Package 'DESeq2'

December 5, 2015

Type Package

Title Differential gene expression analysis based on the negative binomial distribution

Version 1.10.0

Author Michael Love (HSPH Boston), Simon Anders, Wolfgang Huber (EMBL Heidelberg)

Maintainer Michael Love <michaelisaiahlove@gmail.com>

Description Estimate variance-mean dependence in count data from high-throughput sequencing assays and test for differential expression based on a model using the negative binomial distribution.

License LGPL (>= 3)

VignetteBuilder knitr

Imports BiocGenerics (>= 0.7.5), Biobase, BiocParallel, genefilter, methods, locfit, geneplotter, ggplot2, Hmisc

Depends S4Vectors, IRanges, GenomicRanges, SummarizedExperiment (>= 0.2.0), Rcpp (>= 0.10.1), RcppArmadillo (>= 0.3.4.4)

Suggests testthat, knitr, BiocStyle, vsn, pheatmap, RColorBrewer, airway, pasilla (>= 0.2.10), DESeq

LinkingTo Rcpp, RcppArmadillo

biocViews Sequencing, ChIPSeq, RNASeq, SAGE, DifferentialExpression, GeneExpression

NeedsCompilation yes

R topics documented:

`DESeq2-package` *DESeq2 package for differential analysis of count data*

Description

The main functions for differential analysis are `DESeq` and `results`. See the examples at `DESeq` for basic analysis steps. Two transformations offered for count data are the "regularized logarithm", `rlog`, and `varianceStabilizingTransformation`. For more detailed information on usage, see the package vignette, by typing `vignette("DESeq2")`, or the workflow linked to on the first page of the vignette. All support questions should be posted to the Bioconductor support site: `support.bioconductor.org`.

Author(s)

Michael Love, Wolfgang Huber, Simon Anders

References

DESeq2 reference:

Michael I Love, Wolfgang Huber, Simon Anders: Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology 2014, 15:550. http://dx.doi.org/10.1186/s13059-014-0550-8

DESeq reference:

Simon Anders, Wolfgang Huber: Differential expression analysis for sequence count data. Genome Biology 2010, 11:106. http://dx.doi.org/10.1186/gb-2010-11-10-r106

---

coef *Extract a matrix of model coefficients/standard errors*

Description

Note: results tables with log2 fold change, p-values, adjusted p-values, etc. for each gene are best generated using the `results` function. The `coef` function is designed for advanced users who wish to inspect all model coefficients at once.

Usage

```
## S3 method for class 'DESeqDataSet'
coef(object, SE = FALSE, ...)
```

Arguments

`object`	a DESeqDataSet returned by `DESeq`, `nbinomWaldTest`, or `nbinomLRT`.
`SE`	whether to give the standard errors instead of coefficients. defaults to FALSE so that the coefficients are given.
`...`	additional arguments

Details

Estimated model coefficients or estimated standard errors are provided in a matrix form, number of genes by number of parameters, on the log2 scale. The columns correspond to columns of the model matrix for final GLM fitting, i.e., `attr(dds, "modelMatrix")`.

Author(s)

Michael Love

Examples

```
dds <- makeExampleDESeqDataSet(m=4)
dds <- DESeq(dds)
coef(dds)[1,]
coef(dds, SE=TRUE)[1,]
```

---

collapseReplicates *Collapse technical replicates in a RangedSummarizedExperiment or DESeqDataSet*

Description

Collapses the columns in `object` by summing within levels of a grouping factor `groupby`. The purpose of this function is to sum up read counts from technical replicates to create an object with a single column of read counts for each sample. Optionally renames the columns of returned object with the levels of the grouping factor. Note: this function is written very simply and can be easily altered to produce other behavior by examining the source code.

Usage

```
collapseReplicates(object, groupby, run, renameCols = TRUE)
```

Arguments

`object`	A `RangedSummarizedExperiment` or `DESeqDataSet`
`groupby`	a grouping factor, as long as the columns of object
`run`	optional, the names of each unique column in object. if provided, a new column `runsCollapsed` will be added to the `colData` which pastes together the names of `run`
`renameCols`	whether to rename the columns of the returned object using the levels of the grouping factor

Value

the `object` with as many columns as levels in `groupby`. This object has assay/count data which is summed from the various columns which are grouped together, and the `colData` is subset using the first column for each group in `groupby`.

Examples

```
dds <- makeExampleDESeqDataSet(m=12)

# make data with two technical replicates for three samples
dds$sample <- factor(sample(paste0("sample",rep(1:9, c(2,1,1,2,1,1,2,1,1)))))
dds$run <- paste0("run",1:12)

ddsColl <- collapseReplicates(dds, dds$sample, dds$run)
```

```
# examine the colData and column names of the collapsed data
colData(ddsColl)
colnames(ddsColl)

# check that the sum of the counts for "sample1" is the same
# as the counts in the "sample1" column in ddsColl
matchFirstLevel <- dds$sample == levels(dds$sample)[1]
stopifnot(all(rowSums(counts(dds[,matchFirstLevel])) == counts(ddsColl[,1])))
```

---

counts *Accessors for the 'counts' slot of a DESeqDataSet object.*

Description

The counts slot holds the count data as a matrix of non-negative integer count values, one row for each observational unit (gene or the like), and one column for each sample.

Usage

```
## S4 method for signature 'DESeqDataSet'
counts(object, normalized = FALSE,
  replaced = FALSE)

## S4 replacement method for signature 'DESeqDataSet,matrix'
counts(object) <- value
```

Arguments

object	a DESeqDataSet object.
normalized	logical indicating whether or not to divide the counts by the size factors or normalization factors before returning (normalization factors always preempt size factors)
replaced	after a DESeq call, this argument will return the counts with outliers replaced instead of the original counts, and optionally normalized. The replaced counts are stored by DESeq in assays(object)[['replaceCounts']].
value	an integer matrix

Author(s)

Simon Anders

See Also

sizeFactors, normalizationFactors

Examples

```
dds <- makeExampleDESeqDataSet(m=4)
head(counts(dds))

dds <- estimateSizeFactors(dds) # run this or DESeq() first
head(counts(dds, normalized=TRUE))
```

DESeq *Differential expression analysis based on the Negative Binomial (a.k.a. Gamma-Poisson) distribution*

Description

This function performs a default analysis through the steps:

1. estimation of size factors: estimateSizeFactors
2. estimation of dispersion: estimateDispersions
3. Negative Binomial GLM fitting and Wald statistics: nbinomWaldTest

For complete details on each step, see the manual pages of the respective functions. After the DESeq function returns a DESeqDataSet object, results tables (log2 fold changes and p-values) can be generated using the results function. See the manual page for results for information on independent filtering and p-value adjustment for multiple test correction.

Usage

```
DESeq(object, test = c("Wald", "LRT"), fitType = c("parametric", "local",
  "mean"), betaPrior, full = design(object), reduced, quiet = FALSE,
  minReplicatesForReplace = 7, modelMatrixType, parallel = FALSE,
  BPPARAM = bpparam())
```

Arguments

object	a DESeqDataSet object, see the constructor functions DESeqDataSet, DESeqDataSetFromMatrix, DESeqDataSetFromHTSeqCount.
test	either "Wald" or "LRT", which will then use either Wald significance tests (defined by nbinomWaldTest), or the likelihood ratio test on the difference in deviance between a full and reduced model formula (defined by nbinomLRT)
fitType	either "parametric", "local", or "mean" for the type of fitting of dispersions to the mean intensity. See estimateDispersions for description.
betaPrior	whether or not to put a zero-mean normal prior on the non-intercept coefficients See nbinomWaldTest for description of the calculation of the beta prior. By default, the beta prior is used only for the Wald test, but can also be specified for the likelihood ratio test.

`full` for `test="LRT"`, the full model formula, which is restricted to the formula in `design(object)`. alternatively, it can be a model matrix constructed by the user. advanced use: specifying a model matrix for full and `test="Wald"` is possible if `betaPrior=FALSE`

`reduced` for `test="LRT"`, a reduced formula to compare against, i.e., the full formula with the term(s) of interest removed. alternatively, it can be a model matrix constructed by the user

`quiet` whether to print messages at each step

`minReplicatesForReplace`
the minimum number of replicates required in order to use `replaceOutliers` on a sample. If there are samples with so many replicates, the model will be refit after these replacing outliers, flagged by Cook's distance. Set to `Inf` in order to never replace outliers.

`modelMatrixType`
either "standard" or "expanded", which describe how the model matrix, X of the GLM formula is formed. "standard" is as created by `model.matrix` using the design formula. "expanded" includes an indicator variable for each level of factors in addition to an intercept. for more information see the Description of `nbinomWaldTest`. betaPrior must be set to TRUE in order for expanded model matrices to be fit.

`parallel` if FALSE, no parallelization. if TRUE, parallel execution using `BiocParallel`, see next argument `BPPARAM`. A note on running in parallel using `BiocParallel`: it may be advantageous to remove large, unneeded objects from your current R environment before calling `DESeq`, as it is possible that R's internal garbage collection will copy these files while running on worker nodes.

`BPPARAM` an optional parameter object passed internally to `bplapply` when `parallel=TRUE`. If not specified, the parameters last registered with `register` will be used.

Details

The differential expression analysis uses a generalized linear model of the form:

$$K_{ij} \sim \mathrm{NB}(\mu_{ij}, \alpha_i)$$

$$\mu_{ij} = s_j q_{ij}$$

$$\log_2(q_{ij}) = x_{j\cdot}\beta_i$$

where counts K_{ij} for gene i, sample j are modeled using a Negative Binomial distribution with fitted mean μ_{ij} and a gene-specific dispersion parameter α_i. The fitted mean is composed of a sample-specific size factor s_j and a parameter q_{ij} proportional to the expected true concentration of fragments for sample j. The coefficients β_i give the log2 fold changes for gene i for each column of the model matrix X. The sample-specific size factors can be replaced by gene-specific normalization factors for each sample using `normalizationFactors`.

For details on the fitting of the log2 fold changes and calculation of p-values, see `nbinomWaldTest` if using `test="Wald"`, or `nbinomLRT` if using `test="LRT"`.

Experiments without replicates do not allow for estimation of the dispersion of counts around the expected value for each group, which is critical for differential expression analysis. If an experimental design is supplied which does not contain the necessary degrees of freedom for differential analysis, DESeq will provide a message to the user and follow the strategy outlined in Anders and Huber (2010) under the section 'Working without replicates', wherein all the samples are considered as replicates of a single group for the estimation of dispersion. As noted in the reference above: "Some overestimation of the variance may be expected, which will make that approach conservative." Furthermore, "while one may not want to draw strong conclusions from such an analysis, it may still be useful for exploration and hypothesis generation."

The argument minReplicatesForReplace is used to decide which samples are eligible for automatic replacement in the case of extreme Cook's distance. By default, DESeq will replace outliers if the Cook's distance is large for a sample which has 7 or more replicates (including itself). This replacement is performed by the replaceOutliers function. This default behavior helps to prevent filtering genes based on Cook's distance when there are many degrees of freedom. See results for more information about filtering using Cook's distance, and the 'Dealing with outliers' section of the vignette. Unlike the behavior of replaceOutliers, here original counts are kept in the matrix returned by counts, original Cook's distances are kept in assays(dds)[["cooks"]], and the replacement counts used for fitting are kept in assays(object)[["replaceCounts"]].

Note that if a log2 fold change prior is used (betaPrior=TRUE) then expanded model matrices will be used in fitting. These are described in nbinomWaldTest and in the vignette. The contrast argument of results should be used for generating results tables.

Value

a DESeqDataSet object with results stored as metadata columns. These results should accessed by calling the results function. By default this will return the log2 fold changes and p-values for the last variable in the design formula. See results for how to access results for other variables.

Author(s)

Michael Love

References

Michael I Love, Wolfgang Huber, Simon Anders: Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology 2014, 15:550. http://dx.doi.org/10.1186/s13059-014-0550-8

See Also

nbinomWaldTest, nbinomLRT

Examples

```
# see vignette for suggestions on generating
# count tables from RNA-Seq data
cnts <- matrix(rnbinom(n=1000, mu=100, size=1/0.5), ncol=10)
cond <- factor(rep(1:2, each=5))
```

```
# object construction
dds <- DESeqDataSetFromMatrix(cnts, DataFrame(cond), ~ cond)

# standard analysis
dds <- DESeq(dds)
res <- results(dds)

# an alternate analysis: likelihood ratio test
ddsLRT <- DESeq(dds, test="LRT", reduced= ~ 1)
resLRT <- results(ddsLRT)
```

DESeqDataSet-class *DESeqDataSet object and constructors*

Description

DESeqDataSet is a subclass of RangedSummarizedExperiment, used to store the input values, intermediate calculations and results of an analysis of differential expression. The DESeqDataSet class enforces non-negative integer values in the "counts" matrix stored as the first element in the assay list. In addition, a formula which specifies the design of the experiment must be provided. The constructor functions create a DESeqDataSet object from various types of input: a RangedSummarizedExperiment, a matrix, or count files generated by the python package HTSeq. See the vignette for examples of construction from all three input types.

Usage

```
DESeqDataSet(se, design, ignoreRank = FALSE)

DESeqDataSetFromMatrix(countData, colData, design, tidy = FALSE,
  ignoreRank = FALSE, ...)

DESeqDataSetFromHTSeqCount(sampleTable, directory = ".", design,
  ignoreRank = FALSE, ...)
```

Arguments

se	a RangedSummarizedExperiment with columns of variables indicating sample information in colData, and the counts as the first element in the assays list, which will be renamed "counts". A RangedSummarizedExperiment object can be generated by the function summarizeOverlaps in the GenomicAlignments package.
design	a formula which expresses how the counts for each gene depend on the variables in colData. Many R formula are valid, including designs with multiple variables, e.g., ~ group + condition, and designs with interactions, e.g., ~ genotype + treatment + genotype:treatment. See results for a variety of designs and how to extract results tables. By default, the functions in this package will use the last variable in the formula for building results tables and plotting. ~ 1 can be used for no design, although users need to remember to switch to another design for differential testing.

ignoreRank	use of this argument is reserved for DEXSeq developers only. Users will immediately encounter an error upon trying to estimate dispersion using a design with a model matrix which is not full rank.
countData	for matrix input: a matrix of non-negative integers
colData	for matrix input: a DataFrame or data.frame with at least a single column. Rows of colData correspond to columns of countData
tidy	for matrix input: whether the first column of countData is the rownames for the count matrix
...	arguments provided to SummarizedExperiment including rowRanges and metadata. Note that for Bioconductor 3.1, rowRanges must be a GRanges or GRangesList, with potential metadata columns as a DataFrame accessed and stored with mcols. If a user wants to store metadata columns about the rows of the countData, but does not have GRanges or GRangesList information, first construct the DESeqDataSet without rowRanges and then add the DataFrame with mcols(dds).
sampleTable	for htseq-count: a data.frame with three or more columns. Each row describes one sample. The first column is the sample name, the second column the file name of the count file generated by htseq-count, and the remaining columns are sample metadata which will be stored in colData
directory	for htseq-count: the directory relative to which the filenames are specified. defaults to current directory

Value

A DESeqDataSet object.

References

See http://www-huber.embl.de/users/anders/HTSeq for htseq-count

Examples

```
countData <- matrix(1:100,ncol=4)
condition <- factor(c("A","A","B","B"))
dds <- DESeqDataSetFromMatrix(countData, DataFrame(condition), ~ condition)
```

DESeqResults-class *DESeqResults object and constructor*

Description

This constructor function would not typically be used by "end users". This simple class extends the DataFrame class of the IRanges package to allow other packages to write methods for results objects from the DESeq2 package. It is used by results to wrap up the results table.

Usage

```
DESeqResults(DataFrame)
```

Arguments

DataFrame	a DataFrame of results, standard column names are: baseMean, log2FoldChange, lfcSE, stat, pvalue, padj.

Value

a DESeqResults object

DESeqTransform-class *DESeqTransform object and constructor*

Description

This constructor function would not typically be used by "end users". This simple class extends the RangedSummarizedExperiment class of the SummarizedExperiment package. It is used by rlog and varianceStabilizingTransformation to wrap up the results into a class for downstream methods, such as plotPCA.

Usage

```
DESeqTransform(SummarizedExperiment)
```

Arguments

SummarizedExperiment	a RangedSummarizedExperiment

Value

a DESeqTransform object

---

design *Accessors for the 'design' slot of a DESeqDataSet object.*

Description

The design holds the R formula which expresses how the counts depend on the variables in colData. See DESeqDataSet for details.

Usage

```
## S4 method for signature 'DESeqDataSet'
design(object)

## S4 replacement method for signature 'DESeqDataSet,formula'
design(object) <- value
```

Arguments

object	a DESeqDataSet object
value	a formula used for estimating dispersion and fitting Negative Binomial GLMs

Examples

```
dds <- makeExampleDESeqDataSet(m=4)
design(dds) <- formula(~ 1)
```

---

dispersionFunction *Accessors for the 'dispersionFunction' slot of a DESeqDataSet object.*

Description

The dispersion function is calculated by estimateDispersions and used by varianceStabilizingTransformation. Parametric dispersion fits store the coefficients of the fit as attributes in this slot.

Usage

```
dispersionFunction(object, ...)

dispersionFunction(object, ...) <- value

## S4 method for signature 'DESeqDataSet'
dispersionFunction(object)

## S4 replacement method for signature 'DESeqDataSet,`function`'
dispersionFunction(object,
  estimateVar = TRUE) <- value
```

Arguments

object	a DESeqDataSet object.
...	additional arguments
value	a function
estimateVar	whether to estimate the variance of dispersion residuals. setting to FALSE is needed, e.g. within estimateDispersionsMAP when called on a subset of the full dataset in parallel execution.

Details

Setting this will also overwrite mcols(object)$dispFit and the estimate the variance of dispersion residuals, see estimateVar below.

See Also

estimateDispersions

Examples

```
dds <- makeExampleDESeqDataSet(m=4)
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
dispersionFunction(dds)
```

`dispersions` *Accessor functions for the dispersion estimates in a DESeqDataSet object.*

Description

The dispersions for each row of the DESeqDataSet. Generally, these are set by `estimateDispersions`.

Usage

```
dispersions(object, ...)

dispersions(object, ...) <- value

## S4 method for signature 'DESeqDataSet'
dispersions(object)

## S4 replacement method for signature 'DESeqDataSet,numeric'
dispersions(object) <- value
```

Arguments

`object`	a `DESeqDataSet` object.
`...`	additional arguments
`value`	the dispersions to use for the Negative Binomial modeling

Author(s)

Simon Anders

See Also

`estimateDispersions`

`estimateBetaPriorVar` *Steps for estimating the beta prior variance*

Description

These lower-level functions are called within DESeq or nbinomWaldTest. End users should use those higher-level function instead. NOTE: `estimateBetaPriorVar` returns a numeric vector, not a DESEqDataSet! For advanced users: to use these functions, first run `estimateMLEForBetaPriorVar` and then run `estimateBetaPriorVar`.

Usage

```
estimateBetaPriorVar(object, betaPriorMethod = c("weighted", "quantile"),
  upperQuantile = 0.05)

estimateMLEForBetaPriorVar(object, maxit = 100, useOptim = TRUE,
  useQR = TRUE)
```

Arguments

`object`	a DESeqDataSet
`betaPriorMethod`	the method for calculating the beta prior variance, either "quanitle" or "weighted": "quantile" matches a normal distribution using the upper quantile of the finite MLE betas. "weighted" matches a normal distribution using the upper quantile, but weighting by the variance of the MLE betas.
`upperQuantile`	the upper quantile to be used for the "quantile" or "weighted" method of beta prior variance estimation
`maxit`	as defined in `link{nbinomWaldTest}`
`useOptim`	as defined in `link{nbinomWaldTest}`
`useQR`	as defined in `link{nbinomWaldTest}`

Value

for `estimateMLEForBetaPriorVar`, a DESeqDataSet, with the necessary information stored in order to calculate the prior variance. for `estimateBetaPriorVar`, the vector of variances for the prior on the betas in the DESeq GLM

`estimateDispersions` *Estimate the dispersions for a DESeqDataSet*

Description

This function obtains dispersion estimates for Negative Binomial distributed data.

Usage

```
## S4 method for signature 'DESeqDataSet'
estimateDispersions(object, fitType = c("parametric",
  "local", "mean"), maxit = 100, quiet = FALSE, modelMatrix = NULL)
```

Arguments

`object`	a DESeqDataSet
`fitType`	either "parametric", "local", or "mean" for the type of fitting of dispersions to the mean intensity. • parametric - fit a dispersion-mean relation of the form: $$dispersion = asymptDisp + extraPois/mean$$ via a robust gamma-family GLM. The coefficients `asymptDisp` and `extraPois` are given in the attribute `coefficients` of the `dispersionFunction` of the object. • local - use the locfit package to fit a local regression of log dispersions over log base mean (normal scale means and dispersions are input and output for `dispersionFunction`). The points are weighted by normalized mean count in the local regression. • mean - use the mean of gene-wise dispersion estimates.
`maxit`	control parameter: maximum number of iterations to allow for convergence
`quiet`	whether to print messages at each step
`modelMatrix`	an optional matrix which will be used for fitting the expected counts. by default, the model matrix is constructed from `design(object)`

Details

Typically the function is called with the idiom:

`dds <- estimateDispersions(dds)`

The fitting proceeds as follows: for each gene, an estimate of the dispersion is found which maximizes the Cox Reid-adjusted profile likelihood (the methods of Cox Reid-adjusted profile likelihood maximization for estimation of dispersion in RNA-Seq data were developed by McCarthy, et al. (2012), first implemented in the edgeR package in 2010); a trend line capturing the dispersion-mean relationship is fit to the maximum likelihood estimates; a normal prior is determined for the log dispersion estimates centered on the predicted value from the trended fit with variance

equal to the difference between the observed variance of the log dispersion estimates and the expected sampling variance; finally maximum a posteriori dispersion estimates are returned. This final dispersion parameter is used in subsequent tests. The final dispersion estimates can be accessed from an object using dispersions. The fitted dispersion-mean relationship is also used in varianceStabilizingTransformation. All of the intermediate values (gene-wise dispersion estimates, fitted dispersion estimates from the trended fit, etc.) are stored in mcols(dds), with information about these columns in mcols(mcols(dds)).

The log normal prior on the dispersion parameter has been proposed by Wu, et al. (2012) and is also implemented in the DSS package.

In DESeq2, the dispersion estimation procedure described above replaces the different methods of dispersion from the previous version of the DESeq package.

estimateDispersions checks for the case of an analysis with as many samples as the number of coefficients to fit, and will temporarily substitute a design formula ~ 1 for the purposes of dispersion estimation. This treats the samples as replicates for the purpose of dispersion estimation. As mentioned in the DESeq paper: "While one may not want to draw strong conclusions from such an analysis, it may still be useful for exploration and hypothesis generation."

The lower-level functions called by estimateDispersions are: estimateDispersionsGeneEst, estimateDispersionsFit, and estimateDispersionsMAP.

Value

The DESeqDataSet passed as parameters, with the dispersion information filled in as metadata columns, accessible via mcols, or the final dispersions accessible via dispersions.

References

- Simon Anders, Wolfgang Huber: Differential expression analysis for sequence count data. Genome Biology 11 (2010) R106, http://dx.doi.org/10.1186/gb-2010-11-10-r106
- McCarthy, DJ, Chen, Y, Smyth, GK: Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. Nucleic Acids Research 40 (2012), 4288-4297, http://dx.doi.org/10.1093/nar/gks042
- Wu, H., Wang, C. & Wu, Z. A new shrinkage estimator for dispersion improves differential expression detection in RNA-seq data. Biostatistics (2012). http://dx.doi.org/10.1093/biostatistics/kxs033

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
head(dispersions(dds))
```

estimateDispersionsGeneEst *Low-level functions to fit dispersion estimates*

Description

Normal users should instead use `estimateDispersions`. These low-level functions are called by `estimateDispersions`, but are exported and documented for non-standard usage. For instance, it is possible to replace fitted values with a custom fit and continue with the maximum a posteriori dispersion estimation, as demonstrated in the examples below.

Usage

```
estimateDispersionsGeneEst(object, minDisp = 1e-08, kappa_0 = 1,
  dispTol = 1e-06, maxit = 100, quiet = FALSE, modelMatrix = NULL,
  niter = 1)

estimateDispersionsFit(object, fitType = c("parametric", "local", "mean"),
  minDisp = 1e-08, quiet = FALSE)

estimateDispersionsMAP(object, outlierSD = 2, dispPriorVar, minDisp = 1e-08,
  kappa_0 = 1, dispTol = 1e-06, maxit = 100, modelMatrix = NULL,
  quiet = FALSE)

estimateDispersionsPriorVar(object, minDisp = 1e-08, modelMatrix = NULL)
```

Arguments

`object`	a DESeqDataSet
`minDisp`	small value for the minimum dispersion, to allow for calculations in log scale, one order of magnitude above this value is used as a test for inclusion in mean-dispersion fitting
`kappa_0`	control parameter used in setting the initial proposal in backtracking search, higher kappa_0 results in larger steps
`dispTol`	control parameter to test for convergence of log dispersion, stop when increase in log posterior is less than dispTol
`maxit`	control parameter: maximum number of iterations to allow for convergence
`quiet`	whether to print messages at each step
`modelMatrix`	for advanced use only, a substitute model matrix for gene-wise and MAP dispersion estimation
`niter`	number of times to iterate between estimation of means and estimation of dispersion
`fitType`	either "parametric", "local", or "mean" for the type of fitting of dispersions to the mean intensity. See `estimateDispersions` for description.

outlierSD	the number of standard deviations of log gene-wise estimates above the prior mean (fitted value), above which dispersion estimates will be labelled outliers. Outliers will keep their original value and not be shrunk using the prior.
dispPriorVar	the variance of the normal prior on the log dispersions. If not supplied, this is calculated as the difference between the mean squared residuals of gene-wise estimates to the fitted dispersion and the expected sampling variance of the log dispersion

Value

a DESeqDataSet with gene-wise, fitted, or final MAP dispersion estimates in the metadata columns of the object.

`estimateDispersionsPriorVar` is called inside of `estimateDispersionsMAP` and stores the dispersion prior variance as an attribute of `dispersionFunction(dds)`, which can be manually provided to `estimateDispersionsMAP` for parallel execution.

See Also

`estimateDispersions`

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
dds <- estimateDispersionsGeneEst(dds)
dds <- estimateDispersionsFit(dds)
dds <- estimateDispersionsMAP(dds)
plotDispEsts(dds)

# after having run estimateDispersionsFit()
# the dispersion prior variance over all genes
# can be obtained like so:

dispPriorVar <- estimateDispersionsPriorVar(dds)
```

---

estimateSizeFactors *Estimate the size factors for a DESeqDataSet*

Description

This function estimates the size factors using the "median ratio method" described by Equation 5 in Anders and Huber (2010). The estimated size factors can be accessed using `sizeFactors`. Alternative library size estimators can also be supplied using `sizeFactors`.

Usage

```
## S4 method for signature 'DESeqDataSet'
estimateSizeFactors(object, type = c("ratio",
  "iterate"), locfunc = stats::median, geoMeans, controlGenes, normMatrix)
```

Arguments

object	a DESeqDataSet
type	either "ratio" or "iterate". "ratio" uses the standard median ratio method introduced in DESeq. The size factor is the median ratio of the sample over a pseudosample: for each gene, the geometric mean of all samples. "iterate" offers an alternative estimator, which can be used even when all genes contain a sample with a zero. This estimator iterates between estimating the dispersion with a design of ~1, and finding a size factor vector by numerically optimizing the likelihood of the ~1 model.
locfunc	a function to compute a location for a sample. By default, the median is used. However, especially for low counts, the shorth function from the genefilter package may give better results.
geoMeans	by default this is not provided and the geometric means of the counts are calculated within the function. A vector of geometric means from another count matrix can be provided for a "frozen" size factor calculation
controlGenes	optional, numeric or logical index vector specifying those genes to use for size factor estimation (e.g. housekeeping or spike-in genes)
normMatrix	optional, a matrix of normalization factors which do not control for library size (e.g. average transcript length of genes for each sample). Providing normMatrix will estimate size factors on the count matrix divided by normMatrix and store the product of the size factors and normMatrix as normalizationFactors.

Details

Typically, the function is called with the idiom:

```
dds <- estimateSizeFactors(dds)
```

See DESeq for a description of the use of size factors in the GLM. One should call this function after DESeqDataSet unless size factors are manually specified with sizeFactors. Alternatively, gene-specific normalization factors for each sample can be provided using normalizationFactors which will always preempt sizeFactors in calculations.

Internally, the function calls estimateSizeFactorsForMatrix, which provides more details on the calculation.

Value

The DESeqDataSet passed as parameters, with the size factors filled in.

Author(s)

Simon Anders

References

Reference for the median ratio method:

Simon Anders, Wolfgang Huber: Differential expression analysis for sequence count data. Genome Biology 2010, 11:106. http://dx.doi.org/10.1186/gb-2010-11-10-r106

See Also

estimateSizeFactorsForMatrix

Examples

```
dds <- makeExampleDESeqDataSet(n=1000, m=4)
dds <- estimateSizeFactors(dds)
sizeFactors(dds)

dds <- estimateSizeFactors(dds, controlGenes=1:200)

m <- matrix(runif(1000 * 4, .5, 1.5), ncol=4)
dds <- estimateSizeFactors(dds, normMatrix=m)
normalizationFactors(dds)[1:3,]

geoMeans <- exp(rowMeans(log(counts(dds))))
dds <- estimateSizeFactors(dds,geoMeans=geoMeans)
sizeFactors(dds)
```

---

estimateSizeFactorsForMatrix

Low-level function to estimate size factors with robust regression.

Description

Given a matrix or data frame of count data, this function estimates the size factors as follows: Each column is divided by the geometric means of the rows. The median (or, if requested, another location estimator) of these ratios (skipping the genes with a geometric mean of zero) is used as the size factor for this column. Typically, one will not call this function directly, but use estimateSizeFactors.

Usage

```
estimateSizeFactorsForMatrix(counts, locfunc = stats::median, geoMeans,
  controlGenes)
```

Arguments

counts	a matrix or data frame of counts, i.e., non-negative integer values
locfunc	a function to compute a location for a sample. By default, the median is used. However, especially for low counts, the shorth function from genefilter may give better results.
geoMeans	by default this is not provided, and the geometric means of the counts are calculated within the function. A vector of geometric means from another count matrix can be provided for a "frozen" size factor calculation
controlGenes	optional, numeric or logical index vector specifying those genes to use for size factor estimation (e.g. housekeeping or spike-in genes)

Value

a vector with the estimates size factors, one element per column

Author(s)

Simon Anders

See Also

estimateSizeFactors

Examples

```
dds <- makeExampleDESeqDataSet()
estimateSizeFactorsForMatrix(counts(dds))
geoMeans <- exp(rowMeans(log(counts(dds))))
estimateSizeFactorsForMatrix(counts(dds),geoMeans=geoMeans)
```

---

fpkm *FPKM: fragments per kilobase per million mapped fragments*

Description

The following function returns fragment counts normalized per kilobase of feature length per million mapped fragments (by default using a robust estimate of the library size, as in estimateSizeFactors).

Usage

```
fpkm(object, robust = TRUE)
```

Arguments

object	a DESeqDataSet
robust	whether to use size factors to normalize rather than taking the column sums of the raw counts, using the fpm function.

Details

The length of the features (e.g. genes) is calculated one of two ways: if there is a matrix named "avgTxLength" in assays(dds), this will take precedence in the length normalization. Otherwise, feature length is calculated from the rowRanges of the dds object, if a column basepairs is not present in mcols(dds). The calculated length is the number of basepairs in the union of all GRanges assigned to a given row of object, e.g., the union of all basepairs of exons of a given gene.

Note that, when the read/fragment counting has inter-feature dependencies, a strict normalization would not incorporate the basepairs of a feature which overlap another feature. This inter-feature dependence is not taken into consideration in the internal union basepair calculation.

Value

a matrix which is normalized per kilobase of the union of basepairs in the GRangesList or GRanges of the mcols(object), and per million of mapped fragments, either using the robust median ratio method (robust=TRUE, default) or using raw counts (robust=FALSE). Defining a column `mcols(object)$basepairs` takes precedence over internal calculation of the kilobases for each row.

See Also

fpm

Examples

```
# create a matrix with 1 million counts for the
# 2nd and 3rd column, the 1st and 4th have
# half and double the counts, respectively.
m <- matrix(1e6 * rep(c(.125, .25, .25, .5), each=4),
            ncol=4, dimnames=list(1:4,1:4))
mode(m) <- "integer"
se <- SummarizedExperiment(list(counts=m), colData=DataFrame(sample=1:4))
dds <- DESeqDataSet(se, ~ 1)

# create 4 GRanges with lengths: 1, 1, 2, 2.5 Kb
gr1 <- GRanges("chr1",IRanges(1,1000)) # 1kb
gr2 <- GRanges("chr1",IRanges(c(1,1001),c( 500,1500))) # 1kb
gr3 <- GRanges("chr1",IRanges(c(1,1001),c(1000,2000))) # 2kb
gr4 <- GRanges("chr1",IRanges(c(1,1001),c(200,1300))) # 500bp
rowRanges(dds) <- GRangesList(gr1,gr2,gr3,gr4)

# the raw counts
counts(dds)

# the FPM values
fpm(dds)

# the FPKM values
fpkm(dds)
```

---

fpm *FPM: fragments per million mapped fragments*

Description

Calculates either a robust version (default) or the traditional matrix of fragments/counts per million mapped fragments (FPM/CPM). Note: this function is written very simply and can be easily altered to produce other behavior by examining the source code.

Usage

```
fpm(object, robust = TRUE)
```

Arguments

object	a DESeqDataSet
robust	whether to use size factors to normalize rather than taking the column sums of the raw counts. If TRUE, the size factors and the geometric mean of column sums are multiplied to create a robust library size estimate.

Value

a matrix which is normalized per million of mapped fragments, either using the robust median ratio method (robust=TRUE, default) or using raw counts (robust=FALSE).

See Also

fpkm

Examples

```
# generate a dataset with size factors: .5, 1, 1, 2
dds <- makeExampleDESeqDataSet(m = 4, n = 1000,
                               interceptMean=log2(1e3),
                               interceptSD=0,
                               sizeFactors=c(.5,1,1,2),
                               dispMeanRel=function(x) .01)

# add a few rows with very high count
counts(dds)[4:10,] <- 2e5L

# in this robust version, the counts are comparable across samples
round(head(fpm(dds), 3))

# in this column sum version, the counts are still skewed:
# sample1 < sample2 & 3 < sample 4
round(head(fpm(dds, robust=FALSE), 3))

# the column sums of the robust version
# are not equal to 1e6, but the
# column sums of the non-robust version
# are equal to 1e6 by definition

colSums(fpm(dds))/1e6
colSums(fpm(dds, robust=FALSE))/1e6
```

---

makeExampleDESeqDataSet

Make a simulated DESeqDataSet

Description

Constructs a simulated dataset of Negative Binomial data from two conditions. By default, there are no fold changes between the two conditions, but this can be adjusted with the `betaSD` argument.

Usage

```
makeExampleDESeqDataSet(n = 1000, m = 12, betaSD = 0, interceptMean = 4,
  interceptSD = 2, dispMeanRel = function(x) 4/x + 0.1,
  sizeFactors = rep(1, m))
```

Arguments

`n`	number of rows
`m`	number of columns
`betaSD`	the standard deviation for non-intercept betas, i.e. beta ~ N(0,betaSD)
`interceptMean`	the mean of the intercept betas (log2 scale)
`interceptSD`	the standard deviation of the intercept betas (log2 scale)
`dispMeanRel`	a function specifying the relationship of the dispersions on `2^trueIntercept`
`sizeFactors`	multiplicative factors for each sample

Value

a `DESeqDataSet` with true dispersion, intercept and beta values in the metadata columns. Note that the true betas are provided on the log2 scale.

Examples

```
dds <- makeExampleDESeqDataSet()
dds
```

`nbinomLRT` *Likelihood ratio test (chi-squared test) for GLMs*

Description

This function tests for significance of change in deviance between a full and reduced model which are provided as `formula`. Fitting uses previously calculated `sizeFactors` (or `normalizationFactors`) and dispersion estimates.

Usage

```
nbinomLRT(object, full = design(object), reduced, betaPrior = FALSE,
  betaPriorVar, maxit = 100, useOptim = TRUE, quiet = FALSE,
  useQR = TRUE)
```

Arguments

object	a DESeqDataSet
full	the full model formula, this should be the formula in design(object). alternatively, can be a matrix
reduced	a reduced formula to compare against, e.g. the full model with a term or terms of interest removed. alternatively, can be a matrix
betaPrior	whether or not to put a zero-mean normal prior on the non-intercept coefficients While the beta prior is used typically, for the Wald test, it can also be specified for the likelihood ratio test. For more details on the calculation, see nbinomWaldTest.
betaPriorVar	a vector with length equal to the number of model terms including the intercept. which if missing is estimated from the rows which do not have any zeros
maxit	the maximum number of iterations to allow for convergence of the coefficient vector
useOptim	whether to use the native optim function on rows which do not converge within maxit
quiet	whether to print messages at each step
useQR	whether to use the QR decomposition on the design matrix X while fitting the GLM

Details

The difference in deviance is compared to a chi-squared distribution with df = (reduced residual degrees of freedom - full residual degrees of freedom). This function is comparable to the nbinomGLMTest of the previous version of DESeq and an alternative to the default nbinomWaldTest.

Value

a DESeqDataSet with new results columns accessible with the results function. The coefficients and standard errors are reported on a log2 scale.

See Also

DESeq, nbinomWaldTest

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
dds <- nbinomLRT(dds, reduced = ~ 1)
res <- results(dds)
```

nbinomWaldTest *Wald test for the GLM coefficients*

Description

This function tests for significance of coefficients in a Negative Binomial GLM, using previously calculated sizeFactors (or normalizationFactors) and dispersion estimates. See DESeq for the GLM formula.

Usage

```
nbinomWaldTest(object, betaPrior, betaPriorVar, modelMatrix = NULL,
  modelMatrixType, maxit = 100, useOptim = TRUE, quiet = FALSE,
  useT = FALSE, df, useQR = TRUE)
```

Arguments

object	a DESeqDataSet
betaPrior	whether or not to put a zero-mean normal prior on the non-intercept coefficients
betaPriorVar	a vector with length equal to the number of model terms including the intercept. betaPriorVar gives the variance of the prior on the sample betas on the log2 scale. if missing (default) this is estimated from the data
modelMatrix	an optional matrix, typically this is set to NULL and created within the function. only can be supplied if betaPrior=FALSE
modelMatrixType	either "standard" or "expanded", which describe how the model matrix, X of the formula in DESeq, is formed. "standard" is as created by model.matrix using the design formula. "expanded" includes an indicator variable for each level of factors in addition to an intercept. betaPrior must be set to TRUE in order for expanded model matrices to be fit.
maxit	the maximum number of iterations to allow for convergence of the coefficient vector
useOptim	whether to use the native optim function on rows which do not converge within maxit
quiet	whether to print messages at each step
useT	whether to use a t-distribution as a null distribution, for significance testing of the Wald statistics. If FALSE, a standard normal null distribution is used.
df	the degrees of freedom for the t-distribution
useQR	whether to use the QR decomposition on the design matrix X while fitting the GLM

Details

The fitting proceeds as follows: standard maximum likelihood estimates for GLM coefficients (synonymous with "beta", "log2 fold change", "effect size") are calculated. A zero-centered Normal prior distribution is assumed for the coefficients other than the intercept. The variance of the prior distribution for each non-intercept coefficient is calculated using the observed distribution of the maximum likelihood coefficients. The final coefficients are then maximum a posteriori estimates using this prior (Tikhonov/ridge regularization). See below for details on the prior variance and the Methods section of the DESeq2 manuscript for more detail. The use of a prior has little effect on genes with high counts and helps to moderate the large spread in coefficients for genes with low counts. For calculating Wald test p-values, the coefficients are scaled by their standard errors and then compared to a standard Normal distribution.

The prior variance is calculated by matching the 0.05 upper quantile of the observed MLE coefficients to a zero-centered Normal distribution. In a change of methods since the 2014 paper, the weighted upper quantile is calculated using the `wtd.quantile` function from the Hmisc package. The weights are the inverse of the expected variance of log counts, so the inverse of $1/\bar{\mu} + \alpha_{tr}$ using the mean of normalized counts and the trended dispersion fit. The weighting ensures that noisy estimates of log fold changes from small count genes do not overly influence the calculation of the prior variance. The final prior variance for a factor level is the average of the estimated prior variance over all contrasts of all levels of the factor. Another change since the 2014 paper: when interaction terms are present in the design, the prior on log fold changes is turned off (for more details, see the vignette section, "Methods changes since the 2014 DESeq2 paper").

When a log2 fold change prior is used (betaPrior=TRUE), then `nbinomWaldTest` will by default use expanded model matrices, as described in the `modelMatrixType` argument, unless this argument is used to override the default behavior. This ensures that log2 fold changes will be independent of the choice of reference level. In this case, the beta prior variance for each factor is calculated as the average of the mean squared maximum likelihood estimates for each level and every possible contrast. The `results` function without any arguments will automatically perform a contrast of the last level of the last variable in the design formula over the first level. The `contrast` argument of the `results` function can be used to generate other comparisons.

The Wald test can be replaced with the `nbinomLRT` for an alternative test of significance.

Value

a DESeqDataSet with results columns accessible with the `results` function. The coefficients and standard errors are reported on a log2 scale.

See Also

`DESeq`, `nbinomLRT`

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
dds <- nbinomWaldTest(dds)
res <- results(dds)
```

`normalizationFactors` *Accessor functions for the normalization factors in a DESeqDataSet object.*

Description

Gene-specific normalization factors for each sample can be provided as a matrix, which will preempt `sizeFactors`. In some experiments, counts for each sample have varying dependence on covariates, e.g. on GC-content for sequencing data run on different days, and in this case it makes sense to provide gene-specific factors for each sample rather than a single size factor.

Usage

```
normalizationFactors(object, ...)

normalizationFactors(object, ...) <- value

## S4 method for signature 'DESeqDataSet'
normalizationFactors(object)

## S4 replacement method for signature 'DESeqDataSet,matrix'
normalizationFactors(object) <- value
```

Arguments

`object`	a `DESeqDataSet` object.
`...`	additional arguments
`value`	the matrix of normalization factors

Details

Normalization factors alter the model of `DESeq` in the following way, for counts K_{ij} and normalization factors NF_{ij} for gene i and sample j:

$$K_{ij} \sim \text{NB}(\mu_{ij}, \alpha_i)$$

$$\mu_{ij} = NF_{ij} q_{ij}$$

Note

Normalization factors are on the scale of the counts (similar to `sizeFactors`) and unlike offsets, which are typically on the scale of the predictors (in this case, log counts). Normalization factors should include library size normalization. They should have row-wise geometric mean near 1, as is the case with size factors, such that the mean of normalized counts is close to the mean of unnormalized counts. See example code below.

Examples

```
dds <- makeExampleDESeqDataSet(n=100, m=4)

normFactors <- matrix(runif(nrow(dds)*ncol(dds),0.5,1.5),
                      ncol=ncol(dds),nrow=nrow(dds),
                      dimnames=list(1:nrow(dds),1:ncol(dds)))

# the normalization factors matrix should not have 0's in it
# it should have geometric mean near 1 for each row
normFactors <- normFactors / exp(rowMeans(log(normFactors)))
normalizationFactors(dds) <- normFactors

dds <- DESeq(dds)
```

---

normalizeGeneLength *Normalize for gene length*

Description

Normalize for gene length using the output of transcript abundance estimators

Usage

```
normalizeGeneLength(object, files, level = c("tx", "gene"),
  geneIdCol = "gene_id", lengthCol = "length", abundanceCol = "FPKM",
  dropGenes = FALSE, importer, ...)
```

Arguments

object	the DESeqDataSet, before calling DESeq
files	a character vector specifying the filenames of output files containing either transcript abundance estimates with transcript length, or average transcript length information per gene.
level	either "tx" or "gene"
geneIdCol	the name of the column of the files specifying the gene id. This should line up with the rownames(object). The information in the files will be re-ordered to line up with the rownames of the object. See dropGenes for more details.
lengthCol	the name of the column of files specifying the length of the feature, either transcript for level="tx" or the gene for level="gene".
abundanceCol	only needed if level="tx", the name of the column specifying the abundance estimate of the transcript.
dropGenes	whether to drop genes from the object, as labelled by rownames(object), which are not present in the geneIdCol of the files. Defaults to FALSE and gives an error upon finding rownames of the object not present in the geneIdCol of the files. The function will reorder the matching rows of the files to match the rownames of the object.

importer	a function to read the files. fread from the data.table package is quite fast, but other options include read.table, read.csv. One can pre-test with importer(files[1]).
...	further arguments passed to importer

Details

This is a prototype function for importing information about changes in the average transcript length for each gene. The use of this function is only for testing purposes.

The function simply imports or calculates average transcript length for each gene and each sample from external files, and provides this matrix to the normMatrix argument of estimateSizeFactors. By average transcript length, the average refers to a weighted average with respect to the transcript abundances. The RSEM method includes such a column in their gene.results files, but an estimate of average transcript length can be obtained from any software which outputs a file with a row for each transcript, specifying: transcript length, estimate of transcript abundance, and the gene which the transcript belongs to.

Normalization factors accounting for both average transcript length and library size of each sample are generated and then stored within the data object. The analysis can then continue with DESeq; the stored normalization factors will be used instead of size factors in the analysis.

For RSEM genes.results files, specify level="gene", geneIdCol="gene_id", and lengthCol="effective_length"

For Cufflinks isoforms.fpkm_tracking files, specify level="tx", geneIdCol="gene_id", lengthCol="length", and abundanceCol="FPKM".

For Sailfish output files, one can write an importer function which attaches a column gene_id based on Transcript ID, and then specify level="tx", geneIdCol="gene_id", lengthCol="Length" and abundanceCol="RPKM".

Along with the normalization matrix which is stored in normalizationFactors(object), the resulting gene length matrix is stored in assays(dds)[["avgTxLength"]], and will take precedence in calls to fpkm.

Value

a DESeqDataSet with normalizationFactors accounting for average transcript length and library size

Examples

```
n <- 10
files <- c("sample1","sample2")
gene_id <- rep(paste0("gene",seq_len(n)),each=3)
set.seed(1)
sample1 <- data.frame(gene_id=gene_id,length=rpois(3*n,2000),FPKM=round(rnorm(3*n,10,1),2))
sample2 <- data.frame(gene_id=gene_id,length=rpois(3*n,2000),FPKM=round(rnorm(3*n,10,1),2))
importer <- get
dds <- makeExampleDESeqDataSet(n=n, m=2)
dds <- normalizeGeneLength(dds, files=files, level="tx",
  geneIdCol="gene_id", lengthCol="length", abundanceCol="FPKM",
  dropGenes=TRUE, importer=importer)
```

normTransform *Normalized counts transformation*

Description

A simple function for creating a DESeqTransform object after applying: f(count + pc).

Usage

```
normTransform(object, f = log2, pc = 1)
```

Arguments

object	a DESeqDataSet object
f	a function to apply to normalized counts
pc	a pseudocount to add to normalized counts

See Also

varianceStabilizingTransformation, rlog

plotCounts *Plot of normalized counts for a single gene on log scale*

Description

Note: normalized counts plus a pseudocount of 0.5 are shown.

Usage

```
plotCounts(dds, gene, intgroup = "condition", normalized = TRUE,
  transform = FALSE, main, xlab = "group", returnData = FALSE,
  replaced = FALSE, ...)
```

Arguments

dds	a DESeqDataSet
gene	a character, specifying the name of the gene to plot
intgroup	interesting groups: a character vector of names in colData(x) to use for grouping
normalized	whether the counts should be normalized by size factor (default is TRUE)
transform	whether to present log2 counts (TRUE) or to present the counts on the log scale (FALSE, default)
main	as in 'plot'

xlab	as in 'plot'
returnData	should the function only return the data.frame of counts and covariates for custom plotting (default is FALSE)
replaced	use the outlier-replaced counts if they exist
...	arguments passed to plot

Examples

```
dds <- makeExampleDESeqDataSet()
plotCounts(dds, "gene1")
```

---

plotDispEsts *Plot dispersion estimates*

Description

A simple helper function that plots the per-gene dispersion estimates together with the fitted mean-dispersion relationship.

Usage

```
## S4 method for signature 'DESeqDataSet'
plotDispEsts(object, ymin, genecol = "black",
  fitcol = "red", finalcol = "dodgerblue", legend = TRUE, xlab, ylab,
  log = "xy", cex = 0.45, ...)
```

Arguments

object	a DESeqDataSet, with dispersions estimated
ymin	the lower bound for points on the plot, points beyond this are drawn as triangles at ymin
genecol	the color for gene-wise dispersion estimates
fitcol	the color of the fitted estimates
finalcol	the color of the final estimates used for testing
legend	logical, whether to draw a legend
xlab	xlab
ylab	ylab
log	log
cex	cex
...	further arguments to plot

Author(s)

Simon Anders

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
plotDispEsts(dds)
```

---

plotMA *MA-plot from base means and log fold changes*

Description

A simple helper function that makes a so-called "MA-plot", i.e. a scatter plot of log2 fold changes (on the y-axis) versus the mean of normalized counts (on the x-axis).

Usage

```
## S4 method for signature 'DESeqDataSet'
plotMA(object, alpha = 0.1, main = "", ylim, ...)

## S4 method for signature 'DESeqResults'
plotMA(object, alpha, main = "", ylim, MLE = FALSE,
  ...)
```

Arguments

object	a DESeqResults object produced by results; or a DESeqDataSet processed by DESeq, or the individual functions nbinomWaldTest or nbinomLRT
alpha	the significance level for thresholding adjusted p-values
main	optional title for the plot
ylim	optional y limits
...	further arguments passed to plotMA if object is DESeqResults or to results if object is DESeqDataSet
MLE	whether to plot the MLE (unshrunken estimates), defaults to FALSE. Requires that results was run with addMLE=TRUE. Note that the MLE will be plotted regardless of this argument, if DESeq() was run with betaPrior=FALSE.

Details

This function is essentially two lines of code: building a data.frame and passing this to the plotMA method for data.frame from the geneplotter package. The code of this function can be seen with: getMethod("plotMA","DESeqDataSet") If users wish to modify the graphical parameters of the plot, it is recommended to build the data.frame in the same manner and call plotMA.

Author(s)

Michael Love

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- DESeq(dds)
plotMA(dds)
res <- results(dds)
plotMA(res)
```

---

plotPCA *Sample PCA plot for transformed data*

Description

This plot helps to check for batch effects and the like.

Usage

```
## S4 method for signature 'DESeqTransform'
plotPCA(object, intgroup = "condition",
  ntop = 500, returnData = FALSE)
```

Arguments

object	a DESeqTransform object, with data in assay(x), produced for example by either rlog or varianceStabilizingTransformation.
intgroup	interesting groups: a character vector of names in colData(x) to use for grouping
ntop	number of top genes to use for principal components, selected by highest row variance
returnData	should the function only return the data.frame of PC1 and PC2 with intgroup covariates for custom plotting (default is FALSE)

Value

An object created by ggplot, which can be assigned and further customized.

Note

See the vignette for an example of variance stabilization and PCA plots. Note that the source code of plotPCA is very simple. The source can be found by typing DESeq2:::plotPCA.DESeqTransform or getMethod("plotPCA","DESeqTransform"), or browsed on github at https://github.com/Bioconductor-mirror/DESeq2/blob/master/R/plots.R Users should find it easy to customize this function.

Author(s)

Wolfgang Huber

Examples

```
# using rlog transformed data:
dds <- makeExampleDESeqDataSet(betaSD=1)
rld <- rlog(dds)
plotPCA(rld)

# also possible to perform custom transformation:
dds <- estimateSizeFactors(dds)
# shifted log of normalized counts
se <- SummarizedExperiment(log2(counts(dds, normalized=TRUE) + 1),
                           colData=colData(dds))
# the call to DESeqTransform() is needed to
# trigger our plotPCA method.
plotPCA( DESeqTransform( se ) )
```

---

plotSparsity *Sparsity plot*

Description

A simple plot of the concentration of counts in a single sample over the sum of counts per gene. Not technically the same as "sparsity", but this plot is useful diagnostic for datasets which might not fit a negative binomial assumption: genes with many zeros and individual very large counts are difficult to model with the negative binomial distribution.

Usage

```
plotSparsity(x, normalized = TRUE, ...)
```

Arguments

x	a matrix or DESeqDataSet
normalized	whether to normalize the counts from a DESeqDataSEt
...	passed to plot

Examples

```
dds <- makeExampleDESeqDataSet(n=1000,m=4,dispMeanRel=function(x) .5)
dds <- estimateSizeFactors(dds)
plotSparsity(dds)
```

`replaceOutliers` *Replace outliers with trimmed mean*

Description

Note that this function is called within DESeq, so is not necessary to call on top of a DESeq call. See the minReplicatesForReplace argument documented in link{DESeq}.

Usage

```
replaceOutliers(object, trim = 0.2, cooksCutoff, minReplicates = 7,
  whichSamples)

replaceOutliersWithTrimmedMean(object, trim = 0.2, cooksCutoff,
  minReplicates = 7, whichSamples)
```

Arguments

object	a DESeqDataSet object, which has already been processed by either DESeq, nbinomWaldTest or nbinomLRT, and therefore contains a matrix contained in assays(dds)[["cooks"]]. These are the Cook's distances which will be used to define outlier counts.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of the normalized counts for a gene before the mean is computed
cooksCutoff	the threshold for defining an outlier to be replaced. Defaults to the .99 quantile of the F(p, m - p) distribution, where p is the number of parameters and m is the number of samples.
minReplicates	the minimum number of replicate samples necessary to consider a sample eligible for replacement (including itself). Outlier counts will not be replaced if the sample is in a cell which has less than minReplicates replicates.
whichSamples	optional, a numeric or logical index to specify which samples should have outliers replaced. if missing, this is determined using minReplicates.

Details

This function replaces outlier counts flagged by extreme Cook's distances, as calculated by DESeq, nbinomWaldTest or nbinomLRT, with values predicted by the trimmed mean over all samples (and adjusted by size factor or normalization factor). This function replaces the counts in the matrix returned by counts(dds) and the Cook's distances in assays(dds)[["cooks"]]. Original counts are preserved in assays(dds)[["originalCounts"]].

The DESeq function calculates a diagnostic measure called Cook's distance for every gene and every sample. The results function then sets the p-values to NA for genes which contain an outlying count as defined by a Cook's distance above a threshold. With many degrees of freedom, i.e. many more samples than number of parameters to be estimated– it might be undesirable to remove entire genes from the analysis just because their data include a single count outlier. An alternate strategy is to replace the outlier counts with the trimmed mean over all samples, adjusted by the size factor

or normalization factor for that sample. The following simple function performs this replacement for the user, for samples which have at least `minReplicates` number of replicates (including that sample). For more information on Cook's distance, please see the two sections of the vignette: 'Dealing with count outliers' and 'Count outlier detection'.

Value

a DESeqDataSet with replaced counts in the slot returned by `counts` and the original counts preserved in `assays(dds)[["originalCounts"]]`

See Also

`DESeq`

`results` *Extract results from a DESeq analysis*

Description

`results` extracts a result table from a DESeq analysis giving base means across samples, log2 fold changes, standard errors, test statistics, p-values and adjusted p-values; `resultsNames` returns the names of the estimated effects (coefficents) of the model; `removeResults` returns a `DESeqDataSet` object with results columns removed.

Usage

```
results(object, contrast, name, lfcThreshold = 0,
  altHypothesis = c("greaterAbs", "lessAbs", "greater", "less"),
  listValues = c(1, -1), cooksCutoff, independentFiltering = TRUE,
  alpha = 0.1, filter, theta, pAdjustMethod = "BH", filterFun,
  format = c("DataFrame", "GRanges", "GRangesList"), test, addMLE = FALSE,
  tidy = FALSE, parallel = FALSE, BPPARAM = bpparam())

resultsNames(object)

removeResults(object)
```

Arguments

`object`	a DESeqDataSet, on which one of the following functions has already been called: `DESeq`, `nbinomWaldTest`, or `nbinomLRT`
`contrast`	this argument specifies what comparison to extract from the `object` to build a results table. one of either: • a character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change (simplest case)

- a list of 2 character vectors: the names of the fold changes for the numerator, and the names of the fold changes for the denominator. these names should be elements of `resultsNames(object)`. if the list is length 1, a second element is added which is the empty character vector, `character()`. (more general case, can be to combine interaction terms and main effects)
- a numeric contrast vector with one element for each element in `resultsNames(object)` (most general case)

If specified, the `name` argument is ignored.

`name` the name of the individual effect (coefficient) for building a results table. Use this argument rather than `contrast` for continuous variables, individual effects or for individual interaction terms. The value provided to `name` must be an element of `resultsNames(object)`.

`lfcThreshold` a non-negative value, which specifies the test which should be applied to the log2 fold changes. The standard is a test that the log2 fold changes are not equal to zero. However, log2 fold changes greater or less than `lfcThreshold` can also be tested. Specify the alternative hypothesis using the `altHypothesis` argument. If `lfcThreshold` is specified, the results are Wald tests, and LRT p-values will be overwritten.

`altHypothesis` character which specifies the alternative hypothesis, i.e. those values of log2 fold change which the user is interested in finding. The complement of this set of values is the null hypothesis which will be tested. If the log2 fold change specified by `name` or by `contrast` is written as β, then the possible values for `altHypothesis` represent the following alternate hypotheses:

- greaterAbs: $|\beta| >$ lfcThreshold, and p-values are two-tailed
- lessAbs: $|\beta| <$ lfcThreshold, NOTE: this requires that `betaPrior=FALSE` has been specified in the previous `DESeq` call. p-values are the maximum of the upper and lower tests.
- greater: $\beta >$ lfcThreshold
- less: $\beta < -$lfcThreshold

`listValues` only used if a list is provided to `contrast`: a numeric of length two: the log2 fold changes in the list are multiplied by these values. the first number should be positive and the second negative. by default this is `c(1,-1)`

`cooksCutoff` theshold on Cook's distance, such that if one or more samples for a row have a distance higher, the p-value for the row is set to NA. The default cutoff is the .99 quantile of the F(p, m-p) distribution, where p is the number of coefficients being fitted and m is the number of samples. Set to Inf or FALSE to disable the resetting of p-values to NA. Note: this test excludes the Cook's distance of samples belonging to experimental groups with only 2 samples.

`independentFiltering`
logical, whether independent filtering should be applied automatically

`alpha` the significance cutoff used for optimizing the independent filtering (by default 0.1). If the adjusted p-value cutoff (FDR) will be a value other than 0.1, `alpha` should be set to that value.

`filter` the vector of filter statistics over which the independent filtering will be optimized. By default the mean of normalized counts is used.

theta	the quantiles at which to assess the number of rejections from independent filtering
pAdjustMethod	the method to use for adjusting p-values, see ?p.adjust
filterFun	an optional custom function for independent filtering, with arguments alpha, filter, test, theta, and method similar to genefilter::filtered_R, and which returns padj
format	character, either "DataFrame", "GRanges", or "GRangesList", whether the results should be printed as a DESeqResults DataFrame, or if the results DataFrame should be attached as metadata columns to the GRanges or GRangesList rowRanges of the DESeqDataSet. If the rowRanges is a GRangesList, and GRanges is requested, the range of each gene will be returned
test	this is typically automatically detected internally. the one exception is after nbinomLRT has been run, test="Wald" will generate Wald statistics and Wald test p-values.
addMLE	whether the "unshrunken" maximum likelihood estimates (MLE) of log2 fold change should be added as a column to the results table (default is FALSE). only applicable when a beta prior was used during the model fitting. only implemented for 'contrast' for three element character vectors or 'name' for interactions.
tidy	whether to output the results table with rownames as a first column 'row'. the table will also be coerced to data.frame
parallel	if FALSE, no parallelization. if TRUE, parallel execution using BiocParallel, see next argument BPPARAM
BPPARAM	an optional parameter object passed internally to bplapply when parallel=TRUE. If not specified, the parameters last registered with register will be used.

Details

The results table when printed will provide the information about the comparison, e.g. "log2 fold change (MAP): condition treated vs untreated", meaning that the estimates are of log2(treated / untreated), as would be returned by contrast=c("condition","treated","untreated"). Multiple results can be returned for analyses beyond a simple two group comparison, so results takes arguments contrast and name to help the user pick out the comparisons of interest for printing a results table. The use of the contrast argument is recommended for exact specification of the levels which should be compared and their order.

If results is run without specifying contrast or name, it will return the comparison of the last level of the last variable in the design formula over the first level of this variable. For example, for a simple two-group comparison, this would return the log2 fold changes of the second group over the first group (the reference level). Please see examples below and in the vignette.

The argument contrast can be used to generate results tables for any comparison of interest, for example, the log2 fold change between two levels of a factor, and its usage is described below. It can also accomodate more complicated numeric comparisons. The test statistic used for a contrast is:

$$c^t \beta / \sqrt{c^t \Sigma c}$$

The argument name can be used to generate results tables for individual effects, which must be individual elements of resultsNames(object). These individual effects could represent continuous covariates, effects for individual levels, or individual interaction effects.

Information on the comparison which was used to build the results table, and the statistical test which was used for p-values (Wald test or likelihood ratio test) is stored within the object returned by results. This information is in the metadata columns of the results table, which is accessible by calling mcols on the DESeqResults object returned by results.

On p-values:

By default, independent filtering is performed to select a set of genes for multiple test correction which maximizes the number of adjusted p-values less than a given critical value alpha (by default 0.1). The filter used for maximizing the number of rejections is the mean of normalized counts for all samples in the dataset. In version >= 1.10, the threshold chosen is the lowest quantile of the filter for which the number of rejections is close to the peak of a curve fit to the number of rejections over the filter quantiles. 'Close to' is defined as within 1 residual standard deviation.

The adjusted p-values for the genes which do not pass the filter threshold are set to NA. By default, the mean of normalized counts is used to perform this filtering, though other statistics can be provided. Several arguments from the filtered_p function of genefilter are provided here to control or turn off the independent filtering behavior.

By default, results assigns a p-value of NA to genes containing count outliers, as identified using Cook's distance. See the cooksCutoff argument for control of this behavior. Cook's distances for each sample are accessible as a matrix "cooks" stored in the assays() list. This measure is useful for identifying rows where the observed counts might not fit to a Negative Binomial distribution.

For analyses using the likelihood ratio test (using nbinomLRT), the p-values are determined solely by the difference in deviance between the full and reduced model formula. A log2 fold change is included, which can be controlled using the name argument, or by default this will be the estimated coefficient for the last element of resultsNames(object).

Value

For results: a DESeqResults object, which is a simple subclass of DataFrame. This object contains the results columns: baseMean, log2FoldChange, lfcSE, stat, pvalue and padj, and also includes metadata columns of variable information. The lfcSE gives the standard error of the log2FoldChange. For the Wald test, stat is the Wald statistic: the log2FoldChange divided by lfcSE, which is compared to a standard Normal distribution to generate a two-tailed pvalue. For the likelihood ratio test (LRT), stat is the difference in deviance between the reduced model and the full model, which is compared to a chi-squared distribution to generate a pvalue.

For resultsNames: the names of the columns available as results, usually a combination of the variable name and a level

For removeResults: the original DESeqDataSet with results metadata columns removed

References

Richard Bourgon, Robert Gentleman, Wolfgang Huber: Independent filtering increases detection power for high-throughput experiments. PNAS (2010), http://dx.doi.org/10.1073/pnas.0914005107

See Also

DESeq

Examples

```
## Example 1: simple two-group comparison

dds <- makeExampleDESeqDataSet(m=4)

dds <- DESeq(dds)
res <- results(dds)
res[ order(res$padj), ]

## Example 2: two conditions, two genotypes, with an interaction term

dds <- makeExampleDESeqDataSet(n=100,m=12)
dds$genotype <- factor(rep(rep(c("I","II"),each=3),2))

design(dds) <- ~ genotype + condition + genotype:condition
dds <- DESeq(dds)
resultsNames(dds)

# Note: design with interactions terms by default have betaPrior=FALSE

# the condition effect for genotype I (the main effect)
results(dds, contrast=c("condition","B","A"))

# the condition effect for genotype II
# this is, by definition, the main effect *plus* the interaction term
# (the extra condition effect in genotype II compared to genotype I).
results(dds, list( c("condition_B_vs_A","genotypeII.conditionB") ))

# the interaction term, answering: is the condition effect *different* across genotypes?
results(dds, name="genotypeII.conditionB")

## Example 3: two conditions, three genotypes

# ~~~ Using interaction terms ~~~

dds <- makeExampleDESeqDataSet(n=100,m=18)
dds$genotype <- factor(rep(rep(c("I","II","III"),each=3),2))
design(dds) <- ~ genotype + condition + genotype:condition
dds <- DESeq(dds)
resultsNames(dds)

# the condition effect for genotype I (the main effect)
results(dds, contrast=c("condition","B","A"))

# the condition effect for genotype III.
# this is the main effect *plus* the interaction term
# (the extra condition effect in genotype III compared to genotype I).
results(dds, contrast=list( c("condition_B_vs_A","genotypeIII.conditionB") ))
```

```
# the interaction term for condition effect in genotype III vs genotype I.
# this tests if the condition effect is different in III compared to I
results(dds, name="genotypeIII.conditionB")

# the interaction term for condition effect in genotype III vs genotype II.
# this tests if the condition effect is different in III compared to II
results(dds, contrast=list("genotypeIII.conditionB", "genotypeII.conditionB"))

# Note that a likelihood ratio could be used to test if there are any
# differences in the condition effect between the three genotypes.

# ~~~ Using a grouping variable ~~~

# This is a useful construction when users just want to compare
# specific groups which are combinations of variables.

dds$group <- factor(paste0(dds$genotype, dds$condition))
design(dds) <- ~ group
dds <- DESeq(dds)
resultsNames(dds)

# the condition effect for genotypeIII
results(dds, contrast=c("group", "IIIB", "IIIA"))
```

`rlog` *Apply a 'regularized log' transformation*

Description

This function transforms the count data to the log2 scale in a way which minimizes differences between samples for rows with small counts, and which normalizes with respect to library size. The rlog transformation produces a similar variance stabilizing effect as `varianceStabilizingTransformation`, though `rlog` is more robust in the case when the size factors vary widely. The transformation is useful when checking for outliers or as input for machine learning techniques such as clustering or linear discriminant analysis. `rlog` takes as input a `DESeqDataSet` and returns a `RangedSummarizedExperiment` object.

Usage

```
rlog(object, blind = TRUE, intercept, betaPriorVar, fitType = "parametric")

rlogTransformation(object, blind = TRUE, intercept, betaPriorVar,
  fitType = "parametric")
```

Arguments

`object`	a DESeqDataSet, or matrix of counts

blind	logical, whether to blind the transformation to the experimental design. blind=TRUE should be used for comparing samples in an manner unbiased by prior information on samples, for example to perform sample QA (quality assurance). blind=FALSE should be used for transforming data for downstream analysis, where the full use of the design information should be made. blind=FALSE will skip re-estimation of the dispersion trend, if this has already been calculated. If many of genes have large differences in counts due to the experimental design, it is important to set blind=FALSE for downstream analysis.
intercept	by default, this is not provided and calculated automatically. if provided, this should be a vector as long as the number of rows of object, which is log2 of the mean normalized counts from a previous dataset. this will enforce the intercept for the GLM, allowing for a "frozen" rlog transformation based on a previous dataset. You will also need to provide `mcols(object)$dispFit`.
betaPriorVar	a single value, the variance of the prior on the sample betas, which if missing is estimated from the data
fitType	in case dispersions have not yet been estimated for `object`, this parameter is passed on to `estimateDispersions` (options described there).

Details

Note that neither rlog transformation nor the VST are used by the differential expression estimation in DESeq, which always occurs on the raw count data, through generalized linear modeling which incorporates knowledge of the variance-mean dependence. The rlog transformation and VST are offered as separate functionality which can be used for visualization, clustering or other machine learning tasks. See the transformation section of the vignette for more details.

The transformation does not require that one has already estimated size factors and dispersions.

The regularization is on the log fold changes of the count for each sample over an intercept, for each gene. As nearby count values for low counts genes are almost as likely as the observed count, the rlog shrinkage is greater for low counts. For high counts, the rlog shrinkage has a much weaker effect. The fitted dispersions are used rather than the MAP dispersions (so similar to the `varianceStabilizingTransformation`).

The prior variance for the shrinkag of log fold changes is calculated as follows: a matrix is constructed of the logarithm of the counts plus a pseudocount of 0.5, the log of the row means is then subtracted, leaving an estimate of the log fold changes per sample over the fitted value using only an intercept. The prior variance is then calculated by matching the upper quantiles of the observed log fold change estimates with an upper quantile of the normal distribution. A GLM fit is then calculated using this prior. It is also possible to supply the variance of the prior. See the vignette for an example of the use and a comparison with `varianceStabilizingTransformation`.

The transformed values, rlog(K), are equal to $rlog(K_{ij}) = \log_2(q_{ij}) = \beta_{i0} + \beta_{ij}$, with formula terms defined in DESeq.

The parameters of the rlog transformation from a previous dataset can be frozen and reapplied to new samples. See the 'Data quality assessment' section of the vignette for strategies to see if new samples are sufficiently similar to previous datasets. The frozen rlog is accomplished by saving the dispersion function, beta prior variance and the intercept from a previous dataset, and running `rlog` with 'blind' set to FALSE (see example below).

Value

a DESeqTransform if a DESeqDataSet was provided, or a matrix if a count matrix was provided as input. Note that for DESeqTransform output, the matrix of transformed values is stored in assay(rld). To avoid returning matrices with NA values, in the case of a row of all zeros, the rlog transformation returns zeros (essentially adding a pseudocount of 1 only to these rows).

References

Reference for regularized logarithm (rlog):

Michael I Love, Wolfgang Huber, Simon Anders: Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology 2014, 15:550. http://dx.doi.org/10.1186/s13059-014-0550-8

See Also

plotPCA, varianceStabilizingTransformation, normTransform

Examples

```
dds <- makeExampleDESeqDataSet(m=6,betaSD=1)
rld <- rlog(dds)
dists <- dist(t(assay(rld)))
plot(hclust(dists))

# run the rlog transformation on one dataset
design(dds) <- ~ 1
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
rld <- rlog(dds, blind=FALSE)

# apply the parameters to a new sample

ddsNew <- makeExampleDESeqDataSet(m=1)
mcols(ddsNew)$dispFit <- mcols(dds)$dispFit
betaPriorVar <- attr(rld,"betaPriorVar")
intercept <- mcols(rld)$rlogIntercept
rldNew <- rlog(ddsNew, blind=FALSE,
               intercept=intercept,
               betaPriorVar=betaPriorVar)
```

---

show *Show method for DESeqResults objects*

Description

Prints out the information from the metadata columns of the results object regarding the log2 fold changes and p-values, then shows the DataFrame using the standard method.

Usage

```
## S4 method for signature 'DESeqResults'
show(object)
```

Arguments

object	a DESeqResults object

Author(s)

Michael Love

---

sizeFactors *Accessor functions for the 'sizeFactors' information in a DESeqDataSet object.*

Description

The sizeFactors vector assigns to each column of the count matrix a value, the size factor, such that count values in the columns can be brought to a common scale by dividing by the corresponding size factor (as performed by `counts(dds, normalized=TRUE)`). See DESeq for a description of the use of size factors. If gene-specific normalization is desired for each sample, use normalizationFactors.

Usage

```
## S4 method for signature 'DESeqDataSet'
sizeFactors(object)

## S4 replacement method for signature 'DESeqDataSet,numeric'
sizeFactors(object) <- value
```

Arguments

object	a DESeqDataSet object.
value	a numeric vector, one size factor for each column in the count data.

Author(s)

Simon Anders

See Also

estimateSizeFactors

summary *Summarize DESeq results*

Description

Print a summary of the results from a DESeq analysis.

Usage

```
## S3 method for class 'DESeqResults'
summary(object, alpha, ...)
```

Arguments

object	a DESeqResults object
alpha	the adjusted p-value cutoff. if not set, this defaults to the alpha argument which was used in results to set the target FDR for independent filtering.
...	additional arguments

Author(s)

Michael Love

Examples

```
dds <- makeExampleDESeqDataSet(m=4)
dds <- DESeq(dds)
res <- results(dds)
summary(res)
```

varianceStabilizingTransformation *Apply a variance stabilizing transformation (VST) to the count data*

Description

This function calculates a variance stabilizing transformation (VST) from the fitted dispersion-mean relation(s) and then transforms the count data (normalized by division by the size factors or normalization factors), yielding a matrix of values which are now approximately homoskedastic (having constant variance along the range of mean values). The transformation also normalizes with respect to library size. The rlog is less sensitive to size factors, which can be an issue when size factors vary widely. These transformations are useful when checking for outliers or as input for machine learning techniques such as clustering or linear discriminant analysis.

Usage

```
varianceStabilizingTransformation(object, blind = TRUE,
  fitType = "parametric")

getVarianceStabilizedData(object)
```

Arguments

object	a DESeqDataSet or matrix of counts
blind	logical, whether to blind the transformation to the experimental design. blind=TRUE should be used for comparing samples in an manner unbiased by prior information on samples, for example to perform sample QA (quality assurance). blind=FALSE should be used for transforming data for downstream analysis, where the full use of the design information should be made. blind=FALSE will skip re-estimation of the dispersion trend, if this has already been calculated. If many of genes have large differences in counts due to the experimental design, it is important to set blind=FALSE for downstream analysis.
fitType	in case dispersions have not yet been estimated for object, this parameter is passed on to estimateDispersions (options described there).

Details

For each sample (i.e., column of counts(dds)), the full variance function is calculated from the raw variance (by scaling according to the size factor and adding the shot noise). We recommend a blind estimation of the variance function, i.e., one ignoring conditions. This is performed by default, and can be modified using the 'blind' argument.

Note that neither rlog transformation nor the VST are used by the differential expression estimation in DESeq, which always occurs on the raw count data, through generalized linear modeling which incorporates knowledge of the variance-mean dependence. The rlog transformation and VST are offered as separate functionality which can be used for visualization, clustering or other machine learning tasks. See the transformation section of the vignette for more details.

The transformation does not require that one has already estimated size factors and dispersions.

A typical workflow is shown in Section *Variance stabilizing transformation* in the package vignette.

If estimateDispersions was called with:

fitType="parametric", a closed-form expression for the variance stabilizing transformation is used on the normalized count data. The expression can be found in the file 'vst.pdf' which is distributed with the vignette.

fitType="local", the reciprocal of the square root of the variance of the normalized counts, as derived from the dispersion fit, is then numerically integrated, and the integral (approximated by a spline function) is evaluated for each count value in the column, yielding a transformed value.

fitType="mean", a VST is applied for Negative Binomial distributed counts, 'k', with a fixed dispersion, 'a': (2 asinh(sqrt(a k)) - log(a) - log(4))/log(2).

In all cases, the transformation is scaled such that for large counts, it becomes asymptotically (for large values) equal to the logarithm to base 2 of normalized counts.

The variance stabilizing transformation from a previous dataset can be frozen and reapplied to new samples. See the 'Data quality assessment' section of the vignette for strategies to see if new samples are sufficiently similar to previous datasets. The frozen VST is accomplished by saving the dispersion function accessible with `dispersionFunction`, assigning this to the `DESeqDataSet` with the new samples, and running varianceStabilizingTransformation with 'blind' set to FALSE (see example below). Then the dispersion function from the previous dataset will be used to transform the new sample(s).

Limitations: In order to preserve normalization, the same transformation has to be used for all samples. This results in the variance stabilizition to be only approximate. The more the size factors differ, the more residual dependence of the variance on the mean will be found in the transformed data. `rlog` is a transformation which can perform better in these cases. As shown in the vignette, the function `meanSdPlot` from the package **vsn** can be used to see whether this is a problem.

Value

`varianceStabilizingTransformation` returns a `DESeqTransform` if a `DESeqDataSet` was provided, or returns a a matrix if a count matrix was provided. Note that for `DESeqTransform` output, the matrix of transformed values is stored in `assay(vsd)`. `getVarianceStabilizedData` also returns a matrix.

Author(s)

Simon Anders

References

Reference for the variance stabilizing transformation for counts with a dispersion trend:

Simon Anders, Wolfgang Huber: Differential expression analysis for sequence count data. Genome Biology 2010, 11:106. http://dx.doi.org/10.1186/gb-2010-11-10-r106

See Also

`plotPCA`, `rlog`, `normTransform`

Examples

```
dds <- makeExampleDESeqDataSet(m=6)
vsd <- varianceStabilizingTransformation(dds)
dists <- dist(t(assay(vsd)))
plot(hclust(dists))

# learn the dispersion function of a dataset
design(dds) <- ~ 1
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)

# use the previous dispersion function for a new sample
ddsNew <- makeExampleDESeqDataSet(m=1)
ddsNew <- estimateSizeFactors(ddsNew)
dispersionFunction(ddsNew) <- dispersionFunction(dds)
```

```
vsdNew <- varianceStabilizingTransformation(ddsNew, blind=FALSE)
```

Index