Package 'scran'

October 12, 2016

Version 1.0.4

Date 2016-08-10

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, limma, stats, BiocGenerics, methods, Biobase, utils, Matrix

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

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

Description This package 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` *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, normalize=TRUE, 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.
`normalize`	A logical scalar specifying whether the assay values should be normalized for `type="monocle"`.
`get.spikes`	A logical scalar specifying whether rows corresponding to spike-in transcripts should be returned.

Details

This function converts a SCESet 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 analysis pipeline (normalization information is stored through size factors). For monocle, counts are divided by the size factors to

yield (roughly) log-normally distributed expression values. This can be turned off (i.e., to use the raw values in assay) 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. This behaviour can be turned off by setting get.spikes=TRUE, such that all rows are retrieved in the output 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 <- 100
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))
is.spike <- rbinom(ngenes, 1, 0.5)==0L
isSpike(X) <- is.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)

## S4 method for signature 'matrix'
correlatePairs(x, null.dist=NULL, design=NULL, BPPARAM=bpparam(),
    use.names=TRUE, tol=1e-8)

## S4 method for signature 'SCESet'
correlatePairs(x, ..., 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.
`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.
`...`	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. 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.

Value

For `correlateNull`, a numeric vector of length `iters` is returned containing the sorted correlations under the null hypothesis of no correlations.

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`, integer row indices are returned, 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`.

Gene selection and experimental design

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.

If the experiment has known (and uninteresting) factors of variation, e.g., batch effects, cell cycle phase, these can be included in the `design`. Correlations between genes will then be computed using the residual effects of a linear model fitted to the normalized expression values with `design`. Similarly, the null distribution of rho values will be constructed with `ncells` set as the residual degrees of freedom in `design`. This procedure ensures that the uninteresting factors do not drive strong correlations between genes.

Note that tied counts may not lead to tied effects, especially for design matrices with real-valued covariates. As a result, large correlations may be obtained for genes with few non-zero counts.

Some protection is provided by focusing on HVGs, because 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.

For example, counts of zero will have the same normalized log-expression, even if the library sizes are different. This is because the added prior count is scaled by the library size in `cpm.default`, such that the effect of library size cancels out. Thus, all zeroes will have tied ranks (with numerical imprecision handled by `tol`) and will not inflate the correlations. For non-zero counts, correlations may be driven by library size differences between cells. This is, perhaps, less problematic, as two genes with the same counts in both small and large libraries provide evidence for association.

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=bpparam(), verbose=FALSE)
## S4 method for signature 'SCESet'
cyclone(x, ..., 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.
...	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). Pairs of marker genes are trained with sandbag, where the sign of the relative expression between gene in each pair changes across phases. For each phase and each cell, the function calculates the proportion of all marker pairs where the expression of the first gene is greater than the second (pairs with the same expression are ignored). 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.

Cells with G1 and G2M scores above 0.5 should be assigned to the G1 and G2M phases, respectively. 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.

Value

A list of two data frames is returned – `scores`, containing the phase scores for each phase and cell (i.e., each row is a cell); and `normalized.scores`, 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)
## S4 method for signature 'SCESet,list'
decomposeVar(x, fit, ..., 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.
...	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.

Value

A data frame is returned, containing:

`mean`: A numeric vector of mean log-CPMs for all cellular genes.

`total`: A numeric vector of the variances of log-CPMs for all cellular genes.

`bio`: A numeric vector containing the biological component of the variance for all genes.

`tech`: A numeric vector containing the technical component of the variance for all genes.

Rows corresponding to spike-in transcripts are set to `NA` unless `get.spikes=TRUE`.

Author(s)

Aaron Lun

See Also

`trendVar`

Examples

```
set.seed(100)

nspikes <- ncells <- 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 <- 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)
isSpike(y) <- rep(c(FALSE, TRUE), c(ngenes, nspikes))

# Normalizing.
y <- computeSpikeFactors(y) # or computeSumFactors
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)
## S4 method for signature 'SCESet'
computeSumFactors(x, ..., 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.
`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.
`...`	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.

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 `errors=TRUE`, the standard error of the size factor estimates is 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. The standard error of the estimates can be obtained by setting `errors=TRUE`.

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 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.

Author(s)

Aaron Lun and Karsten Bach

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'
spikes(x, assay="counts")
## S4 method for signature 'SCESet'
isSpike(x)
## S4 replacement method for signature 'SCESet'
isSpike(x) <- value
```

Arguments

x	A SCESet object with spike-in data in the colData.
assay	A string specifying whether counts or normalized expression values are to be extracted.
value	A logical vector specifying which rows correspond to spike-ins.

Details

This function extracts the spike-in data from x, into a numeric matrix that can be used for downstream analyses. Users should set a logical vector in isSpike(x) indicating whether each row of x is a spike-in control. If assay="exprs", users should have run x through normalize.

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`.

Author(s)

Aaron Lun

See Also

`normalize`, `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)
isSpike(y) <- rep(c(FALSE, TRUE), c(ngenes, 100))

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

`Normalized Expression` *Compute normalized expression values*

Description

Compute (log-)expression values from read counts, using pre-calculated size factors for normalization.

Usage

```
## S4 method for signature 'SCESet'
normalize(object, log=TRUE, prior.count=1, separate.spikes=TRUE)
```

Arguments

object	A matrix of read counts, or a SCESet object with an assay named "counts".
log	A logical scalar specifying whether the expression should be log-transformed.
prior.count	A numeric scalar indicating the prior count to add prior to log-transformation, to avoid undefined values from zero counts.
separate.spikes	A logical scalar indicating whether spike-in counts should be normalized separately.

Details

This function computes normalized log-expression values by adding prior.count to each count, dividing by the size.factor for that cell, and log-transforming. Size factors are taken from the appropriate field in the colData of object. These size factors can be computed with a number of functions like computeSumFactors or computeSpikeFactors.

If spike-in counts are present in the SCESet object, these will also be converted into normalized values. If separate.spikes=FALSE, this is done with the same set of size factors that was used for the endogenous genes. Otherwise, a separate set of spike-in size factors will be used instead – these are defined by calling computeSpikeFactors.

All size factors are mean-centered so that their geometric mean is equal to unity prior to computing normalized expression values. This ensures that expression values are roughly comparable when different sets of size factors are used.

Value

For normalize,SCESet-method, a SCESet object is returned with an additional assay named "exprs". This contains normalized log-expression values for the endogenous genes. If spike-ins are present, normalized values for the spike-in transcripts are stored in the norm.spikes field of the colData.

Why spikes can be normalized separately

In most cases, it does not make sense to normalize spike-in counts with size factors computed from endogenous genes. This is because the spike-in counts do not (generally) depend on the total amount of endogenous RNA, whereas the size factors do. As such, normalizing the former with the latter would be inappropriate – cells with a lot of endogenous RNA would scale down the spike-in counts, even if the same amount of spike-in RNA was added, captured and sequenced in each cell.

Instead, normalization of the spike-in counts should be performed using size factors computed from those counts, i.e., with computeSpikeFactors. This is the default setting when separate.spikes=TRUE. Normalized log-expression values are made to be roughly comparable to those of endogenous genes, by ensuring *all* sets of size factors are mean-centered prior to normalization.

Author(s)

Aaron Lun

See Also

cpm, SCESet, computeSumFactors, computeSpikeFactors

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)
isSpike(y) <- rep(c(FALSE, TRUE), c(ngenes, 100))

sizeFactors(y) <- colSums(combined) # Library size normalization, basically.
y <- normalize(y)
exprs(y)[1:10,]

y <- computeSpikeFactors(y)
y <- normalize(y)
exprs(y)[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, ...)
## S4 method for signature 'SCESet'
quickCluster(x, ..., 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 `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. 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.

By default, spike-in transcripts are not used 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.

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

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)
## S4 method for signature 'SCESet'
sandbag(x, ..., 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.
...	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 (see Examples). 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, such that the relative expression of genes to spike-ins will not be a reliable predictor. Nonetheless, if all rows are required, users can set get.spikes=TRUE.

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

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"))
```

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)
```

Arguments

x	A SCESet object containing rows corresponding spike-in transcripts.

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. Spike-in counts are assumed to be stored in rows with `fData(x)$is_control_feature` – see `?isSpike<-` for more details. Note that the geometric mean of all size factors is set to unity, for standardization purposes if one were to compare different sets of size factors.

Value

An object of class x is returned, containing a numeric vector of size factors for all cells in `sizeFactors(x)`.

Author(s)

Aaron Lun

See Also

SCESet

Examples

```
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)
isSpike(y) <- rep(c(FALSE, TRUE), c(ngenes, 100))
out.facs <- computeSpikeFactors(y)
```

---

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`. Variance estimates should follow a scaled chi-squared distribution on `df` degrees of freedom, where the scaling factor is equal to `null` divided by `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. 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.

Note that the test statistic is the (scaled) ratio of `total` over `null` for each gene. This may not be ideal when `null` is small, e.g., for high-abundance genes, where a high ratio/low p-value may not represent a large absolute increase in the variance. To avoid detecting irrelevant genes, users can modify `null` by adding a minimum threshold of, say, 0.5. This tests for variances that are significantly greater than `null+0.5` and rules out genes that have high ratios but small absolute increases.

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` *Get the biological variability*

Description

Compute the biological and technical components of the gene-specific variance in single-cell RNA-seq data.

Usage

```
## S4 method for signature 'matrix'
trendVar(x, trend=c("poly", "loess"), df=5, span=0.3, design=NULL)
## S4 method for signature 'SCESet'
trendVar(x, ..., 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.
`df`	An integer scalar specifying the degrees of freedom for polynomial fitting.
`span`	An numeric scalar specifying the span for loess fitting.
`design`	A numeric matrix describing the systematic factors contributing to expression in each cell.
`...`	Additional arguments to pass to `trendVar,matrix,list-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 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 genes, 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 tend to be more robust to genes with strong expression in only one or two outlier cells.

The mean and variance of the log-CPMs is calculated for each spike-in gene. A polynomial or loess trend is then fitted to the variance against the mean for all genes. This 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

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 genes.

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

`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.

Additional notes

By default, a polynomial trend with `df` degrees of freedom is fitted to the spike-in variances as it is more precise than the loess curve. Note that this method is rather dependent on the quality of the spike-ins – the fit will obviously be poor if the coverage of all spike-ins is low. In some data sets, a loess curve may yield better results, though this may require some fiddling with the `span`.

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. A loess curve is recommended here as it is more robust to a subset of genes with very large or very small variances. 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.

Author(s)

Aaron Lun

See Also

`poly`, `loess`, `decomposeVar`, `computeSpikeFactors`, `computeSumFactors`, `normalize`

Examples

```
set.seed(100)

nspikes <- ncells <- 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 <- 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)
isSpike(y) <- rep(c(FALSE, TRUE), c(ngenes, nspikes))

# Normalizing.
y <- computeSpikeFactors(y) # or computeSumFactors.
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