Package 'scran'

April 15, 2017

Version 1.2.2

Date 2017-01-18

Title Methods for Single-Cell RNA-Seq Data Analysis

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

Depends R (>= 3.3.0), BiocParallel, scater

Imports dynamicTreeCut, zoo, edgeR, stats, BiocGenerics, methods, Biobase, utils, Matrix, shiny, graphics, grDevices, statmod

Suggests limSolve, limma, testthat, knitr, BiocStyle, org.Mm.eg.db, DESeq2, monocle, S4Vectors, ggplot2

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

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]

R topics documented:

convertTo . . . 2
correlatePairs . . . 4
cyclone . . . 7
decomposeVar . . . 9
Deconvolution Methods . . . 11
Distance-to-median . . . 13
Get spikes . . . 14
Quick clustering . . . 16
sandbag . . . 18
Selector plot . . . 19
Spike-in normalization . . . 21
technicalCV2 . . . 22
testVar . . . 25
trendVar . . . 26

Index 30

`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

```
ncells <- 200
ngenes <- 1000
count.sizes <- rnbinom(ncells, mu=100, size=5)
dummy <- matrix(count.sizes, ncol=ncells, nrow=ngenes, byrow=TRUE)
rownames(dummy) <- paste0("X", seq_len(ngenes))

X <- newSCESet(countData=data.frame(dummy))
X <- calculateQCMetrics(X, list(Spike=rbinom(ngenes, 1, 0.5)==0L))
isSpike(X) <- "Spike"
sizeFactors(X) <- 2^rnorm(ncells)
X <- normalize(X)

fData(X)$SYMBOL <- paste0("X", seq_len(ngenes))
X$other <- sample(LETTERS, ncells, replace=TRUE)

convertTo(X, type="edgeR")
convertTo(X, type="DESeq2")
convertTo(X, 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, residuals=FALSE, subset.row=NULL)

## S4 method for signature 'SCESet'
correlatePairs(x, subset.row=NULL, ..., 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 describing fixed effects to factorize out.
`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 scalar indicating the rows of `x` to use.
`...`	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.

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.

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, 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.

Rows are sorted by increasing p.value and, if tied, decreasing absolute size of rho.

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 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. Nuisance effects cannot be precisely

removed without placing some quantitative constraints on how observations change in response to those effects. In fact, even with `design=NULL`, there is the assumption that all observations are equally likely to receive any rank when performing the permutations. This will not be the case in practice as some observations are more variable than others, depending on the expected expression in each cell. Thus, caution is required at low counts where the log-expression values will be highly non-normal.)

Gene selection

Users should select their genes in `x` with some care. Using a top set of 100-200 highly variable genes (HVGs) is recommended. 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.

It is also advisable to choose HVGs after filtering on abundance to remove lowly-expressed genes. This is because tied counts may not result in tied normalized expression values or residuals. Uncertainty in model fitting with `design` will introduce differences due to coefficient estimation error (more so if `design` was misspecified). This will break ties in a consistent manner across genes, which may yield large correlations between genes with many zero counts. Focusing on HVGs should avoid detecting such correlations, as genes dominated by zeroes will usually have low variance.

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.

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). Consider classification 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.

To make the proportions comparable between phases, a distribution of proportions is constructed by shuffling the expression values within the cell and recalculating the proportion at each iteration. The phase score for that cell is then defined as the lower tail probability of this distribution. 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. Also, 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.

The same process is repeated for all phases, using the appropriate set of marker pairs in `pairs` for each phase. Cells with G1 or G2M scores above 0.5 should be assigned to the G1 or G2M phases, respectively. (If both are above 0.5, the higher score is used for assignment.) This is based on the interpretation of the score as 1 minus the p-value for the null distribution of proportions. The null hypothesis here is that expression of the marker genes is independent within each cell, i.e., with no cycle-induced correlations between marker pairs. Cells can be assigned to S phase based on the S phase score, but a more reliable approach is to define S phase cells based on those cells 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 `assays`. 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`.

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).

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)

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

# 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 systematic 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 decomposeVar,matrix,list-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 log-CPMs 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-CPM 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-CPMs 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-CPMs 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.

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

See Also

trendVar, testVar

Examples

```
set.seed(100)

nspikes <- ncells <- 200
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 <- 10000
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))))
isSpike(y) <- "Spike"

# Normalizing.
y <- computeSumFactors(y)
y <- computeSpikeFactors(y, general.use=FALSE)
y <- normalize(y)
```

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

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)
```

`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=c(20, 40, 60, 80, 100), 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 – twice the maximum value of sizes is recommended. 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 cell pool 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 a linear system. This is repeated with all window sizes in sizes to obtain an over-determined system that can be solved with methods like the QR decomposition.

In theory, it is possible to obtain negative estimates for the size factors. These are most likely for very small library sizes and are obviously nonsensical. 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.

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.

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

```
set.seed(100)
popsize <- 800
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)
out.facs <- computeSumFactors(counts)
```

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

```
ngenes <- 10000
means <- exp(runif(ngenes, 2, 8))
cv2 <- rgamma(ngenes, 5, 5)
dm.stat <- DM(means, cv2)
```

Get spikes *Construct the spike-in matrix*

Description

Identify rows in the SCESet corresponding to spike-in transcripts, and retrieve a matrix of counts or normalized expression values for those rows.

Usage

```
## S4 method for signature 'SCESet'
isSpike(x, type=NULL)
## S4 method for signature 'SCESet'
spikes(x, assay="counts", type=NULL)

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

Arguments

x	A SCESet object with spike-in data in the colData.
type	A character vector specifying which spike-in set(s) should be extracted.
assay	A string specifying whether counts or normalized expression values are to be extracted.
value	A character vector specifying which control sets are spike-ins. Alternatively a NULL value, to remove existing spike-in specifications.

Details

The isSpike and spikes methods indicate which rows correspond to spike-ins and their expression values, respectively. If multiple spike-in sets are available, users can extract information for specific sets by supplying the names of the set in type. (By default, rows from all spike-in sets are extracted when type=NULL.) If assay="exprs", users should have run x through normalize.

To specify rows as corresponding to spike-ins, we assume that calculateQCMetrics has already been applied to x. Specifically, we assume that spike-ins represent a subset of the control sets supplied as feature_controls in calculateQCMetrics. Users can assign a character vector to isSpike(x)<- containing the names of the control sets that are spike-ins. This will automatically construct a logical vector containing rows from all specified sets, for later retrieval with isSpike(x).

Setting isSpike(x)<-NULL will clear all existing spike-in information in x. Note that direct assignment of a logical vector to isSpike(x)<- is no longer permitted. This is because the names of the spike-in sets are necessary for downstream processing, but will not be included if isSpike(x) is set directly.

Value

For spikes, a numeric matrix of counts or normalized expression values, with one column per cell and one row per spike-in transcript.

For isSpike, a logical vector indicating which rows are spike-ins (or NULL, if this information was not stored in x).

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.

Note on overlapping sets

While it is possible to declare overlapping sets as the spike-in sets with isSpike(x)<-, this is not advisable. This is because some downstream operations assume that each row belongs to only one set (i.e., one of the spike-in sets, or the set of endogenous genes). For example, normalization will use size factors from only one of the sets, so correspondence to multiple sets will not be honoured. A warning will thus be raised if overlapping sets are specified in value.

Author(s)

Aaron Lun

See Also

normalize, calculateQCMetrics, SCESet

Examples

```
set.seed(100)
popsize <- 10
ngenes <- 1000
all.facs <- 2^rnorm(popsize, sd=0.5)
counts <- matrix(rnbinom(ngenes*popsize, mu=10*all.facs, size=1), ncol=popsize, byrow=TRUE)
spikes <- matrix(rnbinom(100*popsize, mu=10*all.facs, size=0.5), ncol=popsize, byrow=TRUE)

combined <- rbind(counts, spikes)
colnames(combined) <- seq_len(popsize)
rownames(combined) <- seq_len(nrow(combined))
y <- newSCESet(countData=combined)
y <- calculateQCMetrics(y, list(IAmASpike=rep(c(FALSE, TRUE), c(ngenes, 100))))
isSpike(y) <- "IAmASpike"

y <- computeSpikeFactors(y)
y <- normalize(y)
spikes(y)[1:10,]
spikes(y, assay="exprs")[1:10,]
isSpike(y)
```

`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, ...)
## 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.
`subset.row`	A logical, integer or character scalar indicating the rows of `x` to use.
`...`	For `quickCluster,matrix-method`, additional arguments to be passed to `cutreeDynamic`. For `quickCluster,SCESet-method`, 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`. A distance matrix is constructed using Spearman's correlation on the counts between cells. (Some manipulation is performed to convert the correlation into a proper distance metric.) Hierarchical clustering is performed and a dynamic tree cut is used to define clusters of cells. A correlation-based approach is preferred here as it is invariant to scaling normalization. This avoids circularity between normalization and clustering.

Note that some cells may not be assigned to any cluster. 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. If so, it is generally safe to ignore this warning and to treat all unassigned cells as a single cluster. Otherwise, it may be necessary to use a custom clustering algorithm.

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`.

Value

A vector of cluster identities for each cell in `counts`. Values of `"0"` are used to indicate cells that are not assigned to any cluster.

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`

Examples

```
set.seed(100)
popsize <- 200
ngenes <- 10000
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)
```

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, is.G1, is.S, is.G2M, gene.names=rownames(x),
    fraction=0.5, subset.row=NULL)

## S4 method for signature 'SCESet'
sandbag(x, is.G1, is.S, is.G2M, 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.
`is.G1, is.S, is.G2M`	A vector indicating which cells are in each phase of the cell cycle.
`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`.

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, is.G1, is.S, is.G2M)

# 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, 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.
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

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.

Author(s)

Aaron Lun

See Also

runApp

Examples

```
example(newSCESet)
pca <- plotPCA(example_sceset)

# Slightly tedious to extract x/y from ggplot objects:
x <- ggplot2::ggplot_build(pca)$data[[1]][,1]
```

```
y <- ggplot2::ggplot_build(pca)$data[[1]][,2]

app <- selectorPlot(x, y)

## Not run:
saved <- shiny::runApp(app)

## 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 `?isSpike<-` 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

See Also

SCESet

Examples

```
# Setting up an example.
set.seed(100)
popsize <- 200
ngenes <- 1000
all.facs <- 2^rnorm(popsize, sd=0.5)
counts <- matrix(rnbinom(ngenes*popsize, mu=all.facs*10, size=1), ncol=popsize, byrow=TRUE)
spikes <- matrix(rnbinom(100*popsize, mu=all.facs*10, size=0.5), ncol=popsize, byrow=TRUE)

combined <- rbind(counts, spikes)
colnames(combined) <- seq_len(popsize)
rownames(combined) <- seq_len(nrow(combined))
y <- newSCESet(countData=combined)
y <- calculateQCMetrics(y, list(IAmASpike=rep(c(FALSE, TRUE), c(ngenes, 100))))
isSpike(y) <- "IAmASpike"

#
y <- computeSpikeFactors(y)
sizeFactors(y)
sizeFactors(y, type="IAmASpike")

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

---

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, only the named spike-in sets 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.

If is.spike or spike.type are NA, 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

```
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

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))
isSpike(X) <- "Spikes"

# Dummying up some size factors.
```

```
sizeFactors(X) <- 1
sizeFactors(X, type="Spikes") <- 1

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)
```

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.

Details

The null hypothesis states that the true variance for each gene is equal to `null`. (Technically, it states that the variance is equal to or less than this value, but the most conservative test is obtained at equality.) Variance estimates should follow a scaled chi-squared distribution on `df` degrees of freedom, where the scaling factor is equal to `null/df`. This can be used to compute a p-value for `total` being greater than `null`. The assumption is that the original observations were normally distributed – using log-CPMs tends to work reasonably well for count data.

The idea 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. Ranking HVGs on p-values is better than ranking on `total - null`, as the latter is less precise when `null` is large.

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

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=list(),
    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 polynomial or loess-based.

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

start
: A list containing starting values for nls.

design
: A numeric matrix describing the systematic 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 log-CPMs 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. By default, the start values are a=5, n=5 and b=1, which can be altered as named arguments in 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.

Author(s)

Aaron Lun

See Also

nls, loess, decomposeVar, computeSpikeFactors, computeSumFactors, normalize

Examples

```
set.seed(100)

nspikes <- ncells <- 200
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 <- 10000
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))))
isSpike(y) <- "Spike"

# Normalizing.
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)
x <- sort(fit$mean)
lines(x, fit$trend(x), col="red", lwd=2)

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

Index

∗Topic **clustering**
  cyclone, 7
  sandbag, 18
∗Topic **correlation**
  correlatePairs, 4
∗Topic **normalization**
  Deconvolution Methods, 11
  Quick clustering, 16
  Spike-in normalization, 21
∗Topic **variance**
  decomposeVar, 9
  Distance-to-median, 13
  technicalCV2, 22
  testVar, 25
  trendVar, 26
∗Topic
  correlatePairs, 4

bpparam, *6*

calculateQCMetrics, *15*
column, *20*
computeSpikeFactors, *13*, *27, 28*
computeSpikeFactors *(*Spike-in normalization*)*, 21
computeSpikeFactors,SCESet-method *(*Spike-in normalization*)*, 21
computeSumFactors, *17*, *28*
computeSumFactors *(*Deconvolution Methods*)*, 11
computeSumFactors,matrix-method *(*Deconvolution Methods*)*, 11
computeSumFactors,SCESet-method *(*Deconvolution Methods*)*, 11
convertTo, 2
convertTo,SCESet-method *(*convertTo*)*, 2
cor, *6*
correlateNull *(*correlatePairs*)*, 4
correlatePairs, 4
correlatePairs,matrix-method *(*correlatePairs*)*, 4
correlatePairs,SCESet-method *(*correlatePairs*)*, 4
cutreeDynamic, *16, 17*

cyclone, 7, *18, 19*
cyclone,matrix-method *(*cyclone*)*, 7
cyclone,SCESet-method *(*cyclone*)*, 7

decomposeVar, 9, *26–28*
decomposeVar,matrix,list-method *(*decomposeVar*)*, 9
decomposeVar,SCESet,list-method *(*decomposeVar*)*, 9
Deconvolution Methods, 11
DESeqDataSetFromMatrix, *3*
DGEList, *3*
Distance-to-median, 13
DM *(*Distance-to-median*)*, 13

estimateSizeFactorsForMatrix, *23, 24*

Get spikes, 14
glmgam.fit, *23, 24*

isSpike *(*Get spikes*)*, 14
isSpike,SCESet-method *(*Get spikes*)*, 14
isSpike<- *(*Get spikes*)*, 14
isSpike<-,SCESet,character-method *(*Get spikes*)*, 14
isSpike<-,SCESet,NULL-method *(*Get spikes*)*, 14

loess, *27, 28*

newCellDataSet, *3*
nls, *27, 28*
normalize, *15*, *27, 28*

Quick clustering, 16
quickCluster, *12, 13*
quickCluster *(*Quick clustering*)*, 16
quickCluster,matrix-method *(*Quick clustering*)*, 16
quickCluster,SCESet-method *(*Quick clustering*)*, 16

runApp, *20*

sandbag, *7, 8*, 18

sandbag,matrix-method (sandbag), 18
sandbag,SCESet-method (sandbag), 18
SCESet, *15*, *22*
Selector plot, 19
selectorPlot (Selector plot), 19
sizeFactors, *23*
Spike-in normalization, 21
spikes (Get spikes), 14
spikes,SCESet-method (Get spikes), 14

technicalCV2, 22
technicalCV2,matrix-method
        (technicalCV2), 22
technicalCV2,SCESet-method
        (technicalCV2), 22
testVar, *10*, 25
trendVar, *9*, *10*, *26*, 26
trendVar,matrix-method (trendVar), 26
trendVar,SCESet-method (trendVar), 26