value

List containing scaling, a vector of scaling factors to be aplied to each embedding; and combined, a numeric matrix formed by scaling each entry of x and then rbinding them together.

Author(s)

Aaron Lun

Examples

```
pcs <- list(
    gene = matrix(rnorm(10000), ncol=200),
    protein = matrix(rnorm(1000, sd=3), ncol=200),
    guide = matrix(rnorm(2000, sd=5), ncol=200)
)

out <- scaleByNeighbors(pcs)
out$scaling
dim(out$combined)
```

---

scoreGeneSet *Score gene set activity for each cell*

Description

Compute per-cell scores for a gene set, defined as the column sums of a rank-1 approximation to the submatrix for the feature set. This uses the same approach implemented in the **GSDecon** package from Jason Hackney.

Usage

```
scoreGeneSet(
  x,
  set,
  rank = 1,
  scale = FALSE,
  block = NULL,
  block.weight.policy = c("variable", "equal", "none"),
  variable.block.weight = c(0, 1000),
  extra.work = 7,
  iterations = 1000,
  seed = 5489,
  realized = TRUE,
  num.threads = 1
)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. Typically, the matrix is expected to contain log-expression values, and the rows should be filtered to relevant (e.g., highly variable) genes.
set	Integer, logical or character vector specifying the rows that belong to the gene set.
rank	Integer scalar specifying the rank of the approximation.
scale	Logical scalar indicating whether to scale all genes to have the same variance.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in x. Alternatively NULL if all cells are from the same block.
block.weight.policy	String specifying the policy to use for weighting different blocks when computing the average for each statistic Only used if block is not NULL.
variable.block.weight	Numeric vector of length 2, specifying the parameters for variable block weighting. The first and second values are used as the lower and upper bounds, respectively, for the variable weight calculation. Only used if block is not NULL and block.weight.policy = "variable".
extra.work	Integer scalar specifying the extra dimensions for the IRLBA workspace.
iterations	Integer scalar specifying the maximum nunber of restart iterations for IRLBA.
seed	Integer scalar specifying the seed for the initial random vector in IRLBA.
realized	Logical scalar indicating whether to realize x into an optimal memory layout for IRLBA. This improves computation time at the cost of increased memory usage.
num.threads	Number of threads to use.

Value

List containing scores, a numeric vector of per-cell scores for each column in x; and weights, a numeric vector of per-feature weights for each feature in set.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/gsdecon/, for more details on the underlying algorithm.

Examples

```
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))
normed <- normalizeCounts(x, size.factors=centerSizeFactors(colSums(x)))
scoreGeneSet(normed, set=c(1,3,5,10,20,100))
```

---

scoreMarkers *Score marker genes*

Description

Score marker genes for each group using a variety of effect sizes from pairwise comparisons between groups. This includes Cohen's d, the area under the curve (AUC), the difference in the means (delta-mean) and the difference in the proportion of detected cells (delta-detected).

Usage

```
scoreMarkers(
  x,
  groups,
  block = NULL,
  block.weight.policy = c("variable", "equal", "none"),
  variable.block.weight = c(0, 1000),
  compute.auc = TRUE,
  threshold = 0,
  all.pairwise = FALSE,
  num.threads = 1
)
```

Arguments

x	A matrix-like object where rows correspond to genes or genomic features and columns correspond to cells. It is typically expected to contain log-expression values, e.g., from normalizeCounts.
groups	A vector specifying the group assignment for each cell in x.
block	Factor specifying the block of origin (e.g., batch, sample) for each cell in x. Alternatively NULL if all cells are from the same block.
block.weight.policy	String specifying the policy to use for weighting different blocks when computing the average for each statistic Only used if block is not NULL.

variable.block.weight
: Numeric vector of length 2, specifying the parameters for variable block weighting. The first and second values are used as the lower and upper bounds, respectively, for the variable weight calculation. Only used if block is not NULL and block.weight.policy = "variable".

compute.auc
: Logical scalar indicating whether to compute the AUC. Setting this to FALSE can improve speed and memory efficiency.

threshold
: Non-negative numeric scalar specifying the minimum threshold on the differences in means (i.e., the log-fold change, if x contains log-expression values). This is incorporated into the effect sizes for Cohen's d and the AUC.

all.pairwise
: Logical scalar indicating whether to report the full effects for every pairwise comparison between groups.

num.threads
: Integer scalar specifying the number of threads to use.

Value

If all.pairwise=FALSE, a list is returned containing:

- mean, a numeric matrix containing the mean expression for each group. Each row is a gene and each column is a group.
- detected, a numeric matrix containing the proportion of detected cells in each group. Each row is a gene and each column is a group.
- cohens.d, a list of data frames where each data frame corresponds to a group. Each row of each data frame represents a gene, while each column contains a summary of Cohen's d from pairwise comparisons to all other groups. This includes the min, mean, median, max and min.rank.
- auc, a list like cohens.d but containing the summaries of the AUCs from each pairwise comparison. Omitted if compute.auc=FALSE.
- delta.mean, a list like cohens.d but containing the summaries of the delta-mean from each pairwise comparison.
- delta.detected, a list like cohens.d but containing the summaries of the delta-detected from each pairwise comparison.

If all.pairwise=TRUE, a list is returned containing:

- mean, a numeric matrix containing the mean expression for each group. Each row is a gene and each column is a group.
- detected, a numeric matrix containing the proportion of detected cells in each group. Each row is a gene and each column is a group.
- cohens.d, a 3-dimensional numeric array containing the Cohen's from each pairwise comparison between groups. The extents of the first two dimensions are equal to the number of groups, while the extent of the final dimension is equal to the number of genes. The entry [i, j, k] represents Cohen's d from the comparison of group j over group i for gene k.
- auc, an array like cohens.d but containing the AUCs from each pairwise comparison. Omitted if compute.auc=FALSE.
- delta.mean, an array like cohens.d but containing the delta-mean from each pairwise comparison.
- delta.detected, an array like cohens.d but containing the delta-detected from each pairwise comparison.

See Also

https://libscran.github.io/scran_markers/, in particular the score_markers_summary function (for all.pairwise=FALSE), the score_markers_pairwise function (for all.pairwise=TRUE), and their blocked equivalents score_markers_summary_blocked and score_markers_pairwise_blocked (when block is not NULL).

summarizeEffects, to summarize the pairwise effects returned when all.pairwise=TRUE.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))
normed <- normalizeCounts(x, size.factors=centerSizeFactors(colSums(x)))

g <- sample(letters[1:4], ncol(x), replace=TRUE)
scores <- scoreMarkers(normed, g)
names(scores)
head(scores$mean)
head(scores$cohens.d[["a"]])
```

---

subsampleByNeighbors *Subsample cells based on their neighbors*

Description

Subsample a dataset by selecting cells to represent all of their nearest neighbors.

Usage

```
subsampleByNeighbors(
  x,
  num.neighbors = 20,
  min.remaining = 10,
  num.threads = 1,
  BNPARAM = AnnoyParam()
)
```

Arguments

x	A numeric matrix where rows are dimensions and columns are cells, typically containing a low-dimensional representation from, e.g., runPca. Alternatively, an index constructed by buildIndex. Alternatively, a list containing existing nearest-neighbor search results. This should contain: • index, an integer matrix where rows are neighbors and columns are cells. Each column contains 1-based indices for the nearest neighbors of the corresponding cell, ordered by increasing distance. • distance, a numeric matrix of the same dimensions as index, containing the distances to each of the nearest neighbors.

The number of neighbors should be equal to num.neighbors, otherwise a warning is raised.

num.neighbors — Integer scalar specifying the number of neighbors to use. Larger values result in greater downsampling. Only used if x does not contain existing nearest-neighbor results.

min.remaining — Integer scalar specifying the minimum number of remaining (i.e., unselected) neighbors that a cell must have in order to be considered for selection. This should be less than or equal to num.neighbors.

num.threads — Integer scalar specifying the number of threads to use for the nearest-neighbor search. Only used if x does not contain existing nearest-neighbor results.

BNPARAM — A BiocNeighborParam object specifying the algorithm to use. Only used if x does not contain existing nearest-neighbor results.

Value

Integer vector with the indices of the selected cells in the subsample.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/nenesub/, for more details on the underlying algorithm.

Examples

```
x <- matrix(rnorm(10000), nrow=2)
keep <- subsampleByNeighbors(x, 10)
plot(x[1,], x[2,])
points(x[1,keep], x[2,keep], col="red")
legend('topright', col=c('black', 'red'), legend=c('all', 'subsample'), pch=1)
```

---

summarizeEffects *Summarize pairwise effect sizes for each group*

Description

For each group, summarize the effect sizes for all pairwise comparisons to other groups. This yields a set of summary statistics that can be used to rank marker genes for each group.

Usage

```
summarizeEffects(effects, num.threads = 1)
```

Arguments

effects	A 3-dimensional numeric containing the effect sizes from each pairwise comparison between groups. The extents of the first two dimensions are equal to the number of groups, while the extent of the final dimension is equal to the number of genes. The entry [i, j, k] represents Cohen's d from the comparison of group j over group i for gene k. See also the output of scoreMarkers with all.pairwise=TRUE.
num.threads	Integer scalar specifying the number of threads to use.

Value

List of data frames containing summary statistics for the effect sizes. Each data frame corresponds to a group, each row corresponds to a gene, and each column contains a single summary.

Author(s)

Aaron Lun

See Also

https://libscran.github.io/scran_markers/, for more details on the statistics.

scoreMarkers, to compute the pairwise effects in the first place.

Examples

```
# Mocking a matrix:
library(Matrix)
x <- round(abs(rsparsematrix(1000, 100, 0.1) * 100))
normed <- normalizeCounts(x, size.factors=centerSizeFactors(colSums(x)))

g <- sample(letters[1:4], ncol(x), replace=TRUE)
effects <- scoreMarkers(normed, g, all.pairwise=TRUE)

summarized <- summarizeEffects(effects$cohens.d)
str(summarized)
```

Index

adt_quality_control, 2
aggregateAcrossCells, 4

BiocNeighborParam, *6*, *15*, *24*, *27*, *29*, *31*, *36*
buildIndex, *6*, *24*, *27, 28*, *35*
buildSnnGraph, 5, *10*, *24*

centerSizeFactors, 7
chooseHighlyVariableGenes, 8
choosePseudoCount, 9
clusterGraph, 10, *24*
clusterKmeans, 11
combineFactors, *4*, 13
computeAdtQcMetrics
        *(*adt_quality_control*)*, 2
computeClrm1Factors, 14
computeCrisprQcMetrics
        *(*crispr_quality_control*)*, 16
computeRnaQcMetrics
        *(*rna_quality_control*)*, 22
correctMnn, 14
crispr_quality_control, 16

DelayedArray, *21*

filterAdtQcMetrics
        *(*adt_quality_control*)*, 2
filterCrisprQcMetrics
        *(*crispr_quality_control*)*, 16
filterRnaQcMetrics
        *(*rna_quality_control*)*, 22
fitVarianceTrend, 18, *20*

igraph, *10*
initializeCpp, *21*

modelGeneVariances, *8*, 19

normalizeCounts, *9*, *19*, 21, *33*

rna_quality_control, 22
runAllNeighborSteps, 24
runPca, *5*, *15*, *24*, 25, *27, 28*, *35*
runTsne, *24*, 27
runUmap, *24*, 28

sanitizeSizeFactors, 29
scaleByNeighbors, 30
scoreGeneSet, 31
scoreMarkers, 33, *37*
subsampleByNeighbors, 35
suggestAdtQcThresholds
        *(*adt_quality_control*)*, 2
suggestCrisprQcThresholds
        *(*crispr_quality_control*)*, 16
suggestRnaQcThresholds
        *(*rna_quality_control*)*, 22
summarizeEffects, *35*, 36

tsnePerplexityToNeighbors *(*runTsne*)*, 27