Package 'DESeq2'

April 5, 2014

Type Package

Title Differential gene expression analysis based on the negative binomial distribution

Version 1.2.10

Author Michael Love (MPIMG Berlin), 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)

biocViews HighThroughputSequencing, ChIPseq, RNAseq, SAGE,DifferentialExpression

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

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

Suggests RUnit, Biobase, parathyroidSE, pasilla (>= 0.2.10), vsn,gplots, BiocStyle

LinkingTo Rcpp, RcppArmadillo

Collate 'AllClasses.R' 'AllGenerics.R' 'core.R' 'methods.R' 'plots.R'
'rlogTransformation.R' 'varianceStabilizingTransformation.R'

R topics documented:

`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
```

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 by calling, in order, the functions: `estimateSizeFactors`, `estimateDispersions`, `nbinomWaldTest`.

Usage

```
  DESeq(object, test = c("Wald", "LRT"),
    fitType = c("parametric", "local", "mean"),
    betaPrior = TRUE, full = design(object), reduced,
    quiet = FALSE)
```

Arguments

`object` a DESeqDataSet object, see the constructor functions `DESeqDataSet`, `DESeqDataSetFromMatrix`, `DESeqDataSetFromHTSeqCount`.

`test` either "Wald" or "LRT", which will then use either Wald tests if coefficients are equal to zero (`nbinomWaldTest`), or the likelihood ratio test on the difference in deviance between a full and reduced model formula (`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. Only used for the Wald test.

`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

`quiet` whether to print messages at each step

Details

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

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

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

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

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"`).

Value

a `DESeqDataSet` object with results stored as metadata columns. The results can be 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

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

`DESeqDataSet` *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)

DESeqDataSetFromMatrix(countData, colData, design, ...)

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

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

`...` 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))
```

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
```

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

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

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

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

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

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

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)
```

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 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 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.

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

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, 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 decade 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

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)
```

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 and stores the information which can be accessed 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

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)
```

---

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,
  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)`
`reduced`	a reduced formula to compare against, e.g. the full model with a term or terms of interest removed
`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

`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,
    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 (Tikhonov/ridge regularization)

`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

`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 are calculated; a zero-mean normal prior distribution is assumed; the variance of the prior distribution for each non-intercept coefficient is calculated as the mean squared maximum likelihood estimates over the genes which do not contain zeros for some condition; 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.

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

`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

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

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

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, ...)
```

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 DESeqDataSet
plotMA(object, lfcColname,
    pvalues, pvalCutoff=.1, ylim, linecol = ”#ff000080”,
    pointcol = c(”black”,”red”), xlab, ylab, log = ”x”,
    cex=0.45, ...)
```

Arguments

`object` a DESeqDataSet processed by `DESeq`, or the individual functions `nbinomWaldTest` or `nbinomLRT`

`lfcColname` the name of the column for log fold changes, if not provided this will default to the last variable in the design formula. for options for this argument, check resultsNames(object).

`pvalues` a vector of the p-values or adjusted p-values to use in coloring the points. If not provided, defaults to the 'padj' column of results(object)

`pvalCutoff` the cutoff for drawing red or black points

`ylim` the limits for the y axis (chosen automatically if not specified)

`linecol` the color of the horizontal line

`pointcol` a vector of length two of the colors for the not significant and significant points, respectively

`xlab` the label for the x axis

`ylab` the label for the y axis

`log` log, defaults to "x", the y-axis is already in log scale

`cex` the size of the points

`...` further arguments to `plot`

Author(s)

Wolfgang Huber

Examples

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

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)
```

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

Note

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

Author(s)

Wolfgang Huber

Examples

```
dds <- makeExampleDESeqDataSet(betaSD=1)
design(dds) <- formula(~ 1)
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
vsd <- varianceStabilizingTransformation(dds)
plotPCA(vsd)
```

replaceOutliersWithTrimmedMean *Replace outliers with trimmed mean*

Description

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 more information on Cook's distance, please see the two sections of the vignette: 'Dealing with count outliers' and 'Count outlier detection'.

Usage

```
replaceOutliersWithTrimmedMean(dds, trim = 0.2,
  cooksCutoff)
```

Arguments

`dds` a DESeqDataSet object, which has already been processed by either DESeq, nbinomWaldTest or nbinomLRT, and therefore contains a matrix 'cooks' contained in assays(dds). 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.

See Also

DESeq

Examples

```
dds <- makeExampleDESeqDataSet(n=100)
dds <- DESeq(dds)
ddsReplace <- replaceOutliersWithTrimmedMean(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` finds available names for results; `removeResults` returns a `DESeqDataSet` object with results columns removed.

Usage

```
results(object, name, contrast, cooksCutoff,
  independentFiltering = TRUE, alpha = 0.1, filter,
  theta = seq(0, 0.95, by = 0.05), 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`

`name` the name of the coefficient for which to report log2 fold changes – and for the Wald test, p-values and adjusted p-values

`contrast` either a character vector with exactly three elements (name of the factor, name of the numerator level, name of the denominator level), or a numeric contrast vector with one element for each element in `resultsNames(object)`, i.e. columns of the model matrix. The DESeqDataSet must be one produced using the Wald test steps in order to use contrasts.

`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 an analysis with multifactor design, so `results` takes an argument `name` as well as the argument `contrast` explained below.

The available names can be checked using `resultsNames`; these are combined variable names and factor levels, potentially with minor changes made by the `make.names` function on column names (e.g. dashes into periods).

By default, results for the last variable will be returned. Information on the variable represented and the test used for p-values (Wald test or likelihood ratio test) is stored in the metadata columns, accessible by calling `mcol` on the `DataFrame` returned by `results`.

By default, independent filtering is performed to select a set of genes which will result in the most genes with adjusted p-values less than a threshold, alpha. 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 to control or turn off the independent filtering behavior.

A contrast can be performed by specifying the variable and factor levels which should be compared, or by specifying the numeric contrast vector. The test statistic is then:

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

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. In this case, the `name` argument only specifies which coefficient should be used for reporting the log2 fold changes.

Cook's distance 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.

Results can be removed from an object by calling `removeResults`

Value

For `results`: a DataFrame of results columns with metadata columns of coefficient and test 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 object 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("DESeq")
results(dds)
resultsNames(dds)
dds <- removeResults(dds)
```

rlogTransformation *Apply a 'regularized log' transformation*

Description

This function uses Tikhonov/ridge regularization, as in `nbinomWaldTest`, to transform the data to the log2 scale in a way which minimizes differences between samples for rows with small counts. The 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.

Usage

```
rlogTransformation(object, blind = TRUE, samplesVector,
  betaPriorVar, intercept)

rlogData(object, samplesVector, betaPriorVar, intercept)
```

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.

`samplesVector` a character vector or factor of the sample identifiers

`betaPriorVar` a single value, the variance of the prior on the sample betas, which if missing is estimated from the data

`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.

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 row means of these log counts are then subtracted, leaving an estimate of the log fold changes per sample. The prior variance is set to the variance of all log fold change estimates. 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 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

for `rlogTransformation`, a SummarizedExperiment with assay data elements equal to $\log_2(q_{ij}) = x_{j.}\beta_i$, see formula at `DESeq`. 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(betaSD=1)
rld <- rlogTransformation(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 <- rlogTransformation(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 <- rlogTransformation(ddsNew, blind=FALSE,
                             betaPriorVar=betaPriorVar,
                             intercept=intercept)
```

---

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
```

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.

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`. 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()
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