DESeq

March 24, 2012

`CountDataSet-class` *Class "CountDataSet" - a container for count data from HTS experiments*

Description

This is the main class for the present package.

Objects from the Class

Objects should be created with calls to `newCountDataSet` (q.v.).

Extends

Class `eSet` (package 'Biobase'), directly. Class `VersionedBiobase` (package 'Biobase'), by class "eSet", distance 2. Class `Versioned` (package 'Biobase'), by class "eSet", distance 3.

Note

Note: This is a summary for reference. For an explanation of the actual usage, see the vignette.

A CountDataSet object stores counts from an HTS data set and offers further slots which are populated during the analysis.

After creation with `newCountDataSet`, a CountDataSet typically contains a count table, i.e., a matrix of integer data, that is accessible with the accessor function `counts`. Each row of the matrix corresponds to a gene (or binding region, or the like), and each colum to an experimental sample. The experimental conditions of the samples are stored in a factor (with one element for each row of the counts matrix), which can be read with the accessor function `conditions`.

In the following analysis steps, further data slots are populated. First, the size factors can be estimated with `estimateSizeFactors`, which are afterwards accessible via `sizeFactors`. Then, the dispersions (variance fits) are estimated with `estimateDispersions`. The resulting estimates are stored in phenoData columns, accessible via `pData`, with the column names staring with `disp_`. The intermediate steps of the fit are stored in the environment-values slot `fitInfo` (see `estimateDispersions` for details).

Internally, the mentioned data is stored in slots as follows:

As `CountDataSet` is derived from `eSet`, it has a `phenoData` slot which allows to store sample annotation. This is used to store the factor with the conditions, as a data frame column named `condition`, and to store the size factors, as an numeric data frame column named `sizeFactor`.

If the user creates an object with multivariate design, i.e., passes a data frame instead of a factor for conditions, this data frame's columns are placed in the phenoData slot instead of the condition column. Furthermore, the function estimateDispersions adds columns with the dispersion values to be used by nbinomTest and fitNbinomGLMs. These columns have names starting with disp_.

The user may add further columns to the phenoData AnnotatedDataFrame.

The counts table is stored in the eSet's assayData locked environment with the name counts.

The slot dispInfo is an environment containing lists, one for each set of estimated dispersion values and the slot dispTable (with accessor dispTable shows the assignment of conditions to dispersion estimates. See estimateDispersions

Examples

```
# See the vignette
```

---

adjustScvForBias *Adjust an SCV value for the bias arising when it is calculated from unbiased estimates of mean and variance.*

Description

Assume that a small sample of i.i.d. random variables from a negative binomial distribution is given, and you have obtained unbiased estimates of mean and raw variance. Then, a new bias is introduced when the squared coefficient of variation (SCV, a.k.a. dispersion) is calculated from these unbiased estimates by dividing the raw variance by the square of the mean. This bias can be calculated by numerical simulation and a pre-calculated adjustment table (or rather a fit through tabulated values) is supplied with the package. The present function uses this to remove the bias from a raw SCV estimate.

This function is used internally in nbinomTest. You will rarely need to call it directly.

Usage

```
adjustScvForBias(scv, nsamples)
```

Arguments

scv	An estimate for the raw squared coefficient of variation (SCV) for negative binomially distributed data, which has been obtained by dividing an unbiased estimate of the raw variance by the square of an unbiased estimate of the mean.
nsamples	The size of the sample used in the estimation.

Value

an unbiased estimate of the raw SCV

Author(s)

Simon Anders

Examples

```
true_mean <- 100
true_scv <- .1
nsamples <- 3
res <- replicate( 1000, {
   mySample <- rnbinom( nsamples, mu=true_mean, size=1/true_scv )
   mu_est <- mean( mySample )
   raw_var_est <- var( mySample ) - mean( mySample )
   raw_scv_est <- raw_var_est / mu_est^2
   unbiased_raw_scv_est <- adjustScvForBias( raw_scv_est, 4 )
   c( raw_scv_est = raw_scv_est, unbiased_raw_scv_est = unbiased_raw_scv_est ) } )
rowMeans( res )
```

---

conditions *Accessor functions for the 'conditions' information in a CountDataSet object.*

Description

The conditions vector is a factor that assigns to each column of the count data a condition (or treatment, or phenotype, or the like). This information is stored in the CountDataSet's "phenoData" slot as a row named "condition".

Usage

```
## S4 method for signature 'CountDataSet'
conditions(object, ...)
## S4 replacement method for signature 'CountDataSet'
conditions(object) <- value
```

Arguments

`object`	a CountDataSet
`value`	a vector of suitable length, i.e. with as many elements as `object` has samples.
`...`	should not be used for this method.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
conditions( cds )
```

counts — *Accessors for the 'counts' slot of a CountDataSet 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 'CountDataSet'
counts(object, normalized=FALSE)
## S4 replacement method for signature 'CountDataSet,matrix'
counts(object) <- value
```

Arguments

object	a CountDataSet object.
normalized	logical indicating whether or not to divide the counts by the size factors before returning.
value	integer matrix.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
head( counts( cds ) )
```

dispTable — *Accessor function for the dispTable information in a CountDataSet*

Description

The dispersion table ("dispTable") is a named vector that assigns to each condition (as name) a dispersion column (as value). If nbinomTest is called to compare two conditions, say "A" and "B", DESeq looks up in the dispTable, which dispersion columns to use. In the standard case (see example), these are just the dispersions for "A" and "B", i.e., the columns disp_A and disp_B in fData(object). If the "pooled" or "blind" variance estimation is used, all conditions are assigned the same column.

Usage

```
dispTable(object,...)
```

Arguments

object	a CountDataSet
...	further argumnts are ignored

Author(s)

Simon Anders, sanders@fs.tum.de

See Also

estimateDispersions, nbinomTest

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds )
dispTable( cds )
```

---

estimateDispersions

Estimate and fit dispersions for a CountDataSet.

Description

This function obtains dispersion estimates for a count data set. For each condition (or collectively for all conditions, see 'method' argument below) it first computes for each gene an empirical dispersion value (a.k.a. a raw SCV value), then fits by regression a dispersion-mean relationship and finally chooses for each gene a dispersion parameter that will be used in subsequent tests from the empirical and the fitted value according to the 'sharingMode' argument.

Usage

```
## S4 method for signature 'CountDataSet'
estimateDispersions( object,
   method = c( "pooled", "per-condition", "blind" ),
   sharingMode = c( "maximum", "fit-only", "gene-est-only" ),
   fitType = c("parametric", "local"),
   locfit_extra_args=list(), lp_extra_args=list(),
   modelFrame = NULL, modelFormula = count ~ condition, ... )
```

Arguments

object	a CountDataSet with size factors.
method	There are three ways how the empirical dispersion can be computed: • pooled - Use the samples from all conditions with replicates to estimate a single pooled empirical dispersion value, called "pooled", and assign it to all samples.

- `per-condition` - For each condition with replicates, compute a gene's empirical dispersion value by considering the data from samples for this condition. For samples of unreplicated conditions, the maximum of empirical dispersion values from the other conditions is used. If `object` has a multivariate design (i.e., if a data frame was passed instead of a factor for the `condition` argument in `newCountDataSet`), this method is not available. (Note: This method was called "normal" in previous versions.)
- `blind` - Ignore the sample labels and compute a gene's empirical dispersion value as if all samples were replicates of a single condition. This can be done even if there are no biological replicates. This method can lead to loss of power; see the vignette for details. The single estimated dispersion condition is called "blind" and used for all samples.

`sharingMode` After the empirical dispersion values have been computed for each gene, a dispersion-mean relationship is fitted for sharing information across genes in order to reduce variability of the dispersion estimates. After that, for each gene, we have two values: the empirical value (derived only from this gene's data), and the fitted value (i.e., the dispersion value typical for genes with an average expression similar to those of this gene). The `sharingMode` argument specifies which of these two values will be written to the `featureData`'s `disp_` columns and hence will be used by the functions `nbinomTest` and `fitNbinomGLMs`.

- `fit-only` - use only the fitted value, i.e., the empirical value is used only as input to the fitting, and then ignored. Use this only with very *few* replicates, and when you are not too concerned about false positives from dispersion outliers, i.e. genes with an unusually high variability.
- `maximum` - take the maximum of the two values. This is the conservative or prudent choice, recommended once you have at least three or four replicates and maybe even with only two replicates.
- `gene-est-only` - No fitting or sharing, use only the empirical value. This method is preferable when the number of replicates is large and the empirical dispersion values are sufficiently reliable. If the number of replicates is small, this option may lead to many cases where the dispersion of a gene is accidentally underestimated and a false positive arises in the subsequent testing.

`fitType`

- `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 `dispFunc` in the `fitInfo` (see below).
- `local` - Use the locfit package to fit a dispersion-mean relation, as described in the DESeq paper.

`locfit_extra_args, lp_extra_args`
(only for `fitType=local`) Options to be passed to the `locfit` and to the `lp` function of the locfit package. Use this to adjust the local fitting. For example, you may pass a value for `nn` different from the default (0.7) if the fit seems too smooth or too rough by setting `lp_extra_agrs=list(nn=0.9)`. As another example, you can set `locfit_extra_args=list(maxk=200)` if you get the error that locfit ran out of nodes. See the documentation of the `locfit` package for details. In most cases, you will not need to provide these parameters, as the defaults seem to work quite well.

`modelFrame` By default, the information in `conditions(object)` or `pData(object)` is used to determine which samples are replicates (see `newCountDataSet`).

For `method="pooled"`, a data frame can be passed here, and all rows that are identical in this data frame are considered to indicate replicate samples in `object`. For `method="pooled-CR"`, the data frame is used in the fits. For the other methods, this argument is ignored.

`modelFormula` For `method="pooled-CR"`, this is the formual used for the dispersion fits. For all other methods, this argument is ignored.

`...` extra arguments are ignored

Details

Behaviour for `method="per-condition"`: For each replicated condition, a list, named with the condition's name, is placed in the environment `object@fitInfo`. This list has five named elements: The vector `perGeneDispEsts` contains the empirical dispersions. The function `dispFunc` is the fitted function, i.e., it takes as its argument a normalized mean expression value and returns the corresponding fitted dispersion. The values fitted according to this function are in the third element `fittedDispEst`, a vector of the same length as `perGeneDispEsts`. The fourt element, `df`, is an integer, indicating the number of degrees of freedom of the per-gene estimation. The fifth element, `sharingMode`, stores the value of the `sharingMode` argument to `esimateDispersions`.

Behaviour for `method="blind"` and `method="pooled"`: Only one list is produced, named `"blind"` or `"pooled"` and placed in `object@fitInfo`.

For each list in the `fitInfo` environment, the dispersion values that are intended to be used in subsequent testing are computed according to the value of `sharingMode` and are placed in the `featureData` data frame, in a column named with the same name, prefixed with `"disp_"`.

Then, the `dispTable` (see there) is filled to assign to each condition the appropriate dispersion column in the phenoData frame.

Note: Up to DESeq version 1.4.x (Bioconductor release 2.8), this function was called `estimateVarianceFunction` stored its result differently and had did not yet have the arguments `sharingMode` and `fitType`. `estimatevarianceFunction`'s behaviour corresponded to the settings `sharingMode="fit-only"` and `fitType="local"`. Note that these are not the default, because the new functionalities sharingMode="maximum" and fitType="local" tend to give better results in standard cases as they are more robust

Value

The `CountDataSet` cds, with the slots `fitInfo` and `featureData` updated as described in Details.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds )
str( fitInfo( cds ) )
head( fData( cds ) )
```

`estimateSizeFactors` *Estimate the size factors for a CountDataSet*

Description

Estimate the size factors for a CountDataSet

Usage

```
## S4 method for signature 'CountDataSet'
estimateSizeFactors( object, locfunc=median, ... )
```

Arguments

`object`	a CountDataSet
`locfunc`	a function to compute a location for a sample. By default, the median is used. However, especially for low counts, the `shorth` may give better results.
`...`	extra arguments are ignored

Details

You need to call this function right after `newCountDataSet` unless you have manually specified size factors.

Typically, the function is called with the idiom

```
cds <- estimateSizeFactors( cds )
```

This estimates the size factors and stores the information in the object.

Internally, the function calls `estimateSizeFactorsForMatrix`. See there for more details on the calculation.

Value

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

Author(s)

Simon Anders, sanders@fs.tum.de

See Also

`estimateSizeFactorsForMatrix`

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
sizeFactors( cds )
```

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, ir 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)
```

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` may give better results.

Value

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

Author(s)

Simon Anders, sanders@fs.tum.de

See Also

`estimateSizeFactors`

Examples

```
cds <- makeExampleCountDataSet()
estimateSizeFactorsForMatrix( counts(cds) )
```

estimateVarianceFunctions

REMOVED

Description

This function has been removed. Instead, use `estimateDispersions`.

`fitInfo` *Accessor function for the fitInfo objects in a CountDataSet*

Description

After calling `estimateDispersions`, a CountDataSet object is populated with one or (in case of a "per-condition" estimation) several fitInfo objects, which can be accessed with this function.

Usage

```
fitInfo( cds, name=NULL )
```

Arguments

`cds`	a CountDataSet
`name`	if `estimateDispersion` was called with `method="per-condition"` a name hasd to specified. Try `ls(cds@fitInfo`.

Author(s)

Simon Anders, sanders@fs.tum.de

See Also

`estimateDispersions`

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds )
str( fitInfo( cds ) )
```

`fitNbinomGLMs` *Fit a generalized linear model (GLM) for each gene.*

Description

Use this function to estimate coefficients and calculate deviance from a GLM for each gene. The GLM uses the `nbkd.sf` family, with the dispersion estimate according to `getVarianceFunction(cds)`. Note that this requires that the variance functions were estimated with method "pooled" or "blind".

Usage

```
fitNbinomGLMs( cds, modelFormula, glmControl=list() )
```

Arguments

`cds`	a CountDataSet
`modelFormula`	a formula. The left hand side must be 'count' (not 'counts'!), the right hand side can involve any column of `pData(cds)`, i.e., `pData(cds)` is used as the model frame. If you have passed just a single factor to the 'conditions' argument of `newCountDataSet`, it can be referred to as 'condition' in the formula. If you have passed a data frame to 'conditions', all columns of this data frame will be available.
`glmControl`	list of additional parameters to be passed to `glm.control`

Value

A data frame with one row for each gene and columns as follows:

- one column for each estimated coefficient, on a log2 scale (i.e., the natural log reported by `glm` is rescaled to base 2)
- a column 'deviance', with the deviance of the fit
- a boolean column 'converged', indicating whether the fit converged

Furthermore, the data frame has a scalar attribute 'df.residual' that contains the number of residual degrees of freedom.

Author(s)

Simon Anders (sanders@fs.tum.de)

See Also

`newCountDataSet`,`nbinomGLMTest`, `nbkd.sf`

Examples

```
# see nbinomGLMTest for an example
```

`fitNbinomGLMsForMatrix` *Fit negative binomial GLMs to a count matrix.*

Description

This is a low-level function that is wrapped by `nbinomGLMTest`.

Usage

```
fitNbinomGLMsForMatrix(counts, sizeFactors, rawScv, modelFormula,
   modelFrame, quiet = FALSE, reportLog2 = TRUE, glmControl = list() )
```

Arguments

counts	a matrix of integer counts. Rows for genes, Columns for samples.
sizeFactors	a vector with a size factor for each column in 'counts'.
rawScv	a vector with a raw SCV (i.e., a dispersion) for each row in 'counts'.
modelFormula	a model formula. The left hand side should read 'count ~'.
modelFrame	a model frame (with one row for each column in 'counts')
quiet	whether to not print dots
reportLog2	whether to convert reported coefficients from natural log to log2 scale
glmControl	list of additional parameters to be passed to glm.control

Value

A data frame with one row for each gene and columns as follows:

- one column for each estimated coefficient, on a log2 scale (i.e., the natural log reported by glm is rescaled to base 2)
- a column 'deviance', with the deviance of the fit
- a boolean column 'converged', indicating whether the fit converged

Furthermore, the data frame has a scalar attribute 'df.residual' that contains the number of residual degrees of freedom.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
# See the code of fitNbinomGLMs for an example.
```

---

getBaseMeansAndVariances

Perform row-wise estimates of base-level means and variances for count data.

Description

This function is called internally by a number of other functions. You will need to call it directly only in very special cases.

Usage

```
getBaseMeansAndVariances(counts, sizeFactors)
```

Arguments

counts	a matrix of data frame of count data. All the columns of this matrix will be considered as replicates of the same condition.
sizeFactors	the size factors of the columns, as estimated e.g. with estimateSizeFactorsForMatrix

Details

This function is kept for backwards compatibility. See the example below for an alternative and more self-explanatory way to get the same data.

Value

A data frame with one row for each row in 'counts' and two columns:

`baseMean`	The base mean for each row. This is the mean of the counts after they have been divided by the size factors
`comp2`	The base variance for each row. This is the variance of the counts after they have been divided by the size factors

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
head( getBaseMeansAndVariances( counts(cds), sizeFactors(cds) ) )

# You can get the same as follows
head( rowMeans( counts( cds, normalized=TRUE ) ) )
head( genefilter::rowVars( counts( cds, normalized=TRUE ) ) )
```

`getVarianceStabilizedData` *Perform a variance stabilising transformation (VST) on the count data*

Description

This function calculates a variance stabilising transformations (VST) from the fitted dispersion-mean reltions and then transforms the count data (after normalization by division by the size factor), yielding a matrix of values which are now approximately homoskedastic. This is useful as input to statistical analyses requiring homoskedasticity.

Usage

```
getVarianceStabilizedData(cds)
```

Arguments

`cds`	a CountDataSet with estimated variance functions

Details

For each sample (i.e., column of `counts(cds)`), the full variance function is calculated from the raw variance (by scaling according to the size factor and adding the shot noise). The function requires a blind estimate of the variance function, i.e., one ignoring conditions, and hence, you need to call `estimateDispersions` with `method="blind"` before calling it.

If `estimateDispersions` was called with `fitType="parametric"`, the following variance stabilizing transformation is used on the normalized count data, using the coefficients `asymptDisp` and `extraPois` from the dispersion fit:

```
vst( q ) = 2/log(2) * log( asymptDisp * sqrt(q) + asymptDisp * sqrt(
1 + extraPois + asymptDisp * q ) )
```

This transformation is scaled such that for large counts, it become asymptotically equal to log2.

If `estimateDispersions` was called with `fitType="locfit"`, the reciprocal of the square root of the variance of the normalized counts, as derived from the dispersion fit, is then numerically integrated up, and the integral (approximated by a spline function) evaluated for each count value in the column, yielding a transformed value. Note that the asymptotic property mentioned above does not hold in this case.

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. (Compare the variance of the upper half of your transformed data with the lower half to see whether this is a problem in your case.)

Value

A matrix of the same dimension as the count data, containing the transformed data.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds, method="blind" )
vsd <- getVarianceStabilizedData( cds )
colsA <- conditions(cds) == "A"
plot( rank( rowMeans( vsd[,colsA] ) ), genefilter::rowVars( vsd[,colsA] ) )
```

---

makeExampleCountDataSet

make a simple example CountDataSet with random data

Description

This function returns an example CountDataSet. It is used for the examples in the package help pages.

Usage

```
makeExampleCountDataSet()
```

Value

a CountDataSet that has been constructed as follows: First, true base mean values for 10,000 genes are drawn from an exponential distribution with rate 1/250. Then, certain genes are declared (with probability 0.3 per gene) as truly differentially expressed (tDE). For these genes, the true base mean is split into two values, one for condition "A" and one for condition "B", such that the log2 fold change from "A" to "B" follows a zero-centred normal distribution with standard deviation 2. Then, counts are drawn for each gene for 5 samples, the first three corresponding to condition "A" and the remaining two for condition "B". The counts are drawn from a negative binomial with the specified mean, multiplied by the size factor for the sample, with a constant raw SCV (dispersion) of 0.2 (i.e., a 'size' parameter of 1/0.2). The true size factors are fixed to c(1., 1.3, .7, .9, 1.6).

All these values were chosen to give data that at least somewhat resembles what one might encounter in an actual experiment. Note that this function is not meant to verify the package by simulation. For this purpose the parameters and distribution choices should be more varied.

Author(s)

Simon Anders, anders@embl.de

Examples

```
cds <- makeExampleCountDataSet()
```

---

nbinomGLMTest *Perform chi-squared tests comparing two sets of GLM fits*

Description

For each gene, the function calculates a chi-square p value by simply calculating: `1 - pchisq(resReduced$devia - resFull$deviance, attr(resReduced, "df.residual") - attr(resFull, "df.residual"))`

Usage

```
nbinomGLMTest(resFull, resReduced)
```

Arguments

`resFull, resReduced`
: GLM fit data frames, as returned by `fitNbinomGLMs`, first the full, then the reduced model.

Value

a vector of p values

Author(s)

Simon Anders, anders@embl.de

See Also

fitNbinomGLMs

Examples

```
cds <- makeExampleCountDataSet()[ 1:100, ]
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds, method="pooled" )
fit1 <- fitNbinomGLMs( cds, count ~ condition )
fit0 <- fitNbinomGLMs( cds, count ~ 1 )
nbinomGLMTest( fit1, fit0 )
```

---

nbinomTest *Test for differences between the base means for two conditions*

Description

This function tests for differences between the base means of two conditions (i.e., for differential expression in the case of RNA-Seq).

Usage

```
nbinomTest(cds, condA, condB, pvals_only = FALSE, eps=NULL )
```

Arguments

cds	a CountDataSet with size factors and raw variance functions
condA	one of the conditions in 'cds'
condB	another one of the conditions in 'cds'
pvals_only	return only a vector of (unadjusted) p values instead of the data frame described below.
eps	This argument is no longer used. Do not use it.

Details

See nbinomTestForMatrices for more technical informations

Value

A data frame with the following columns:

id	The ID of the observable, taken from the row names of the counts slots.
baseMean	The base mean (i.e., mean of the counts divided by the size factors) for the counts for both conditions
baseMeanA	The base mean (i.e., mean of the counts divided by the size factors) for the counts for condition A
baseMeanB	The base mean for condition B
foldChange	The ratio meanB/meanA
log2FoldChange	The log2 of the fold change
pval	The p value for rejecting the null hypothesis 'meanA==meanB'
padj	The adjusted p values (adjusted with 'p.adjust(pval, method="BH")')

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds )
head( nbinomTest( cds, "A", "B" ) )
```

---

nbinomTestForMatrices

Perform row-wise tests for differences between the base means of two count matrices.

Description

This function is called by `nbinomTest`. Call it directly only if the S4 interface is unsuitable for your task.

Usage

```
nbinomTestForMatrices(countsA, countsB, sizeFactorsA, sizeFactorsB, dispsA, disp
```

Arguments

`countsA`	A matrix of counts, where each column is a replicate
`countsB`	Another matrix of counts, where each column is a replicate
`sizeFactorsA`	Size factors for the columns of the matrix 'countsA'
`sizeFactorsB`	Size factors for the columns of the matrix 'countsB'
`dispsA`	The dispersions for 'countsA', a vector with one value per gene
`dispsB`	The same for 'countsB'

Details

See the vignette for an exact description of the null hypothesis tested.

Value

A vector of unadjusted p values, one for each row in the counts matrices.

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
cds <- estimateDispersions( cds, method="per-condition" )
colsA <- conditions(cds) == "A"
colsB <- conditions(cds) == "B"
bmvA <- getBaseMeansAndVariances( counts(cds)[,colsA], sizeFactors(cds)[colsA] )
bmvB <- getBaseMeansAndVariances( counts(cds)[,colsB], sizeFactors(cds)[colsB] )
pvals <- nbinomTestForMatrices(
   counts(cds)[,colsA],
   counts(cds)[,colsB],
   sizeFactors(cds)[colsA],
   sizeFactors(cds)[colsB],
   fitInfo(cds,"A")$dispFunc( rowMeans( counts( cds, normalized=TRUE ) ) ),
   fitInfo(cds,"B")$dispFunc( rowMeans( counts( cds, normalized=TRUE ) ) ) )
names( pvals ) <- row.names( counts(cds) )
head( pvals )

# This here should give the same results:
head( nbinomTest( cds, "A", "B" )$pval )
```

`nbkd.sf` *GLM family for a negative binomial with known dispersion and log link with size factors*

Description

A distribution family for use with `glm`. It describes a negative binomial (as `negative.binomial` in the MASS package), but with a special link function, namely eta[i] = log(mu[i] / sf[i]), i.e., each count value is divided by its size factor before the log is taken. This is used internally by `fitNbinomGLMs`.

Usage

```
nbkd.sf(r, sf)
```

Arguments

`r`	The 'size' argument (see `dnbinom`), i.e., the reciprocal of the dispersion.
`sf`	A vector of size factors.

Value

A GLM family object.

Author(s)

Simon Anders, anders@embl.de

`newCountDataSet` *Create a CountDataSet object*

Description

This function creates a CountDataSet object from a matrix or data frame of count data.

Usage

```
newCountDataSet(countData, conditions, sizeFactors = NULL, phenoData = NULL, fea
```

Arguments

`countData`	A matrix or data frame of count data, i.e., of non-negative integer values. The rows correspond to observations (e.g., number of reads that were assigned to a gene), the columns correspond to samples (or experiments). Note that biological replicates should each get their own column, while the counts of technical replicates (i.e., several sequencing ruins/lanes from the same sample) have to be summed up into a single column.
`conditions`	A factor of experimental conditions (or treatments, or tissue types, or phenotypes, or the like). The length of the factor has to be equal to the number of columns of the countData matrix, assigning a condition to each sample. If 'conditions' is not a factor, it will be converted to one. Alternatively, you may pass a data frame, that will be placed in pData(cds) as is and can then be used with the modes "pooled" and "blind" in `estimateVarianceFunctions` and its columns can be refered top in a model formula provided to `fitNbinomGLMs`.
`sizeFactors`	This argument is deprecated. Do not use it. (Size factors should always be estimated from the data with `estimateSizeFactors`. If you need to set size factors manually for some reasons, change the `pData(cds)$sizeFactor`.
`phenoData`	You may pass an AnnotatedDataFrame here to describe the columns of the count matrix. Note that the package always adds two rows (or creates a new AnnotatedDataFrame with only these two rows in case you do not supply one) with names "condition" and "sizeFactor" to store this information.
`featureData`	You may pass an AnnotatedDataFrame here to describe the rows of the count matrix. The package will just pass through this information without using it. Note that further columns will be added to feature data later, when estimating dispersions.

Details

See also `CountDataSet-class` and the documentation of `eSet` (package Biobase) for the meaning of the other slots, which CountDataSet inherits from eSet (but which the present package does not use).

Value

an object of class CountDataSet

Author(s)

Simon Anders, sanders@fs.tum.de

Examples

```
countsTable <- counts( makeExampleCountDataSet() )
cds <- newCountDataSet( countsTable, c( "A", "A", "A", "B", "B" ) )
```

`residualsEcdfPlot` *REMOVED*

Description

This function has been removed. Please see the vignette for our newer suggestions on how to check fit quality.

Usage

```
residualsEcdfPlot(...)
```

Arguments

`...`	dummy argument

`scvPlot` *REMOVED*

Description

This function has been removed. Please see the vignette for our newer suggestions on how to check fit quality.

Usage

```
scvPlot( ... )
```

Arguments

`...`	dummy argument

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

Description

The sizeFactors vector assigns to each column of the count data 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.

Usage

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

Arguments

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

Author(s)

Simon Anders, sanders@fs.tum.de

See Also

`estimateSizeFactors`

Examples

```
cds <- makeExampleCountDataSet()
cds <- estimateSizeFactors( cds )
sizeFactors(cds)
```

`varianceFitDiagnostics` *REMOVED*

Description

This function has been removed. Please see the vignette for our newer suggestions on how to check fit quality.

Usage

```
varianceFitDiagnostics( ... )
```

Arguments

`...`	dummy argument

Index