edgeR

April 20, 2011

`approx.expected.info`
Approximate Expected Information (Fisher Information)

Description

Using a linear fit (for simplicity), the expected information from the conditional log likelihood of the dispersion parameter of the negative binomial is calculated over all genes.

Usage

```
approx.expected.info(object, d, pseudo, robust = FALSE)
```

Arguments

`object`	`DGEList` object containing the raw counts with (at least) elements `counts` (table of counts), `group` (vector indicating group) and `lib.size` (vector of library sizes)
`d`	numeric vector giving the delta parameter for negative binomial - `phi/(phi+1)` ; either of length 1 or of length equal to the number of tags/transcripts (i.e. number of rows of `object$counts`.
`pseudo`	numeric matrix of pseudocounts from output of `estimateDispIter`
`robust`	logical on whether to use a robust fit, default `FALSE`

Value

numeric vector of approximate values of the Fisher information for each tag/transcript (with length same as the number of rows of the original counts)

Author(s)

Mark Robinson

See Also

This function is used in the algorithm for estimating an appropriate amount of smoothing for the dipsersion estimates carried out by `estimateSmoothing`.

Examples

```
set.seed(0)
y<-matrix(rnbinom(40,size=1,mu=10),ncol=4)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
d<-estimateCommonDisp(d)
d<-estimateTagwiseDisp(d,prior.n=10)
exp.inf<-approx.expected.info(d,1/(1 + d$common.dispersion),d$pseudo.alt)
```

`betaApproxNBTest` *An Approximate Exact Test for Differences between Two Negative Binomial Groups*

Description

Approximate the tail probabilities of a conditional negative binomial exact test of equality of means between groups.

Usage

```
betaApproxNBTest(x1, x2, dispersion)
```

Arguments

`x1`	vector of observed negative binomial variables for group one
`x2`	vector of observed negative binomial variables for group two
`dispersion`	vector or scalar providing the value of the NB dispersion parameter for each tag to be used for calculating p-values for differences in mean between the two groups.

Details

`exactTest` is the user-level function for computing p-values for differential expression between groups in DGE data. However, for tags with extremely large counts, the computation of the tail propbabilities of the conditional negative binomial exact test can be unstable. For such tags, the tail probabilities are well approximated by using a transformed beta distribution (Anderson and Boullion, 1972).

Value

Vector of p-values providing the extent of evidence for difference in means between the two groups.

Author(s)

Davis McCarthy

References

Anderson, Dwane E. and Boullion, Thomas L. Homogeneity test for two negative binomial populations. IEEE Transactions on Reliability, Vol. R-21, No. 2, May 1972.

See Also

Computing p-values for differential expression for each transcript between two (only) digital gene expression libraries can also be done using the `sage.test` function in the `statmod` package.

Examples

```
# generate raw counts from NB, create list object
x1<-rnbinom(20,size=1,mu=1000)
x2<-rnbinom(20, size=1, mu=1500)
betaApproxNBTest(x1, x2, dispersion=1)
```

`calcNormFactors` *Calculates Normalization Factors for a Matrix of Count Data*

Description

Using a reference sample, calculate the normalization factors, over and above accounting for library size.

Usage

```
calcNormFactors(object, method=c("TMM","RLE","quantile"), refColumn = NULL, logr
```

Arguments

`object`	either a `matrix` of raw (read) counts or a `DGEList` object
`method`	method to use to calculate the scale factors
`refColumn`	column to use as reference, only used when `method="TMM"`
`logratioTrim`	amount of trim to use on log-ratios ("M" values), only used when `method="TMM"`
`sumTrim`	amount of trim to use on the combined absolute levels ("A" values), only used when `method="TMM"`
`doWeighting`	logical, whether to compute (asymptotic binomial precision) weights, only used when `method="TMM"`
`Acutoff`	cutoff on "A" values to use before trimming, only used when `method="TMM"`
`quantile`	quantile used to compute scale factors from, only used when `method="Quantile"`

Details

When `method="TMM"`, the weighted trimmed mean of M values (to the reference) is used as the normalization factor, where the weights are from the delta method on Binomial data. If `refColumn` is unspecified, the library whose upper quartile is closest to the mean upper quartile is used. When `method="RLE"` (which stands for relative log expression), a median library is calculated from the geometric mean of all columns and the median ratio of each sample to the median library is taken as the scale factor (this is the implementation proposed by the DESeq package). When `method="Quantile"`, the scale factors are calculated from the quantiles (default=75

For symmetry, normalization factors are adjusted to multiply to 1.

Value

If a `matrix` is given for `object`, the output is a vector with length `ncol(object)` giving the relative normalization factors. If a `DGEList` object is given for `object`, the output is a `DGEList` object containing the normalization factors in the `samples$norm.factors` element.

Author(s)

Mark Robinson

Examples

```
d <- matrix( rpois(1000, lambda=5), nrow=200 )
f <- calcNormFactors(d)
```

---

commonCondLogLikDerDelta

Conditional Log-Likelihoods in Terms of Delta

Description

Common conditional log-likelihood parameterized in terms of delta (`phi / (phi+1)`)

Usage

```
commonCondLogLikDerDelta(y, delta, der = 0, doSum = FALSE)
```

Arguments

`y`	list with elements comprising the matrices of count data (or pseudocounts) for the different groups
`delta`	delta (`phi / (phi+1)`) parameter of negative binomial
`der`	derivative, either 0 (the function), 1 (first derivative) or 2 (second derivative)
`doSum`	logical, whether to sum over samples or not (default `FALSE`

Details

The common conditional log-likelihood is constructed by summing over all of the individual tag conditional log-likelihoods. The common conditional log-likelihood is taken as a function of the dispersion parameter (`phi`), and here parameterized in terms of delta (`phi / (phi+1)`). The value of delta that maximizes the common conditional log-likelihood is converted back to the `phi` scale, and this value is the estimate of the common dispersion parameter used by all tags.

Value

numeric scalar of function/derivative evaluated at given delta

Author(s)

Davis McCarthy

See Also

estimateCommonDisp is the user-level function for estimating the common dispersion parameter.

Examples

```
counts<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=counts,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
y<-splitIntoGroups(d)
ll1<-commonCondLogLikDerDelta(y,delta=0.5,der=0,doSum=FALSE)
ll2<-commonCondLogLikDerDelta(y,delta=0.5,der=1)
```

---

condLogLikDerDelta *Conditional Log-Likelihood in Terms of Delta*

Description

Conditional negative binomial log-likelihood parameterized in terms of delta (phi / (phi+1))

Usage

```
condLogLikDerDelta(y, delta, grid = TRUE, der = 1, doSum = TRUE)
```

Arguments

y	matrix with count data (or pseudocounts)
delta	delta (phi / (phi+1))parameter of negative binomial
grid	logical, whether to calculate a grid over the values of delta
der	derivative, either 0 (the function), 1 (first derivative) or 2 (second derivative)
doSum	logical, whether to sum over samples or not (default TRUE

Details

This function computes the individual tag conditional log-likelihood for each tag. It is necessary for computing both the common conditional log-likelihood and the weighted conditional log-likelihood, which are used to find the common and tagwise (moderated) estimates of the dipsersion parameter. The delta scale for convenience (delta is bounded between 0 and 1).

Value

vector or matrix of function/derivative evaluations

Author(s)

Mark Robinson, Davis McCarthy

See Also

commonCondLogLikDerDelta and weightedCondLogLikDerDelta rely on condLogLikDerDelta, and at a user level, estimateCommonDisp and estimateTagwiseDisp are used to estimate the common and (moderated) tagwise dispersion estimates, respectively. condLogLikDerDelta calls condLogLikDerSize, the function that does the mathematical calculations.

Examples

```
y1<-matrix(rnbinom(10,size=1,mu=10),nrow=5)
v1<-seq(.1,.9,length=9)
ll1<-condLogLikDerDelta(y1,v1,grid=TRUE,der=0,doSum=FALSE)
ll2<-condLogLikDerDelta(y1,delta=.5,grid=FALSE,der=0)
```

---

condLogLikDerSize *Log-Likelihood of the Common Dispersion for a Single Equalized Group*

Description

Derivatives of the conditional negative-binomial log-likelihood (for each tag/transcript) with respect to the common dispersion parameter, for a single group of replicate libraries of the same size. Parameterized in terms of size or precision (1/phi).

Usage

```
condLogLikDerSize(y, r, der=1)
```

Arguments

y	matrix of (pseudo) count data
r	size parameter of negative binomial distribution
der	order of derivative required, either 0 (the function), 1 (first derivative) or 2 (second derivative)

Details

The library sizes must be equalized before running this function. This function carries out the actual mathematical computations for the conditional log-likelihood and its derivatives, calculating the conditional log-likelihood for each tag/transcript.

Value

vector of function/derivative evaluations, one for each transcript

Author(s)

Mark Robinson, Davis McCarthy

Examples

```
y <- matrix(rnbinom(10,size=1,mu=10),nrow=5)
condLogLikDerSize(y,r=1,der=1)
```

`decideTestsDGE` *Multiple Testing Across Genes and Contrasts*

Description

Classify a series of related differential expression statistics as up, down or not significant. A number of different multiple testing schemes are offered which adjust for multiple testing down the genes as well as across contrasts for each gene.

Usage

```
decideTestsDGE(object, adjust.method="BH", p.value=0.05)
```

Arguments

`object`	`deDGElist` object, output from `exactTest`, or `DGELRT` object, output from `DGELRT`, from which p-values for differential expression and log-fold change values may be extracted.
`adjust.method`	character string specifying p-value adjustment method. Possible values are `"none"`, `"BH"`, `"fdr"` (equivalent to `"BH"`), `"BY"` and `"holm"`. See `p.adjust` for details.
`p.value`	numeric value between 0 and 1 giving the desired size of the test

Details

These functions implement multiple testing procedures for determining whether each log-fold change in a matrix of log-fold changes should be considered significantly different from zero.

Value

An object of class `TestResults` (see `TestResults`). This is essentially a numeric matrix with elements `-1`, `0` or `1` depending on whether each DE p-value is classified as significant with negative log-fold change, not significant or significant with positive log-fold change, respectively.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

Adapted from `decideTests` in the limma package.

DGEExact-class *differential expression of Digital Gene Expression data - class*

Description

A simple list-based class for storing results of differential expression analysis for DGE data

Slots/List Components

Objects of this class contain the following list components:

`table`: data frame containing the log-concentration (i.e. expression level), the log-fold change in expression between the two groups/conditions and the exact p-value for differential expression, for each tag.

`comparison`: vector giving the two experimental groups/conditions being compared.

`genes`: a data frame containing information about each transcript (can be `NULL`).

Methods

This class inherits directly from class `list` so any operation appropriate for lists will work on objects of this class. `DGEExact` objects also have a `show` method.

Author(s)

Mark Robinson, Davis McCarthy

DGEGLM-class *Digital Gene Expression Generalized Linear Model results - class*

Description

A simple list-based class for storing results of a GLM fit to each tag/gene in a DGE dataset.

Slots/List Components

Objects of this class contain the following list components:

`coefficients`: matrix containing the coefficients computed from fitting the model defined by the design matrix to each gene/tag in the dataset.

`df.residual`: vector containing the residual degrees of freedom for the model fit to each tag/gene in the dataset.

`deviance`: vector giving the deviance from the model fit to each tag/gene.

`design`: design matrix for the full model from the likelihood ratio test.

`offset`: scalar, vector or matrix of offset values to be included in the GLMs for each tag/gene.

`samples`: data frame containing information about the samples comprising the dataset.

`genes`: data frame containing information about the genes or tags for which we have DGE data (can be `NULL` if there is no information available).

`dispersion`: scalar or vector providing the value of the dispersion parameter used in the negative binomial GLM for each tag/gene.

`lib.size`: vector providing the effective library size for each sample in the dataset.

`weights`: matrix of weights used in the GLM fitting for each tag/gene.

`fitted.values`: the fitted (expected) values–here they are counts–from the GLM for each tag/gene.

`abundance`: vector of gene/tag abundances (expression level), on the log2 scale, computed from the mean count for each gene/tag after scaling count by normalized library size.

Methods

This class inherits directly from class `list` so any operation appropriate for lists will work on objects of this class. `DGEGLM` objects also have a `show` method.

Author(s)

Davis McCarthy

`DGEList-class` *Digital Gene Expression data - class*

Description

A simple list-based class for storing read counts from digital gene expression technologies and other important information for the analysis of DGE data.

Slots/List Components

Objects of this class contain (at least) the following list components:

`counts`: numeric matrix containing the read counts.

`samples`: data.frame containing the library size and group labels.

Methods

This class inherits directly from class `list` so any operation appropriate for lists will work on objects of this class. `DGEList` objects also have a `show` method.

Author(s)

Mark Robinson

See Also

`DGEList`

DGEList *DGEList Constructor*

Description

A function to create a DGEList object from a table of counts (rows=features, columns=samples), group indicator for each column, library size (optional) and a table of annotation (optional)

Usage

```
DGEList(counts = matrix(0, 0, 0), lib.size = NULL, norm.factors = NULL, group =
```

Arguments

counts	numeric matrix containing the read counts.
lib.size	numeric vector containing the total to normalize against for each sample (optional)
norm.factors	numeric vector containing normalization factors (optional, defaults to all 1)
group	vector giving the experimental group/condition for each sample/library
genes	data frame containing annotation information for the tags/transcripts/genes for which we have count data (optional).
remove.zeros	whether to remove rows that have 0 total count; default is FALSE so as to retain all information in the dataset

Details

If no lib.size argument is passed to the constructor, the column totals are used.

The optional genes argument is meant to be an annotation data.frame, with rows matching those in the counts argument.

Value

a DGEList object

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

See Also

DGEList

Examples

```
y <- matrix(rnbinom(10000,mu=5,size=2),ncol=4)
d <- DGEList(counts=y, group=rep(1:2,each=2), lib.size=colSums(y))
```

DGELRT-class *Digital Gene Expression Likelihood Ratio Test data and results - class*

Description

A simple list-based class for storing results of a GLM-based differential expression analysis for DGE data, with evidence for differential expression assessed using a likelihood ratio test.

Slots/List Components

Objects of this class contain the following list components:

`table`: data frame containing the log-concentration (i.e. expression level), the log-fold change in expression between the two groups/conditions and the exact p-value for differential expression, for each tag.

`coefficients.full`: matrix containing the coefficients computed from fitting the full model (fit using `glmFit` and a given design matrix) to each gene/tag in the dataset.

`coefficients.null`: matrix containing the coefficients computed from fitting the null model to each gene/tag in the dataset. The null model is the model to which the full model is compared, and is fit using `glmFit` and dropping selected column(s) (i.e. coefficient(s)) from the design matrix for the full model.

`design`: design matrix for the full model from the likelihood ratio test.

`...`: if the argument `y` to `glmLRT` (which produces the `DGELRT` object) was itself a `DGEList` object, then the `DGELRT` will contain all of the elements of `y`, except for the table of counts and the table of pseudocounts.

Methods

This class inherits directly from class `list` so any operation appropriate for lists will work on objects of this class. `DGELRT` objects also have a `show` method.

Author(s)

Davis McCarthy

dglmStdResid *Visualize the mean-variance relationship in DGE data using standardized residuals*

Description

Appropriate modelling of the mean-variance relationship in DGE data is important for making inferences about differential expression. However, the standard approach to visualizing the mean-variance relationship is not appropriate for general, complicated experimental designs that require generalized linear models (GLMs) for analysis. Here are functions to compute standardized residuals from a Poisson GLM and plot them for bins based on overall expression level of tags as a way to visualize the mean-variance relationship. A rough estimate of the dispersion parameter can also be obtained from the standardized residuals.

Usage

```
dglmStdResid(y, design, dispersion=0, offset=0, nbins=100, make.plot=TRUE, xlab=
getDispersions(binned.object)
```

Arguments

y	numeric matrix of counts, each row represents one tag, each column represents one DGE library.
design	numeric matrix giving the design matrix of the GLM. Assumed to be full column rank.
dispersion	numeric scalar or vector giving the dispersion parameter for each GLM. Can be a scalar giving one value for all tags, or a vector of length equal to the number of tags giving tag-wise dispersions.
offset	numeric vector or matrix giving the offset that is to be included in teh log-linear model predictor. Can be a vector of length equal to the number of libraries, or a matrix of the same size as y.
nbins	scalar giving the number of bins (formed by using the quantiles of the genewise mean expression levels) for which to compute average means and variances for exploring the mean-variance relationship. Default is 100 bins
make.plot	logical, whether or not to plot the mean standardized residual for binned data (binned on expression level). Provides a visualization of the mean-variance relationship. Default is TRUE.
xlab	character string giving the label for the x-axis. Standard graphical parameter. If left as the default, then the x-axis label will be set to "Mean".
ylab	character string giving the label for the y-axis. Standard graphical parameter. If left as the default, then the y-axis label will be set to "Ave. binned standardized residual".
...	further arguments passed on to plot
binned.object	list object, which is the output of dglmStdResid.

Details

This function is useful for exploring the mean-variance relationship in the data. Raw or pooled variances cannot be used for complex experimental designs, so instead we can fit a Poisson model using the appropriate design matrix to each tag and use the standardized residuals in place of the pooled variance (as in plotMeanVar) to visualize the mean-variance relationship in the data. The function will plot the average standardized residual for observations split into nbins bins by overall expression level. This provides a useful summary of how the variance of the counts change with respect to average expression level (abundance). A line showing the Poisson mean-variance relationship (mean equals variance) is always shown to illustrate how the genewise variances may differ from a Poisson mean-variance relationship. A log-log scale is used for the plot.

The function mglmLS is used to fit the Poisson models to the data. This code is fast for fitting models, but does not compute the value for the leverage, technically required to compute the standardized residuals. Here, we approximate the standardized residuals by replacing the usual denominator of (1 - leverage) by (1 - p/n) , where n is the number of observations per tag (i.e. number of libraries) and p is the number of parameters in the model (i.e. number of columns in the full-rank design matrix.

Value

`dglmStdResid` produces a mean-variance plot based on standardized residuals from a Poisson model fitfor each tag for the DGE data. `dglmStdResid` returns a list with the following elements:

`ave.means` vector of the average expression level within each bin of observations

`ave.std.resid` vector of the average standardized Poisson residual within each bin of tags

`bin.means` list containing the average (mean) expression level (given by the fitted value from the given Poisson model) for observations divided into bins based on amount of expression

`bin.std.resid` list containing the standardized residual from the given Poisson model for observations divided into bins based on amount of expression

`means` vector giving the fitted value for each observed count

`standardized.residuals` vector giving approximate standardized residual for each observed count

`bins` list containing the indices for the observations, assigning them to bins

`nbins` scalar giving the number of bins used to split up the observed counts

`ngenes` scalar giving the number of genes/tags in the dataset

`nlibs` scalar giving the number of libraries in the dataset

`getDispersions` computes the dispersion from the standardized residuals and returns a list with the following components:

`bin.dispersion` vector giving the estimated dispersion value for each bin of observed counts, computed using the average standardized residual for the bin

`bin.dispersion.used` vector giving the actual estimated dispersion value to be used. Some computed dispersions using the method in this function can be negative, which is not allowed. We use the dispersion value from the nearest bin of higher expression level with positive dispersion value in place of any negative dispersions.

`dispersion` vector giving the estimated dispersion for each observation, using the binned dispersion estimates from above, so that all of the observations in a given bin get the same dispersion value.

Author(s)

Davis McCarthy

See Also

`plotMeanVar`, `plotMDS.dge`, `plotSmear` and `maPlot` provide more ways of visualizing DGE data.

Examples

```
y <- matrix(rnbinom(1000,mu=10,size=2),ncol=4)
design <- model.matrix(~c(0,0,1,1)+c(0,1,0,1))
binned <- dglmStdResid(y, design, dispersion=0.5)

getDispersions(binned)$bin.dispersion.used # Look at the estimated dispersions for the bi
```

dimnames *Retrieve the Dimension Names of a DGEList Object*

Description

Retrieve the dimension names of a digital gene expression data object.

Usage

```
## S3 method for class 'DGEList':
dimnames(x)
## S3 replacement method for class 'DGEList':
dimnames(x) <- value
```

Arguments

x	an object of class DGEList
value	a possible value for dimnames(x): see dimnames

Details

The dimension names of a microarray object are the same as those of the most important matrix component of that object.

A consequence is that rownames and colnames will work as expected.

Value

Either NULL or a list of length 2. If a list, its components are either NULL or a character vector the length of the appropriate dimension of x.

Author(s)

Gordon Smyth

See Also

dimnames in the base package.

02.Classes gives an overview of data classes used in LIMMA.

dim *Retrieve the Dimensions of a DGEList, DGEExact, DGEGLM, DGELRT or TopTags Object*

Description

Retrieve the number of rows (transcripts) and columns (libraries) for an DGEList, DGEExact or TopTags Object.

Usage

```
## S3 method for class 'DGEList':
dim(x)
## S3 method for class 'DGEList':
length(x)
```

Arguments

x	an object of class DGEList, DGEExact, TopTags, DGEGLM or DGELRT

Details

Digital gene expression data objects share many analogies with ordinary matrices in which the rows correspond to transcripts or genes and the columns to arrays. These methods allow one to extract the size of microarray data objects in the same way that one would do for ordinary matrices.

A consequence is that row and column commands `nrow(x)`, `ncol(x)` and so on also work.

Value

Numeric vector of length 2. The first element is the number of rows (genes) and the second is the number of columns (arrays).

Author(s)

Gordon Smyth, Davis McCarthy

See Also

`dim` in the base package.

02.Classes gives an overview of data classes used in LIMMA.

Examples

```
M <- A <- matrix(11:14,4,2)
rownames(M) <- rownames(A) <- c("a","b","c","d")
colnames(M) <- colnames(A) <- c("A1","A2")
MA <- new("MAList",list(M=M,A=A))
dim(M)
ncol(M)
nrow(M)
length(M)
```

edgeR-package *Empirical analysis of digital gene expression data in R*

Description

edgeR is a library for the analysis of digital gene expression data arising from RNA sequencing technologies such as SAGE, CAGE, Tag-seq or RNA-seq, with emphasis on testing for differential expression.

Particular strengths of the package include the ability to estimate biological variation between replicate libraries, and to conduct exact tests of significance which are suitable for small counts. The package is able to make use of even minimal numbers of replicates.

A User's Guide is available as well as the usual help page documentation for each of the individual functions.

The library implements statistical methodology developed by Robinson and Smyth (2007, 2008).

Author(s)

Mark Robinson <mrobinson@wehi.edu.au>, Davis McCarthy <dmccarthy@wehi.edu.au>, Gordon Smyth

References

Robinson MD and Smyth GK (2007). Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics* 23, 2881-2887

Robinson MD and Smyth GK (2008). Small-sample estimation of negative binomial dispersion, with applications to SAGE data. *Biostatistics*, 9, 321-332

Robinson MD, McCarthy DJ and Smyth GK (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 26, 139-140

equalizeLibSizes *Quantile Adjustment to Equalize Library Sizes for a Fixed Value of the Dispersion Parameter*

Description

A function that uses a NB quantile-to-quantile method to adjust the libraries of counts so that library sizes are equal for a fixed value of the dispersion parameter.

Usage

```
equalizeLibSizes(object, disp=0, N=exp(mean(log(object$samples$lib.size*object$s
```

Arguments

`object`	`DGEList` object containing the raw counts with elements `counts` (table of counts), `group` (vector indicating group) and `lib.size` (vector of library sizes)
`disp`	numeric scalar or vector of `dispersion` parameters; if a scalar, then a common dispersion parameter is used for all tags
`N`	numeric scalar, the library size to normalize to; default is the geometric mean of the original library sizes
`null.hypothesis`	logical, whether to calculate the `input.mean` and `output.mean` under the null hypothesis; default is `FALSE`

Details

The function `equalizeLibSizes` provides the necessary framework and calculations to call `q2qnbinom`, for given value(s) of the dispersion parameter. The function `q2qnbinom` actually generates the pseudocounts, the counts that have been adjusted for normalized library sizes. These pseudocounts are required to estimate the dispersion parameter, as the methods used by `estimateCommonDisp` and `estimateTagwiseDisp` rely on the assumption of equal library sizes. This function calls `estimatePs` to estimate the expression proportion for each tag, which is needed to calculate the `input.mean` and `output.mean` for each tag, which are passed to `q2qnbinom` along with the unadjusted counts and the fixed value(s) for the dispersion parameter.

Value

A list with elements

`pseudo`	numeric matrix of pseudocounts, i.e. adjusted counts for equalized libraries
`conc`	list with elements `conc.common` (vector giving overall proportion/concentration for each tag), and `conc.group` (matrix with columns giving estimates of tag/gene concentrations (proportion of total RNA for that group that that particular tag/gene contributes) for different groups); output from `estimatePs`
`N`	normalized library size

Author(s)

Mark Robinson, Davis McCarthy

Examples

```
y<-matrix(rnbinom(10000,size=2,mu=10),ncol=4)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000,1010),2))
ps<-estimatePs(d,r=2)
q2q.out<-equalizeLibSizes(d,disp=0.5,null.hypothesis=FALSE)
```

`estimateCommonDisp` *Estimates the Negative Binomial Common Dispersion by Maximizing the Negative Binomial Conditional Common Likelihood*

Description

Maximizes the negative binomial conditional common likelihood to give the estimate of the common dispersion across all tags for the unadjusted counts provided.

Usage

```
estimateCommonDisp(object, tol=1e-06, rowsum.filter=5)
```

Arguments

`object`	`DGEList` object with (at least) elements `counts` (table of unadjusted counts), and `samples` (vector indicating group) and `lib.size` (vector of library sizes)
`tol`	numeric scalar providing the tolerance to be passed to `optimize`; default value is `1e-06`
`rowsum.filter`	numeric scalar giving a value for the filtering out of low abundance tags in the estimation of the common dispersion. Only tags with total sum of counts above this value are used in the estimation of the common dispersion. Low abundance tags can adversely affect the estimation of the common dispersion, so this argument allows the user to select an appropriate filter threshold for the tag abundance.

Details

The method of conditional maximum likelihood assumes that library sizes are equal, which is not true in general, so pseudocounts (counts adjusted so that the library sizes are equal) need to be calculated. The function `equalizeLibSizes` is called to adjust the counts using a quantile-to-quantile method, but this requires a fixed value for the common dispersion parameter. To obtain a good estimate for the common dispersion, pseudocounts are calculated under the Poisson model (dispersion is zero) and these pseudocounts are used to give an estimate of the common dispersion. This estimate of the common dispersion is then used to recalculate the pseudocounts, which are used to provide a final estimate of the common dispersion.

Value

`estimateCommonDisp` produces an object of class `DGEList` with the following components.

`common.dispersion`	estimate of the common dispersion; the value for `phi`, the dispersion parameter in the NB model, that maximizes the negative binomial common likelihood on the `phi` scale
`counts`	table of unadjusted counts
`group`	vector indicating the group to which each library belongs
`lib.size`	vector containing the unadjusted size of each library

pseudo.alt — table of adjusted counts; quantile-to-quantile method (see q2qnbinom) used to adjust the raw counts so that library sizes are equal; adjustment here done under the alternative hypothesis that there is a true difference between groups

conc — list containing the estimates of the concentration of each tag in the underlying sample; conc$p.common gives estimates under the null hypothesis of no difference between groups; conc$p.group gives the estimate of the concentration for each tag within each group; concentration is a measure of abundance and thus expression level for the tags

common.lib.size — the common library size to which the count libraries have been adjusted

Author(s)

Mark Robinson, Davis McCarthy

References

Robinson MD and Smyth GK (2008). Small-sample estimation of negative binomial dispersion, with applications to SAGE data. *Biostatistics*, 9, 321-332

See Also

estimateTagwiseDisp can be used to estimate a value for the dispersion parameter for each tag/transcript. The estimates are stabilized by squeezing the estimates towards the common value calculated by estimateCommonDisp.

Examples

```
y<-matrix(rnbinom(1000,mu=10,size=2),ncol=4)
d<-DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
cmdisp<-estimateCommonDisp(d)
```

---

estimateCRDisp — *Estimate the dispersion parameter for a negative binomial model using Cox-Reid approximate conditional inference*

Description

Estimates the common dispersion parameter for a DGE dataset for general experimental designs by using Cox-Reid approximate conditional inference for a negative binomial generalized linear model for each transcript (tag) with the unadjusted counts and design matrix provided.

Usage

```
estimateCRDisp(y, design=NULL, offset=0, npts=10, min.disp=0, max.disp=2, nselec
rowsum.filter=5, tagwise=FALSE, prior.n=10, trend=FALSE, lib.size=NULL, verbose=
adjustedProfileLik(dispersion, y, design, offset)
```

Arguments

y	an object that contains the raw counts for each library (the measure of expression level); it can either be a matrix of counts, or a DGEList object with (at least) elements counts (table of unadjusted counts) and samples (data frame containing information about experimental group, library size and normalization factor for the library size)
design	numeric matrix giving the design matrix for the GLM that is to be fit.
offset	numeric scalar, vector or matrix giving the offset (in addition to the log of the effective library size) that is to be included in the NB GLM for the transcripts. If a scalar, then this value will be used as an offset for all transcripts and libraries. If a vector, it should be have length equal to the number of libraries, and the same vector of offsets will be used for each transcript. If a matrix, then each library for each transcript can have a unique offset, if desired. In adjustedProfileLik the offset must be a matrix with the same dimension as the table of counts.
npts	scalar, the number of points at which to place knots for the spline-based estimation of the common and tagwise dispersion estimates.
min.disp	scalar, the minimum possible value for the dispersion. May need to be set smaller (e.g. 1e-04 or less) if there is no biological variability in the data.
max.disp	scalar, the maximum possible value for the dispersion.
nselect	scalar, the number of genes/tags to be used to get an initial 'ballpark' estimate of the magnitude of the dispersions in the dataset. Used to finesse the calculation of the estimates using all the data.
rowsum.filter	numeric scalar giving a value for the filtering out of low abundance tags in the estimation of the common dispersion. Only tags with total sum of counts above this value are used in the estimation of the common dispersion. Low abundance tags can adversely affect the estimation of the common dispersion, so this argument allows the user to select an appropriate filter threshold for the tag abundance.
tagwise	logical scalar, if FALSE (default) then the tagwise dispersions are not calculated, if TRUE then the tagwise dispersions are calculated.
prior.n	numeric scalar, smoothing parameter that indicates the weight to give to the common likelihood compared to the individual tag's likelihood; default 10 means that the common likelihood is given 10 times the weight of the individual tag/gene's likelihood in the estimation of the tag/genewise dispersion
trend	logical scalar, if FALSE (default) then the abundance-dispersion trend is not considered in calculating both the common dispersion and the tagwise dispersions, if TRUE then such trend is introduced in calculating both dispersions.
lib.size	optional vector providing the (effective) library size for each library (must have length equal to the number of columns, or libraries, in the matrix of counts). If NULL, then a default is used. If y is a DGEList object then the default for lib.size is the product of the library sizes and the normalization factors (in the samples slot of the object). If y is a simple matrix of counts, then the default for lib.size is the vector of column sums of y.
verbose	logical scalar, if TRUE (default) then certain notification messages are displayed in some circumstances, if FALSE then these messages are not displayed.

`dispersion` numeric scalar providing the common value for the dispersion parameter (the 'size' parameter in the GLM fit is equal to `1/dispersion`) that is used in fitting the GLM for each transcript. Poisson GLM is fitted if `dispersion` is set at 0. `estimateCRDisp` maximizes the Cox-Reid adjusted profile likelihood over dispersion to obtain the estimate for the common dispersion.

Details

To obtain estimates of the common and tagwise (i.e., genewise) dispersion parameters for negative binomial GLMs we use Cox-Reid approximate conditional inference. The approach is to maximize the adjusted profile likelihood over the dispersion value, for both the common and tagwise models and use these values as the common and tagwise dispersion parameters for differential signal testing in downstream analysis.

Value

`estimateCRDisp` produces a `DGEList` object, which contains the estimate of the common dispersion parameter for the negative binomial model that maximizes the Cox-Reid adjusted profile likelihood, and also the tagwise Cox-Reid dispersion estimates.

`adjustedProfileLik` produces a vector of the tagwise Cox-Reid adjusted profile likelihood for the given counts, dispersion value, offset and design matrices (i.e. the APL for each gene/tag).

Author(s)

Yunshun Chen, Gordon Smyth

References

Cox DR and Reid N (1987). Parameter orthogonality and approximate conditional inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, 49, 1, 1–39.

See Also

`estimateTagwiseDisp`, and `estimateCommonDisp` can be used to estimate a value for the dispersion parameter for each tag/transcript and a common dispersion value, respectively. The estimates are stabilized by squeezing the estimates towards the common value calculated by `estimateCommonDisp`. These functions use exact conditional methods, but are restricted to less complicated experimental designs; they can deal with multiple groups, but nothing more complicated.

Examples

```
y<-matrix(rnbinom(1000,mu=10,size=2),ncol=4)
d<-DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
design <- model.matrix(~d$samples$group) # Define the design matrix for the full model
d<-estimateCRDisp(d, design)
d
```

`estimatePs` *Estimate Expression Levels*

Description

Estimate expression levels (i.e. proportion of all sample mRNA corresponding to each tag; or, concentration of mRNA for each tag in sample mRNA) using maximum likelihood with dispersion parameter fixed based on the negative binomial model for each tag/gene and sample group. Expression proportions are used to determine overall abundance of each tag/gene and differential expression of tags/genes between groups.

Usage

```
estimatePs(object, r, tol = 1e-10, maxit = 30)
```

Arguments

`object`	list containing (at least) the elements `counts` (table of counts), `group` (vector or factor indicating group) and `lib.size` (numeric vector of library sizes)
`r`	numeric vector providing the size parameter of negative binomial model (`size` = `1/phi` where `phi` is the dispersion parameter in the NB model)
`tol`	numeric scalar, tolerance between iterations
`maxit`	positive integer scalar, maximum number of iterations

Details

The Newton-Raphson method is used to calculate iteratively the maximum likelihood estimate of the expression level (i.e. concentration of mRNA for a particular tag in the sample mRNA) for each tag/gene.

Value

A list with elements:

`conc.common`	numeric vector giving overall proportion/concentration for each tag
`conc.group`	numeric matrix with columns giving estimates of tag/gene concentrations (proportion of total RNA for that group that that particular tag/gene contributes) for different groups)

Author(s)

Mark Robinson, Davis McCarthy

Examples

```
set.seed(0)
y<-matrix(rnbinom(40,size=1,mu=10),ncol=4)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
conc<-estimatePs(d,r=1)
```

`estimateSmoothing` *Estimate the Prior Weight*

Description

Estimate the prior weight, prior.n, using an approximate empirical Bayes rule given the estimate of the common dispersion. The prior weight determines how much smoothing takes place to squeeze tag/genewise estimates of the dispersion closer to the estimate of the common dispersion.

Usage

```
estimateSmoothing(object,verbose=TRUE)
```

Arguments

`object`	`DGEList` object, output of `estimateCommonDisp`
`verbose`	logical, whether to write comments, default `true`

Details

We are not recommending this function for routine use at the moment, as it has given unexpected results on some deep-sequenced data sets. It should be considered experimental. We are instead recommending that `prior.n` be chosen by the user. Values in the range 10-50 give good results in practice.

Value

`estimateSmoothing` produces an object of class `DGEList` with the following components.

`prior.n`	scalar; estimate of the prior weight, i.e. the smoothing parameter that indicates the weight to put on the common likelihood compared to the individual tag's likelihood; prior.n of 10 means that the common likelihood is given 10 times the weight of the individual tag/gene's likelihood in the estimation of the tag/genewise dispersion

Author(s)

Mark Robinson, Davis McCarthy

Examples

```
y<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
d<-estimateCommonDisp(d)
prior.n<-estimateSmoothing(d)
```

`estimateTagwiseDisp`
Maximizes the Negative Binomial Weighted Conditional Likelihood

Description

Maximizes the negative binomial weighted likelihood (a weighted version using the common likelihood given weight according the the smoothing parameter prior.n and the individual tag/gene likelihood) for each tag from the pseudocounts provided (i.e. assuming library sizes are equal), to give an estimate of the dispersion parameter for each tag (i.e. tagwise dispersion estimation).

Usage

```
estimateTagwiseDisp(object, prior.n=10, trend=FALSE, prop.used=NULL, tol=1e-06,
```

Arguments

`object`	a `DGEList` object containing (at least) the elements `counts` (table of raw counts), `group` (factor indicating group), `lib.size` (numeric vector of library sizes) and `pseudo,alt` (numeric matrix of quantile-adjusted pseudocounts calculated under the alternative hypothesis of a true difference between groups; recommended to use the `DGEList` object provided as the output of `estimateCommonDisp`
`prior.n`	numeric scalar, smoothing parameter that indicates the weight to give to the common likelihood compared to the individual tag's likelihood; default `10` means that the common likelihood is given 10 times the weight of the individual tag/gene's likelihood in the estimation of the tag/genewise dispersion
`trend`	logical, whether or not to let the tagwise dispersion estimates vary with tag/gene abundance (expression level), that is, whether or not to allow a trend with tag abundance in the tagwise dispersion estimates
`prop.used`	optional scalar giving the proportion of all tags/genes to be used for the locally weighted estimation of the tagwise dispersion, allowing the dispersion estimates to vary with abundance (expression level). If `NULL`, then a default value of 0.4 (i.e. 40 per cent of tags) are used. That means that for each tag/gene the estimate of its dispersion is based on the closest 40 per cent of all of the genes to that gene, where 'closeness' is based on similarity in expression level.
`tol`	numeric scalar, if `grid=FALSE`, tolerance for Newton-Rhapson iterations
`grid`	logical, whether to use a grid search (default = `TRUE`); if `FALSE`, uses `optimize`, but this is very slow if there is a large number of tags/genes to be analysed (i.e. more than 5000)
`grid.length`	if `grid=TRUE`, the number of points at which the likelihood is evaluated for each tag, so larger values improve the accuracy of the dispersion estimates; default `1000`
`verbose`	logical, whether to write comments, default `TRUE`

Value

`estimateSmoothing` produces an object of class `DGEList` with the following components.

`common.dispersion`
: estimate of the common dispersion; the value for `phi`, the dispersion parameter in the NB model, that maximizes the negative binomial common likelihood on the `phi` scale

`prior.n`
: estimate of the prior weight, i.e. the smoothing parameter that indicates the weight to put on the common likelihood compared to the individual tag's likelihood; prior.n of 10 means that the common likelihood is given 10 times the weight of the individual tag/gene's likelihood in the estimation of the tag/genewise dispersion

`tagwise.dispersion`
: tag- or gene-wise estimates of the dispersion parameter

`counts`
: table of unadjusted counts

`group`
: vector indicating the group to which each library belongs

`lib.size`
: vector containing the unadjusted size of each library

`pseudo.altn`
: table of adjusted counts; quantile-to-quantile method (see `q2qnbinom`) used to adjust the raw counts so that library sizes are equal; adjustment here done under the alternative hypothesis that there is a true difference between groups

`conc`
: list containing the estimates of the concentration of each tag in the underlying sample; `conc$p.common` gives estimates under the null hypothesis of no difference between groups; `conc$p.group` gives the estimate of the concentration for each tag within each group; concentration is a measure of abundance and thus expression level for the tags

`common.lib.size`
: the common library size to which the count libraries have been adjusted

Author(s)

Mark Robinson, Davis McCarthy

References

Robinson MD and Smyth GK (2007). Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics* 23, 2881-2887

See Also

`estimateCommonDisp` estimates a common value for the dispersion parameter for all tags/genes - should generally be run before `estimateTagwiseDisp`.

Examples

```
y<-matrix(rnbinom(1000,mu=10,size=2),ncol=4)
d<-DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
d<-estimateCommonDisp(d)
tgwdisp<-estimateTagwiseDisp(d, prior.n=10)
```

exactTest *An Exact Test for Differences between Two Negative Binomial Groups*

Description

Carry out an exact test for differences between two negative binomial groups, based on conditioning on sums of (quantile-adjusted pseudo-)counts; calculations performed by `exactTest.matrix`

Usage

```
exactTest(object,pair=NULL,dispersion=NULL,common.disp=TRUE)
exactTest.matrix(y1,y2,mus,r,all.zeros=rep(FALSE,nrow(y1)))
```

Arguments

`object`	a `DGEList` object, output of `estimateCommonDisp`, on which to compute Fisher-like exact statistics for the pair of groups specified.
`pair`	vector of length two, either numeric or character, providing the pair of groups to be compared; if a character vector, then should be the names of two groups (e.g. two levels of `object$samples$group`); if numeric, then groups to be compared are chosen by finding the levels of `object$samples$group` corresponding to those numeric values and using those levels as the groups to be compared; if `NULL`, then first two levels of `object$samples$group` (a factor) are used.
`dispersion`	optional vector either of length 1 or the same length as the number of tags. If not `NULL` (default), then the supplied value(s) will be used as the dispersion parameter for calculating p-values for differential expression. If `NULL`, then either the common or tagwise dispersion estimates from the `DGEList` object will be used, according to the value of `common.disp`. If `dispersion` is zero, then p-values are equivalent to exact Poisson rather than NB p-values.
`common.disp`	logical, if `TRUE`, then testing carried out using common dispersion for each tag/gene, if `FALSE` then tag-wise estimates of the dispersion parameter are used; default `TRUE`.
`y1`	numeric matrix of counts for one of the two given experimental groups to be tested for differences. Libraries are assumed to be equal in size - e.g. adjusted pseudocounts from the output of `equalizeLibSizes`.
`y2`	numeric matrix of counts for one of the two given experimental groups to be tested for differences. Libraries are assumed to be equal in size - e.g. adjusted pseudocounts from the output of `equalizeLibSizes`. Must have the same number of rows as `y1`.
`mus`	vector of count means for each tag/transcript under the null hypothesis (of no difference between groups)
`r`	vector of negative binomial `size` parameter values (`size = 1/phi` where `phi` is the dispersion parameter in the NB model); if `r` is of length 1, then a common value of the dispersion is used for all transcripts, otherwise, must be a vector with length equal to the number of rows of `y1` and `y2`. If you want to run a Poisson test, set r very large (e.g. 1000)
`all.zeros`	logical vector indicating for each tag whether it has zero counts in each library (`TRUE`) or not (`FALSE`), with the default being not to remove any tags.

Details

For each transcript, conditioning on the total sum of counts within each group and the total sum of counts across all groups allows us to construct an exact test for differences between two group. The conditional distribution for the sum of counts in a group is known (given the values for the mean counts, `mus`, and the dispersion parameter, 1/`r`), exact p-values can be computed by summing over all sums of counts that have a probability less than the probability under the null hypothesis of the observed sum of counts.

`exactTest.matrix` is the function that actually computes the exact p-values. `exactTest` is intended to have a more object-orientated flavor as it produces objects containing all the necessary components for downstream analysis.

Value

`exactTest` produces an object of class `DGEExact` containing the following elements.

`table`	a data frame containing the elements `logConc`, the log-average concentration/abundance for each tag in the two groups being compared, `logFC`, the log-abundance ratio, i.e. fold change, for each tag in the two groups being compared, `p.value`, exact p-value for differential expression using the NB model
`comparison`	a vector giving the names of the two groups being compared
`genes`	a data frame containing information about each transcript; taken from `object` and can be `NULL`

`exactTest.matrix` produces a numeric vector of exact p-values with length equal to the number of transcripts, taken to be the number of rows of `y1`.

Author(s)

Mark Robinson, Davis McCarthy

References

Robinson MD and Smyth GK (2008). Small-sample estimation of negative binomial dispersion, with applications to SAGE data. *Biostatistics*, 9, 321-332

See Also

Computing p-values for differential expression for each transcript between two (only) digital gene expression libraries can also be done using the `sage.test` function in the `statmod` package.

Examples

```
# generate raw counts from NB, create list object
y<-matrix(rnbinom(80,size=1,mu=10),nrow=20)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
rownames(d$counts)<-paste("tagno",1:nrow(d$counts),sep=".")

# estimate common dispersion and find differences in expression
d<-estimateCommonDisp(d)
de<-exactTest(d)

# example using exactTest.matrix directly
y<-matrix(rnbinom(20,mu=10,size=1.5),nrow=5)
group<-factor(c(1,1,2,2))
```

```
y<-splitIntoGroupsPseudo(y,group,pair=c(1,2))
mus<-rep(10,5)
f<-exactTest.matrix(y$y1,y$y2,mus,r=1.5,all.zeros=rep(FALSE,length=nrow(y$y1)))
```

`getCounts` *Extract Table of Counts from DGEList Object*

Description

Returns the `counts` slot of a DGEList object

Usage

```
getCounts(object)
```

Arguments

`object`	`DGEList` object containing (at least) the elements `counts` (table of raw counts), `group` (factor indicating group) and `lib.size` (numeric vector of library sizes)

Value

`getCounts` returns a matrix of counts (presumably integers)

Author(s)

Mark Robinson, Davis McCarthy

See Also

`DGEList` for more information about the `DGEList` class.

Examples

```
# generate raw counts from NB, create list object
y<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
# should be 5x4
print(dim(getCounts(d)))
```

`glmFit` *Fit negative binomial generalized linear model for each transcript*

Description

Fit a negative binomial generalized linear model for each transcript (tag) with the unadjusted counts provided, a value for the dispersion parameter and, optionally, offsets and weights for different libraries or transcripts.

Usage

```
glmFit(y, design, dispersion, offset=NULL, weights=NULL, lib.size=NULL)
glmLRT(y, glmfit, coef=ncol(glmfit$design), contrast=NULL)
```

Arguments

`y`	an object that contains the raw counts for each library (the measure of expression level); alternatively, a matrix of counts, or a `DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`design`	numeric matrix giving the design matrix for the GLM that is to be fit. Must be of full column rank.
`dispersion`	numeric scalar or vector providing the value for the dispersion parameter that is used in fitting the GLM for each transcript. Can be a common value for all tags, or a vector of values can provide a unique dispersion value for each tag.
`offset`	numeric scalar, vector or matrix giving the offset that is to be included in the NB GLM for the transcripts. Only one of `offset` and `lib.size` should be supplied—if both are supplied then `offset` will be used and `lib.size` will be ignored. If a scalar, then this value will be used as an offset for all transcripts and libraries. If a vector, it should be have length equal to the number of libraries, and the same vector of offsets will be used for each transcript. If a matrix, then each library for each transcript can have a unique offset, if desired. If `NULL` (the default) then the log of the effective library size (library size multiplied by normalization factors) will be used as the offsets in the GLMs.
`weights`	optional numeric matrix giving the matrix of weights for the observations (for each library and transcript) to be used in the GLM calculations. Not currently used in the GLM calculations.
`lib.size`	optional vector providing the (effective) library size for each library (must have length equal to the number of columns, or libraries, in the matrix of counts). If `NULL`, then a default is used. If `y` is a `DGEList` object then the default for `lib.size` is the product of the library sizes and the normalization factors (in the `samples` slot of the object). If `y` is a simple matrix of counts, then the default for `lib.size` is the vector of column sums of `y`.
`glmfit`	a `DGEGLM` object, the output from `glmFit`.
`coef`	scalar or vector indicating the column(s) of `design` that are to be dropped when creating the null model for the Likelihood Ratio (LR) Test. The `glmLRT` fits the null model and then conducts an LR test of the model fit provided in `glmfit` against the null model defined by the choice of `coef`.

contrast — contrast vector for which the test is required, of length equal to the number of columns of design. If specified, then takes precedence over coef.

Details

Given a fixed value for the dispersion parameter, a negative binomial model can be fitted to the counts for each tag/transcript in a dataset. The function glmFit calls the in-built function glm.fit to fit the NB GLM for each tag. Once we have a fit for a given design matrix, glmLRT can be run with a given coefficient or contrast specified and evidence for differential expression assessed using a likelihood ratio test. Tags can be ranked in order of evidence for differential expression, based on the p-value computed for each tag.

Value

glmFit produces an object of class DGEGLM with the following components:

coefficients — matrix of estimated coefficients from the NB model

df.residual — vector giving the residual degrees of freedom for each tag. In theory it can be different for different tags (if there are missing values), but in practice these will usually be identical for each tag.

deviance — vector giving the deviance from the NB model fit for each tag.

design — design matrix used in the NB model fit for each tag.

offset — scalar, vector or matrix giving the offset to use in the NB model for each tag.

samples — data frame providing information about the samples (libraries) in the experiment; taken from the object y.

genes — vector or data frame providing gene information for each tag; taken from the object y.

dispersion — scalar or vector giving the the value of the dispersion parameter used in each tag's NB model fit.

lib.size — vector of library sizes used in the model fit.

weights — matrix of final weights used in the NB model fits for each tag.

fitted.values — matrix of fitted values from the NB model for each tag.

abundance — vector of gene/tag abundances (expression level), on the log2 scale, computed from the mean count for each gene/tag after scaling count by normalized library size.

glmLRT produces an object of class DGELRT with the following components:

table — data frame (table) containing the abundance of each tag (log-concentration, logConc), the log-fold change of expression between conditions/contrasts being tested (logFC), the likelihood ratio statistic (LR.statistic) and the p-value from the LR test (p.value), for each tag in the dataset.

coefficients — matrix of coefficients for the full model defined by the design matrix (i.e. for the full model).

dispersion.used — scalar or vector of the dispersion value(s) used in the GLM fits and LR test.

The DGELRT object also contains all the elements of y except for the table of counts (raw data) and the table of pseudo-counts (if applicable).

Author(s)

Davis McCarthy and Gordon Smyth

See Also

estimateCRDisp for estimating the negative binomial dispersion.

topTags for displaying results from glmLRT.

Examples

```
nlibs <- 3
ntags <- 100
dispersion.true <- 0.1

# Make first transcript respond to covariate x
x <- 0:2
design <- model.matrix(~x)
beta.true <- cbind(Beta1=2,Beta2=c(2,rep(0,ntags-1)))
mu.true <- 2^(beta.true %*% t(design))

# Generate count data
y <- rnbinom(ntags*nlibs,mu=mu.true,size=1/dispersion.true)
y <- matrix(y,ntags,nlibs)
colnames(y) <- c("x0","x1","x2")
rownames(y) <- paste("Gene",1:ntags,sep="")
d <- DGEList(y)

# Normalize
d <- calcNormFactors(d)

# Fit the NB GLMs
fit <- glmFit(d, design, dispersion=dispersion.true)

## Likelihood ratio tests for trend
results <- glmLRT(d, fit, coef=2)
topTags(results)
```

---

goodTuring *Good-Turing Frequency Estimation*

Description

Non-parametric empirical Bayes estimates of the frequencies of observed (and unobserved) species.

Usage

```
goodTuring(x, plot=FALSE)
```

Arguments

x	numeric vector of non-negative integers, representing the observed frequency of each species.
plot	logical, whether to plot log-probability (i.e., log frequencies of frequencies)versus log-frequency.

Details

Observed counts are assumed to be Poisson. Using an non-parametric empirical Bayes strategy, the algorithm evaluates the posterior expectation of each species mean given its observed count. The posterior means are then converted to proportions. In the empirical Bayes step, the counts are smoothed by assuming a log-linear relationship between frequencies and frequencies of frequencies. The basics of the algorithm are from Good (1953). Gale and Sampson (1995) proposed a simplied algorithm with a rule for switching between the observed and smoothed frequencies, and it is Gale and Sampson's simplified algorithm that is implemented here. The number of zero values in `x` are not used in the algorithm, but is returned by this function.

Sampson gives a C code version on his webpage at http://www.grsampson.net/RGoodTur.html which gives identical results to this function.

Value

A list with components

`count`	observed frequencies, i.e., the unique positive values of `x`
`proportion`	estimated proportion of species given the count
`P0`	estimated combined proportion of all undetected species
`n0`	number of zeros found in `x`

Author(s)

Gordon Smyth

References

Gale, WA, and Sampson, G (1995). Good-Turing frequency estimation without tears. *Journal of Quantitative Linguistics* 2, 217-237.

Examples

```
#  True means of observed species
lambda <- rnbinom(10000,mu=2,size=1/10)
lambda <- lambda[lambda>1]

#  Oberved frequencies
Ntrue <- length(lambda)
x <- rpois(Ntrue, lambda=lambda)
freq <- goodTuring(x, plot=TRUE)
```

`logLikDerP` *Log-Likelihood for Proportion*

Description

Log-likelihood and derivatives for the proportion parameter (i,e, expression level) of negative binomial (mean = library size * proportion)

Usage

```
logLikDerP(p, y, lib.size, r, der = 0)
```

Arguments

p	vector of proportion parameters to be evaluated
y	matrix of counts
lib.size	vector of library sizes
r	size parameter of negative binomial distribution
der	derivative, either 0 (the function), 1 (first derivative) or 2 (second derivative)

Value

vector of the likelihood or specified derivative evaluations for each tag/gene

Author(s)

Mark Robinson, Davis McCarthy

See Also

estimatePs calls logLikDerP as part of the procedure for estimating the expression level(s) of each tag.

Examples

```
y<-matrix(rnbinom(20,size=1.5,mu=10),nrow=5)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))

this.p<-rowMeans( y/ outer(rep(1,nrow(y)),d$samples$lib.size) )
d1p<-logLikDerP(this.p,y,d$samples$lib.size,r=1.5,der=1)
```

---

maPlot — *Plots Log-Fold Change versus Log-Concentration (or, M versus A) for Count Data*

Description

To represent counts that were low (e.g. zero in 1 library and non-zero in the other) in one of the two conditions, a 'smear' of points at low A value is presented.

Usage

```
maPlot(x, y, logAbundance=NULL, logFC=NULL, normalize=FALSE, smearWidth = 1, col
```

Arguments

x	vector of counts or concentrations (group 1)
y	vector of counts or concentrations (group 2)
logAbundance	vector providing the abundance of each tag on the log2 scale. Purely optional (default is NULL), but in combination with logFC provides a more direct way to create an MA-plot if the log-abundance and log-fold change are available.

logFC	vector providing the log-fold change for each tag for a given experimental contrast. Default is NULL, only to be used together with logAbundance as both need to be non-null for their values to be used.
normalize	logical, whether to divide x and y vectors by their sum
smearWidth	scalar, width of the smear
col	vector of colours for the points (if NULL, uses allCol and lowCol)
allCol	colour of the non-smeared points
lowCol	colour of the smeared points
deCol	colour of the DE (differentially expressed) points
de.tags	indices for tags identified as being differentially expressed; use exactTest to identify DE genes
smooth.scatter	logical, whether to produce a 'smooth scatter' plot using the KernSmooth::smoothScatter function or just a regular scatter plot; default is FALSE, i.e. produce a regular scatter plot
lowess	logical, indicating whether or not to add a lowess curve to the MA-plot to give an indication of any trend in the log-fold change with log-concentration
...	further arguments passed on to plot

Details

The points to be smeared are identified as being equal to the minimum in one of the two groups. The smear is created by using random uniform numbers of width smearWidth to the left of the minimum A value.

Value

a plot to the current device

Author(s)

Mark Robinson, Davis McCarthy

See Also

plotSmear

Examples

```
y <- matrix(rnbinom(10000,mu=5,size=2),ncol=4)
maPlot(y[,1], y[,2])
```

meanvar *Explore the mean-variance relationship for DGE data*

Description

Appropriate modelling of the mean-variance relationship in DGE data is important for making inferences about differential expression. Here are functions to compute tag/gene means and variances, as well at looking at these quantities when data is binned based on overall expression level.

Usage

```
plotMeanVar(object, meanvar=NULL, show.raw.vars=FALSE, show.tagwise.vars=FALSE,
binMeanVar(x, conc=NULL, group, nbins=100, common.dispersion=FALSE, object=NULL)
pooledVar(y,group)
```

Arguments

object	DGEList object containing the raw data and dispersion value. According the method desired for computing the dispersion, either CRDisp or estimateCommonDisp and (possibly) estimateTagwiseDisp should be run on the DGEList object before using plotMeanVar. The argument object must be supplied in the function binMeanVar if common dispersion values are to be computed for each bin.
meanvar	list (optional) containing the output from binMeanVar or the returned value of plotMeanVar. Providing this object as an argument will save time in computing the tag/gene means and variances when producing a mean-variance plot.
show.raw.vars	logical, whether or not to display the raw (pooled) gene/tag variances on the mean-variance plot. Default is FALSE.
show.tagwise.vars	logical, whether or not to display the estimated genewise/tagwise variances on the mean-variance plot. Default is FALSE.
show.binned.common.disp.vars	logical, whether or not to compute the common dispersion for each bin of tags and show the variances computed from those binned common dispersions and the mean expression level of the respective bin of tags. Default is TRUE.
show.ave.raw.vars	logical, whether or not to show the average of the raw variances for each bin of tags plotted against the average expression level of the tags in the bin. Likely to be biased, so the default is FALSE.
dispersion.method	character string giving the method that has been used to estimate the common and tagwise dispersion values used to calculate the estimated variances. Default is "coxreid" indicating that the Cox-Reid method for GLMs has been used to compute the dispersions; other option is "qcml" to indicate that conditional inference methods (e.g. estimateCommonDisp and estimateTagwiseDisp were used.

scalar	vector (optional) of scaling values to divide counts by. Would expect to have this the same length as the number of columns in the count matrix (i.e. the number of libraries).
NBline	logical, whether or not to add a line on the graph showing the mean-variance relationship for a NB model with common dispersion.
nbins	scalar giving the number of bins (formed by using the quantiles of the genewise mean expression levels) for which to compute average means and variances for exploring the mean-variance relationship. Default is 100 bins
log	character vector indicating if any of the axes should use a log scale. Default is "xy", which makes both y and x axes on the log scale. Other valid options are "x" (log scale on x-axis only), "y" (log scale on y-axis only) and "" (linear scale on x- and y-axis).
xlab	character string giving the label for the x-axis. Standard graphical parameter. If left as the default NULL, then the x-axis label will be set to "logConc".
ylab	character string giving the label for the y-axis. Standard graphical parameter. If left as the default NULL, then the x-axis label will be set to "logConc".
...	further arguments passed on to plot
x	matrix of count data, with rows representing tags/genes and columns representing samples
conc	vector (optional) of values for the concentration (i.e. abundance) of each tag
group	factor giving the experimental group or condition to which each sample (i.e. column of x or element of y) belongs
common.dispersion	logical, whether or not to compute the common dispersion for each bin of tags.
y	vector of count data

Details

This function is useful for exploring the mean-variance relationship in the data. Raw variances are, for each gene, the pooled variance of the counts from each sample, divided by a scaling factor (by default the effective library size). The function will plot the average raw variance for tags split into nbins bins by overall expression level. This provides a useful summary of how the variance of the gene counts change with respect to average expression level (abundance). A line showing the Poisson mean-variance relationship (mean equals variance) is always shown to illustrate how the genewise variances may differ from a Poisson mean-variance relationship. Optionally, the raw variances and estimated tagwise variances can also be plotted. Estimated tagwise variances can be calculated using either qCML estimates of the tagwise dispersions (estimateTagwiseDisp) or Cox-Reid conditional inference estimates (CRDisp). A log-log scale is used for the plot.

Value

plotMeanVar produces a mean-variance plot for the DGE data using the options described above. plotMeanVar and binMeanVar both return a list with the following components:

avemeans	vector of the average expression level within each bin of genes
avevars	vector of the average raw pooled gene-wise variance within each bin of genes
bin.means	list containing the average (mean) expression level for genes divided into bins based on amount of expression
bin.vars	list containing the pooled variance for genes divided into bins based on amount of expression

means	vector giving the mean expression level for each gene
vars	vectore giving the pooled variance for each gene

pooledVar returns a scalar for the pooled variance of the given data vector.

Author(s)

Davis McCarthy

See Also

plotMDS.dge, plotSmear and maPlot provide more ways of visualizing DGE data.

Examples

```
y <- matrix(rnbinom(1000,mu=10,size=2),ncol=4)
d <- DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
plotMeanVar(d) # Produce a straight-forward mean-variance plot
meanvar <- plotMeanVar(d, show.raw.vars=TRUE) # Produce a mean-variance plot with the raw

## If we want to show estimated tagwise variances on the plot, we must first estimate the
d <- estimateCommonDisp(d) # Obtain an estimate of the dispersion parameter
d <- estimateTagwiseDisp(d)  # Obtain tagwise dispersion estimates
plotMeanVar(d, meanvar=meanvar, show.tagwise.vars=TRUE, NBline=TRUE, dispersion.method="q
## We could also estimate common/tagwise dispersions using the Cox-Reid methods using CRD
```

---

mglm *Fit a negative binomial generalized linear model to multiple response vectors*

Description

Fit the same log-link negative binomial or Poisson generalized linear model to each row of a matrix of counts.

Usage

```
mglmLS(y, design, dispersion=0, offset=0, start=NULL, tol=1e-5, maxit=50, trace=
mglmOneGroup(y, dispersion=0, offset=0, maxit=50, trace=FALSE)
mglmSimple(y, design, dispersion=0, offset=0, weights=NULL)
deviances.function(dispersion)
```

Arguments

y	numeric matrix containing the negative binomial counts. Rows for tags and columns for libraries.
design	numeric matrix giving the design matrix of the GLM. Assumed to be full column rank.
dispersion	numeric scalar or vector giving the dispersion parameter for each GLM. Can be a scalar giving one value for all tags, or a vector of length equal to the number of tags giving tag-wise dispersions.

offset	numeric vector or matrix giving the offset that is to be included in the log-linear model predictor. Can be a vector of length equal to the number of libraries, or a matrix of the same size as y.
weights	numeric vector or matrix of non-negative quantitative weights. Can be a vector of length equal to the number of libraries, or a matrix of the same size as y.
start	numeric matrix of starting values for the GLM coefficients. Number of rows should agree with y and number of columns should agree with design.
tol	numeric scalar giving the convergence tolerance.
maxit	scalar giving the maximum number of iterations for the Fisher scoring algorithm.
trace	logical, whether or not to information should be output at each iteration.

Details

The functions mglmLS, mglmOneGroup and mglmSimple all fit negative binomial generalized linear models, with the same design matrix but possibly different dispersions, offsets and weights, to a series of response vectors. mglmLS and mglmOneGroup are vectorized in R for fast execution, while mglmSimple simply makes tagwise calls to glm.fit in the stats package. The functions are all low-level functions in that they operate on atomic objects such as matrices. They are used as work-horses by higher-level functions in the edgeR package.

mglmOneGroup fits the null model, with intercept term only, to each response vector. In other words, it treats the libraries as belonging to one group. It implements Fisher scoring with a score-statistic stopping criterion for each tag. Excellent starting values are available for the null model, so this function seldom has any problems with convergence. It is used by other edgeR functions to compute the overall abundance for each tag.

mglmLS fits an arbitrary log-linear model to each response vector. It implements a vectorized approximate scoring algorithm with a likelihood derivative stopping criterion for each tag. A simple line search strategy is used to ensure that the residual deviance is reduced at each iteration. This function is the work-horse of other edgeR functions such as glmFit and glmLRT.

mglmSimple is not vectorized, and simply makes tag-wise calls to glm.fit. This has the advantage that it accesses all the usual information generated by glm.fit. Unfortunately, glm.fit does not always converge, and the tag-wise fitting is relatively slow.

All these functions treat the dispersion parameter of the negative binomial distribution as a known input.

deviances.function simply chooses the appropriate deviance function to use given a scalar or vector of dispersion parameters. If the dispersion values are zero, then the Poisson deviance function is returned; if the dispersion values are positive, then the negative binomial deviance function is returned.

Value

mglmOneGroup produces a vector of length equal to the number of tags/genes (number of rows of y) providing the single coefficent from the GLM fit for each tag/gene. This can be interpreted as a measure of the 'average expression' level of the tag/gene.

mglmLS produces a list with the following components:

coefficients	matrix of estimated coefficients for the linear models
fitted	matrix of fitted values
fail	vector of indices of tags that fail the line search, in that the maximum number of step-halvings in exceeded

not.converged
: vector of indices of tags that exceed the iteration limit before satisying the convergence criterion

mglmSimple produces a list with the following components:

coefficients
: matrix of estimated coefficients for the linear models

df.residual
: vector of residual degrees of freedom for the linear models

deviance
: vector of deviances for the linear models

design
: matrix giving the experimental design that was used for each of the linear models

offset
: scalar, vector or matrix of offset values used for the linear models

dispersion
: scalar or vector of the dispersion values used for the linear model fits

weights
: matrix of final weights for the observations from the linear model fits

fitted.values
: matrix of fitted values

deviances.function returns a function to calculate the deviance as appropriate for the given values of the dispersion.

Author(s)

Davis McCarthy, Yunshun Chen, Gordon Smyth

See Also

glmFit, for more complicated GLM modelling for DGE data.

Examples

```
y<-matrix(rnbinom(1000,mu=10,size=2),ncol=4)
dispersion <- 0.1
## Fit the NB GLM to the counts
ave.expression <- mglmOneGroup(y, dispersion=dispersion)
head(ave.expression)
## Fit the NB GLM to the counts with a given design matrix
f1<-factor(c(1,1,2,2))
f2<-factor(c(1,2,1,2))
x<-model.matrix(~f1+f2)
ave.expression <- mglmLS(y, x, dispersion=dispersion)
head(ave.expression$coef)
```

---

plotMDS.dge *Multidimensional scaling plot of SAGE data*

Description

Plot the sample relations based on Multidimensional Scaling.

Usage

```
plotMDS.dge(x, top=500, labels=colnames(x), col=NULL, cex=1, dim.plot=c(1,2), nd
```

Arguments

x	any matrix or DGEList object.
top	number of top genes used to calculate pairwise distances.
labels	character vector of sample names or labels. If x has no column names, then defaults the index of the samples.
col	numeric or character vector of colors for the plotting characters.
cex	numeric vector of plot symbol expansions.
dim.plot	which two dimensions should be plotted, numeric vector of length two.
ndim	number of dimensions in which data is to be represented
...	any other arguments are passed to plot.

Details

This function is a variation on the usual multdimensional scaling (or principle coordinate) plot, in that a distance measure particularly appropriate for the digital gene expression (DGE) context is used. The distance between each pair of samples (columns) is the square root of the common dispersion for the top top genes which best distinguish that pair of samples. These top top genes are selected according to the tagwise dispersion of all the samples.

See text for possible values for col and cex.

Value

A plot is created on the current graphics device.

Author(s)

Yunshun Chen and Gordon Smyth

Examples

```
# Simulate DGE data for 1000 genes(tags) and 6 samples.
# Samples are in two groups
# First 300 genes are differentially expressed in second group

x <- 10*runif(1000)
counts <- rnbinom(6000, size = 5, mu = x)
m <- matrix(counts, 1000, 6)
rownames(m) <- paste("Gene",1:1000)
m[1:300,4:6] <- m[1:300,4:6] + 10
plotMDS.dge(m)

# Indexes of samples are plotted.
plotMDS.dge(m,  col=c(rep("black",3), rep("red",3)) )
```

`plotSmear` *Plots log-Fold Change versus log-Concentration (or, M versus A) for Count Data*

Description

Both of these functions plot the log-fold change (i.e. the log of the ratio of expression levels for each tag between two experimental groups) against the log-concentration (i.e. the overall average expression level for each tag across the two groups). To represent counts that were low (e.g. zero in 1 library and non-zero in the other) in one of the two conditions, a 'smear' of points at low A value is presented in `plotSmear`.

Usage

```
plotSmear(object, pair = NULL, de.tags=NULL, xlab = "logConc", ylab =
"logFC", pch = 19, cex = 0.2, smearWidth = 0.5, panel.first=grid(),
smooth.scatter=FALSE, lowess=FALSE, ...)
```

Arguments

`object`	`DGEList` or `DGELRT` object containing data to produce an MA-plot.
`pair`	pair of experimental conditions to plot (if `NULL`, the first two conditions are used)
`de.tags`	rownames for tags identified as being differentially expressed; use `exactTest` to identify DE genes
`xlab`	x-label of plot
`ylab`	y-label of plot
`pch`	scalar or vector giving the character(s) to be used in the plot; default value of `19` gives a round point.
`cex`	character expansion factor, numerical value giving the amount by which plotting text and symbols should be magnified relative to the default; default `cex=0.2` to make the plotted points smaller
`smearWidth`	width of the smear
`panel.first`	an expression to be evaluated after the plot axes are set up but before any plotting takes place; the default `grid()` draws a background grid to aid interpretation of the plot
`smooth.scatter`	logical, whether to produce a 'smooth scatter' plot using the KernSmooth::smoothScatter function or just a regular scatter plot; default is `FALSE`, i.e. produce a regular scatter plot
`lowess`	logical, indicating whether or not to add a lowess curve to the MA-plot to give an indication of any trend in teh log-fold change with log-concentration
`...`	further arguments passed on to `plot`

Details

plotSmear is a more sophisticated and superior way to produce an 'MA plot'. plotSmear resolves the problem of plotting tags that have a total count of zero for one of the groups by adding the 'smear' of points at low A value. The points to be smeared are identified as being equal to the minimum estimated concentration in one of the two groups. The smear is created by using random uniform numbers of width smearWidth to the left of the minimum A. plotSmear also allows easy highlighting of differentially expressed (DE) tags.

Value

A plot to the current device

Author(s)

Mark Robinson, Davis McCarthy

See Also

maPlot

Examples

```
y <- matrix(rnbinom(10000,mu=5,size=2),ncol=4)
d <- DGEList(counts=y, group=rep(1:2,each=2), lib.size=colSums(y))
rownames(d$counts) <- paste("tag",1:nrow(d$counts),sep=".")
d <- estimateCommonDisp(d)
plotSmear(d)

# find differential expression
de<-exactTest(d)

# highlighting the top 500 most DE tags
de.tags <- rownames(topTags(de, n=500)$table)
plotSmear(d, de.tags=de.tags)
```

---

q2qnbinom — *Quantile to Quantile Mapping between Negative-Binomial Distributions*

Description

Approximate quantile to quantile mapping between negative-binomial distributions with the same dispersion but different means. The Poisson distribution is a special case.

Usage

```
q2qpois(x, input.mean, output.mean)
q2qnbinom(x, input.mean, output.mean, dispersion=0)
```

Arguments

x	numeric matrix of unadjusted count data from a DGEList object
input.mean	numeric matrix of estimated mean counts for tags/genes in unadjusted libraries
output.mean	numeric matrix of estimated mean counts for tags/genes in adjusted (equalized) libraries, the same for all tags/genes in a particular group, different between groups
dispersion	numeric scalar, vector or matrix of dispersion parameters

Details

This function finds the quantile with the same left and right tail probabilities relative to the output mean as x has relative to the input mean. q2qpois is equivalent to q2qnbinom with dispersion=0.

This is the function that actually generates the pseudodata for equalizeLibSizes and required by estimateCommonDisp to adjust (normalize) the library sizes and estimate the dispersion parameter. The function takes fixed values of the estimated mean for the unadjusted libraries (input.mean) and the estimated mean for the equalized libraries (output.mean) for each tag, as well as a fixed (tagwise or common) value for the dispersion parameter (phi).

The function calculates the percentiles that the counts in the unadjusted library represent for the normal and gamma distributions with mean and variance defined by the negative binomial rules: mean=input.mean and variance=input.mean*(1+dispersion*input.mean). The percentiles are then used to obtain quantiles from the normal and gamma distributions respectively, with mean and variance now defined as above but using output.mean instead of input.mean. The function then returns as the pseudodata, i.e., equalized libraries, the arithmetic mean of the quantiles for the normal and the gamma distributions. As the actual negative binomial distribution is not used, we refer to this as a "poor man's" NB quantile adjustment function, but it has the advantage of not producing Inf values for percentiles or quantiles as occurs using the equivalent NB functions. If, for any tag, the dispersion parameter for the negative binomial model is 0, then it is equivalent to using a Poisson model. Lower tails of distributions are used where required to ensure accuracy.

Value

numeric matrix of the same size as x with quantile-adjusted pseudodata

Author(s)

Gordon Smyth

Examples

```
y<-matrix(rnbinom(10000,size=2,mu=10),ncol=4)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000,1010),2))
conc<-estimatePs(d,r=2)
N<-exp(mean(log(d$samples$lib.size)))
in.mean<-matrix(0,nrow=nrow(d$counts),ncol=ncol(d$counts))
out.mean<-matrix(0,nrow=nrow(d$counts),ncol=ncol(d$counts))
for(i in 1:2) {
in.mean[,d$samples$group==i]<-outer(conc$conc.group[,i],d$samples$lib.size[d$samples$grou
out.mean[,d$samples$group==i]<-outer(conc$conc.group[,i],rep(N,sum(d$samples$group==i)))
}
pseudo<-q2qnbinom(d$counts, input.mean=in.mean, output.mean=out.mean, dispersion=0.5)
```

readDGE *Read and Merge a Set of Files Containing DGE Data*

Description

Reads and merges a set of text files containing digital gene expression data.

Usage

```
readDGE(files, path=NULL, columns=c(1,2), group=NULL, labels=NULL, ...)
```

Arguments

files	character vector of filenames, or alternatively a data.frame with a column containing the file names of the files containing the libraries of counts and, optionally, columns containing the group to which each library belongs, descriptions of the other samples and other information.
path	character string giving the directory containing the files. The default is the current working directory.
columns	numeric vector stating which two columns contain the tag names and counts, respectively
group	vector, or preferably a factor, indicating the experimental group to which each library belongs. If group is not NULL, then this argument overrides any group information included in the files argument.
labels	character vector giving short names to associate with the libraries. Defaults to the file names.
...	other are passed to read.delim

Details

Each file is assumed to contained digital gene expression data for one sample (or library), with transcript identifiers in the first column and counts in the second column. Transcript identifiers are assumed to be unique and not repeated in any one file. By default, the files are assumed to be tab-delimited and to contain column headings. The function forms the union of all transcripts and creates one big table with zeros where necessary.

Value

DGEList object

Author(s)

Mark Robinson and Gordon Smyth

See Also

DGEList provides more information about the DGEList class and the function DGEList, which can also be used to construct a DGEList object, if readDGE is not required to read in and construct a table of counts from separate files.

Examples

```
#  Read all .txt files from current working directory

## Not run: files <- dir(pattern="*\\.txt$")
RG <- readDGE(files)
## End(Not run)
```

`splitIntoGroups` *Split the Counts or Pseudocounts from a DGEList Object According To Group*

Description

Split the counts from a DGEList object according to group, creating a list where each element consists of a numeric matrix of counts for a particular experimental group. Given a pair of groups, split pseudocounts for these groups, creating a list where each element is a matrix of pseudocounts for a particular gourp.

Usage

```
splitIntoGroups(object)
splitIntoGroupsPseudo(pseudo, group, pair)
```

Arguments

`object`	`DGEList`, object containing (at least) the elements `counts` (table of raw counts), `group` (factor indicating group) and `lib.size` (numeric vector of library sizes)
`pseudo`	numeric matrix of quantile-adjusted pseudocounts to be split
`group`	factor indicating group to which libraries/samples (i.e. columns of `pseudo` belong; must be same length as ncol(pseudo)
`pair`	vector of length two stating pair of groups to be split for the pseudocounts

Value

`splitIntoGroups` outputs a list in which each element is a matrix of count counts for an individual group. `splitIntoGroupsPseudo` outputs a list with two elements, in which each element is a numeric matrix of (pseudo-)count data for one of the groups specified.

Author(s)

Davis McCarthy

Examples

```
# generate raw counts from NB, create list object
y<-matrix(rnbinom(80,size=1,mu=10),nrow=20)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
rownames(d$counts)<-paste("tagno",1:nrow(d$counts),sep=".")
z1<-splitIntoGroups(d)

z2<-splitIntoGroupsPseudo(d$counts,d$group,pair=c(1,2))
```

subsetting *Subset DGEList and DGEExact Objects*

Description

Extract a subset of a DGEList or DGEExact object.

Usage

```
## S3 method for class 'DGEList':
object[i, j, ...]
## S3 method for class 'DGEExact':
object[i, j, ...]
```

Arguments

object	object of class DGEList or DGEExact, respectively
i,j	elements to extract. i subsets the tags or genes while j subsets the libraries. Note, columns of DGEExact objects cannot be subsetted.
...	not used

Details

i,j may take any values acceptable for the matrix components of object of class DGEList. See the Extract help entry for more details on subsetting matrices. For DGEExact objects, only rows (i.e. i) may be subsetted.

Value

An object of class DGEList or DGEExact as appropriate, holding data from the specified subset of tags/genes and libraries.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

Extract in the base package.

Examples

```
d <- matrix(rnbinom(16,size=1,mu=10),4,4)
rownames(d) <- c("a","b","c","d")
colnames(d) <- c("A1","A2","B1","B2")
d <- DGEList(counts=d,group=factor(c("A","A","B","B")))
d[1:2,]
d[1:2,2]
d[,2]
d <- estimateCommonDisp(d)
results <- exactTest(d)
results[1:2,]
# NB: cannot subset columns for DGEExact objects
```

topTags *Table of the Top Differentially Expressed Tags*

Description

Extracts the top DE tags in a data frame for a given pair of groups, ranked by p-value or absolute log-fold change.

Usage

```
topTags(object, n=10, adjust.method="BH", sort.by="p.value")
```

Arguments

object	a DGEExact object (output from exactTest) or a DGELRT object (output from glmLRT), containing the (at least) the elements table: a data frame containing the log-concentration (i.e. expression level), the log-fold change in expression between the two groups/conditions and the p-value for differential expression, for each tag. If it is a DGEExact object, then topTags will also use the comparison element, which is a vector giving the two experimental groups/conditions being compared. The object may contain other elements that are not used by topTags.
n	scalar, number of tags to display/return
adjust.method	character string stating the method used to adjust p-values for multiple testing, passed on to p.adjust
sort.by	character string, indicating whether tags should be sorted by p-value ("p.value") or absolute log-fold change ("logFC"); default is to sort by p-value.

Value

an object of class TopTags containing the following elements for the top n most differentially expressed tags as determined by sort.by.

table	a data frame containing the elements logConc, the log-average concentration/abundance for each tag in the two groups being compared, logFC, the log-abundance ratio, i.e. fold change, for each tag in the two groups being compared, p.value, exact p-value for differential expression using the NB model, adj.p.val, the p-value adjusted for multiple testing as found using p.adjust using the method specified
comparison	a vector giving the names of the two groups being compared

There is a show method for this class.

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

References

Robinson MD, Smyth GK. 'Small-sample estimation of negative binomial dispersion, with applications to SAGE data.' Biostatistics. 2008 Apr;9(2):321-32.

Robinson MD, Smyth GK. 'Moderated statistical tests for assessing differences in tag abundance.' Bioinformatics. 2007 Nov 1;23(21):2881-7.

See Also

`exactTest`, `glmLRT`, `p.adjust`.

Analogous to `topTable` in the limma package.

Examples

```
# generate raw counts from NB, create list object
y <- matrix(rnbinom(80,size=1,mu=10),nrow=20)
d <- DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
rownames(d$counts) <- paste("tag",1:nrow(d$counts),sep=".")

# estimate common dispersion and find differences in expression
# here we demonstrate the 'exact' methods, but the use of topTags is
# the same for a GLM analysis
d<-estimateCommonDisp(d)
de<-exactTest(d)

# look at top 10
topTags(de)
# Can specify how many tags to view
tp <- topTags(de, n=15)
# Here we view top 15
tp
# Or order by fold change instead
topTags(de,sort.by="logFC")
```

Tu102 *Raw Data for Several SAGE Libraries from the Zhang 1997 Science Paper.*

Description

SAGE dataset for 2 tumour samples, 2 normal samples.

Usage

```
data(Tu102)
```

Format

Data frames with 22713, 18794, 16270 and 17703 observations (for Tu102, Tu98, NC2, NC1, respectively) on the following 2 variables.

`Tag_Sequence` a character vector

`Count` a numeric vector

Source

Zhang et al. (1997) Gene Expression Profiles in Normal and Cancer Cells. *Science*, 276, 1268-72.

`weightedComLik` *Weighted Common Log-Likelihood*

Description

Allow a flexible approach to accounting for a potential dependence of the dispersion on the abundance (expression level) of tags/genes by calculating a weighted 'common' log-likelihood for each gene.

Usage

```
weightedComLik(object,l0,prop.used=0.25)
```

Arguments

`object`	`DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`l0`	matrix of the conditional log-likelihood evaluated at a variety of values for the dispersion (on the delta scale, `phi/(1 + phi)`) for each tag/gene. The matrix has number of rows equal to the number of tags/genes and number of columns equal to the number of grid values (between 0 and 1) for the dispersion at which the conditional log-likelihood is evaluated.
`prop.used`	scalar giving the proportion of tags/genes in the whole dataset to use in computing the weighted common log-likelihood for each tag/gene. Default value is 0.25, i.e. a quarter of the tags/genes in the dataset.

Details

Genes are ordered based on abundance (expression level) and for a given gene, a proportion of the genes close to it are used to compute the common log-likelihood with decreasing weight given to the genes further from the given gene. Weighting is done using the tricube weighting function. Computation can be slow relative to other functions in `edgeR`, especially if the number of genes or the number of grid values (i.e. the dimensions of l0) are large.

Value

matrix of weighted common log-likelihood values computed for each gene at each grid value for the dispersion. The matrix returned has the same dimensions as l0.

Author(s)

Davis McCarthy

Examples

```
counts<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=counts,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
d<-estimateCommonDisp(d)
ntags<-nrow(d$counts)
y<-splitIntoGroups(new("DGEList",list(counts=d$pseudo.alt,samples=d$samples)))
grid.vals<-seq(0.001,0.999,length.out=10)
l0<-0
for(i in 1:length(y)) {
    l0<-condLogLikDerDelta(y[[i]],grid.vals,der=0,doSum=FALSE)+l0
}
m0 <- ntags*weightedComLik(d,l0,prop.used=0.25) # Weights sum to 1, so need to multiply b
```

---

weightedCondLogLikDerDelta

Weighted Conditional Log-Likelihood in Terms of Delta

Description

Weighted conditional log-likelihood parameterized in terms of delta (`phi / (phi+1)`) for a given tag/gene - maximized to find the smoothed (moderated) estimate of the dispersion parameter

Usage

```
weightedCondLogLikDerDelta(y, delta, tag, prior.n=10, ntags=nrow(y[[1]]), der=0,
```

Arguments

`y`	list with elements comprising the matrices of count data (or pseudocounts) for the different groups
`delta`	delta (`phi / (phi+1)`)parameter of negative binomial
`tag`	tag/gene at which the weighted conditional log-likelihood is evaluated
`prior.n`	smoothing paramter that indicates the weight to put on the common likelihood compared to the individual tag's likelihood; default `10` means that the common likelihood is given 10 times the weight of the individual tag/gene's likelihood in the estimation of the tag/genewise dispersion
`ntags`	numeric scalar number of tags/genes in the dataset to be analysed
`der`	derivative, either 0 (the function), 1 (first derivative) or 2 (second derivative)
`doSum`	logical, whether to sum over samples or not (default `FALSE`

Details

This function computes the weighted conditional log-likelihood for a given tag, parameterized in terms of delta. The value of delta that maximizes the weighted conditional log-likelihood is converted back to the `phi` scale, and this value is the estimate of the smoothed (moderated) dispersion parameter for that particular tag. The delta scale for convenience (delta is bounded between 0 and 1).

Value

numeric scalar of function/derivative evaluated for the given tag/gene and delta

Author(s)

Mark Robinson, Davis McCarthy

Examples

```
counts<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=counts,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
y<-splitIntoGroups(d)
ll1<-weightedCondLogLikDerDelta(y,delta=0.5,tag=1,prior.n=10,der=0)
ll2<-weightedCondLogLikDerDelta(y,delta=0.5,tag=1,prior.n=10,der=1)
```

Index