Package 'DESeq2'

October 7, 2014

Type Package

Title Differential gene expression analysis based on the negative binomial distribution

Version 1.4.5

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 GPL (>= 3)

VignetteBuilder knitr

Imports BiocGenerics (>= 0.7.5), methods, locfit, genefilter,geneplotter, RColorBrewer, lattice

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

Suggests RUnit, gplots, knitr, Biobase, BiocStyle, parathyroidSE,pasilla (>= 0.2.10), DESeq, vsn, GenomicAlignments,GenomicFeatures, Rsamtools, biomaRt

LinkingTo Rcpp, RcppArmadillo

biocViews Sequencing, ChIPSeq, RNASeq, SAGE, DifferentialExpression

R topics documented:

`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.
`...`	ignored

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

```
example("DESeq")
coef(dds)[1,]
coef(dds, SE=TRUE)[1,]
```

---

collapseReplicates *Collapse replicates in a SummarizedExperiment or DESeqDataSet*

Description

Collapses the columns in `object` by summing within levels of a grouping factor `groupby`. 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 `SummarizedExperiment` 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)

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

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

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)
value	an integer matrix

Author(s)

Simon Anders

See Also

sizeFactors, normalizationFactors

Examples

```
dds <- makeExampleDESeqDataSet()
head(counts(dds))
```

DESeq *Differential expression analysis based on the negative binomial 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)
```

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 (Tikhonov/ridge regularization) 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	the full model formula, this should be the formula in design(object), only used by the likelihood ratio test

`reduced` a reduced formula to compare against, e.g. the full model with a term or terms of interest removed, only used by the likelihood ratio test

`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.

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` (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

DESeq2 reference:

Michael I Love, Wolfgang Huber, Simon Anders: Moderated estimation of fold change and dispersion for RNA-Seq data with DESeq2. bioRxiv preprint (2014) http://dx.doi.org/10.1101/002832

DESeq reference:

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

See Also

nbinomWaldTest, nbinomLRT

Examples

```
dds <- makeExampleDESeqDataSet(betaSD=1, n=100)
dds <- DESeq(dds)
res <- results(dds)
ddsLRT <- DESeq(dds, test="LRT", reduced= ~ 1)
resLRT <- results(ddsLRT)
```

DESeqDataSet-class *DESeqDataSet object and constructors*

Description

The DESeqDataSet is a subclass of SummarizedExperiment, 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 SummarizedExperiment, 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, ignoreRank = FALSE, ...)

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

Arguments

se	a SummarizedExperiment with at least one column in colData, and the counts as the first element in the assays list, which will be renamed "counts". A SummarizedExperiment object can be generated by the function summarizeOverlaps in the GenomicRanges package.
design	a formula which specifies the design of the experiment, taking the form formula(~ x + y + z). By default, the functions in this package will use the last variable in the formula (e.g. z) for presenting results (fold changes, etc.) and plotting.
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.
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
ignoreRank	for advanced use only, allows creation of a DESeqDataSet which is not of full rank
...	arguments provided to SummarizedExperiment including rowData and exptData

Value

A DESeqDataSet object.

References

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

Examples

```
countData <- matrix(1:4,ncol=2)
colData <- data.frame(condition=factor(c("a","b")))
dds <- DESeqDataSetFromMatrix(countData, colData, formula(~ condition))
```

DESeqResults-class *DESeqResults object and constructor*

Description

This class extends the DataFrame class of the IRanges package simply to allow other packages to write methods for results objects from the DESeq2 package.

Usage

```
DESeqResults(DataFrame)
```

Arguments

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

Value

a DESeqResults object

---

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

Description

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

Usage

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

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

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

Arguments

object	a DESeqDataSet object
value	a formula used for estimating dispersion and fitting negative binomial GLMs

Examples

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

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

Arguments

object	a DESeqDataSet object.
value	a function

Examples

```
example("estimateDispersions")
dispersionFunction(dds)
```

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

Description

The dispersions for each row of the DESeqDataSet. Generally, these should be set only 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

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

Arguments

`object`	a `DESeqDataSet` object.
`value`	the dispersions to use for the negative binomial modeling

Author(s)

Simon Anders

See Also

`estimateDispersions`

Examples

```
example("estimateDispersions")
dispersions(dds)
```

`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)

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

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

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, 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, quiet = FALSE)
```

Arguments

`object`	a DESeqDataSet
`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
`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

Value

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

See Also

estimateDispersions

Examples

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

---

estimateSizeFactors *Estimate the size factors for a DESeqDataSet*

Description

Estimate the size factors for a DESeqDataSet

Usage

```
## S4 method for signature DESeqDataSet
estimateSizeFactors(object,locfunc=median,geoMeans)

## S4 method for signature DESeqDataSet
estimateSizeFactors(object, locfunc = median,
  geoMeans)
```

Arguments

object	a DESeqDataSet
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

Details

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

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. You need to call this function after `DESeqDataSet` unless you have manually specified `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

- 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

See Also

`estimateSizeFactorsForMatrix`

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors(dds)
sizeFactors(dds)
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, you will not call this function directly, but use `estimateSizeFactors`.

Usage

```
estimateSizeFactorsForMatrix(counts, locfunc = median, geoMeans)
```

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

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

Note: the kilobase length of the features is calculated from the `rowData` if a column `basepairs` is not present in `mcols(dds)`. This 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. When the read/fragment counting is interfeature dependent, a strict normalization would not incorporate the basepairs of a feature which overlap another feature. This interfeature 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(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))
gr2 <- GRanges("chr1",IRanges(c(1,501),c(500,1000)))
gr3 <- GRanges("chr1",IRanges(c(1,1001),c(1000,2000)))
gr4 <- GRanges("chr1",IRanges(c(1,1001,2001),c(500,3000,3000)))
rowData(dds) <- GRangesList(gr1,gr2,gr3,gr4)

# the raw counts
counts(dds)

# the FPKM values
fpkm(dds)

# held constant per 1 million fragments
counts(dds) <- counts(dds) * 2L
round(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.

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 = 1e3,
                               interceptMean=log2(1e3),
                               interceptSD=0,
                               sizeFactors=c(.5,1,1,2),
                               dispMeanRel=function(x) .01)

# examine the column sums
# then add 1 million counts over rows 11:15 for each sample
colSums(counts(dds))/1e6
counts(dds)[11:15,] <- 2e5L
colSums(counts(dds))/1e6

# the robust FPM treats the samples
# relatively equally
head(fpm(dds), 3)

# the non-robust version is thrown
# off by the 5 rows with large counts
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

# the total sum is equal for both methods
sum(fpm(dds))
sum(fpm(dds, robust=FALSE))
```

`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, modelMatrixType, maxit = 100, useOptim = TRUE,
  quiet = FALSE, useQR = TRUE, betaPriorUpperQuantile = 0.05)
```

Arguments

object	a DESeqDataSet
full	the full model formula, this should be the formula in design(object)
reduced	a reduced formula to compare against, e.g. the full model with a term or terms of interest removed
betaPrior	whether or not to put a zero-mean normal prior on the non-intercept coefficients (Tikhonov/ridge regularization). 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

`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 with 3 or more levels in addition to an intercept, in order to ensure that the log2 fold changes are independent of the choice of base level. 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

`useQR` whether to use the QR decomposition on the design matrix X while fitting the GLM

`betaPriorUpperQuantile` only used when betaPrior=TRUE, which is not the default. the upper quantile to use for calculating the variance of the beta prior. by default the 0.05 upper quantile of the absolute value of the MLE betas is matched to the 0.025 upper quantile of a zero-centered normal.

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 = TRUE, betaPriorVar, modelMatrixType,
  maxit = 100, useOptim = TRUE, quiet = FALSE, useT = FALSE, df,
  useQR = TRUE, betaPriorUpperQuantile = 0.05)
```

Arguments

`object`	a DESeqDataSet
`betaPrior`	whether or not to put a zero-mean normal prior on the non-intercept coefficients (Tikhonov/ridge regularization)
`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
`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
`betaPriorUpperQuantile`	the upper quantile to use for calculating the variance of the beta prior. by default the 0.05 upper quantile of the absolute value of the MLE betas is matched to the 0.025 upper quantile of a zero-centered normal.

Details

The fitting proceeds as follows: standard maximum likelihood estimates for GLM coefficients (synonymous with beta, log2 fold change) are calculated; a zero-mean normal prior distribution is assumed; the variance of the prior distribution for each non-intercept coefficient is calculated by matching the upper quantile of the MLE coefficients with a zero-centered normal distribution; the final coefficients are then maximum a posteriori estimates (using Tikhonov/ridge regularization) using this prior. 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 normal distribution. From examination of Wald statistics for real datasets, the effect of the prior on dispersion estimates results in a Wald statistic distribution which is approximately normal.

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 or unless there the design contains 2 level factors and an interaction term. This ensures that log2 fold changes will be independent of the choice of base 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.

When interaction terms are present, the prior on log fold changes the calculated beta prior variance will only be used for the interaction terms (non-interaction log2 fold changes receive a prior variance of 1e3).

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

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

Arguments

object	a DESeqDataSet object.
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 size factor normalization and should have a mean around 1, as is the case with size factors.

Examples

```
dds <- makeExampleDESeqDataSet()
normFactors <- matrix(runif(nrow(dds)*ncol(dds),0.5,1.5),
                      ncol=ncol(dds),nrow=nrow(dds))
normalizationFactors(dds) <- normFactors
dds <- estimateDispersions(dds)
dds <- nbinomWaldTest(dds)
```

`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, ...)

## 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
`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 DESeqResults
plotMA(object, alpha, main, ylim, ...)
## S4 method for signature DESeqDataSet
plotMA(object, alpha, main, ylim, ...)

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

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

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

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

Value

A `trellis` object.

Author(s)

Michael Love

Examples

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

`plotPCA` *Sample PCA plot from variance-stabilized data*

Description

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

Usage

```
plotPCA(x, intgroup = "condition", ntop = 500, col)
```

Arguments

`x`	a SummarizedExperiment, with data in `assay(x)`, produced for example by either `varianceStabilizingTransformation` or `rlogTransformation`
`intgroup`	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
`col`	a vector of colors for each level of intgroup

Value

A `trellis` object.

Note

See the vignette for an example of variance stabilization and PCA plots.

Author(s)

Wolfgang Huber

Examples

```
dds = makeExampleDESeqDataSet(betaSD=1)
vsd = varianceStabilizingTransformation(dds)
p = plotPCA(vsd)
print(p)

## Add text labels (for presentation graphics, consider additional
## layout operations that avoid overplotting, such as the FField package on CRAN)
names = colData(vsd)$sample

p = update(p, panel = function(x, y, ...) {
      lattice::panel.xyplot(x, y, ...);
      lattice::ltext(x=x, y=y, labels=names, pos=1, offset=1, cex=0.8)
    })
print(p)
```

`replaceOutliers` *Replace outliers with trimmed mean*

Description

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"]]`.

Usage

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

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

Arguments

`dds`	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

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

Examples

```
dds <- makeExampleDESeqDataSet(n=100)
dds <- DESeq(dds)
ddsReplace <- replaceOutliers(dds)
```

---

results *Extract results from a DESeq analysis*

Description

results extracts results 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")

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 log2 fold change, and the name of the denominator level for the log2 fold change (most simple case) • a list of two character vectors: the names of the effects for the numerator, and the names of the effects for denominator. these names should be elements of resultsNames(object). one list element can be the empty 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 - $\lvert\beta\rvert >$ 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, giving the values to assign to the first and second elements of the list, which should be positive and negative, respectively, to specify the numerator and denominator. 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 whose removal would result in rank deficient design matrix and 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

`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`

Details

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 comparison of interest for printing the results table. 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 base 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 mcol on the DESeqResults object returned by results.

By default, independent filtering is performed to select a set of genes for multiple test correction which will optimize the number of adjusted p-values less than a given critical value alpha (by default 0.1). 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.

In addition, results by default 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.

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

```
# minimal example with simple two-group comparison
example("DESeq")
results(dds)
resultsNames(dds)
dds <- removeResults(dds)

# two conditions, two groups, with interaction term
dds <- makeExampleDESeqDataSet(n=100,m=12)
dds$group <- factor(rep(rep(c("X","Y"),each=3),2))
design(dds) <- ~ group + condition + group:condition
dds <- DESeq(dds)
```

```
resultsNames(dds)
results(dds, contrast=c("condition","B","A"))
results(dds, contrast=c("group","Y","X"))
# extract the interaction term simply with name
results(dds, name="groupY.conditionB")

# two conditions, three groups, with interaction term
dds <- makeExampleDESeqDataSet(n=100,m=18)
dds$group <- factor(rep(rep(c("X","Y","Z"),each=3),2))
design(dds) <- ~ group + condition + group:condition
dds <- DESeq(dds)
resultsNames(dds)

# results tables for various comparisons:

# the condition effect over all groups
results(dds, contrast=c("condition","B","A"))
# which is equivalent to
results(dds, contrast=list("conditionB","conditionA"))
# which is equivalent to
results(dds, contrast=c(0, 0,0,0, -1,1, 0,0,0, 0,0,0))

# the group Z effect compared to the average of group X and Y
# here we use listValues to multiply group X and
# group Y by -1/2 in the numeric contrast
results(dds, contrast=list("groupZ",c("groupX","groupY")), listValues=c(1,-1/2))

# the individual effect for group Z, compared to the intercept
results(dds, name="groupZ")

# the interaction effect of condition for group Z.
# if this term is non-zero, then group Z has a
# different condition effect than the overall condition effect
results(dds, contrast=list("groupZ.conditionB","groupZ.conditionA"))

# the condition effect for group Z:
# this is the sum of the main effect for condition
# and the interaction effect for group Z
results(dds, contrast=list(
               c("conditionB","groupZ.conditionB"),
               c("conditionA","groupZ.conditionA")))
```

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

Description

This function uses Tikhonov/ridge regularization to transform the data to the log2 scale in a way which minimizes differences between samples for rows with small counts. The rlog transformation produces a similar variance stabilizing effect as `varianceStabilizingTransformation`, though

rlogTransformation 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. Note: rlog is equivalent to rlogTransformation, which take as input a DESeqDataSet and return a SummarizedExperiment object.

Usage

```
rlog(object, blind = TRUE, fast = FALSE, intercept, betaPriorVar, B)

rlogTransformation(object, blind = TRUE, fast = FALSE, intercept,
  betaPriorVar, B)

rlogData(object, intercept, betaPriorVar)

rlogDataFast(object, intercept, betaPriorVar, B)
```

Arguments

object	a DESeqDataSet
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. If many of genes have large differences in counts due to the experimental design, it is important to set blind=FALSE for downstream analysis.
fast	if TRUE, an approximation to the rlog transformation is made, wherein an optimal amount of shrinkage of sample-wise log fold changes is calculated for genes divided into 12 batched based on mean counts. then shrinkage is performed by interpolating between these optimal values. optimal is defined by maximizing the same posterior as in the standard rlog transformation.
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.
betaPriorVar	a single value, the variance of the prior on the sample betas, which if missing is estimated from the data
B	for the fast method only. 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. this is the amount of shrinkage from 0 to 1 for each row, and takes precedence over internal calculation of B using betaPriorVar.

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 'regularization' referred to here corresponds to the maximum a posteriori solution to the GLM with a prior on the coefficients for each sample. The fitted dispersions are used rather than the MAP dispersions (so similar to the varianceStabilizingTransformation) as the blind dispersion estimation would otherwise shrink large, true log fold changes. The prior variance 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 as with nbinomWaldTest, by matching the upper quantiles of the observed log fold change estimates with an upper quantile of the normal distribution. A second and final GLM fit is then performed 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 $rlog(K_{ij}) = \log_2(q_{ij}) = x_{j\cdot}\beta_i$, 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 "freezing" is accomplished by saving the dispersion function, beta prior variance and the intercept from a previous dataset, and running rlogTransformation with 'blind' set to FALSE (see example below).

Value

rlog, equivalent to rlogTransformation, returns a SummarizedExperiment. The matrix of transformed values are accessed by assay(rld). for rlogData, a matrix of the same dimension as the count data, containing the transformed values. To avoid returning matrices with NA values where there were zeros for all rows of the unnormalized counts, rlogTransformation returns instead all zeros (essentially adding a pseudocount of one, only to those rows in which all samples have zero).

See Also

plotPCA, varianceStabilizingTransformation

Examples

```
dds <- makeExampleDESeqDataSet(m=6,betaSD=1)
rld <- rlog(dds, blind=TRUE)
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. 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

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

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

Examples

```
dds <- makeExampleDESeqDataSet()
dds <- estimateSizeFactors( dds )
sizeFactors(dds)
```

---

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 rlogTransformation is less sensitive to size factors, which can be an issue when size factors vary widely. This transformation is useful when checking for outliers or as input for machine learning techniques such as clustering or linear discriminant analysis.

Usage

```
varianceStabilizingTransformation(object, blind = TRUE)

getVarianceStabilizedData(object)
```

Arguments

object	a DESeqDataSet, with design(object) <- formula(~ 1) and size factors (or normalization factors) and dispersions estimated using local or parametric fitType.

`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. If many of genes have large differences in counts due to the experimental design, it is important to set blind=FALSE for downstream analysis.

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.

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.

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.

If `estimateDispersions` was called with `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.

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

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 "freezing" 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). See `estimateSizeFactors` for details on how to "freeze" size factor estimation.

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 you will find in the transformed data. As shown in the vignette, you can use the function `meanSdPlot` from the package **vsn** to see whether this is a problem for your data.

Value

for `varianceStabilizingTransformation`, a `SummarizedExperiment`. The matrix of transformed values are accessed by `assay(vsd)`. for `getVarianceStabilizedData`, a `matrix` of the same dimension as the count data, containing the transformed values.

Author(s)

Simon Anders

See Also

plotPCA, rlogTransformation

Examples

```
dds <- makeExampleDESeqDataSet(m=6)
vsd <- varianceStabilizingTransformation(dds, blind=TRUE)
par(mfrow=c(1,2))
plot(rank(rowMeans(counts(dds))), genefilter::rowVars(log2(counts(dds)+1)),
     main="log2(x+1) transform")
plot(rank(rowMeans(assay(vsd))), genefilter::rowVars(assay(vsd)),
     main="VST")

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

Index