Package ‘scran’

October 18, 2017

Version 1.4.5

Date 2017-06-17

Title Methods for Single-Cell RNA-Seq Data Analysis

Maintainer Aaron Lun <alun@wehi.edu.au>

Depends R (>= 3.4), BiocParallel, scater

Imports Biobase, stats, methods, utils, graphics, grDevices, Matrix, edgeR, limma, dynamicTreeCut, FNN, igraph, shiny, zoo, statmod, splines, ggplot2, MASS, DT, viridis

Suggests testthat, knitr, BiocStyle, limSolve, org.Mm.eg.db, DESeq2, monocle, S4Vectors, Rtsne, pracma

biocViews Normalization, Sequencing, RNASeq, Software, GeneExpression, Transcriptomics, SingleCell, Visualization, BatchEffect

Description Implements a variety of low-level analyses of single-cell RNA-seq data. Methods are provided for normalization of cell-specific biases, assignment of cell cycle phase, and detection of highly variable and significantly correlated genes.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

Author Aaron Lun [aut, cre], Karsten Bach [aut], Jong Kyoung Kim [ctb], Antonio Scialdone [ctb], Laleh Haghverdi [ctb]

R topics documented:

`buildSNNGraph` *Build a SNN graph*

Description

Build a shared-nearest-neighbors graph for cells based on their expression profiles.

Usage

```
## S4 method for signature 'matrix'
buildSNNGraph(x, k=10, d=50, subset.row=NULL)

## S4 method for signature 'SCESet'
buildSNNGraph(x, ..., subset.row=NULL, assay="exprs", get.spikes=FALSE)
```

Arguments

`x`	A SCESet object, or a matrix containing expression values for each gene (row) in each cell (column).
`k`	An integer scalar specifying the number of nearest neighbors to consider during graph construction.
`d`	An integer scalar specifying the number of dimensions to use for the k-NN search.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	Additional arguments to pass to `buildSNNGraph,matrix-method`.
`assay`	A string specifying which assay values to use, e.g., `counts` or `exprs`.
`get.spikes`	A logical specifying whether spike-in transcripts should be used.

Details

This function builds a SNN graph using cells as nodes. Each cell is connected to its `k` nearest neighbors, based on Euclidean distances in their expression profiles. The weight of the edge between two cells is determined by the ranking of their shared nearest neighbors. More shared neighbors, or shared neighbors that are close to both cells, will yield larger weights.

The aim is to use the SNN graph to perform community-based clustering, using various methods in the **igraph** package. This is faster/more memory efficient than hierarchical clustering for large numbers of cells. In particular, it avoids the need to construct a distance matrix for all pairs of cells. The choice of k can be roughly interpreted as the minimum cluster size.

In practice, PCA is performed on x to obtain the first d principal components. This is necessary in order to perform the k-NN search (done using the get.knn function) in reasonable time. By default, the first 50 components are chosen, which should retain most of the substructure in the data set. If d is NA or less than the number of cells, no dimensionality reduction is performed.

Expression values should typically be on the log-scale, e.g., log-transformed counts. Ranks can also be used for greater robustness, e.g., from quickCluster with get.ranks=TRUE. (Dimensionality reduction is still okay when ranks are provided - running PCA on ranks is equivalent to running MDS on the distance matrix derived from Spearman's rho.)

By default, spike-in transcripts are removed from the expression matrix in buildSNNGraph,SCESet-method. However, any non-NULL setting of subset.row will override get.spikes.

Note that the setting of k here is slightly different from that used in SNN-Cliq. The original implementation considers each cell to be its first nearest neighbor that contributes to k. In buildSNNGraph, the k nearest neighbours refers to the number of *other* cells.

Value

An igraph-type graph, where nodes are cells and weighted edges represent connections between nearest neighbors.

Author(s)

Aaron Lun

References

Xu C and Su Z (2015). Identification of cell types from single-cell transcriptomes using a novel clustering method. *Bioinformatics* 31:1974-80

See Also

get.knn, make_graph

Examples

```
exprs <- matrix(rnorm(100000), ncol=100)
g <- buildSNNGraph(exprs)

library(igraph) # lots of algorithms can be used
clusters <- cluster_fast_greedy(g)$membership
```

---

combineVar *Combine variance decompositions*

Description

Combine the results of multiple variance decompositions, usually generated for the same genes across separate batches of cells.

Usage

```
combineVar(..., method=c("fisher", "simes", "berger"))
```

Arguments

`...`	Two or more data frames, each produced by `decomposeVar`.
`method`	A string specifying how p-values are to be combined.

Details

This function is designed to merge results from multiple calls to `decomposeVar`, usually computed for different batches of cells. Separate variance decompositions are necessary in cases where different concentrations of spike-in have been added to the cells in each batch. This affects the technical mean-variance relationship and precludes the use of a common trend fit.

The default setting is to use `method="fisher"`, where Fisher's method is used to combine p-values across batches. This aims to detect genes that are highly variable in *any* batch and assumes that the test outcome is independent between batches. If independence does not hold, Simes' method should be used by setting `method="simes"`, as it is more robust to correlations between tests. To identify genes that are detected as highly variable in *all* batches, Berger's IUT can be used by setting `method="simes"`.

Value

A data frame with the same numeric fields as that produced by `decomposeVar`. Each field contains the average across all batches except for `p.value`, which contains the combined p-value based on `method`; and `FDR`, which contains the adjusted p-value using the BH method.

Author(s)

Aaron Lun

References

Simes RJ (1986). An improved Bonferroni procedure for multiple tests of significance. *Biometrika* 73:751-754.

Berger RL and Hsu JC (1996). Bioequivalence trials, intersection-union tests and equivalence confidence sets. *Statist. Sci.* 11, 283-319.

Fisher, R.A. (1925). *Statistical Methods for Research Workers.* Oliver and Boyd (Edinburgh).

See Also

`decomposeVar`

Examples

```
example(computeSpikeFactors) # Using the mocked-up data 'y' from this example.
y <- computeSumFactors(y) # Size factors for the the endogenous genes.
y <- computeSpikeFactors(y, general.use=FALSE) # Size factors for spike-ins.

y1 <- y[,1:100]
y1 <- normalize(y1) # normalize separately after subsetting.
fit1 <- trendVar(y1)
results1 <- decomposeVar(y1, fit1)
```

```
y2 <- y[,1:100 + 100]
y2 <- normalize(y2) # normalize separately after subsetting.
fit2 <- trendVar(y2)
results2 <- decomposeVar(y2, fit2)

head(combineVar(results1, results2))
head(combineVar(results1, results2, method="simes"))
head(combineVar(results1, results2, method="berger"))
```

convertTo *Convert to other classes*

Description

Convert a SCESet object into other classes for entry into other analysis pipelines.

Usage

```
## S4 method for signature 'SCESet'
convertTo(x, type=c("edgeR", "DESeq2", "monocle"),
    fData.col=NULL, pData.col=NULL, ..., assay,
    use.all.sf=TRUE, normalize=TRUE, subset.row=NULL, get.spikes=FALSE)
```

Arguments

x	A SCESet object.
type	A string specifying the analysis for which the object should be prepared.
fData.col	Any set of indices specifying which columns of fData(x) should be retained in the returned object.
pData.col	Any set of indices specifying which columns of pData(x) should be retained.
...	Other arguments to be passed to pipeline-specific constructors.
assay	A string specifying which assay of x should be put in the returned object.
use.all.sf	A logical scalar indicating whether multiple size factors should be used to generate the returned object.
normalize	A logical scalar specifying whether the assay values should be normalized for type="monocle".
subset.row	A logical, integer or character scalar indicating the rows of x to return.
get.spikes	A logical scalar specifying whether rows corresponding to spike-in transcripts should be returned.

Details

This function converts an SCESet object into various other classes in preparation for entry into other analysis pipelines, as specified by type. Gene- and cell-specific data fields can be retained in the output object by setting fData.col and pData.col, respectively. Other arguments can be passed to the relevant constructors through the ellipsis.

By default, for edgeR and DESeq2, assay is set to "counts" such that count data is stored in the output object. This is consistent with the required inputs to these analyses. Information about normalization is instead transmitted via size or normalization factors in the output object. For monocle, assay is ignored and counts are divided by the size factors to yield (roughly) log-normally distributed expression values. Values in assay can be used directly by setting normalize=FALSE.

In all cases, rows corresponding to spike-in transcripts are removed from the output object by default. As such, rows in the returned object may not correspond directly to rows in x. Users should consider this when retrieving analysis results from these pipelines, e.g., match on row names in x before comparing to other results. This behaviour can be turned off by setting get.spikes=TRUE, such that all rows are retrieved in the output object. Users can also set subset.row to extract specific rows, in which case get.spikes is ignored.

By default, different size factors for different rows (e.g., for spike-in sets) will be respected. For edgeR, an offset matrix will be constructed containing mean-centred log-size factors for each row. For DESeq2, a similar matrix will be constructed containing size factors scaled to have a geometric mean of unity. For monocle, counts for each row will be divided by the size factors for that row. This behaviour can be turned off with use.all.sf=FALSE, such that only sizeFactors(x) is used for normalization for all type. (For edgeR and DESeq2, the offset matrix is not generated if all rows correspond to sizeFactors(x), as this information is already stored in the object.)

Value

For type="edgeR", a DGEList object is returned containing the count matrix. Size factors are converted to normalization factors. Gene-specific fData is stored in the genes element, and cell-specific pData is stored in the samples element.

For type="DESeq2", a DESeqDataSet object is returned containing the count matrix and size factors. Additional gene- and cell-specific data is stored in the mcols and colData respectively.

For type="monocle", a CellDataSet object is returned containing the unlogged expression values. Additional gene- and cell-specific data is stored in the fData and pData respectively.

Author(s)

Aaron Lun

See Also

DGEList, DESeqDataSetFromMatrix, newCellDataSet

Examples

```
example(computeSpikeFactors) # Using the mocked up data 'y' from this example.
sizeFactors(y) <- 2^rnorm(ncells) # Adding some additional embellishments.
fData(y)$SYMBOL <- paste0("X", seq_len(nrow(y)))
y$other <- sample(LETTERS, ncells, replace=TRUE)

# Converting to various objects.
convertTo(y, type="edgeR")
convertTo(y, type="DESeq2")
convertTo(y, type="monocle")
```

correlatePairs *Test for significant correlations*

Description

Identify pairs of genes that are significantly correlated based on a modified Spearman's rho.

Usage

```
correlateNull(ncells, iters=1e6, design=NULL, residuals=FALSE)

## S4 method for signature 'matrix'
correlatePairs(x, null.dist=NULL, design=NULL, BPPARAM=SerialParam(),
    use.names=TRUE, tol=1e-8, iters=1e6, residuals=FALSE,
    subset.row=NULL, per.gene=FALSE)

## S4 method for signature 'SCESet'
correlatePairs(x, subset.row=NULL, use.names=TRUE, per.gene=FALSE,
    ..., assay="exprs", get.spikes=FALSE)
```

Arguments

`ncells`	An integer scalar indicating the number of cells in the data set.
`iters`	An integer scalar specifying the number of values in the null distribution.
`design`	A numeric design matrix containing uninteresting factors to be ignored.
`residuals`	A logical scalar indicating whether correlations should be calculated from residuals when `design!=NULL`.
`x`	A numeric matrix of normalized expression values, where rows are genes and columns are cells. Alternatively, a SCESet object containing such a matrix.
`null.dist`	A numeric vector of rho values under the null hypothesis.
`BPPARAM`	A BiocParallelParam object to use in `bplapply` for parallel processing.
`use.names`	A logical scalar specifying whether the row names of `exprs` should be used in the output. Alternatively, a character vector containing the names to use.
`tol`	A numeric scalar indicating the maximum difference under which two expression values are tied.
`subset.row`	A logical, integer or character vector indicating the rows of `x` to use. Alternatively, a list of 2 such vectors, or a integer/character matrix with 2 columns - see below for details.
`per.gene`	A logical scalar specifying whether statistics should be summarized per gene.
`...`	Additional arguments to pass to `correlatePairs,matrix-method`.
`assay`	A string specifying which assay values to use.
`get.spikes`	A logical specifying whether spike-in transcripts should be used.

Details

The aim of the correlatePairs function is to identify significant correlations between all pairs of genes in x. This allows prioritization of genes that are driving systematic substructure in the data set. By definition, such genes should be correlated as they are behaving in the same manner across cells. In contrast, genes driven by random noise should not exhibit any correlations with other genes.

An approximation of Spearman's rho is used to quantify correlations robustly based on ranks. To identify correlated gene pairs, the significance of non-zero correlations is assessed using a permutation test. The null hypothesis is that the (ranking of) normalized expression across cells should be independent between genes. This allows us to construct a null distribution by randomizing (ranked) expression within each gene.

The correlateNull function constructs an empirical null distribution for rho computed with ncells cells. When design=NULL, this is done by shuffling the ranks, calculating the rho and repeating until iters values are obtained. The p-value for each gene pair is defined as the tail probability of this distribution at the observed correlation (with some adjustment to avoid zero p-values). Correction for multiple testing is done using the BH method.

For correlatePairs, a pre-computed empirical distribution can be supplied as null.dist if available. Otherwise, it will be automatically constructed via correlateNull with ncells set to the number of columns in exprs. If the limited field is TRUE in the returned dataframe, it may be possible to obtain lower p-values by increasing iters. This should be examined for non-significant pairs, in case some correlations are overlooked due to computational limitations. The function will automatically raise a warning if any genes are limited in their significance at a FDR of 5%.

For correlatePairs,SCESet-method, correlations should be computed for normalized expression values in the specified assay. By default, rows corresponding to spike-in transcripts are removed with get.spikes=FALSE. This avoids picking up strong technical correlations between pairs of spike-in transcripts. Users can also set subset.row to specify which genes to test, which will override any setting of get.spikes.

If per.gene=TRUE, results are summarized on a per-gene basis. For each gene, all of its pairs are identified, and the corresponding p-values are combined using Simes' method. This tests whether the gene is involved in significant correlations to *any* other gene. Setting per.gene=TRUE is useful for identifying correlated genes without regard to what they are correlated with (e.g., during feature selection).

Value

For correlateNull, a numeric vector of length iters is returned containing the sorted correlations under the null hypothesis of no correlations. Arguments to design and residuals are stored in the attributes.

For correlatePairs with per.gene=FALSE, a dataframe is returned with one row per gene pair and the following fields:

gene1, gene2: Character or integer fields specifying the genes in the pair. If use.names=FALSE, integers are returned representing row indices of x, otherwise gene names are returned.

rho: A numeric field containing the approximate Spearman's rho.

p.value, FDR: Numeric fields containing the permutation p-value and its BH-corrected equivalent.

limited: A logical scalar indicating whether the p-value is at its lower bound, defined by iters.

Rows are sorted by increasing p.value and, if tied, decreasing absolute size of rho. The exception is if subset.row is a matrix, in which case each row in the dataframe correspond to a row of subset.row.

For correlatePairs with per.gene=TRUE, a dataframe is returned with one row per gene. For each row, the rho field contains the correlation with the largest magnitude across all gene pairs involving the corresponding gene. The p.value field contains the Simes p-value, and the FDR field contains the corresponding adjusted p-value. No sorting is performed, so if subset.row=NULL, each row of the returned dataframe corresponds to a row of x. For the SCESet method, if get.spikes=FALSE, rows corresponding to spike-in features will have NA entries for most fields.

Accounting for uninteresting variation

If the experiment has known (and uninteresting) factors of variation, these can be included in design. These factors will be regressed out to ensure that they do not drive strong correlations between genes. Examples might be to block on batch effects or cell cycle phase, which may have substantial but uninteresting effects on expression.

The approach used to remove these factors depends on the design matrix. If there is only one factor in design, the levels of the factor are defined as separate groups. For each pair of genes, correlations are computed within each group, and a weighted mean (based on the group size) of the correlations is taken across all groups. The same strategy is used to generate the null distribution where ranks are computed and shuffled within each group.

For designs containing multiple factors or covariates, a linear model is fitted to the (log-normalized) expression values with design. The correlation between a pair of genes is then computed from the residuals of the fitted model. Similarly, to obtain a null distribution of rho values, normally-distributed random errors are simulated in a fitted model based on design; the corresponding residuals are generated from these errors; and the correlation between sets of residuals is computed at each iteration. This approach can also be used for one-way layouts by setting residuals=TRUE.

(The second procedure assumes normality, during both linear modelling and generation of the null distribution. This is why it is not used for the simpler one-way layouts by default. However, this assumption is largely unavoidable for complex designs, where some quantitative constraints are required to remove nuisance effects.)

Gene selection

We recommend setting subset.row to contain only the subset of highly variable genes (HVGs). This will focus on genes contributing to cell-to-cell heterogeneity (and thus more likely to be involved in driving substructure). There is no need to account for HVG pre-selection in multiple testing, because rank correlations are unaffected by the variance. For more genes, set BPPARAM to use more workers and reduce computational time.

Lowly-expressed genes can also cause problems when design is non-NULL and residuals=TRUE. Tied counts, and zeroes in particular, may not result in tied residuals after fitting of the linear model. Model fitting may break ties in a consistent manner across genes, yielding large correlations between genes with many zero counts. Focusing on HVGs should mitigate the detection of these uninteresting correlations, as genes dominated by zeroes will usually have low variance.

If subset.row is a list of two vectors, correlations will be computed between one gene in the first vector and another gene in the second vector. This improves efficiency if the only correlations of interest are those between two pre-defined sets of genes. Alternatively, if subset.row is an integer/character matrix of two columns, each row is assumed to specify a gene pair. Correlations will then be computed for only those gene pairs, and the returned dataframe will *not* be sorted by p-value.

Approximating Spearman's rho with tied values

As previously mentioned, an approximate version of Spearman's rho is used. Specifically, untied ranks are randomly assigned to any tied values. This means that a common empirical distribution can be used for all gene pairs, rather than having to do new permutations for every pair to account for the different pattern of ties. Generally, this modification has little effect on the results for expressed genes (and in any case, differences in library size break ties for normalized expression values). Some correlations may end up being spuriously large, but this should be handled by the error control machinery after multiplicity correction.

Author(s)

Aaron Lun

References

Phipson B and 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.

Simes RJ (1986). An improved Bonferroni procedure for multiple tests of significance. *Biometrika* 73:751-754.

See Also

`bpparam`, `cor`

Examples

```
set.seed(0)
ncells <- 100
null.dist <- correlateNull(ncells, iters=100000)
exprs <- matrix(rpois(ncells*100, lambda=10), ncol=ncells)
out <- correlatePairs(exprs, null.dist=null.dist)
hist(out$p.value)
```

cyclone *Cell cycle phase classification*

Description

Classify single cells into their cell cycle phases based on gene expression data.

Usage

```
## S4 method for signature 'matrix'
cyclone(x, pairs, gene.names=rownames(x), iter=1000, min.iter=100, min.pairs=50,
    BPPARAM=SerialParam(), verbose=FALSE, subset.row=NULL)

## S4 method for signature 'SCESet'
cyclone(x, pairs, subset.row=NULL, ..., assay="counts", get.spikes=FALSE)
```

Arguments

`x` A numeric matrix of gene expression values where rows are genes and columns are cells. Alternatively, a SCESet object containing such a matrix.

`pairs` A list of data.frames produced by `sandbag`, containing pairs of marker genes.

`gene.names` A character vector of gene names.

`iter` An integer scalar specifying the number of iterations for random sampling to obtain a cycle score.

`min.iter` An integer scalar specifying the minimum number of iterations for score estimation.

`min.pairs` An integer scalar specifying the minimum number of pairs for cycle estimation.

`BPPARAM` A BiocParallelParam object to use in `bplapply` for parallel processing.

`verbose` A logical scalar specifying whether diagnostics should be printed to screen.

`subset.row` A logical, integer or character scalar indicating the rows of `x` to use.

`...` Additional arguments to pass to `cyclone,matrix-method`.

`assay` A string specifying which assay values to use, e.g., `counts` or `exprs`.

`get.spikes` A logical specifying whether spike-in transcripts should be used.

Details

This function implements the classification step of the pair-based prediction method described by Scialdone et al. (2015). To illustrate, consider classification of cells into G1 phase. Pairs of marker genes are identified with `sandbag`, where the expression of the first gene in the training data is greater than the second in G1 phase but less than the second in all other phases. For each cell, `cyclone` calculates the proportion of all marker pairs where the expression of the first gene is greater than the second in the new data `x` (pairs with the same expression are ignored). A high proportion suggests that the cell is likely to belong in G1 phase, as the expression ranking in the new data is consistent with that in the training data.

Proportions are not directly comparable between phases due to the use of different sets of gene pairs for each phase. Instead, proportions are converted into scores (see below) that account for the size and precision of the proportion estimate. The same process is repeated for all phases, using the corresponding set of marker pairs in `pairs`. Cells with G1 or G2M scores above 0.5 are assigned to the G1 or G2M phases, respectively. (If both are above 0.5, the higher score is used for assignment.) Cells can be assigned to S phase based on the S score, but a more reliable approach is to define S phase cells as those with G1 and G2M scores below 0.5.

For `cyclone,SCESet-method`, the matrix of counts is used but can be replaced with expression values by setting `assay`. By default, `get.spikes=FALSE` which means that any rows corresponding to spike-in transcripts will not be considered for score calculation. This is for the same reasons as described in `?sandbag`.

Users can also manually set `subset.row` to specify which rows of `x` are to be used. This is better than subsetting `x` directly, as it reduces memory usage and also subsets `gene.names` at the same time. If this is specified, it will overwrite any setting of `get.spikes`.

While this method is described for cell cycle phase classification, any biological groupings can be used here – see `?sandbag` for details. However, for non-cell cycle phase groupings, the output `phases` will be an empty character vector. Users should manually apply their own score thresholds for assigning cells into specific groups.

Value

A list is returned containing:

`phases`**:** A character vector containing the predicted phase for each cell.

`scores`**:** A data frame containing the numeric phase scores for each phase and cell (i.e., each row is a cell).

`normalized.scores`**:** A data frame containing the row-normalized scores (i.e., where the row sum for each cell is equal to 1).

Description of the score calculation

To make the proportions comparable between phases, a distribution of proportions is constructed by shuffling the expression values within each cell and recalculating the proportion. The phase score is defined as the lower tail probability at the observed proportion. High scores indicate that the proportion is greater than what is expected by chance if the expression of marker genes were independent (i.e., with no cycle-induced correlations between marker pairs within each cell).

By default, shuffling is performed `iter` times to obtain the distribution from which the score is estimated. However, some iterations may not be used if there are fewer than `min.pairs` pairs with different expression, such that the proportion cannot be calculated precisely. A score is only returned if the distribution is large enough for stable calculation of the tail probability, i.e., consists of results from at least `min.iter` iterations.

Note that the score calculation in `cyclone` is slightly different from that described originally by Scialdone et al. The original code shuffles all expression values within each cell, while in this implementation, only the expression values of genes in the marker pairs are shuffled. This modification aims to use the most relevant expression values to build the null score distribution.

Author(s)

Antonio Scialdone, with modifications by Aaron Lun

References

Scialdone A, Natarajana KN, Saraiva LR et al. (2015). Computational assignment of cell-cycle stage from single-cell transcriptome data. *Methods* 85:54–61

See Also

`sandbag`

Examples

```
example(sandbag) # Using the mocked-up data in this example.

# Classifying (note: test.data!=training.data in real cases)
test <- training
assignments <- cyclone(test, out)
head(assignments$scores)
head(assignments$phases)

# Visualizing
col <- character(ncells)
col[is.G1] <- "red"
col[is.G2M] <- "blue"
```

```
col[is.S] <- "darkgreen"
plot(assignments$score$G1, assignments$score$G2M, col=col, pch=16)
```

`decomposeVar` *Decompose the gene-level variance*

Description

Decompose the gene-specific variance into biological and technical components for single-cell RNA-seq data.

Usage

```
## S4 method for signature 'matrix,list'
decomposeVar(x, fit, design=NA, subset.row=NULL, ...)

## S4 method for signature 'SCESet,list'
decomposeVar(x, fit, subset.row=NULL, ..., assay="exprs", get.spikes=FALSE)
```

Arguments

`x`	A numeric matrix of normalized log-expression values, where each column corresponds to a cell and each row corresponds to an endogenous gene. Alternatively, a SCESet object containing such a matrix.
`fit`	A list containing the output of `trendVar`, run on log-expression values for spike-in genes.
`design`	A numeric matrix describing the uninteresting factors contributing to expression in each cell.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	For `decomposeVar,matrix,list-method`, additional arguments to pass to `testVar`. For `decomposeVar,SCESet,list-method`, additional arguments to pass to the matrix method.
`assay`	A string specifying which assay values to use, e.g., `counts` or `exprs`.
`get.spikes`	A logical scalar specifying whether decomposition should be performed for spike-ins.

Details

This function computes the variance of the normalized log-counts for each endogenous gene. The technical component of the variance for each gene is determined by interpolating the fitted trend in `fit` at the mean log-count for that gene. This represents variance due to sequencing noise, variability in capture efficiency, etc. The biological component is determined by subtracting the technical component from the total variance.

Highly variable genes (HVGs) can be identified as those with large biological components. Unlike other methods for decomposition, this approach estimates the variance of the log-counts rather than of the counts themselves. The log-transformation blunts the impact of large positive outliers and ensures that the HVG list is not dominated by outliers. Interpretation is not compromised – HVGs will still be so, regardless of whether counts or log-counts are considered.

The design matrix can be set if there are factors that should be blocked, e.g., batch effects, known (and uninteresting) clusters. If NULL, it will be set to an all-ones matrix, i.e., all cells are replicates. If NA, it will be extracted from fit$design, assuming that the same cells were used to fit the trend.

Users can also directly specify which rows to use with subset.row. This is equivalent to running decomposeVar on x[subset.row,], but is more efficient as it avoids the construction of large temporary matrices.

If assay="exprs" and the size factors are not centred at unity, a warning will be raised - see ?trendVar for details.

Value

A data frame is returned where each row corresponds to and is named after a row of x (if subset.row=NULL; otherwise, each row corresponds to an element of subset.row). This contains the numeric fields:

mean: Mean normalized log-count per gene.

total: Variance of the normalized log-counts per gene.

bio: Biological component of the variance.

tech: Technical component of the variance.

p.value, FDR: Raw and adjusted p-values for the test against the null hypothesis that bio=0.

Rows corresponding to spike-in transcripts have their p-value and FDR fields set to NA unless get.spikes=TRUE.

Author(s)

Aaron Lun

References

Lun ATL, McCarthy DJ and Marioni JC (2016). A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.* 5:2122

See Also

trendVar, testVar

Examples

```
example(computeSpikeFactors) # Using the mocked-up data 'y' from this example.
y <- computeSumFactors(y) # Size factors for the the endogenous genes.
y <- computeSpikeFactors(y, general.use=FALSE) # Size factors for spike-ins.
y <- normalize(y) # Normalizing the counts by the size factors.

# Decomposing technical and biological noise.
fit <- trendVar(y)
results <- decomposeVar(y, fit)
head(results)

plot(results$mean, results$total)
o <- order(results$mean)
lines(results$mean[o], results$tech[o], col="red", lwd=2)

plot(results$mean, results$bio)
```

```
# A trend fitted to endogenous genes can also be used, pending assumptions.
fit.g <- trendVar(y, use.spikes=FALSE)
results.g <- decomposeVar(y, fit.g)
head(results.g)
```

Deconvolution Methods *Normalization by deconvolution*

Description

Methods to normalize single-cell RNA-seq data by deconvolving size factors from cell pools.

Usage

```
## S4 method for signature 'matrix'
computeSumFactors(x, sizes=seq(20, 100, 5), clusters=NULL,
    ref.clust=NULL, positive=FALSE, errors=FALSE, subset.row=NULL)
## S4 method for signature 'SCESet'
computeSumFactors(x, subset.row=NULL, ..., assay="counts",
    get.spikes=FALSE, sf.out=FALSE)
```

Arguments

`x`	A numeric count matrix where rows are genes and columns are cells. Alternatively, a SCESet object containing such a matrix.
`sizes`	A numeric vector of pool sizes, i.e., number of cells per pool.
`clusters`	An optional factor specifying which cells belong to which cluster, for deconvolution within clusters.
`ref.clust`	A level of `clusters` to be used as the reference cluster for inter-cluster normalization.
`positive`	A logical scalar indicating whether linear inverse models should be used to enforce positive estimates.
`errors`	A logical scalar indicating whether the standard error should be returned.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	Additional arguments to pass to `computeSumFactors,matrix-method`.
`assay`	A string specifying which assay values to use, e.g., `counts` or `exprs`.
`get.spikes`	A logical scalar specifying whether spike-in transcripts should be used.
`sf.out`	A logical scalar indicating whether only size factors should be returned.

Value

For `computeSumFactors,matrix-method`, a numeric vector of size factors for all cells in `x` is returned.

For `computeSumFactors,SCESet-method`, an object of class `x` is returned containing the vector of size factors in `sizeFactors(x)`, if `sf.out=FALSE`. Otherwise, the vector of size factors is returned directly.

If `errors=TRUE`, the standard errors of the size factor estimates are stored as the `"standard.error"` field of the attributes of the returned vector.

Overview of the deconvolution method

The computeSumFactors function provides an implementation of the deconvolution strategy for normalization. Briefly, a pool of cells is selected and the counts for those cells are summed together. The count sums for this pool is normalized against an average reference pseudo-cell, constructed by averaging the counts across all cells. This defines a size factor for the pool as the median ratio between the count sums and the average across all genes.

Now, the bias for the pool is equal to the sum of the biases for the constituent cells. The same applies for the size factors (which are effectively estimates of the bias for each cell). This means that the size factor for the pool can be written as a linear equation of the size factors for the cells. Repeating this process for multiple pools will yield a linear system that can be solved to obtain the size factors for the individual cells.

In this manner, pool-based factors are deconvolved to yield the relevant cell-based factors. The advantage is that the pool-based estimates are more accurate, as summation reduces the number of stochastic zeroes and the associated bias of the size factor estimate. This accuracy will feed back into the deconvolution process, thus improving the accuracy of the cell-based size factors.

Normalization within and between clusters

In general, it is more appropriate to pool more similar cells to avoid violating the assumption of a non-DE majority of genes across the data set. This can be done by specifying the clusters argument where cells in each cluster have similar expression profiles. Deconvolution is subsequently applied on the cells within each cluster. Each cluster should contain a sufficient number of cells for pooling – an error is thrown if the number of cells is less than the maximum value of sizes. A convenince function quickCluster is provided for rapid clustering based on Spearman's rank correlation.

Size factors computed within each cluster must be rescaled for comparison between clusters. This is done by normalizing between clusters to identify the rescaling factor. One cluster is chosen as a "reference" (by default, that with the median of the mean per-cell library sizes is used) to which all others are normalized. Ideally, a cluster that is not extremely different from all other clusters should be used as the reference. This can be specified using ref.clust if there is prior knowledge about which cluster is most suitable, e.g., from PCA or t-SNE plots.

Additional details about pooling and deconvolution

Within each cluster (if not specified, all cells are put into a single cluster), cells are sorted by increasing library size and a sliding window is applied to this ordering. Each location of the window defines a pool of cells with similar library sizes. This avoids inflated estimation errors for very small cells when they are pooled with very large cells. Sliding the window will construct an overdetermined linear system that can be solved with methods like the QR decomposition.

Window sliding is repeated with different window sizes to construct the linear system, as specified by sizes. By default, the number of cells in each window ranges from 20 to 100. Using a range of window sizes improves the precision of the estimates, at the cost of increased computational complexity. The defaults are chosen to provide a reasonable compromise between these two considerations.

The smallest window should be large enough so that the pool-based size factors are, on average, non-zero. We recommend window sizes no lower than 20 for UMI data, though smaller windows may be possible for read count data.

Dealing with negative size factors

In theory, it is possible to obtain negative estimates for the size factors. These values are obviously nonsensical and computeSumFactors will raise a warning if they are encountered. Negative estimates are mostly commonly generated from low quality cells with few expressed features, such that most counts are zero even after pooling. They may also occur if insufficient filtering of low-abundance genes was performed.

To avoid negative size factors, the best solution is to increase the stringency of the filtering.

- If only a few negative size factors are present, they are likely to correspond to a few low-quality cells with few expressed features. Such cells are difficult to normalize reliably under any approach, and can be removed by increasing the stringency of the quality control.
- If many negative size factors are present, it is probably due to insufficient filtering of low-abundance genes. This results in many zero counts and pooled size factors of zero, and can be fixed by filtering out more genes.

A complementary approach is to increase in the number of sizes to improve the precision of the estimates. This reduces the chance of obtaining negative size factors due to estimation error for cells with near-zero true size factors.

As a last resort, some protection can be provided by setting positive=TRUE, which will use linear inverse models to solve the system. This ensures that non-negative values for the size factors will always be obtained. Note that some cells may still have size factors of zero and should be removed prior to downstream analysis. Such occurrences are unavoidable – rather, the aim is to prevent negative values from affecting the estimates for all other cells.

Gene selection

By default, get.spikes=FALSE in quickCluster,SCESet-method which means that spike-in transcripts are not included in the set of genes used for deconvolution. This is because they can behave differently from the endogenous genes. Users wanting to perform spike-in normalization should see computeSpikeFactors instead.

Users can also set subset.row to specify which rows of x are to be used to calculate correlations. This is equivalent to but more efficient than subsetting x directly, as it avoids constructing a (potentially large) temporary matrix. If this is specified, it will overwrite any setting of get.spikes.

Note that pooling does not eliminate the need to filter out low-abundance genes. As mentioned above, if too many genes have consistently low counts across all cells, even the pool-based size factors will be zero. This results in zero or negative size factor estimates for many cells. Filtering ensures that this is not the case, e.g., by removing genes with average counts below 1.

Author(s)

Aaron Lun and Karsten Bach

References

Lun ATL, Bach K and Marioni JC (2016). Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biol.* 17:75

See Also

quickCluster

Examples

```
# Mocking up some data.
set.seed(100)
popsize <- 200
ngenes <- 10000
all.facs <- 2^rnorm(popsize, sd=0.5)
counts <- matrix(rnbinom(ngenes*popsize, mu=all.facs*10, size=1), ncol=popsize, byrow=TRUE)

# Computing the size factors.
out.facs <- computeSumFactors(counts)
head(out.facs)
plot(colSums(counts), out.facs, log="xy")
```

Denoise with PCA *Denoise expression with PCA*

Description

Denoise log-expression data by removing principal components corresponding to technical noise.

Usage

```
## S4 method for signature 'matrix'
denoisePCA(x, technical, design=NULL, subset.row=NULL)

## S4 method for signature 'SCESet'
denoisePCA(x, ..., subset.row=NULL, assay="exprs", get.spikes=FALSE)
```

Arguments

`x` A numeric matrix of log-expression values for `denoisePCA,matrix-method`, or a SCESet object containing such values for `denoisePCA,SCESet-method`.

`technical` A named numeric vector containing the technical variance for each gene in `x`. Alternatively, a function that accepts the average log-expression and returns the technical variance.

`design` A numeric matrix containing the experimental design. If `NULL`, all cells are assumed to belong to a single group.

`subset.row` A logical, integer or character vector indicating the rows of `x` to use. All genes are used by default.

`...` Further arguments to pass to `denoisePCA,matrix-method`.

`assay` A string specifying which assay values to use.

`get.spikes` A logical specifying whether spike-in transcripts should be used. This is ignored if `subset.row` is specified.

Details

The aim of this function is to perform a principal components analysis to reduce random technical noise in the data. Random noise is uncorrelated across genes and should be captured by later PCs, as the variance in the data explained by any single gene is low. In contrast, biological substructure should be correlated and captured by earlier PCs, as this explains more variance for sets of genes. The idea is to discard later PCs to remove technical noise and improve the resolution of substructure.

The choice of the number of PCs to discard is based on the estimates of technical variance in `technical`. This either uses the trend function obtained from `trendVar` or the technical components from `decomposeVar`. (Obviously, the log-expression values used in those functions should be the same as those used here.) The total technical variance is estimated by summing the values across genes.

PCs are discarded in order of increasing variance, stopping before the discarded variance is greater than the total technical variance. This eliminates the last PCs that should contain only uncorrelated noise. In this manner, it is possible to make a justified choice for the number of PCs to retain. The retained PCs should now be enriched for biological signal (both variance and substructure).

If `design` is specified, the residuals of a linear model fitted to each gene are computed. Because variances computed from residuals are usually underestimated, the residuals are scaled up so that their variance is equal to the residual variance of the model fit. This ensures that the sum of variances is not understated, which would lead to more PCs being discarded than appropriate.

Value

For `denoisePCA,matrix-method`, a numeric matrix is returned containing the selected PCs (columns) for all cells (rows).

For `denoisePCA,SCESet-method`, a SCESet object is returned containing the PCs in the `reducedDimension` slot.

Author(s)

Aaron Lun

See Also

`trendVar`, `decomposeVar`

Examples

```
# Mocking up some data.
ngenes <- 1000
is.spike <- 1:100
means <- 2^runif(ngenes, 6, 10)
dispersions <- 10/means + 0.2
nsamples <- 50
counts <- matrix(rnbinom(ngenes*nsamples, mu=means, size=1/dispersions), ncol=nsamples)
rownames(counts) <- paste0("Gene", seq_len(ngenes))

# Fitting a trend.
lcounts <- log2(counts + 1)
fit <- trendVar(lcounts, subset.row=is.spike)
dec <- decomposeVar(lcounts, fit)

# Denoising (not including the spike-ins in the PCA;
# spike-ins are automatically removed with the SCESet method).
```

```
pcs <- denoisePCA(lcounts, technical=fit$trend, subset.row=-is.spike)
dim(pcs)

# Gets the same result as above, if you don't have the function.
pcs2 <- denoisePCA(lcounts, technical=setNames(dec$tech,
    rownames(dec)), subset.row=-is.spike) # same result
dim(pcs2)

# With a design matrix.
design <- model.matrix(~factor(rep(0:1, length.out=nsamples)))
fit3 <- trendVar(lcounts, design=design, subset.row=is.spike)
pcs3 <- denoisePCA(lcounts, technical=fit3$trend, design=design, subset.row=-is.spike)
dim(pcs3)
```

Distance-to-median *Compute the distance-to-median statistic*

Description

Compute the distance-to-median statistic for the CV2 residuals of all genes

Usage

```
DM(mean, cv2, win.size=50)
```

Arguments

mean	A numeric vector of average counts for each gene.
cv2	A numeric vector of squared coefficients of variation for each gene.
win.size	An integer scalar specifying the window size for median-based smoothing.

Details

This function will compute the distance-to-median (DM) statistic described by Kolodziejczyk et al. (2015). Briefly, a median-based trend is fitted to the log-transformed cv2 against the log-transformed mean. The DM is defined as the residual from the trend for each gene. This statistic is a measure of the relative variability of each gene, after accounting for the empirical mean-variance relationship. Highly variable genes can then be identified as those with high DM values.

Value

A numeric vector of DM statistics for all genes.

Author(s)

Jong Kyoung Kim, with modifications by Aaron Lun

References

Kolodziejczyk AA, Kim JK, Tsang JCH et al. (2015). Single cell RNA-sequencing of pluripotent states unlocks modular transcriptional variation. *Cell Stem Cell* 17(4), 471–85.

Examples

```
# Mocking up some data
ngenes <- 1000
ncells <- 100
gene.means <- 2^runif(ngenes, 0, 10)
dispersions <- 1/gene.means + 0.2
counts <- matrix(rnbinom(ngenes*ncells, mu=gene.means, size=1/dispersions), nrow=ngenes)

# Computing the DM.
means <- rowMeans(counts)
cv2 <- apply(counts, 1, var)/means^2
dm.stat <- DM(means, cv2)
head(dm.stat)
```

`Explore Data` *Shiny app for explorative data analysis*

Description

Generate an interactive Shiny app to explore gene expression patterns in single-cell data

Usage

```
exploreData(x, cell.data, gene.data, red.dim, run=TRUE)
```

Arguments

`x`	A numeric matrix of expression values to be visualized.
`cell.data`	A data frame of cell information, where each row corresponds to a column of `x`.
`gene.data`	A data frame of gene information, where each row corresponds to a row of `x`.
`red.dim`	A numeric matrix with two colums, specifying the reduced-dimension coordinates for each cell.
`run`	A logical scalar specifying whether the app should be run immediately.

Details

This function will return a Shiny app object that can be run with `runApp`. The app allows the user to interactively explore gene expression patterns in single-cell RNA-seq data. Explorative analysis is focused on comparing gene exression between different groups of cells, as defined by the covariates of `cell.data`.

Three plots are shown in the app:

- a scatterplot of cell locations based on the `red.dim` coordinates, colored by a selected covariate
- a scatterplot of cell locations based on the `red.dim` coordinates, colored by expression of a selected gene
- boxplot(s) of expression values for a selected gene, grouped by a selected covariate.

Several options are available within the app:

"Color by": Covariate to be used for coloring the first scatter plot.

“Group by”: Covariate with which expression values are grouped in the boxplots.

In addition, the gene.data data frame is rendered as an interactive table using the JavaScript library DataTable. This allows the user to subset/search the feature data and select a gene by clicking on the corresponding row.

Value

If run=FALSE, a Shiny app object is returned, which can be run with runApp. If run=TRUE, a Shiny app object is created and run.

Author(s)

Karsten Bach

See Also

runApp,

Examples

```
# Set up example data
example(newSCESet)
x <- exprs(example_sceset)
cell.data <- pData(example_sceset)
gene.data <- data.frame("meanExp"=rowMeans(x))
rownames(gene.data) <- rownames(x)

# Mocking up  some reduced dimensions.
library(Rtsne)
tsn <- Rtsne(t(x), perplexity=10)
red.dim <- tsn$Y[,1:2]

# Creating the app object.
app <- exploreData(x, cell.data, gene.data, red.dim, run=FALSE)
if (interactive()) { shiny::runApp(app) }

## Not run: # Running directly from the function.
saved <- exploreData(x, cell.data, gene.data, red.dim)

## End(Not run)
```

findMarkers *Find marker genes*

Description

Find candidate marker genes for clusters of cells, by testing for differential expression between clusters.

Usage

```
## S4 method for signature 'matrix'
findMarkers(x, clusters, design=NULL, subset.row=NULL)

## S4 method for signature 'SCESet'
findMarkers(x, ..., subset.row=NULL, assay="exprs", get.spikes=FALSE)
```

Arguments

`x`	A numeric matrix of normalized log-expression values, where each column corresponds to a cell and each row corresponds to an endogenous gene. Alternatively, a SCESet object containing such a matrix.
`clusters`	A vector of cluster identities for all cells.
`design`	A numeric matrix containing blocking terms, i.e., uninteresting factors driving expression across cells.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	Additional arguments to pass to the matrix method.
`assay`	A string specifying which assay values to use, e.g., `counts` or `exprs`.
`get.spikes`	A logical scalar specifying whether decomposition should be performed for spike-ins.

Details

This function uses **limma** to test for differentially expressed genes (DEGs) between pairs of clusters. For each cluster, the log-fold changes and other statistics from all relevant pairwise comparisons are combined into a single table. A list of such tables is returned for all clusters to define a set of potential marker genes.

Each table is sorted by the `Top` value, which specifies the size of the candidate marker set. Taking all rows with `Top` values no greater than some integer X will yield a set containing the top X genes (ranked by significance) from each pairwise comparison. For example, if X is 5, the set will consist of the *union* of the top 5 genes from each pairwise comparison. This set allows the cluster to be distinguished from the others based on the expression of at least one gene. It should be stressed that markers are not explicitly chosen to be uniquely expressed in the cluster. Such a strategy is often too stringent, especially in cases involving overclustering or cell types defined by combinatorial gene expression.

The `FDR` value is calculated by combining p-values across contrasts for each gene using Simes' method, and then applying the BH method across genes. The null hypothesis here is that the gene is not DE in any of the contrasts. However, this is intended only as a rough measure of significance. Properly correcting for multiple testing is not generally possible when `clusters` is determined from the same `x` used for DE testing.

The application of **limma** uses the "trend" approach on the normalized log-expression values, as described by Law et al. (2015). This is fast and avoids putting too much weight on outliers or cells with large library sizes. Uninteresting factors of variation (e.g., preparation time, sequencing batch) will be blocked if they are stored in `design`. Note that the presence of factors that are confounded with `clusters` will raise a warning about unestimable coefficients.

By default, spike-in transcripts are ignored in `findMarkers,SCESet-method` with `get.spikes=FALSE`. This is overridden by any non-`NULL` value of `subset.row`.

Value

A named list of data frames, where each data frame corresponds to a cluster and contains a ranked set of potential marker genes. In each data frame, the log-fold change of the cluster against every other cluster Y is also reported, under the column named Y.

Author(s)

Aaron Lun

References

Law CW, Chen Y, Shi W and Smyth, GK (2014). voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biol.* 15:R29

Simes RJ (1986). An improved Bonferroni procedure for multiple tests of significance. *Biometrika* 73:751-754.

See Also

`normalize`

Examples

```
example(computeSpikeFactors) # Using the mocked-up data 'y' from this example.

kout <- kmeans(t(exprs(y)), centers=2)

out <- findMarkers(y, clusters=kout$cluster)
```

`Get spikes` *Set spike-in rows*

Description

Specify the rows in the SCESet corresponding to spike-in transcripts.

Usage

```
## S4 replacement method for signature 'SCESet'
isSpike(x) <- value
```

Arguments

`x`	A SCESet object with spike-in data in the `colData`.
`value`	A character vector specifying which control sets are spike-ins. Alternatively a `NULL` value, to remove existing spike-in specifications.

Details

This function is deprecated, use `setSpike` instead.

Value

For isSpike<-, x is modified to store a spike-specifying vector in fData(x)$is_feature_spike. A logical vector indicating which controls are spike-ins is also stored in the featureControlInfo slot of x.

Author(s)

Aaron Lun

See Also

setSpike,

Examples

```
example(computeSpikeFactors)
setSpike(y) <- "Spike"
```

improvedCV2 *Stably model the technical coefficient of variation*

Description

Model the decompose coefficient of variation as a function of the mean, and determine the significance of highly variable genes. This is intended to be a more stable version of technicalCV2.

Usage

```
## S4 method for signature 'matrix'
improvedCV2(x, is.spike, sf.cell=NULL, sf.spike=NULL,
    log.prior=NULL, df=4, robust=FALSE, use.spikes=FALSE)

## S4 method for signature 'SCESet'
improvedCV2(x, spike.type=NULL, ..., assay="exprs", logged=NULL)
```

Arguments

x	A numeric matrix of counts or log-expression values, where each column corresponds to a cell and each row corresponds to a spike-in transcript. Alternatively, a SCESet object that contains such values.
is.spike	A vector indicating which rows of x correspond to spike-in transcripts.
sf.cell	A numeric vector containing size factors for endogenous genes.
sf.spike	A numeric vector containing size factors for spike-in transcripts.
log.prior	A numeric scalar specifying the pseudo-count added prior to log-transformation. If this is set, x is assumed to contain log-expression values, otherwise it is assumed to contain counts.
df	An integer scalar indicating the number of degrees of freedom for the spline fit with ns.
robust	A logical scalar indicating whether robust fitting should be performed with rlm.

`use.spikes`	A logical scalar indicating whether p-values should be returned for spike-in transcripts.
`spike.type`	A character vector containing the names of the spike-in sets to use.
`...`	Additional arguments to pass to `improvedCV2,matrix-method`.
`assay`	A string specifying which assay values to use.
`logged`	A logical scalar indicating if `assay` contains log-expression values. This is automatically determined if `assay="counts"` or `"exprs"`.

Details

This function will estimate the squared coefficient of variation (CV2) and mean for each spike-in transcript. Both values are log-transformed and a mean-dependent trend is fitted to the log-CV2 values, using a linear model with a natural spline of degree `df`. The trend is used to obtain the technical contribution to the CV2 for each gene. The biological contribution is computed by subtracting the technical contribution from the total CV2.

Deviations from the trend are identified by modelling the CV2 estimates for the spike-in transcripts as log-normally distributed around the fitted trend. This accounts for sampling variance as well as any variability in the true dispersions (e.g., due to transcript-specific amplification biases). The p-value for each gene is calculated from a one-sided Z-test on the log-CV2, using the fitted value as the mean and the robust scale estimate as the standard deviation. A Benjamini-Hochberg adjustment is applied to correct for multiple testing.

If `log.prior` is specified, `x` is assumed to contain log-expression values. These are converted back to the count scale prior to calculation of the CV2. Otherwise, `x` is assumed to contain raw counts, which need to be normalized with `sf.cell` and `sf.spike` prior to calculating the CV2. Note that both sets of size factors are set to 1 by default if their values are not supplied to the function.

For any given data set, the maximum CV2 that can be achieved is equal to the number of cells. (This occurs when only one cell has a non-zero expression value - proof via Holder's inequality.) Genes with CV2 values equal to the maximum are ignored during trend fitting. This ensures that the trend is not distorted by the presence of an upper bound on CV2 values, especially at low means.

For `improvedCV2,matrix-method`, the rows corresponding to spike-in transcripts are specified with `is.spike`. These rows will be used for trend fitting, while all other rows are treated as endogenous genes. By default, p-values are set to `NA` for the spike-in transcripts, such that they do not contribute to the multiple testing correction. This behaviour can be modified with `use.spikes=TRUE`, which will return p-values for all features.

For `improvedCV2,SCESet-method`, transcripts from spike-in sets named in `spike.type` will be used for trend fitting. If `spike.type=NULL`, all spike-in sets listed in `x` will be used. Size factors for endogenous genes are automatically extracted via `sizeFactors`. Spike-in-specific size factors for `spike.type` are extracted from `x`, if available; otherwise they are set to the size factors for the endogenous genes. Note that the spike-in-specific factors must be the same for each set in `spike.type`.

Users can also set `is.spike` to `NA` in `improvedCV2,matrix-method`; or `spike.type` to `NA` in `decomposeCV2,SCESet-method`. In such cases, all rows will be used for trend fitting, and (adjusted) p-values will be reported for all rows. This should be used in cases where there are no spike-ins. Here, the assumption is that most endogenous genes do not exhibit high biological variability and thus can be used to model decompose variation.

Value

A data frame is returned containing one row per row of `x` (including both endogenous genes and spike-in transcripts). Each row contains the following information:

`mean`: A numeric field, containing mean (scaled) counts for all genes and transcripts.

`var`: A numeric field, containing the variances for all genes and transcripts.

`cv2`: A numeric field, containing CV2 values for all genes and transcripts.

`trend`: A numeric field, containing the fitted value of the trend in the CV2 values. Note that the fitted value is reported for all genes and transcripts, but the trend is only fitted using the transcripts.

`p.value`: A numeric field, containing p-values for all endogenous genes (`NA` for rows corresponding to spike-in transcripts).

`FDR`: A numeric field, containing adjusted p-values for all genes.

Author(s)

Aaron Lun

See Also

`ns`, `technicalCV2`

Examples

```
# Mocking up some data.
ngenes <- 10000
nsamples <- 50
means <- 2^runif(ngenes, 6, 10)
dispersions <- 10/means + 0.2
counts <- matrix(rnbinom(ngenes*nsamples, mu=means, size=1/dispersions), ncol=nsamples)
is.spike <- logical(ngenes)
is.spike[seq_len(500)] <- TRUE

# Running it directly on the counts.
out <- improvedCV2(counts, is.spike)
head(out)
plot(out$mean, out$cv2, log="xy")
points(out$mean, out$trend, col="red", pch=16, cex=0.5)

# Same again with an SCESet.
rownames(counts) <- paste0("X", seq_len(ngenes))
colnames(counts) <- paste0("Y", seq_len(nsamples))
X <- newSCESet(countData=counts)
X <- calculateQCMetrics(X, list(Spikes=is.spike))
setSpike(X) <- "Spikes"

# Dummying up some size factors (for convenience only, use computeSumFactors() instead).
sizeFactors(X) <- 1
X <- computeSpikeFactors(X, general.use=FALSE)

# Running it.
out <- improvedCV2(X, spike.type="Spikes")
head(out)
```

mnnCorrect *Mutual nearest neighbors correction*

Description

Correct for batch effects in single-cell expression data using the mutual nearest neighbors method.

Usage

```
mnnCorrect(..., inquiry.genes=NULL, hvg.genes=NULL, k=20, sigma=0.1,
    cos.norm=TRUE, svd.dim=2, order=NULL)
```

Arguments

`...` Two or more expression matrices where genes correspond to rows and cells correspond to columns. Each matrix should contain cells from the same batch; multiple matrices represent separate batches of cells. Each matrix should contain the same number of rows, corresponding to the same genes (in the same order).

`inquiry.genes` A vector specifying the genes in all batches on which the correction is to be performed. All genes are used by default.

`hvg.genes` A vector specifying the genes in all batches with which to identify the mutual nearest neighbours. All genes are used by default.

`k` An integer scalar specifying the number of nearest neighbors to consider when identifying mutual nearest neighbors.

`sigma` A numeric scalar specifying the bandwidth of the Gaussian smoothing kernel used to compute the correction vector for each cell.

`cos.norm` A logical scalar indicating whether cosine normalization should be performed.

`svd.dim` An integer scalar specifying the number of dimensions to use for summarizing biological substructure within each batch.

`order` An integer vector specifying the order in which batches are to be corrected.

Details

This function is designed for batch correction of single-cell RNA-seq data where the batches are partially confounded with biological conditions of interest. It does so by identifying pairs of mutual nearest neighbors (MNN) in the high-dimensional expression space. Each MNN pair represents cells in different batches that are of the same cell type/state, assuming that batch effects are mostly orthogonal to the biological manifold. Correction vectors are calculated from the pairs of MNNs and corrected expression values are returned for use in clustering and dimensionality reduction.

The concept of a MNN pair can be explained by considering cells in each of two batches. For each cell in one batch, the set of `k` nearest cells in the other batch is identified, based on the Euclidean distance in expression space. Two cells in different batches are considered to be MNNs if each cell is in the other's set. The size of `k` can be interpreted as the minimum size of a subpopulation in each batch. The algorithm is generally robust to the choice of `k`, though values that are too small will not yield enough MNN pairs, while values that are too large will ignore substructure within each batch.

For each MNN pair, a pairwise correction vector is computed based on the difference in the expression profiles. The correction vector for each cell is computed by applying a Gaussian smoothing kernel with bandwidth `sigma` is the pairwise vectors. This stabilizes the vectors across many MNN pairs and extends the correction to those cells that do not have MNNs. The choice of `sigma` determines the extent of smoothing - a value of 1 is used by default to reflect the boundaries of the space after cosine normalization.

Value

A named list containing two components:

corrected A list of length equal to the number of batches, containing matrices of corrected expression values for each cell in each batch. The order of batches is the same as supplied in `...`, and the order of cells in each matrix is also unchanged.

mnn An integer matrix with two columns, specifying the number of MNNs used for correction of each batch. Each row corresponds to a batch, and the first and second columns contain the number of MNNs in the reference and current batch, respectively.

Further options

The input expression values should generally be log-transformed, e.g., log-counts, see `normalize` for details. By default, a further cosine normalization step is performed on the supplied expression data prior to identifying MNNs, etc. This can be turned off with `cos.norm=FALSE`, though we do not advise doing so.

The function depends on a shared biological manifold, i.e., one or more cell types/states being present in multiple batches. If this is not true, MNNs may be incorrectly identified. We protect against this by raising a warning if there are no shared biological subspaces between batches. We also remove components of the correction vectors that are parallel to the biological subspaces in each batch, to avoid over-correction of interesting biology. The biological subspace in each batch is identified with SVD, analogous to taking the first several principal components after PCA. The number of dimensions of this subspace can be controlled with `svd.dim`.

Users should note that the order in which batches are corrected will affect the final results. The first batch in `order` is used as the reference batch against which the second batch is corrected. Corrected values of the second batch are added to the reference batch, against which the third batch is corrected, and so on. This strategy maximizes the chance of detecting sufficient MNN pairs for stable calculation of correction vectors. We would consider 20 cells involved in MNN pairs to be the minimum number required for batch correction.

Author(s)

Laleh Haghverdi, with modifications by Aaron Lun

See Also

`get.knnx`

Examples

```
B1 <- matrix(rnorm(10000), ncol=50) # Batch 1
B2 <- matrix(rnorm(10000), ncol=50) # Batch 2
out <- mnnCorrect(B1, B2) # corrected values
```

overlapExprs *Overlap expression profiles*

Description

Compute the gene-specific overlap in expression profiles between two groups of cells.

Usage

```
## S4 method for signature 'matrix'
overlapExprs(x, groups, design=NULL, residuals=FALSE, tol=1e-8, subset.row=NULL)

## S4 method for signature 'SCESet'
overlapExprs(x, ..., subset.row=NULL, assay="exprs", get.spikes=FALSE)
```

Arguments

`x`	A numeric matrix of expression values, where each column corresponds to a cell and each row corresponds to an endogenous gene. Alternatively, a SCESet object containing such a matrix.
`groups`	A vector of group assignments for all cells.
`design`	A numeric matrix containing blocking terms, i.e., uninteresting factors driving expression across cells.
`residuals`	A logical scalar indicating whether overlaps should be computed between residuals of a linear model.
`tol`	A numeric scalar specifying the tolerance with which ties are considered.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	Additional arguments to pass to the matrix method.
`assay`	A string specifying which assay values to use, e.g., `counts` or `exprs`.
`get.spikes`	A logical scalar specifying whether decomposition should be performed for spike-ins.

Details

For two groups of cells A and B, consider the distribution of expression values for gene X across those cells. The overlap proportion is defined as the probability that a randomly selected cell in A has a greater expression value of X than a randomly selected cell in B. Overlap proportions near 0 or 1 indicate that the expression distributions are well-separated. In particular, large proportions indicate that most cells of the first group (A) express the gene more highly than most cells of the second group (B).

This function computes, for each gene, the overlap proportions between all pairs of groups in `groups`. It is designed to complement `findMarkers`, which reports the log-fold changes between groups. This is useful for prioritizing candidate markers that are distinctive to one group or another, without needing to plot the expression values.

Expression values that are tied between groups are considered to be 50% likely to be greater in either group. Thus, if all values were tied, the overlap proportion would be 0.5. The tolerance with which ties are considered can be set by changing `tol`.

By default, spike-in transcripts are ignored in `overlapExprs,SCESet-method` with `get.spikes=FALSE`. This is overridden by any non-`NULL` value of `subset.row`.

Value

A named list of numeric matrices. Each matrix corresponds to a group (A) in `groups` and contains one row per gene in `x` (or the subset specified by `subset.row`). Each column corresponds to another group (B) in `groups`. The matrix entries contain overlap proportions between groups A and B for each gene.

Accounting for uninteresting variation

If the experiment has known (and uninteresting) factors of variation, these can be included in `design`. The approach used to remove these factors depends on the design matrix. If there is only one factor in `design`, the levels of the factor are defined as separate blocks. Overlaps between groups are computed within each block, and a weighted mean (based on the number of cells in each block) of the overlaps is taken across all blocks.

This approach avoids the need for linear modelling and the associated assumptions regarding normality and correct model specification. In particular, it avoids problems with breaking of ties when counts or expression values are converted to residuals. However, it also makes less use of information, e.g., we ignore any blocks containing cells from only one group. `NA` proportions may be observed for a pair of groups if there is no block that contains cells from that pair.

For designs containing multiple factors or covariates, a linear model is fitted to the (log-normalized) expression values with `design`. The overlaps are then computed from the residuals of the fitted model. This approach is not ideal (see `?correlatePairs` for a related discussion) but is unavoidable for covariates. It can also be used for one-way layouts by setting `residuals=TRUE`.

Author(s)

Aaron Lun

See Also

`findMarkers`

Examples

```
example(computeSpikeFactors) # Using the mocked-up data 'y' from this example.
groups <- sample(3, ncol(y), replace=TRUE)
out <- overlapExprs(y, groups, subset.row=1:10)
```

`Quick clustering` *Quick clustering of cells*

Description

Cluster similar cells based on rank correlations in their gene expression profiles.

Usage

```
## S4 method for signature 'matrix'
quickCluster(x, min.size=200, subset.row=NULL, get.ranks=FALSE,
    method=c("hclust", "igraph"), ...)

## S4 method for signature 'SCESet'
quickCluster(x, subset.row=NULL, ..., assay="counts", get.spikes=FALSE)
```

Arguments

`x` A numeric count matrix where rows are genes and columns are cells. Alternatively, a SCESet object containing such a matrix.

`min.size` An integer scalar specifying the minimum size of each cluster for `method="hclust"`.

`subset.row` A logical, integer or character scalar indicating the rows of `x` to use.

`get.ranks` A logical scalar specifying whether a matrix of adjusted ranks should be returned.

`method` A string specifying the clustering method to use.

`...` For `quickCluster,matrix-method`, additional arguments to be passed to `cutreeDynamic` for `method="hclust"`, or `buildSNNGraph` for `method="igraph"`. For `quickCluster,SCESet-meth` additional arguments to pass to `quickCluster,matrix-method`.

`assay` A string specifying which assay values to use, e.g., `counts` or `exprs`.

`get.spikes` A logical specifying whether spike-in transcripts should be used.

Details

This function provides a correlation-based approach to quickly define clusters of a minimum size `min.size`. Two clustering strategies are available:

- If `method="hclust"`, a distance matrix is constructed using Spearman's rho on the counts between cells. (Some manipulation is performed to convert Spearman's rho into a proper distance metric.) Hierarchical clustering is performed and a dynamic tree cut is used to define clusters of cells.
- If `method="igraph"`, a shared nearest neighbor graph is constructed using the `buildSNNGraph` function. This is used to define clusters based on highly connected communities in the graph, using the `cluster_fast_greedy` function. Again, neighbors are identified using distances based on Spearman's rho.

A correlation-based approach is preferred here as it is invariant to scaling normalization. This avoids circularity between normalization and clustering, e.g., in `computeSumFactors`.

When using `cutreeDynamic` some cells may not be assigned to any cluster, and are assigned identities of `"0"` in the output vector. In most cases, this is because those cells belong in a separate cluster with fewer than `min.size` cells. The function will not be able to call this as a cluster as the minimum threshold on the number of cells has not been passed. Users are advised to check that the unassigned cells do indeed form their own cluster. Otherwise, it may be necessary to use a custom clustering algorithm.

Using `method="igraph"` should be used in situations where there are too many cells for construction of a distance matrix. It can also be used in cases with few cells, though it is less customizable than `method="hclust"` - for example, there are no options to control the minimum cluster size. (When there are many cells, the “quick” in the function's name refers to the number of commands you have to call rather than the computation speed.)

In `quickCluster,SCESet-method`, spike-in transcripts are not used by default as they provide little information on the biological similarities between cells. This may not be the case if subpopulations differ by total RNA content, in which case setting `get.spikes=TRUE` may provide more discriminative power. Users can also set `subset.row` to specify which rows of `x` are to be used to calculate correlations. This is equivalent to but more efficient than subsetting `x` directly, as it avoids constructing a (potentially large) temporary matrix. Note that if `subset.row` is specified, it will overwrite any setting of `get.spikes`.

Users can also set get.ranks=TRUE, in which case a matrix of ranks will be returned. Each column contains the ranks for the expression values within a single cell after standardization and mean-centring. Computing Euclidean distances between the rank vectors for pairs of cells will yield the same correlation-based distance as that used above. This allows users to apply their own clustering algorithms on the ranks, which protects against outliers and is invariant to scaling (at the cost of sensitivity).

Value

If get.ranks=FALSE, a character vector of cluster identities for each cell in counts is returned.

If get.ranks=TRUE, a numeric matrix is returned where each column contains ranks for the expression values in each cell.

Author(s)

Aaron Lun and Karsten Bach

References

van Dongen S and Enright AJ (2012). Metric distances derived from cosine similarity and Pearson and Spearman correlations. *arXiv* 1208.3145

Lun ATL, Bach K and Marioni JC (2016). Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biol.* 17:75

See Also

cutreeDynamic, computeSumFactors, buildSNNGraph

Examples

```
set.seed(100)
popsize <- 200
ngenes <- 1000
all.facs <- 2^rnorm(popsize, sd=0.5)
counts <- matrix(rnbinom(ngenes*popsize, mu=all.facs, size=1), ncol=popsize, byrow=TRUE)

clusters <- quickCluster(counts, min.size=20)
clusters <- quickCluster(counts, method="igraph")
```

sandbag *Cell cycle phase training*

Description

Use gene expression data to train a classifier for cell cycle phase.

Usage

```
## S4 method for signature 'matrix'
sandbag(x, phases, gene.names=rownames(x),
    fraction=0.5, subset.row=NULL)

## S4 method for signature 'SCESet'
sandbag(x, phases, subset.row=NULL, ...,
    assay="counts", get.spikes=FALSE)
```

Arguments

`x` A numeric matrix of gene expression values where rows are genes and columns are cells. Alternatively, a SCESet object containing such a matrix.

`phases` A list of subsetting vectors specifying which cells are in each phase of the cell cycle. This should typically be of length 3, with elements named as `"G1"`, `"S"` and `"G2M"`.

`gene.names` A character vector of gene names.

`fraction` A numeric scalar specifying the minimum fraction to define a marker gene pair.

`subset.row` A logical, integer or character scalar indicating the rows of `x` to use.

`...` Additional arguments to pass to `sandbag,matrix-method`.

`assay` A string specifying which assay values to use, e.g., `counts` or `exprs`.

`get.spikes` A logical specifying whether spike-in transcripts should be used.

Details

This function implements the training step of the pair-based prediction method described by Scialdone et al. (2015). Pairs of genes (A, B) are identified from a training data set where in each pair, the fraction of cells in phase G1 with expression of A > B (based on expression values in `training.data`) and the fraction with B > A in each other phase exceeds `fraction`. These pairs are defined as the marker pairs for G1. This is repeated for each phase to obtain a separate marker pair set.

Pre-defined sets of marker pairs are provided for mouse and human (see Examples). The mouse set was generated as described by Scialdone et al. (2015), while the human training set was generated with data from Leng et al. (2015). Classification from test data can be performed using the `cyclone` function. For each cell, this involves comparing expression values between genes in each marker pair. The cell is then assigned to the phase that is consistent with the direction of the difference in expression in the majority of pairs.

For `sandbag,SCESet-method`, the matrix of counts is used but can be replaced with expression values by setting `assays`. By default, `get.spikes=FALSE` which means that any rows corresponding to spike-in transcripts will not be considered when picking markers. This is because the amount of spike-in RNA added will vary between experiments and will not be a robust predictor. Nonetheless, if all rows are required, users can set `get.spikes=TRUE`. Users can also manually select which rows to use via `subset.row`, which will override any setting of `get.spikes`.

While `sandbag` and its partner function `cyclone` were originally designed for cell cyclone phase classification, the same computational strategy can be used to classify cells into any mutually exclusive groupings. Any number and nature of groups can be specified in `phases`, e.g., differentiation lineages, activation states. Only the names of `phases` need to be modified to reflect the biology being studied.

Value

A named list of data.frames, where each data frame corresponds to a cell cycle phase and contains the names of the genes in each marker pair.

Author(s)

Antonio Scialdone, with modifications by Aaron Lun

References

Scialdone A, Natarajana KN, Saraiva LR et al. (2015). Computational assignment of cell-cycle stage from single-cell transcriptome data. *Methods* 85:54–61

Leng N, Chu LF, Barry C et al. (2015). Oscope identifies oscillatory genes in unsynchronized single-cell RNA-seq experiments. *Nat. Methods* 12:947–50

See Also

`cyclone`

Examples

```
ncells <- 50
ngenes <- 20
training <- matrix(rnorm(ncells*ngenes), ncol=ncells)
rownames(training) <- paste0("X", seq_len(ngenes))

is.G1 <- 1:20
is.S <- 21:30
is.G2M <- 31:50
out <- sandbag(training, list(G1=is.G1, S=is.S, G2M=is.G2M))
str(out)

# Getting pre-trained marker sets
mm.pairs <- readRDS(system.file("exdata", "mouse_cycle_markers.rds", package="scran"))
hs.pairs <- readRDS(system.file("exdata", "human_cycle_markers.rds", package="scran"))
```

`Selector plot` *Construct a selector plot via Shiny*

Description

Generate an interactive Shiny plot in which cells can be selected for further analysis.

Usage

```
selectorPlot(x, y, persist=FALSE, plot.width=5, plot.height=500, run=TRUE, pch=16, ...)
```

Arguments

`x, y`	Numeric vectors of x-y coordinates, of length equal to the number of cells.
`persist`	A logical scalar indicating whether selections should persist after stopping the app.
`plot.width`	A numeric scalar specifying the plot width, see `width` in `?column`.
`plot.height`	A numeric scalar specifying the plot height in pixels.
`run`	A logical scalar specifying whether the Shiny app should be run.
`pch, ...`	Other arguments to pass to `plot`.

Details

This function will return a Shiny app object that can be run with `runApp`. The aim is to perform dimensionality reduction to obtain coordinates for each cell, e.g., from PCA or t-SNE. These coordinates can be plotted with `selectorPlot`, and subpopulations of interest can be interactively selected. The selections can then be saved for further manipulation in R.

The app allows users to select groups of cells; mark them as cells of interest; and then save the marked cells into a list. Currently marked cells will be shown in red, previously saved cells are shown in orange, and all other cells are shown in grey. The distribution of saved cells is also shown in a separate plot indicating the list element to which they were saved. This can be repeated multiple times to obtain several groups of interest.

Several buttons are available within the app:

“Select”: Marks the current selection of cells.

“Deselect”: Unmarks the current selection of cells.

“Clear selection”: Unmarks all currently marked cells.

“Add to list”: Saves currently marked cells into a list.

“Reset all”: Removes all marking, removes all saved cells from the list.

“Save list to R”: Stops the app and returns the list of saved cells to R.

Value

If `run=FALSE`, a Shiny app object is returned, which can be run with `runApp`. This transfers control to a browser window where cells can be selected. Upon stopping the app with the “Save list to R” button, control is transferred back to R and the list of saved cells is returned. Each element of the list is a logical vector indicating which cells were saved in that group of interest.

If `run=TRUE`, a Shiny app object is created and run. This returns a list of saved cells upon stopping the app as previously described.

Author(s)

Aaron Lun

See Also

`runApp`

Examples

```
example(newSCESet)
pca <- plotPCA(example_sceset)
x <- pca$data$PC1
y <- pca$data$PC2

# Creating the app object.
app <- selectorPlot(x, y, run=FALSE)
if (interactive()) { saved <- shiny::runApp(app) }

## Not run: # Running the app directly from the function.
saved <- selectorPlot(x, y)

## End(Not run)
```

`Spike-in normalization` *Normalization with spike-in counts*

Description

Compute size factors based on the coverage of spike-in transcripts.

Usage

```
## S4 method for signature 'SCESet'
computeSpikeFactors(x, type=NULL, sf.out=FALSE, general.use=TRUE)
```

Arguments

`x`	A SCESet object containing rows corresponding spike-in transcripts.
`type`	A character vector specifying which spike-in sets to use.
`sf.out`	A logical scalar indicating whether only size factors should be returned.
`general.use`	A logical scalar indicating whether the size factors should be stored for general use by all genes.

Details

The size factor for each cell is defined as the sum of all spike-in counts in each cell. This is equivalent to normalizing to equalize spike-in coverage between cells. Size factors are scaled so that the mean of all size factors is unity, for standardization purposes if one were to compare different sets of size factors.

Spike-in counts are assumed to be stored in the rows specified by `isSpike(x)`. This specification should have been performed by supplying the names of the spike-in sets – see ?`setSpike` for more details. By default, if multiple spike-in sets are available, all of them will be used to compute the size factors. The function can be restricted to a subset of the spike-ins by specifying the names of the desired spike-in sets in `type`.

By default, the function will store several copies of the same size factors in the output object. One copy will be stored in `sizeFactors(x)` for normalization of all genes – this can be disabled by setting `general.use=FALSE`. One copy will also be stored in `sizeFactors(x, type=s)`, where s is the name of a spike-in set in type. (If type=NULL, a copy is stored for every spike-in set, as all of them would be used to compute the size factors.) Separate storage allows spike-in-specific normalization in normalize,SCESet-method.

Value

If sf.out=TRUE, a numeric vector of size factors is returned directly.

Otherwise, an object of class x is returned, containing size factors for all cells. A copy of the vector is stored for each spike-in set that was used to compute the size factors. If general.use=TRUE, a copy is also stored for use by non-spike-in genes.

Author(s)

Aaron Lun

References

Lun ATL, McCarthy DJ and Marioni JC (2016). A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.* 5:2122

See Also

isSpike, setSpike

Examples

```
################
# Mocking up some data.
set.seed(100)
ncells <- 200

nspikes <- 100
spike.means <- 2^runif(nspikes, 3, 8)
spike.disp <- 100/spike.means + 0.5
spike.data <- matrix(rnbinom(nspikes*ncells, mu=spike.means, size=1/spike.disp), ncol=ncells)

ngenes <- 2000
cell.means <- 2^runif(ngenes, 2, 10)
cell.disp <- 100/cell.means + 0.5
cell.data <- matrix(rnbinom(ngenes*ncells, mu=cell.means, size=1/cell.disp), ncol=ncells)

combined <- rbind(cell.data, spike.data)
colnames(combined) <- seq_len(ncells)
rownames(combined) <- seq_len(nrow(combined))
y <- newSCESet(countData=combined)
y <- calculateQCMetrics(y, list(Spike=rep(c(FALSE, TRUE), c(ngenes, nspikes))))
setSpike(y) <- "Spike"

################
# Computing and storing spike-in size factors.
y2 <- computeSpikeFactors(y)
head(sizeFactors(y2))
head(sizeFactors(y2, type="Spike"))

# general.use=FALSE does not modify general size factors
sizeFactors(y2) <- 1
sizeFactors(y2, type="Spike") <- 1
y2 <- computeSpikeFactors(y2, general.use=FALSE)
head(sizeFactors(y2))
head(sizeFactors(y2, type="Spike"))
```

`technicalCV2` *Model the technical coefficient of variation*

Description

Model the technical coefficient of variation as a function of the mean, and determine the significance of highly variable genes.

Usage

```
## S4 method for signature 'matrix'
technicalCV2(x, is.spike, sf.cell=NULL, sf.spike=NULL,
    cv2.limit=0.3, cv2.tol=0.8, min.bio.disp=0.25)

## S4 method for signature 'SCESet'
technicalCV2(x, spike.type=NULL, ..., assay="counts")
```

Arguments

`x` A numeric matrix of counts, where each column corresponds to a cell and each row corresponds to a spike-in transcript. Alternatively, a SCESet object that contains such values.

`is.spike` A vector indicating which rows of `x` correspond to spike-in transcripts.

`sf.cell` A numeric vector containing size factors for endogenous genes.

`sf.spike` A numeric vector containing size factors for spike-in transcripts.

`cv2.limit, cv2.tol`
Numeric scalars that determine the minimum mean abundance for the spike-in transcripts to be used for trend fitting.

`min.bio.disp` A numeric scalar specifying the minimum biological dispersion.

`spike.type` A character vector containing the names of the spike-in sets to use.

`...` Additional arguments to pass to `technicalCV2,matrix-method`.

`assay` A string specifying which assay values to use.

Details

This function will estimate the squared coefficient of variation (CV2) and mean for each spike-in transcript. A mean-dependent trend is fitted to the CV2 values for the transcripts using a Gamma GLM with `glmgam.fit`. Only high-abundance transcripts are used for stable trend fitting. (Specifically, a mean threshold is selected by taking all transcripts with CV2 above `cv2.limit`, and taking the quantile of this subset at `cv2.tol`. A warning will be thrown and all spike-ins will be used if the subset is empty.)

The trend is used to determine the technical CV2 for each endogenous gene based on its mean. To identify highly variable genes, the null hypothesis is that the total CV2 for each gene is less than or equal to the technical CV2 plus `min.bio.disp`. Deviations from the null are identified using a chi-squared test. The additional min.bio.disp is necessary for a ratio-based test, as otherwise genes with large relative (but small absolute) CV2 would be favoured.

For technicalCV2,matrix-method, the rows corresponding to spike-in transcripts are specified with is.spike. These rows will be used for trend fitting, while all other rows are treated as endogenous genes. If either sf.cell or sf.spike are not specified, the estimateSizeFactorsForMatrix function is applied to compute size factors.

For technicalCV2,SCESet-method, transcripts from spike-in sets named in spike.type will be used for trend fitting. If spike.type=NULL, all spike-in sets listed in x will be used. Size factors for the endogenous genes are automatically extracted via sizeFactors. Spike-in-specific size factors for spike.type are extracted from x, if available; otherwise they are set to the size factors for the endogenous genes. Note that the spike-in-specific factors must be the same for each set in spike.type.

Users can also set is.spike to NA in technicalCV2,matrix-method; or spike.type to NA in technicalCV2,SCESet-method. In such cases, all rows will be used for trend fitting, and (adjusted) p-values will be reported for all rows. This should be used in cases where there are no spike-ins. Here, the assumption is that most endogenous genes do not exhibit high biological variability and thus can be used to model technical variation.

Value

A data frame is returned containing one row per row of x (including both endogenous genes and spike-in transcripts). Each row contains the following information:

mean: A numeric field, containing mean (scaled) counts for all genes and transcripts.

var: A numeric field, containing the variances for all genes and transcripts.

cv2: A numeric field, containing CV2 values for all genes and transcripts.

trend: A numeric field, containing the fitted value of the trend in the CV2 values. Note that the fitted value is reported for all genes and transcripts, but the trend is only fitted using the transcripts.

p.value: A numeric field, containing p-values for all endogenous genes (NA for rows corresponding to spike-in transcripts).

FDR: A numeric field, containing adjusted p-values for all genes.

Author(s)

Aaron Lun, based on code from Brennecke et al. (2013)

References

Brennecke P, Anders S, Kim JK et al. (2013). Accounting for technical noise in single-cell RNA-seq experiments. *Nat. Methods* 10:1093-95

See Also

glmgam.fit, estimateSizeFactorsForMatrix

Examples

```
# Mocking up some data.
ngenes <- 10000
means <- 2^runif(ngenes, 6, 10)
dispersions <- 10/means + 0.2
```

```
nsamples <- 50
counts <- matrix(rnbinom(ngenes*nsamples, mu=means, size=1/dispersions), ncol=nsamples)
is.spike <- logical(ngenes)
is.spike[seq_len(500)] <- TRUE

# Running it directly on the counts.
out <- technicalCV2(counts, is.spike)
head(out)
plot(out$mean, out$cv2, log="xy")
points(out$mean, out$trend, col="red", pch=16, cex=0.5)

# Same again with an SCESet.
rownames(counts) <- paste0("X", seq_len(ngenes))
colnames(counts) <- paste0("Y", seq_len(nsamples))
X <- newSCESet(countData=counts)
X <- calculateQCMetrics(X, list(Spikes=is.spike))
setSpike(X) <- "Spikes"

# Dummying up some size factors (for convenience only, use computeSumFactors() instead).
sizeFactors(X) <- 1
X <- computeSpikeFactors(X, general.use=FALSE)

# Running it.
out <- technicalCV2(X, spike.type="Spikes")
head(out)
```

testVar *Test for significantly large variances*

Description

Test for whether the total variance exceeds that expected under some null hypothesis, for sample variances estimated from normally distributed observations.

Usage

```
testVar(total, null, df, design=NULL, test=c("chisq", "f"), second.df=NULL)
```

Arguments

`total`	A numeric vector of total variances for all genes.
`null`	A numeric scalar or vector of expected variances under the null hypothesis for all genes.
`df`	An integer scalar specifying the degrees of freedom on which the variances were estimated.
`design`	A design matrix, used to determine the degrees of freedom if `df` is missing.
`test`	A string specifying the type of test to perform.
`second.df`	A numeric scalar specifying the second degrees of freedom for the F-distribution when `test="f"`.

Details

The null hypothesis is that the true variance for each gene is equal to `null`. (Technically, it is that the variance is equal to or less than this value, but the most conservative test is obtained at equality.) If `test="chisq"`, variance estimates are assumed to follow a chi-squared distribution on `df` degrees of freedom and scaled by `null/df`. This is used to compute a p-value for `total` being greater than `null`. The underlying assumption is that the observations are normally distributed under the null, which is reasonable for log-counts with low-to-moderate dispersions.

The aim is to use this function to identify significantly highly variable genes (HVGs). For example, the `null` vector can be set to the values of the trend fitted to the spike-in variances. This will identify genes with variances significantly greater than technical noise. Alternatively, it can be set to the trend fitted to the cellular variances, which will identify those that are significantly more variable than the bulk of genes. Selecting HVGs on p-values is better than using `total - null`, as the latter is less precise when `null` is large.

If `test="f"`, the true variance of each spike-in transcript is assumed to be sampled from a scaled inverse chi-squared distribution. This accounts for any inflated scatter around the trend due to differences in amplification efficiency between transcripts. As a result, the gene-wise variance estimates are should be F-distributed around the trend under the null. The second degrees of freedom is estimated from the scatter around the trend in `trendVar` using `fitFDistRobustly`, and needs to be supplied to `second.df` to calculate an appropriate p-value.

Value

A numeric vector of p-values for all genes.

Author(s)

Aaron Lun

References

Law CW, Chen Y, Shi W and Smyth GK (2014). voom: precision weights unlock linear model analysis tools for RNA-seq read counts *Genome Biol.* 15(2), R29.

See Also

`trendVar`, `decomposeVar`, `fitFDistRobustly`

Examples

```
set.seed(100)
null <- 100/runif(1000, 50, 2000)
df <- 30
total <- null * rchisq(length(null), df=df)/df

# Direct test:
out <- testVar(total, null, df=df)
hist(out)

# Rejecting the null:
alt <- null * 5 * rchisq(length(null), df=df)/df
out <- testVar(alt, null, df=df)
plot(alt[order(out)]-null)

# Focusing on genes that have high absolute increases in variability:
```

```
out <- testVar(alt, null+0.5, df=df)
plot(alt[order(out)]-null)
```

`trendVar` *Fit a variance trend*

Description

Fit a mean-dependent trend to the gene-specific variances in single-cell RNA-seq data.

Usage

```
## S4 method for signature 'matrix'
trendVar(x, trend=c("loess", "semiloess"), span=0.3,
    family="symmetric", degree=1, start=NULL, design=NULL, subset.row=NULL)

## S4 method for signature 'SCESet'
trendVar(x, subset.row=NULL, ..., assay="exprs", use.spikes=TRUE)
```

Arguments

`x` A numeric matrix of normalized expression values, where each column corresponds to a cell and each row corresponds to a spike-in transcript. Alternatively, a SCESet object that contains such values.

`trend` A string indicating whether the trend should be fully loess-based, or a mixture of loess and parametric fitting.

`span, family, degree`
Arguments to pass to `loess`.

`start` A named list of numeric scalars, containing starting values for parametric fitting with `nls`. This is automatically generated with the most suitable values if set to `NULL`.

`design` A numeric matrix describing the uninteresting factors contributing to expression in each cell.

`subset.row` A logical, integer or character scalar indicating the rows of `x` to use.

`...` Additional arguments to pass to `trendVar,matrix-method`.

`assay` A string specifying which assay values to use, e.g., `counts` or `exprs`.

`use.spikes` A logical scalar specifying whether the trend should be fitted to variances for spike-in transcripts or endogenous genes.

Details

The strategy is to fit an abundance-dependent trend to the variance of the log-normalized expression for the spike-in transcripts, using `trendVar`. For SCESet objects, these expression values can be computed by `normalize` after setting the size factors, e.g., with `computeSpikeFactors`. Log-transformed values are used as these are more robust to genes/transcripts with strong expression in only one or two outlier cells.

The mean and variance of the normalized log-counts is calculated for each spike-in transcript, and a trend is fitted to the variance against the mean for all transcripts. The fitted value of this trend represents technical variability due to sequencing, drop-outs during capture, etc. Variance decomposition to biological and technical components for endogenous genes can then be performed later with decomposeVar.

The design matrix can be set if there are factors that should be blocked, e.g., batch effects, known (and uninteresting) clusters. Otherwise, it will default to an all-ones matrix, effectively treating all cells as part of the same group.

Value

A named list is returned, containing:

mean: A numeric vector of mean log-CPMs for all spike-in transcripts.

var: A numeric vector of the variances of log-CPMs for all spike-in transcripts.

trend: A function that returns the fitted value of the trend at any mean log-CPM.

design: A numeric matrix, containing the design matrix that was used.

Trend fitting options

By default, a robust loess curve is used for trend fitting via loess. This protects against genes with very large or very small variances. Some experimentation with span, degree or family may be required to obtain satisfactory results. The fit is also dependent on the quality of the spike-ins – the fit will obviously be poor if the coverage of all spike-ins is low.

Alternatively, when trend="semiloess", a non-linear curve of the form

$$y = \frac{ax}{x^n + b}$$

is fitted to the variances against the means using nls, and a loess curve is then fitted to the log-ratios of the observed to fitted values. The parametric curve reduces the sharpness of the trend for easier loess fitting. Conversely, the parametric form is not exact, so the loess curve models any remaining trends in the residuals.

In general, the semi-loess setting tends to give smoother curves than loess alone. It is more robust to the uneven distribution of spike-in transcripts across the covariate range. However, it tends to be susceptible to convergence issues, and may require some fiddling with the start values to converge properly. Reasonably good starting values for a, n and b are chosen automatically but these can be set manually with start.

Additional notes on row selection

Spike-in transcripts can be selected in trendVar,SCESet-method using the use.spikes method. By default, use.spikes=TRUE which means that only rows labelled as spike-ins with isSpike(x) will be used.

When spike-ins are not available, trendVar can also be applied directly to the counts for endogenous genes by setting use.spikes=FALSE (or by manually supplying a matrix of normalized expression for endogenous genes, for trendVar,matrix-method). This assumes that most genes exhibit technical variation and little biological variation, e.g., in a homogeneous population.

If use.spikes=NA, every row will be used for trend fitting, regardless of whether it corresponds to a spike-in transcript or endogenous gene. Users can also directly specify which rows to use with subset.row. This will override any setting of use.spikes.

Warning on size factor centring

If assay="exprs", trendVar,SCESet-method will attempt to determine if the expression values were computed from counts via normalize. If so, a warning will be issued if the size factors are not centred at unity. This is because different size factors are typically used for endogenous genes and spike-in transcripts. If these size factor sets are not centred at the same value, there will be systematic differences in abundance between these features. This precludes the use of a spike-in fitted trend with abundances for endogenous genes in decomposeVar.

For other expression values and in trendVar,matrix-method, the onus is on the user to ensure that normalization preserves differences in abundance. In other words, the scaling factors used to normalize each feature should have the same mean. This ensures that spurious differences in abundance are not introduced by the normalization process.

Author(s)

Aaron Lun

References

Lun ATL, McCarthy DJ and Marioni JC (2016). A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.* 5:2122

See Also

nls, loess, decomposeVar, computeSpikeFactors, computeSumFactors, normalize

Examples

```
example(computeSpikeFactors) # Using the mocked-up data 'y' from this example.

# Normalizing (gene-based factors for genes, spike-in factors for spike-ins)
y <- computeSumFactors(y)
y <- computeSpikeFactors(y, general.use=FALSE)
y <- normalize(y)

# Fitting a trend to the spike-ins.
fit <- trendVar(y)
plot(fit$mean, fit$var)
curve(fit$trend(x), col="red", lwd=2, add=TRUE)

# Fitting a trend to the endogenous genes.
fit.g <- trendVar(y, use.spikes=FALSE)
plot(fit.g$mean, fit.g$var)
curve(fit.g$trend(x), col="red", lwd=2, add=TRUE)
```

Index