Package ‘edgeR’

October 9, 2013

Version 3.2.4

Date 2013/07/14

Title Empirical analysis of digital gene expression data in R

Author Mark Robinson <mrobinson@wehi.edu.au>, Davis McCarthy <dmccarthy@wehi.edu.au>, Yunshun Chen <yuchen@wehi.edu.au>, Aaron Lun <alun@wehi.edu.au>, Gordon Smyth <smyth@wehi.edu.au>

Maintainer Mark Robinson <mrobinson@wehi.edu.au>, Davis McCarthy <dmccarthy@wehi.edu.au>, Yunshun Chen <yuchen@wehi.edu.au>,Gordon Smyth <smyth@wehi.edu.au>

Depends R (>= 2.15.0), methods, limma

Suggests MASS, statmod, splines, locfit, KernSmooth

biocViews
Bioinformatics, DifferentialExpression, SAGE,HighThroughputSequencing, RNAseq, ChIPseq

Description Differential expression analysis of RNA-seq and digital gene expression profiles with biological replication. Uses empirical Bayes estimation and exact tests based on the negative binomial distribution. Also useful for differential signal analysis with other types of genome-scale count data.

License GPL (>=2)

R topics documented:

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

Description

edgeR is a package 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.

An extensive User's Guide is available, and can be opened by typing `edgeRUsersGuide()` at the R prompt. Detailed help pages are also provided for each individual function.

The edgeR package implements original statistical methodology described in the publications below.

Author(s)

Mark Robinson <mrobinson@wehi.edu.au>, Davis McCarthy <dmccarthy@wehi.edu.au>, Yunshun Chen <yuchen@wehi.edu.au>, Aaron Lun <alun@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

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297.

Lund, SP, Nettleton, D, McCarthy, DJ, Smyth, GK (2012). Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates. *Statistical Applications in Genetics and Molecular Biology*. (Accepted 31 July 2012)

`adjustedProfileLik` *Adjusted Profile Likelihood for the Negative Binomial Dispersion Parameter*

Description

Compute adjusted profile-likelihoods for estimating the dispersion parameters of genewise negative binomial glms.

Usage

```
adjustedProfileLik(dispersion, y, design, offset, adjust=TRUE)
```

Arguments

`dispersion`	numeric scalar or vector of dispersions.
`y`	numeric matrix of counts.
`design`	numeric matrix giving the design matrix.
`offset`	numeric matrix of same size as `y` giving offsets for the log-linear models. Can be a scalor or a vector of length `ncol(y)`, in which case it is expanded out to a matrix.
`adjust`	logical, if `TRUE` then Cox-Reid adjustment is made to the log-likelihood, if `FALSE` then the log-likelihood is returned without adjustment.

Details

For each row of data, compute the adjusted profile-likelihood for estimating the dispersion parameter of the negative binomial glm. The adjusted profile likelihood is described by McCarthy et al (2012), and is based on the method of Cox and Reid (1987).

The adjusted profile likelihood is an approximate log-likelihood for the dispersion parameter, conditional on the estimated values of the coefficients in the NB log-linear models. The conditional likelihood approach is a technique for adjusting the likelihood function to allow for the fact that nuisance parameters have to be estimated in order to evaluate the likelihood. When estimating the dispersion, the nuisance parameters are the coefficients in the linear model.

This implementation calls the LAPACK library to perform the Cholesky decomposition during adjustment estimation.

Value

vector of adjusted profile log-likelihood values, one for each row of `y`.

Author(s)

Yunshun Chen, Gordon Smyth, Aaron Lun

References

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

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

`glmFit`

Examples

```
y <- matrix(rnbinom(1000, mu=10, size=2), ncol=4)
design <- matrix(1, 4, 1)
dispersion <- 0.5
apl <- adjustedProfileLik(dispersion, y, design, offset=0)
apl
```

`as.data.frame` *Turn a TopTags Object into a Dataframe*

Description

Turn a `TopTags` object into a `data.frame`.

Usage

```
## S3 method for class 'TopTags'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	an object of class `TopTags`
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If `TRUE`, setting row names and converting column names (to syntactic names) is optional.
`...`	additional arguments to be passed to or from methods.

Details

This method combines all the components of `x` which have a row for each tag (transcript) into a `data.frame`.

Value

A data.frame.

Author(s)

Gordon Smyth

See Also

`as.data.frame` in the base package.

`as.matrix` *Turn a DGEList Object into a Matrix*

Description

Turn a digital gene expression object into a numeric matrix by extracting the count values.

Usage

```
## S3 method for class 'DGEList'
as.matrix(x,...)
```

Arguments

`x`	an object of class `DGEList`.
`...`	additional arguments, not used for these methods.

Details

This method extracts the matrix of counts.

This involves loss of information, so the original data object is not recoverable.

Value

A numeric matrix.

Author(s)

Gordon Smyth

See Also

`as.matrix` in the base package or `as.matrix.RGList` in the limma package.

aveLogCPM — *Average Log Counts Per Million*

Description

Compute average log2 counts-per-million for each row of counts.

Usage

```
## S3 method for class 'DGEList'
aveLogCPM(y, normalized.lib.sizes=TRUE, prior.count=2, dispersion=0.05, ...)
## Default S3 method:
aveLogCPM(y, lib.size=NULL, prior.count=2, dispersion=0.05, ...)
```

Arguments

`y`	numeric matrix containing counts. Rows for tags and columns for libraries.
`normalized.lib.sizes`	logical, use normalized library sizes?
`lib.size`	numeric vector of library sizes. Defaults to `colSums(y)`.
`prior.count`	average value to be added to each count, to avoid infinite values on the log-scale.
`dispersion`	numeric scalar or vector of negative-binomial dispersions.
`...`	other arguments are not currently used

Details

This function uses `mglmOneGroup` to compute average counts-per-million (AveCPM) for each row of counts, and returns log2(AveCPM). An average value of `prior.count` is added to the counts before running `mglmOneGroup`.

This function is similar to `rowMeans(cpm(y, log=TRUE, ...))`, but with the refinement that larger library sizes are given more weight in the average. This function converges to `rowMeans(cpm(y, log=TRUE, ...))` for large values of `dispersion`,

Value

Numeric vector giving log2(AveCPM) for each row of `y`.

Author(s)

Gordon Smyth

See Also

See `cpm` for individual logCPM values, rather than tagwise averages.

The computations for `aveLogCPM` are done by `mglmOneGroup`.

Examples

```
y <- matrix(c(0,100,30,40),2,2)
lib.size <- c(1000,10000)

# With disp large, the function is equivalent to row-wise averages of individual cpms:
aveLogCPM(y, dispersion=1e4)
cpm(y, log=TRUE, prior.count=2)

# With disp=0, the function is equivalent to pooling the counts before dividing by lib.size:
aveLogCPM(y,prior.count=0,dispersion=0)
cpms <- rowSums(y)/sum(lib.size)*1e6
log2(cpms)
```

---

binomTest — *Exact Binomial Tests for Comparing Two Digital Libraries*

Description

Computes p-values for differential abundance for each tag between two digital libraries, conditioning on the total count for each tag. The counts in each group as a proportion of the whole are assumed to follow a binomial distribution.

Usage

```
binomTest(y1, y2, n1=sum(y1), n2=sum(y2), p=n1/(n1+n2))
```

Arguments

`y1`	integer vector giving counts in first library. Non-integer values are rounded to the nearest integer.
`y2`	integer vector giving counts in second library. Of same length as `x`. Non-integer values are rounded to the nearest integer.
`n1`	total number of tags in first library. Non-integer values are rounded to the nearest integer. Not required if `p` is supplied.
`n2`	total number of tags in second library. Non-integer values are rounded to the nearest integer. Not required if `p` is supplied.
`p`	expected proportion of `y1` to the total under the null hypothesis.

Details

This function can be used to compare two libraries from SAGE, RNA-Seq, ChIP-Seq or other sequencing technologies with respect to technical variation.

An exact two-sided binomial test is computed for each tag. This test is closely related to Fisher's exact test for 2x2 contingency tables but, unlike Fisher's test, it conditions on the total number of counts for each tag. The null hypothesis is that the expected counts are in the same proportions as the library sizes, i.e., that the binomial probability for the first library is `n1/(n1+n2)`.

The two-sided rejection region is chosen analogously to Fisher's test. Specifically, the rejection region consists of those values with smallest probabilities under the null hypothesis.

When the counts are reasonably large, the binomial test, Fisher's test and Pearson's chisquare all give the same results. When the counts are smaller, the binomial test is usually to be preferred in this context.

This function replaces the earlier `sage.test` functions in the statmod and sagenhaft packages. It produces the same results as `binom.test` in the stats packge, but is much faster.

Value

Numeric vector of p-values.

Author(s)

Gordon Smyth

References

http://en.wikipedia.org/wiki/Binomial_test

http://en.wikipedia.org/wiki/Fisher's_exact_test

http://en.wikipedia.org/wiki/Serial_analysis_of_gene_expression

http://en.wikipedia.org/wiki/RNA-Seq

See Also

`sage.test` (statmod package), `binom.test` (stats package)

Examples

```
binomTest(c(0,5,10),c(0,30,50),n1=10000,n2=15000)
#  Univariate equivalents:
binom.test(5,5+30,p=10000/(10000+15000))$p.value
binom.test(10,10+50,p=10000/(10000+15000))$p.value
```

---

calcNormFactors *Calculate Normalization Factors to Align Columns of a Count Matrix*

Description

Calculate normalization factors to scale the raw library sizes.

Usage

```
calcNormFactors(object, method=c("TMM","RLE","upperquartile","none"), refColumn = NULL,
      logratioTrim = .3, sumTrim = 0.05, doWeighting=TRUE, Acutoff=-1e10, p=0.75)
```

Arguments

`object`	either a `matrix` of raw (read) counts or a `DGEList` object
`method`	normalization method to be used
`refColumn`	column to use as reference for `method="TMM"`. Can be a column number or a numeric vector of length `nrow(object))`.
`logratioTrim`	amount of trim to use on log-ratios ("M" values) for `method="TMM"`
`sumTrim`	amount of trim to use on the combined absolute levels ("A" values) for `method="TMM"`
`doWeighting`	logical, whether to compute (asymptotic binomial precision) weights for `method="TMM"`
`Acutoff`	cutoff on "A" values to use before trimming for `method="TMM"`
`p`	percentile (between 0 and 1) of the counts that is aligned when `method="upperquartile"`

Details

`method="TMM"` is the weighted trimmed mean of M-values (to the reference) proposed by Robinson and Oshlack (2010), 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.

`method="RLE"` is the scaling factor method proposed by Anders and Huber (2010). We call it "relative log expression", as 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.

`method="upperquartile"` is the upper-quartile normalization method of Bullard et al (2010), in which the scale factors are calculated from the 75% quantile of the counts for each library, after removing transcripts which are zero in all libraries. This idea is generalized here to allow scaling by any quantile of the distributions.

If `method="none"`, then the normalization factors are set to 1.

For symmetry, normalization factors are adjusted to multiply to 1. The effective library size is then the original library size multiplied by the scaling factor.

Note that rows that have zero counts for all columns are trimmed before normalization factors are computed. Therefore rows with all zero counts do not affect the estimated factors.

Value

If `object` is a `matrix`, the output is a vector with length `ncol(object)` giving the relative normalization factors. If `object` is a `DGEList`, then it is returned as output with the relative normalization factors in `object$samples$norm.factors`.

Author(s)

Mark Robinson, Gordon Smyth

References

Anders, S, Huber, W (2010). Differential expression analysis for sequence count data *Genome Biology* 11, R106.

Bullard JH, Purdom E, Hansen KD, Dudoit S. (2010) Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC Bioinformatics* 11, 94. A scaling normalization method for differential expression analysis of RNA-seq data.

Robinson MD, Oshlack A (2010). *Genome Biology* 11, R25.

Examples

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

`camera.DGEList` *Competitive Gene Set Test for Digital Gene Expression Data Accounting for Inter-gene Correlation*

Description

Test whether a set of genes is highly ranked relative to other genes in terms of differential expression, accounting for inter-gene correlation.

Usage

```
## S3 method for class 'DGEList'
camera(y, index, design, contrast=ncol(design), weights=NULL, use.ranks=FALSE, allow.neg.cor=TRUE, tre
```

Arguments

`y`	`DGEList` object.
`index`	an index vector or a list of index vectors. Can be any vector such that `y[indices,]` selects the rows corresponding to the test set.
`design`	design matrix.
`contrast`	contrast of the linear model coefficients for which the test is required. Can be an integer specifying a column of `design`, or else a numeric vector of same length as the number of columns of `design`.
`weights`	can be a numeric matrix of individual weights, of same size as `y`, or a numeric vector of array weights with length equal to `ncol(y)`.
`use.ranks`	do a rank-based test (`TRUE`) or a parametric test (`FALSE`?
`allow.neg.cor`	should reduced variance inflation factors be allowed for negative correlations?
`trend.var`	logical, should an empirical Bayes trend be estimated? See `eBayes` for details.
`sort`	logical, should the results be sorted by p-value?

Details

This function implements a method proposed by Wu and Smyth (2012) for the digital gene expression data, eg. RNA-Seq data. `camera` performs a *competitive* test in the sense defined by Goeman and Buhlmann (2007). It tests whether the genes in the set are highly ranked in terms of differential expression relative to genes not in the set. It has similar aims to `geneSetTest` but accounts for inter-gene correlation. See `roast.DGEList` for an analogous *self-contained* gene set test.

The function can be used for any sequencing experiment which can be represented by a Negative Binomial generalized linear model. The design matrix for the experiment is specified as for the `glmFit` function, and the contrast of interest is specified as for the `glmLRT` function. This allows users to focus on differential expression for any coefficient or contrast in a model by giving the vector of test statistic values.

`camera` estimates p-values after adjusting the variance of test statistics by an estimated variance inflation factor. The inflation factor depends on estimated genewise correlation and the number of genes in the gene set.

Value

A data.frame. See `camera` for details.

Author(s)

Yunshun Chen, Gordon Smyth

References

Wu, D, and Smyth, GK (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Research* 40, e133. http://nar.oxfordjournals.org/content/40/17/e133

Goeman, JJ, and Buhlmann, P (2007). Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics* 23, 980-987.

See Also

`roast.DGEList`, `camera`.

Examples

```
mu <- matrix(10, 100, 4)
group <- factor(c(0,0,1,1))
design <- model.matrix(~group)

# First set of 10 genes that are genuinely differentially expressed
iset1 <- 1:10
mu[iset1,3:4] <- mu[iset1,3:4]+10

# Second set of 10 genes are not DE
iset2 <- 11:20

# Generate counts and create a DGEList object
```

```
y <- matrix(rnbinom(100*4, mu=mu, size=10),100,4)
y <- DGEList(counts=y, group=group)

# Estimate dispersions
y <- estimateDisp(y, design)

camera(y, iset1, design)
camera(y, iset2, design)

camera(y, list(set1=iset1,set2=iset2), design)
```

`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)
```

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)

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)
ll2<-commonCondLogLikDerDelta(y,delta=0.5,der=1)
```

`condLogLikDerSize` *Conditional Log-Likelihood of the Dispersion for a Single Group of Replicate Libraries*

Description

Derivatives of the negative-binomial log-likelihood with respect to the dispersion parameter for each tag/transcript, conditional on the mean count, for a single group of replicate libraries of the same size.

Usage

```
condLogLikDerSize(y, r, der=1L)
condLogLikDerDelta(y, delta, der=1L)
```

Arguments

`y`	matrix of counts, all counts in each row having the same population mean
`r`	numeric vector or scalar, size parameter of negative binomial distribution, equal to 1/dispersion
`delta`	numeric vector or scalar, delta parameter of negative binomial, equal to dispersion/(1+dispersion)
`der`	integer specifying 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. Derivatives are with respect to either the size or the delta parametrization of the dispersion.

Value

vector of function/derivative evaluations, one for each transcript,with respect to

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

Examples

```
y <- matrix(rnbinom(10,size=1,mu=10),nrow=5)
condLogLikDerSize(y,r=1,der=1)
condLogLikDerDelta(y,delta=0.5,der=1)
```

---

cpm *Counts per Million or Reads per Kilobase per Million*

Description

Computes counts per million (CPM) or reads per kilobase per million (RPKM) values.

Usage

```
## S3 method for class 'DGEList'
cpm(x, normalized.lib.sizes=TRUE, log=FALSE, prior.count=0.25, ...)
## Default S3 method:
cpm(x, lib.size=NULL, log=FALSE, prior.count=0.25, ...)
rpkm(x, gene.length, normalized.lib.sizes=TRUE, log=FALSE, prior.count=0.25)
```

Arguments

`x`	matrix of counts or a `DGEList` object
`normalized.lib.sizes`	logical, use normalized library sizes?
`lib.size`	library size, defaults to `colSums(x)`.
`log`	logical, if `TRUE` then `log2` values are returned.
`prior.count`	average count to be added to each observation to avoid taking log of zero. Used only if `log=TRUE`.
`gene.length`	vector of length `nrow(x)` giving gene length in bases.
`...`	other arguments are not currently used

Details

CPM or RPKM values are useful descriptive measures for the expression level of a gene or transcript. By default, the normalized library sizes are used in the computation for `DGEList` objects but simple column sums for matrices.

If log-values are computed, then a small count, given by `prior.count` but scaled to be proportional to the library size, is added to `x` to avoid taking the log of zero.

Value

numeric matrix of CPM or RPKM values.

Note

`aveLogCPM(x)`, `rowMeans(cpm(x,log=TRUE))` and `log2(rowMeans(cpm(x))` all give slightly different results.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

`aveLogCPM`

Examples

```
y <- matrix(rnbinom(20,size=1,mu=10),5,4)
cpm(y)
d <- DGEList(counts=y, lib.size=1001:1004)
cpm(d)
cpm(d,log=TRUE)
```

`cutWithMinN` *Cut numeric vector into non-empty intervals*

Description

Discretizes a numeric vector. Divides the range of `x` into intervals, so that each interval contains a minimum number of values, and codes the values in `x` according to which interval they fall into.

Usage

```
cutWithMinN(x, intervals=2, min.n=1)
```

Arguments

`x`	numeric vector.
`intervals`	number of intervals required.
`min.n`	minimum number of values in any interval. Must be greater than `length(x)/intervals`.

Details

This function strikes a compromise between the base functions `cut`, which by default cuts a vector into equal length intervals, and `quantile`, which is suited to finding equally populated intervals. It finds a partition of the `x` values that is as close as possible to equal length intervals while keeping at least `min.n` values in each interval.

Tied values of `x` are broken by random jittering, so the partition may vary slightly from run to run if there are many tied values.

Value

A list with components:

`group`	integer vector of same length as `x` indicating which interval each value belongs to.
`breaks`	numeric vector of length `intervals+1` giving the left and right limits of each interval.

Author(s)

Gordon Smyth

See Also

`cut`, `quantile`.

Examples

```
x <- c(1,2,3,4,5,6,7,100)
cutWithMinN(x,intervals=3,min.n=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 list-based S4 class for for storing results of a differential expression analysis for DGE data.

List Components

For objects of this class, rows correspond to genomic features and columns to statistics associated with the differential expression analysis. The genomic features are called genes, but in reality might correspond to transcripts, tags, exons etc.

Objects of this class contain the following list components:

`table`	data frame containing columns for the log2-fold-change, `logFC`, the average log2-counts-per-million, `logCPM`
`comparison`	vector giving the two experimental groups/conditions being compared.
`genes`	a data frame containing information about each gene (can be `NULL`).

Methods

This class inherits directly from class `list`, so `DGEExact` objects can be manipulated as if they were ordinary lists. However they can also be treated as if they were matrices for the purposes of subsetting.

The dimensions, row names and column names of a `DGEExact` object are defined by those of `table`, see `dim.DGEExact` or `dimnames.DGEExact`.

`DGEExact` objects can be subsetted, see `subsetting`.

`DGEExact` objects also have a `show` method so that printing produces a compact summary of their contents.

Author(s)

edgeR team. First created by Mark Robinson and Davis McCarthy

See Also

Other classes defined in edgeR are `DGEList-class`, `DGEGLM-class`, `DGELRT-class`, `TopTags-class`

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

Description

A list-based S4 class for storing results of a GLM fit to each gene in a DGE dataset.

List Components

For objects of this class, rows correspond to genomic features and columns to coefficients in the linear model. The genomic features are called genes, but in reality might correspond to transcripts, tags, exons etc.

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 ge
`df.residual`	vector containing the residual degrees of freedom for the model fit to each gene in the dataset.
`deviance`	vector giving the deviance from the model fit to each 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 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 th
`dispersion`	scalar or vector providing the value of the dispersion parameter used in the negative binomial GLM for ea
`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 gene.
`fitted.values`	the fitted (expected) values from the GLM for each gene.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene.

Methods

This class inherits directly from class `list` so any operation appropriate for lists will work on objects of this class.

The dimensions, row names and column names of a `DGEGLM` object are defined by those of the dataset, see `dim.DGEGLM` or `dimnames.DGEGLM`.

`DGEGLM` objects can be subsetted, see `subsetting`.

`DGEGLM` objects also have a `show` method so that printing produces a compact summary of their contents.

Author(s)

edgeR team. First created by Davis McCarthy.

See Also

Other classes defined in edgeR are DGEList-class, DGEExact-class, DGELRT-class, TopTags-class

---

DGEList *DGEList Constructor*

Description

Creates a DGEList object from a table of counts (rows=features, columns=samples), group indicator for each column, library size (optional) and a table of feature annotation (optional).

Usage

```
DGEList(counts = matrix(0, 0, 0), lib.size = colSums(counts), norm.factors = rep(1,ncol(counts)),
     group = rep(1,ncol(counts)), genes = NULL, remove.zeros = FALSE)
```

Arguments

counts	numeric matrix of read counts.
lib.size	numeric vector giving the total count (sequence depth) for each library.
norm.factors	numeric vector of normalization factors that modify the library sizes.
group	vector or factor giving the experimental group/condition for each sample/library.
genes	data frame containing annotation information for the tags/transcripts/genes.
remove.zeros	logical, whether to remove rows that have 0 total count.

Details

To facilitate programming pipelines, NULL values can be input for lib.size, norm.factors or group, in which case the default value is used as if the argument had been missing.

Value

a DGEList object

Author(s)

edgeR team. First created by Mark Robinson.

See Also

DGEList-class

Examples

```
y <- matrix(rnbinom(10000,mu=5,size=2),ncol=4)
d <- DGEList(counts=y, group=rep(1:2,each=2))
dim(d)
colnames(d)
d$samples
```

DGEList-class *Digital Gene Expression data - class*

Description

A list-based S4 class for storing read counts and associated information from digital gene expression or sequencing technologies.

List Components

For objects of this class, rows correspond to genomic features and columns to samples. The genomic features are called genes, but in reality might correspond to transcripts, tags, exons etc. Objects of this class contain the following essential list components:

`counts`	numeric matrix of read counts, one row for each gene and one column for each sample.
`samples`	data.frame with a row for each sample and columns `group`, `lib.size` and `norm.factors` containing the group la

Optional components include:

`genes`	data.frame giving annotation information for each gene. Same number of rows as `counts`.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene.
`common.dispersion`	numeric scalar giving the overall dispersion estimate.
`trended.dispersion`	numeric vector giving trended dispersion estimates for each gene.
`tagwise.dispersion`	numeric vector giving tagwise dispersion estimates for each gene.
`offset`	numeric matrix of same size as `counts` giving offsets for use in log-linear models.

Methods

This class inherits directly from class `list`, so `DGEList` objects can be manipulated as if they were ordinary lists. However they can also be treated as if they were matrices for the purposes of subsetting.

The dimensions, row names and column names of a `DGEList` object are defined by those of `counts`, see `dim.DGEList` or `dimnames.DGEList`.

`DGEList` objects can be subsetted, see `subsetting`.

`DGEList` objects also have a `show` method so that printing produces a compact summary of their contents.

Author(s)

edgeR team. First created by Mark Robinson.

See Also

`DGEList` constructs DGEList objects. Other classes defined in edgeR are `DGEExact-class`, `DGEGLM-class`, `DGELRT-class`, `TopTags-class`

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

Description

A list-based S4 class for storing results of a GLM-based differential expression analysis for DGE data.

List Components

For objects of this class, rows correspond to genomic features and columns to statistics associated with the differential expression analysis. The genomic features are called genes, but in reality might correspond to transcripts, tags, exons etc.

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
`coefficients.full`	matrix containing the coefficients computed from fitting the full model (fit using `glmFit` and a given
`coefficients.null`	matrix containing the coefficients computed from fitting the null model to each gene in the dataset. T
`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 th

Methods

This class inherits directly from class `list`, so `DGELRT` objects can be manipulated as if they were ordinary lists. However they can also be treated as if they were matrices for the purposes of subsetting.

The dimensions, row names and column names of a `DGELRT` object are defined by those of `table`, see `dim.DGELRT` or `dimnames.DGELRT`.

`DGELRT` objects can be subsetted, see `subsetting`.

`DGELRT` objects also have a `show` method so that printing produces a compact summary of their contents.

Author(s)

edgeR team. First created by Davis McCarthy

See Also

Other classes defined in edgeR are `DGEList-class`, `DGEExact-class`, `DGEGLM-class`, `TopTags-class`

`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="Mean", ylab="Ave. binned standardized residual", ...)
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.DGEList`, `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 bins
```

`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)
```

`dimnames` *Retrieve the Dimension Names of a DGE 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`, `DGEExact`, `DGEGLM`, `DGELRT` or `TopTags`
`value`	a possible value for `dimnames(x)`, see `dimnames`

Details

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

Setting dimension names is currently only permitted for `DGEList` or `DGEGLM` objects.

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.

`dispBinTrend` *Estimate Dispersion Trend by Binning for NB GLMs*

Description

Estimate the abundance-dispersion trend by computing the common dispersion for bins of genes of similar AveLogCPM and then fitting a smooth curve.

Usage

```
dispBinTrend(y, design=NULL, offset=NULL, df = 5, span=0.3, min.n=400, method.bin="CoxReid",
          method.trend="spline", AveLogCPM=NULL, ...)
```

Arguments

`y`	numeric matrix of counts
`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.
`df`	degrees of freedom for spline curve.
`span`	span used for loess curve.

`min.n`	minimim number of genes in a bins.
`method.bin`	method used to estimate the dispersion in each bin. Possible values are `"CoxReid"`, `"Pearson"` or `"deviance"`.
`method.trend`	type of curve to smooth the bins. Possible values are `"spline"` for a natural cubic regression spline or `"loess"` for a linear lowess curve.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene
`...`	other arguments are passed to `estimateGLMCommonDisp`

Details

Estimate a dispersion parameter for each of many negative binomial generalized linear models by computing the common dispersion for genes sorted into bins based on overall AveLogCPM. A regression natural cubic splines or a linear loess curve is used to smooth the trend and extrapolate a value to each gene.

If there are fewer than `min.n` rows of `y` with at least one positive count, then one bin is used. The number of bins is limited to 1000.

Value

list with the following components:

`AveLogCPM`	numeric vector containing the overall AveLogCPM for each gene
`dispersion`	numeric vector giving the trended dispersion estimate for each gene
`bin.AveLogCPM`	numeric vector of length equal to `nbins` giving the average (mean) AveLogCPM for each bin
`bin.dispersion`	numeric vector of length equal to `nbins` giving the estimated common dispersion for each bin

Author(s)

Davis McCarthy and Gordon Smyth

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

`estimateGLMTrendedDisp`

Examples

```
ntags <- 1000
nlibs <- 4
means <- seq(5,10000,length.out=ntags)
y <- matrix(rnbinom(ntags*nlibs,mu=rep(means,nlibs),size=0.1*means),nrow=ntags,ncol=nlibs)
keep <- rowSums(y) > 0
y <- y[keep,]
group <- factor(c(1,1,2,2))
design <- model.matrix(~group) # Define the design matrix for the full model
out <- dispBinTrend(y, design, min.n=100, span=0.3)
with(out, plot(AveLogCPM, sqrt(dispersion)))
```

`dispCoxReid` *Estimate Common Dispersion for Negative Binomial GLMs*

Description

Estimate a common dispersion parameter across multiple negative binomial generalized linear models.

Usage

```
dispCoxReid(y, design=NULL, offset=NULL, interval=c(0,4), tol=1e-5, min.row.sum=5,
            subset=10000, AveLogCPM=NULL)
dispDeviance(y, design=NULL, offset=NULL, interval=c(0,4), tol=1e-5, min.row.sum=5,
            subset=10000, AveLogCPM=NULL, robust=FALSE, trace=FALSE)
dispPearson(y, design=NULL, offset=NULL, min.row.sum=5, subset=10000,
            AveLogCPM=NULL, tol=1e-6, trace=FALSE, initial.dispersion=0.1)
```

Arguments

`y`	numeric matrix of counts. A glm is fitted to each row.
`design`	numeric design matrix, as for `glmFit`.
`offset`	numeric vector or matrix of offsets for the log-linear models, as for `glmFit`.
`interval`	numeric vector of length 2 giving allowable values for the dispersion, passed to `optimize`.
`tol`	the desired accuracy, see `optimize` or `uniroot`.
`min.row.sum`	integer. Only rows with at least this number of counts are used.
`subset`	integer, number of rows to use in the calculation. Rows used are chosen evenly spaced by AveLogCPM.
`AveLogCPM`	numeric vector giving average log2 counts per million.
`trace`	logical, should iteration information be output?
`robust`	logical, should a robust estimator be used?
`initial.dispersion`	starting value for the dispersion

Details

These are low-level (non-object-orientated) functions called by `estimateGLMCommonDisp`.

`dispCoxReid` maximizes the Cox-Reid adjusted profile likelihood (Cox and Reid, 1987). `dispPearson` sets the average Pearson goodness of fit statistics to its (asymptotic) expected value. This is also known as the *pseudo-likelihood* estimator. `dispDeviance` sets the average residual deviance statistic to its (asymptotic) expected values. This is also known as the *quasi-likelihood* estimator.

Robinson and Smyth (2008) and McCarthy et al (2011) showed that the Pearson (pseudo-likelihood) estimator typically under-estimates the true dispersion. It can be seriously biased when the number of libraries (`ncol(y)` is small. On the other hand, the deviance (quasi-likelihood) estimator typically over-estimates the true dispersion when the number of libraries is small. Robinson and Smyth (2008) and McCarthy et al (2011) showed the Cox-Reid estimator to be the least biased of the three options.

`dispCoxReid` uses `optimize` to maximize the adjusted profile likelihood. `dispDeviance` uses `uniroot` to solve the estimating equation. The robust options use an order statistic instead the mean statistic, and have the effect that a minority of tags with very large (outlier) dispersions should have limited influence on the estimated value. `dispPearson` uses a globally convergent Newton iteration.

Value

Numeric vector of length one giving the estimated common dispersion.

Author(s)

Gordon Smyth

References

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

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

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research*. `http://nar.oxfordjournals.org/content/early/2012/02/06/nar.gks042` (Published online 28 January 2012)

See Also

`estimateGLMCommonDisp`, `optimize`, `uniroot`

Examples

```
ntags <- 100
nlibs <- 4
y <- matrix(rnbinom(ntags*nlibs,mu=10,size=10),nrow=ntags,ncol=nlibs)
group <- factor(c(1,1,2,2))
lib.size <- rowSums(y)
design <- model.matrix(~group)
disp <- dispCoxReid(y, design, offset=log(lib.size), subset=100)
```

dispCoxReidInterpolateTagwise

Estimate Tagwise Dispersion for Negative Binomial GLMs by Cox-Reid Adjusted Profile Likelihood

Description

Estimate tagwise dispersion parameters across multiple negative binomial generalized linear models using weighted Cox-Reid Adjusted Profile-likelihood and cubic spline interpolation over a tagwise grid.

Usage

```
dispCoxReidInterpolateTagwise(y, design, offset=NULL, dispersion, trend=TRUE, AveLogCPM=NULL,
        min.row.sum=5, prior.df=10, span=0.3, grid.npts=11, grid.range=c(-6,6))
```

Arguments

`y`	numeric matrix of counts
`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.
`dispersion`	numeric scalar or vector giving the dispersion(s) towards which the tagwise dispersion parameters are shrunk.
`trend`	logical, whether abundance-dispersion trend is used for smoothing.
`AveLogCPM`	numeric vector giving average log2 counts per million for each tag.
`min.row.sum`	numeric scalar giving a value for the filtering out of low abundance tags. Only tags with total sum of counts above this value are used. 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.
`prior.df`	numeric scalar, prior degsmoothing parameter that indicates the weight to give to the common likelihood compared to the individual tag's likelihood; default `getPriorN(object)` gives a value for `prior.n` that is equivalent to giving the common likelihood 20 prior degrees of freedom in the estimation of the tag/genewise dispersion.
`span`	numeric parameter between 0 and 1 specifying proportion of data to be used in the local regression moving window. Larger numbers give smoother fits.
`grid.npts`	numeric scalar, the number of points at which to place knots for the spline-based estimation of the tagwise dispersion estimates.
`grid.range`	numeric vector of length 2, giving relative range, in terms of `log2(dispersion)`, on either side of trendline for each tag for spline grid points.

Details

In the edgeR context, `dispCoxReidInterpolateTagwise` is a low-level function called by `estimateGLMTagwiseDisp`.

`dispCoxReidInterpolateTagwise` calls the function `maximizeInterpolant` to fit cubic spline interpolation over a tagwise grid.

Value

`dispCoxReidInterpolateTagwise` produces a vector of tagwise dispersions having the same length as the number of genes in the count data.

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* 49, 1-39.

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

`estimateGLMTagwiseDisp`, `maximizeInterpolant`

Examples

```
y <- matrix(rnbinom(1000, mu=10, size=2), ncol=4)
design <- matrix(1, 4, 1)
dispersion <- 0.5
d <- dispCoxReidInterpolateTagwise(y, design, dispersion=dispersion)
d
```

---

dispCoxReidSplineTrend *Estimate Dispersion Trend for Negative Binomial GLMs*

Description

Estimate trended dispersion parameters across multiple negative binomial generalized linear models using Cox-Reid adjusted profile likelihood.

Usage

```
dispCoxReidSplineTrend(y, design, offset=NULL, df = 5, subset=10000, AveLogCPM=NULL,
                          method.optim="Nelder-Mead", trace=0)
dispCoxReidPowerTrend(y, design, offset=NULL, subset=10000, AveLogCPM=NULL,
                          method.optim="Nelder-Mead", trace=0)
```

Arguments

`y`	numeric matrix of counts
`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.
`df`	integer giving the degrees of freedom of the spline function, see `ns` in the splines package.
`subset`	integer, number of rows to use in the calculation. Rows used are chosen evenly spaced by AveLogCPM using `cutWithMinN`.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene
`method.optim`	the method to be used in `optim`. See `optim` for more detail.
`trace`	logical, should iteration information be output?

Details

In the `edgeR` context, these are low-level functions called by `estimateGLMTrendedDisp`.

`dispCoxReidSplineTrend` and `dispCoxReidPowerTrend` fit abundance trends to the tagwise dispersions. `dispCoxReidSplineTrend` fits a regression spline whereas `dispCoxReidPowerTrend` fits a log-linear trend of the form `a*exp(abundance)^b+c`. In either case, `optim` is used to maximize the adjusted profile likelihood (Cox and Reid, 1987).

Value

List containing numeric vectors `dispersion` and `abundance` containing the estimated dispersion and abundance for each transcript. The vectors are of the same length as `nrow(y)`.

Author(s)

Yunshun Chen, Davis McCarthy, Gordon Smyth

References

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

See Also

`estimateGLMTrendedDisp`

Examples

```
design <- matrix(1,4,1)
y <- matrix((rnbinom(400,mu=100,size=5)),100,4)
d1 <- dispCoxReidSplineTrend(y, design, df=3)
d2 <- dispCoxReidPowerTrend(y, design)
with(d2,plot(AveLogCPM,sqrt(dispersion)))
```

`edgeRUsersGuide` *View edgeR User's Guide*

Description

Finds the location of the edgeR User's Guide and optionally opens it.

Usage

```
edgeRUsersGuide(view=TRUE)
```

Arguments

`view`	logical, should the document be opened using the default PDF document reader?

Details

The function `vignette("edgeR")` will find the short edgeR Vignette which describes how to obtain the Limma User's Guide. The User's Guide is not itself a true vignette because it is not automatically generated using `Sweave` during the package build process. This means that it cannot be found using `vignette`, hence the need for this special function.

If the operating system is other than Windows, then the PDF viewer used is that given by `Sys.getenv("R_PDFVIEWER")`. The PDF viewer can be changed using `Sys.putenv(R_PDFVIEWER=)`.

Value

Character string giving the file location. If `view=TRUE`, the PDF document reader is started and the User's Guide is opened, as a side effect.

Author(s)

Gordon Smyth

See Also

`system`

Examples

```
# To get the location:
edgeRUsersGuide(view=FALSE)
# To open in pdf viewer:
## Not run: edgeRUsersGuide()
```

`equalizeLibSizes` *Equalize Library Sizes by Quantile-to-Quantile Normalization*

Description

Adjusts counts so that the effective library sizes are equal, preserving fold-changes between groups and preserving biological variability within each group.

Usage

```
equalizeLibSizes(object, dispersion=0, common.lib.size)
```

Arguments

`object`	`DGEList` object
`dispersion`	numeric scalar or vector of `dispersion` parameters; if a scalar, then a common dispersion parameter is used for all tags
`common.lib.size`	numeric scalar, the library size to normalize to; default is the geometric mean of the original effective library sizes

Details

Thus function implements the quantile-quantile normalization method of Robinson and Smyth (2008). It computes normalized counts, or pseudo-counts, used by `exactTest` and `estimateCommonDisp`.

Note that the output common library size is a theoretical quantity. The column sums of the normalized counts, while to be exactly equal, nor are they intended to be. However, the expected counts for each tag are equal under the null hypothesis of no differential expression.

Value

A list with components

`pseudo.counts`	numeric matrix of normalized pseudo-counts
`common.lib.size`	normalized library size

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

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

`q2qnbinom`

Examples

```
ngenes <- 1000
nlibs <- 2
counts <- matrix(0,ngenes,nlibs)
colnames(counts) <- c("Sample1","Sample2")
counts[,1] <- rpois(ngenes,lambda=10)
counts[,2] <- rpois(ngenes,lambda=20)
summary(counts)
y <- DGEList(counts=counts)
out <- equalizeLibSizes(y)
summary(out$pseudo.counts)
```

`estimateCommonDisp` *Estimate Common Negative Binomial Dispersion by Conditional Maximum Likelihood*

Description

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

Usage

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

Arguments

`object`	`DGEList` object
`tol`	the desired accuracy, passed to `optimize`
`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.
`verbose`	logical, if `TRUE` estimated dispersion and BCV will be printed to standard output.

Details

Implements the method of Robinson and Smyth (2008) for estimating a common dispersion parameter by conditional maximum likelihood. 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

Returns `object` with the following added components:

`common.dispersion`	estimate of the common dispersion.
`pseudo.counts`	numeric matrix of quantile-quantile normalized counts. These are counts adjusted so that the library sizes are equal, while preserving differences between groups and variability within each group.
`pseudo.lib.size`	the common library size to which the counts have been adjusted

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

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

`equalizeLibSizes`

Examples

```
# True dispersion is 1/5=0.2
y <- matrix(rnbinom(1000,mu=10,size=5),ncol=4)
d <- DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
d <- estimateCommonDisp(d, verbose=TRUE)
```

`estimateDisp` *Estimate Common, Trended and Tagwise Negative Binomial dispersions by weighted likelihood empirical Bayes*

Description

Maximizes the negative binomial likelihood to give the estimate of the common, trended and tagwise dispersions across all tags.

Usage

```
estimateDisp(y, design=NULL, offset=NULL, prior.df=NULL, trend.method="locfit", span=NULL, grid.lengt
```

Arguments

`y`	`DGEList` object
`design`	numeric design matrix
`offset`	numeric vector or matrix of offsets for the log-linear models
`prior.df`	prior degrees of freedom. It is used in calculating `prior.n`.
`trend.method`	method for estimating dispersion trend. Possible values are `"none"`, `"movingave"`, `"loess"` and `"locfit"`.
`span`	width of the smoothing window, as a proportion of the data set.
`grid.length`	the number of points on which the interpolation is applied for each tag.
`grid.range`	the range of the grid points around the trend on a log2 scale.
`robust`	logical, should the estimation of `prior.df` be robustified against outliers?
`winsor.tail.p`	numeric vector of length 1 or 2, giving left and right tail proportions of the deviances to Winsorize when estimating `prior.df`.
`tol`	the desired accuracy, passed to `optimize`

Details

This function calculates a matrix of likelihoods for each gene at a set of dispersion grid points, and then applies weighted likelihood empirical Bayes method to obtain posterior dispersion estimates. If there is no design matrix, it calculates the quantile conditional likelihood for each gene (tag) and then maximize it. The method is same as in the function `estimateCommonDisp` and `estimateTagwiseDisp`. If a design matrix is given, it then calculates the adjusted profile log-likelihood for each gene (tag) and then maximize it. It is similar to the functions `estimateGLMCommonDisp`, `estimateGLMTrendedDisp` and `estimateGLMTagwiseDisp`.

Value

Returns `object` with the following added components:

`common.dispersion`	estimate of the common dispersion.
`trended.dispersion`	estimates of the trended dispersions.
`tagwise.dispersion`	tag- or gene-wise estimates of the dispersion parameter.
`logCPM`	the tag abundance in log average counts per million.
`prior.df`	prior degrees of freedom. It is a vector when robust method is used.
`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.
`span`	width of the smoothing window used in estimating dispersions.

Author(s)

Yunshun Chen, Gordon Smyth

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

Robinson, MD, and Smyth, GK (2007). Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics* 23, 2881-2887. `http://bioinformatics.oxfordjournals.org/content/23/21/2881`

See Also

`estimateCommonDisp`, `estimateTagwiseDisp`, `estimateGLMCommonDisp`, `estimateGLMTrendedDisp`, `estimateGLMTagwiseDisp`

Examples

```
# True dispersion is 1/5=0.2
y <- matrix(rnbinom(1000, mu=10, size=5), ncol=4)
group <- c(1,1,2,2)
design <- model.matrix(~group)
d <- DGEList(counts=y, group=group)
d1 <- estimateDisp(d)
d2 <- estimateDisp(d, design)
```

estimateExonGenewiseDisp

Estimate Genewise Dispersions from Exon-Level Count Data

Description

Estimate a dispersion value for each gene from exon-level count data by collapsing exons into the genes to which they belong.

Usage

```
estimateExonGenewiseDisp(y, geneID, group=NULL)
```

Arguments

`y`	either a matrix of exon-level counts or a `DGEList` object with (at least) elements `counts` (table of counts summarized at the exon level) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size). Each row of `y` should represent one exon.
`geneID`	vector of length equal to the number of rows of `y`, which provides the gene identifier for each exon in `y`. These identifiers are used to group the relevant exons into genes for the gene-level analysis of splice variation.
`group`	factor supplying the experimental group/condition to which each sample (column of `y`) belongs. If `NULL` (default) the function will try to extract if from `y`, which only works if `y` is a `DGEList` object.

Details

This function can be used to compute genewise dispersion estimates (for an experiment with a one-way, or multiple group, layout) from exon-level count data. `estimateCommonDisp` and `estimateTagwiseDisp` are used to do the computation and estimation, and the default arguments for those functions are used.

Value

`estimateExonGenewiseDisp` returns a vector of genewise dispersion estimates, one for each unique `geneID`.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

`estimateCommonDisp` and related functions for estimating the dispersion parameter for the negative binomial model.

Examples

```
# generate exon counts from NB, create list object
y<-matrix(rnbinom(40,size=1,mu=10),nrow=10)
d<-DGEList(counts=y,group=rep(1:2,each=2))
genes <- rep(c("gene.1","gene.2"), each=5)
estimateExonGenewiseDisp(d, genes)
```

`estimateGLMCommonDisp` *Estimate Common Dispersion for Negative Binomial GLMs*

Description

Estimates a common negative binomial dispersion parameter for a DGE dataset with a general experimental design.

Usage

```
## S3 method for class 'DGEList'
estimateGLMCommonDisp(y, design=NULL, offset=NULL, method="CoxReid", subset=10000, AveLogCPM=NULL, ve
## Default S3 method:
estimateGLMCommonDisp(y, design=NULL, offset=NULL, method="CoxReid", subset=10000, AveLogCPM=NULL, ve
```

Arguments

`y`	object containing read counts, as for `glmFit`.
`design`	numeric design matrix, as for `glmFit`.
`offset`	numeric vector or matrix of offsets for the log-linear models, as for `glmFit`.
`method`	method for estimating the dispersion. Possible values are `"CoxReid"`, `"Pearson"` or `"deviance"`.
`subset`	maximum number of rows of `y` to use in the calculation. Rows used are chosen evenly spaced by AveLogCPM using `systematicSubset`.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene
`verbose`	logical, if `TRUE` estimated dispersion and BCV will be printed to standard output.
`...`	other arguments are passed to lower-level functions. See `dispCoxReid`, `dispPearson` and `dispDeviance` for details.

Details

This function calls `dispCoxReid`, `dispPearson` or `dispDeviance` depending on the `method` specified. See `dispCoxReid` for details of the three methods and a discussion of their relative performance.

Value

The default method returns a numeric vector of length 1 containing the estimated dispersion.

The `DGEList` method returns the same `DGEList` `y` as input but with `common.dispersion` as an added component.

Author(s)

Gordon Smyth, Davis McCarthy, Yunshun Chen

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. http://nar.oxfordjournals.org/content/40/10/4288

See Also

dispCoxReid, dispPearson, dispDeviance

estimateGLMTrendedDisp for trended dispersion and estimateGLMTagwiseDisp for tagwise dispersions in the context of a generalized linear model.

estimateCommonDisp for common dispersion or estimateTagwiseDisp for tagwise dispersion in the context of a multiple group experiment (one-way layout).

Examples

```
#  True dispersion is 1/size=0.1
y <- matrix(rnbinom(1000,mu=10,size=10),ncol=4)
d <- DGEList(counts=y,group=c(1,1,2,2))
design <- model.matrix(~group, data=d$samples)
d1 <- estimateGLMCommonDisp(d, design, verbose=TRUE)

#  Compare with classic CML estimator:
d2 <- estimateCommonDisp(d, verbose=TRUE)

#  See example(glmFit) for a different example
```

---

estimateGLMTagwiseDisp *Empirical Bayes Tagwise Dispersions for Negative Binomial GLMs*

Description

Compute an empirical Bayes estimate of the negative binomial dispersion parameter for each tag or transcript, with expression levels specified by a log-linear model.

Usage

```
## S3 method for class 'DGEList'
estimateGLMTagwiseDisp(y, design=NULL, offset=NULL, dispersion=NULL, prior.df=10,
          trend=!is.null(y$trended.dispersion), span=NULL, AveLogCPM=NULL, ...)
## Default S3 method:
estimateGLMTagwiseDisp(y, design=NULL, offset=NULL, dispersion, prior.df=10,
          trend=TRUE, span=NULL, AveLogCPM=NULL, ...)
```

Arguments

`y`	matrix of counts or a `DGEList` object.
`design`	numeric design matrix, as for `glmFit`.
`trend`	logical. Should the prior be the trended dispersion (`TRUE`) or the common dispersion (`FALSE`)?
`offset`	offset matrix for the log-linear model, as for `glmFit`. Defaults to the log-effective library sizes.
`dispersion`	common or trended dispersion estimates, used as an initial estimate for the tagwise estimates. By default uses values stored in the `DGEList` object.
`prior.df`	prior degrees of freedom.
`span`	width of the smoothing window, in terms of proportion of the data set. Default value decreases with the number of tags.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene
`...`	other arguments are passed to `dispCoxReidInterpolateTagwise`.

Details

This function implements the empirical Bayes strategy proposed by McCarthy et al (2012) for estimating the tagwise negative binomial dispersions. The experimental conditions are specified by design matrix allowing for multiple explanatory factors. The empirical Bayes posterior is implemented as a conditional likelihood with tag-specific weights, and the conditional likelihood is computed using Cox-Reid approximate conditional likelihood (Cox and Reid, 1987).

The prior degrees of freedom determines the weight given to the global dispersion trend. The larger the prior degrees of freedom, the more the tagwise dispersions are squeezed towards the global trend.

This function calls the lower-level function `dispCoxReidInterpolateTagwise`.

Value

`estimateGLMTagwiseDisp.DGEList` produces a `DGEList` object, which contains the tagwise dispersion parameter estimate for each tag for the negative binomial model that maximizes the Cox-Reid adjusted profile likelihood. The tagwise dispersions are simply added to the `DGEList` object provided as the argument to the function.

`estimateGLMTagwiseDisp.default` returns a vector of the tagwise dispersion estimates.

Author(s)

Gordon Smyth, Davis McCarthy

References

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

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

estimateGLMCommonDisp for common dispersion and estimateGLMTrendedDisp for trended dispersion in the context of a generalized linear model.

estimateCommonDisp for common dispersion or estimateTagwiseDisp for tagwise dispersion in the context of a multiple group experiment (one-way layout).

Examples

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

---

estimateGLMTrendedDisp *Estimate Trended Dispersion for Negative Binomial GLMs*

Description

Estimates the abundance-dispersion trend by Cox-Reid approximate profile likelihood.

Usage

```
## S3 method for class 'DGEList'
estimateGLMTrendedDisp(y, design=NULL, offset=NULL, AveLogCPM=NULL, method="auto", ...)
## Default S3 method:
estimateGLMTrendedDisp(y, design=NULL, offset=NULL, AveLogCPM=NULL, method="auto", ...)
```

Arguments

y	a matrix of counts or a DGEList object.)
design	numeric design matrix, as for glmFit.
method	method (low-level function) used to estimated the trended dispersions. Possible values are "auto" (default, switch to "bin.spline" method if the number of tags is great than 200 and "power" method otherwise),"bin.spline", "bin.loess" (which both result in a call to dispBinTrend), "power" (call to dispCoxReidPowerTrend), or "spline" (call to dispCoxReidSplineTrend).
offset	numeric scalar, vector or matrix giving the linear model offsets, as for glmFit.
AveLogCPM	numeric vector giving average log2 counts per million for each gene.
...	other arguments are passed to lower-level functions dispBinTrend, dispCoxReidPowerTrend or dispCoxReidSplineTrend.

Details

Estimates the dispersion parameter for each transcript (tag) with a trend that depends on the overall level of expression for the transcript 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.

The function provides an object-orientated interface to lower-level functions.

Value

When the input object is a `DGEList`, `estimateGLMTrendedDisp` produces a `DGEList` object, which contains the estimates of the trended dispersion parameter for the negative binomial model according to the method applied.

When the input object is a numeric matrix, the output of one of the lower-level functions `dispBinTrend`, `dispCoxReidPowerTrend` of `dispCoxReidSplineTrend` is returned.

Author(s)

Gordon Smyth, Davis McCarthy, Yunshun Chen

References

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

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

`dispBinTrend`, `dispCoxReidPowerTrend` and `dispCoxReidSplineTrend` for details on how the calculations are done.

Examples

```
ntags <- 250
nlibs <- 4
y <- matrix(rnbinom(ntags*nlibs,mu=10,size=10),ntags,nlibs)
d <- DGEList(counts=y,group=c(1,1,2,2),lib.size=c(1000:1003))
design <- model.matrix(~group, data=d$samples)
disp <- estimateGLMTrendedDisp(d, design, min.n=25, df=3)
plotBCV(disp)
```

estimateTagwiseDisp *Estimate Empirical Bayes Tagwise Dispersion Values*

Description

Estimates tagwise dispersion values by an empirical Bayes method based on weighted conditional maximum likelihood.

Usage

```
estimateTagwiseDisp(object, prior.df=10, trend="movingave", span=NULL, method="grid",
          grid.length=11, grid.range=c(-6,6), tol=1e-06, verbose=FALSE)
```

Arguments

`object`	object of class `DGEList` 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.df`	prior degrees of freedom.
`trend`	method for estimating dispersion trend. Possible values are `"none"`, `"movingave"` and `"loess"`.
`span`	width of the smoothing window, as a proportion of the data set.
`method`	method for maximizing the posterior likelihood. Possible values are `"grid"` for interpolation on grid points or `"optimize"` to call the function of the same name.
`grid.length`	for `method="grid"`, the number of points on which the interpolation is applied for each tag.
`grid.range`	for `method="grid"`, the range of the grid points around the trend on a log2 scale.
`tol`	for `method="optimize"`, the tolerance for Newton-Rhapson iterations.
`verbose`	logical, if `TRUE` then diagnostic ouput is produced during the estimation process.

Details

This function implements the empirical Bayes strategy proposed by Robinson and Smyth (2007) for estimating the tagwise negative binomial dispersions. The experimental design is assumed to be a oneway layout with one or more experimental groups. The empirical Bayes posterior is implemented as a conditional likelihood with tag-specific weights.

The prior values for the dispersions are determined by a global trend. The individual tagwise dispersions are then squeezed towards this trend. The prior degrees of freedom determines the weight given to the prior. The larger the prior degrees of freedom, the more the tagwise dispersions are squeezed towards the global trend. If the number of libraries is large, the prior becomes less important and the tagwise dispersion are determined more by the individual tagwise data.

If `trend="none"`, then the prior dispersion is just a constant, the common dispersion. Otherwise, the trend is determined by a moving average (`trend="movingave"`) or loess smoother applied to the tagwise conditional log-likelihood. `method="loess"` applies a loess curve of degree 0 as implemented in `loessByCol`.

`method="optimize"` is not recommended for routine use as it is very slow. It is included for testing purposes.

Value

An object of class `DGEList` with the same components as for `estimateCommonDisp` plus the following:

`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

Author(s)

Mark Robinson, Davis McCarthy, Yunshun Chen and Gordon Smyth

References

Robinson, MD, and Smyth, GK (2007). Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics* 23, 2881-2887. `http://bioinformatics.oxfordjournals.org/content/23/21/2881`

See Also

`estimateCommonDisp` is usually run before `estimateTagwiseDisp`.

`movingAverageByCol` and `loessByCol` implement the moving average or loess smoothers.

Examples

```
# See exactTest
```

estimateTrendedDisp *Estimate Empirical Bayes Trended Dispersion Values*

Description

Estimates trended dispersion values by an empirical Bayes method.

Usage

```
estimateTrendedDisp(object, method="bin.spline", df=5, span=2/3)
```

Arguments

`object`	object of class `DGEList` 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`
`method`	method used to estimated the trended dispersions. Possible values are `"spline"`, and `"loess"`.
`df`	integer giving the degrees of freedom of the spline function if `"spline"` method is used, see `ns` in the splines package. Default is 5.
`span`	scalar, passed to `loess` to determine the amount of smoothing for the loess fit when `"loess"` method is used. Default is `2/3`.

Details

This function takes the binned common dispersion and abundance, and fits a smooth curve through these binned values using either natural cubic splines or loess. From this smooth curve it predicts the dispersion value for each gene based on the gene's overall abundance. This results in estimates for the NB dispersion parameter which have a dependence on the overall expression level of the gene, and thus have an abundance-dependent trend.

Value

An object of class `DGEList` with the same components as for `estimateCommonDisp` plus the trended dispersion estimates for each gene or tag.

Author(s)

Yunshun Chen and Gordon Smyth

See Also

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

Examples

```
y <- matrix(rnbinom(6000, mu=100, size=10), 1000, 6)
group <- c(0,0,0,1,1,1)
d <- DGEList(y, group=group)
d <- estimateCommonDisp(d)
d <- estimateTrendedDisp(d)
```

exactTest	*Exact Tests for Differences between Two Groups of Negative-Binomial Counts*

Description

Compute genewise exact tests for differences in the means between two groups of negative-binomially distributed counts.

Usage

```
exactTest(object, pair=1:2, dispersion="auto", rejection.region="doubletail",
          big.count=900, prior.count=0.125)
exactTestDoubleTail(y1, y2, dispersion=0, big.count=900)
exactTestBySmallP(y1, y2, dispersion=0, big.count=900)
exactTestByDeviance(y1, y2, dispersion=0, big.count=900)
exactTestBetaApprox(y1, y2, dispersion=0)
```

Arguments

`object`	an object of class `DGEList`.
`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. Note that the first group listed in the pair is the baseline for the comparison—so if the pair is `c("A","B")` then the comparison is `B - A`, so genes with positive log-fold change are up-regulated in group B compared with group A (and vice versa for genes with negative log-fold change).
`dispersion`	either a numeric vector of dispersions or a character string indicating that dispersions should be taken from the data object. If a numeric vector, then can be either of length one or of length equal to the number of tags. Allowable character values are `"common"`, `"trended"`, `"tagwise"` or `"auto"`. Default behavior (`"auto"` is to use most complex dispersions found in data object.
`rejection.region`	type of rejection region for two-sided exact test. Possible values are `"doubletail"`, `"smallp"` or `"deviance"`.
`big.count`	count size above which asymptotic beta approximation will be used.
`prior.count`	average prior count used to shrink log-fold-changes. Larger values produce more shrinkage.
`y1`	numeric matrix of counts for the first the two experimental groups to be tested for differences. Rows correspond to genes or transcripts and columns to libraries. Libraries are assumed to be equal in size - e.g. adjusted pseudocounts from the output of `equalizeLibSizes`.

`y2`	numeric matrix of counts for the second of the two experimental groups to be tested for differences. Rows correspond to genes or transcripts and columns to libraries. 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`.

Details

The functions test for differential expression between two groups of count libraries. They implement the exact test proposed by Robinson and Smyth (2008) for a difference in mean between two groups of negative binomial random variables. The functions accept two groups of count libraries, and a test is performed for each row of data. For each row, the test is conditional on the sum of counts for that row. The test can be viewed as a generalization of the well-known exact binomial test (implemented in `binomTest`) but generalized to overdispersed counts.

`exactTest` is the main user-level function, and produces an object containing all the necessary components for downstream analysis. `exactTest` calls one of the low level functions `exactTestDoubleTail`, `exactTestBetaApprox`, `exactTestBySmallP` or `exactTestByDeviance` to do the p-value computation. The low level functions all assume that the libraries have been normalized to have the same size, i.e., to have the same expected column sum under the null hypothesis. `exactTest` equalizes the library sizes using `equalizeLibSizes` before calling the low level functions.

The functions `exactTestDoubleTail`, `exactTestBySmallP` and `exactTestByDeviance` correspond to different ways to define the two-sided rejection region when the two groups have different numbers of samples. `exactTestBySmallP` implements the method of small probabilities as proposed by Robinson and Smyth (2008). This method corresponds exactly to `binomTest` as the dispersion approaches zero, but gives poor results when the dispersion is very large. `exactTestDoubleTail` computes two-sided p-values by doubling the smaller tail probability. `exactTestByDeviance` uses the deviance goodness of fit statistics to define the rejection region, and is therefore equivalent to a conditional likelihood ratio test.

Note that `rejection.region="smallp"` is no longer recommended. It is preserved as an option only for backward compatiblity with early versions of edgeR. `rejection.region="deviance"` has good theoretical statistical properties but is relatively slow to compute. `rejection.region="doubletail"` is just slightly more conservative than `rejection.region="deviance"`, but is recommended because of its much greater speed. For general remarks on different types of rejection regions for exact tests see Gibbons and Pratt (1975).

`exactTestBetaApprox` implements an asymptotic beta distribution approximation to the conditional count distribution. It is called by the other functions for rows with both group counts greater than `big.count`.

Value

`exactTest` produces an object of class `DGEExact` containing the following components:

`table`	data frame containing columns for the log2-fold-change, `logFC`, the average log2-counts-per-million, `logCPM`, and the two-sided p-value `PValue`
`comparison`	character vector giving the names of the two groups being compared
`genes`	optional data frame containing annotation for transcript; taken from `object`

The low-level functions, `exactTestDoubleTail` etc, produce a numeric vector of genewise p-values, one for each row of `y1` and `y2`.

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

References

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

Gibbons, JD and Pratt, JW (1975). P-values: interpretation and methodology. *The American Statistician* 29, 20-25.

See Also

`equalizeLibSizes`, `binomTest`

Examples

```
# generate raw counts from NB, create list object
y <- matrix(rnbinom(80,size=1/0.2,mu=10),nrow=20,ncol=4)
d <- DGEList(counts=y, group=c(1,1,2,2), lib.size=rep(1000,4))

de <- exactTest(d, dispersion=0.2)
topTags(de)

# same p-values using low-level function directly
p.value <- exactTestDoubleTail(y[,1:2], y[,3:4], dispersion=0.2)
sort(p.value)[1:10]
```

`expandAsMatrix` *expandAsMatrix*

Description

Expand scalar or vector to a matrix.

Usage

```
expandAsMatrix(x, dim)
```

Arguments

`x`	scalar, vector or matrix. If a vector, length must match one of the output dimensions.
`dim`	required dimension for the output matrix.

Details

This function expands a row or column vector to be a matrix. It is used internally in edgeR to convert offsets to a matrix.

Value

Numeric matrix of dimension `dim`.

Author(s)

Gordon Smyth

See Also

`mglmLS`.

Examples

```
expandAsMatrix(1:3,c(4,3))
expandAsMatrix(1:4,c(4,3))
```

`getCounts` *Extract Specified Component of a DGEList Object*

Description

`getCounts(y)` returns the matrix of read counts `y$counts`.

`getOffset(y)` returns offsets for the log-linear predictor account for sequencing depth and possibly other normalization factors. Specifically it returns the matrix `y$offset` if it is non-null, otherwise it returns the log product of `lib.size` and `norm.factors` from `y$samples`.

`getDispersion(y)` returns the most complex dispersion estimates (common, trended or tagwise) found in `y`.

Usage

```
getCounts(y)
getOffset(y)
getDispersion(y)
```

Arguments

`y`	`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 the matrix of counts. `getOffset` returns a numeric matrix or vector. `getDispersion` returns vector of dispersion values.

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

See Also

`DGEList-class`

Examples

```
# generate raw counts from NB, create list object
y <- matrix(rnbinom(20,size=5,mu=10),5,4)
d <- DGEList(counts=y, group=c(1,1,2,2), lib.size=1001:1004)
getCounts(d)
getOffset(d)
d <- estimateCommonDisp(d)
getDispersion(d)
```

`getPriorN` *Get a Recommended Value for Prior N from DGEList Object*

Description

Returns the `lib.size` component of the `samples` component of DGEList object multiplied by the `norm.factors` component

Usage

```
getPriorN(y, design=NULL, prior.df=20)
```

Arguments

`y`	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 (optional argument) giving the design matrix for the GLM that is to be fit. Must be of full column rank. If provided `design` is used to determine the number of parameters to be fit in the statistical model and therefore the residual degrees of freedom. If left as the default (`NULL`) then the `y$samples$group` element of the `DGEList` object is used to determine the residual degrees of freedom.
`prior.df`	numeric scalar giving the weight, in terms of prior degrees of freedom, to be given to the common parameter likelihood when estimating tagwise dispersion estimates.

Details

When estimating tagwise dispersion values using `estimateTagwiseDisp` or `estimateGLMTagwiseDisp` we need to decide how much weight to give to the common parameter likelihood in order to smooth (or stabilize) the dispersion estimates. The best choice of value for the `prior.n` parameter varies between datasets depending on the number of samples in the dataset and the complexity of the model to be fit. The value of `prior.n` should be inversely proportional to the residual degrees of freedom. We have found that choosing a value for `prior.n` that is equivalent to giving the common parameter likelihood 20 degrees of freedom generally gives a good amount of smoothing for the tagwise dispersion estimates. This function simply recommends an appropriate value for `prior.n`—to be used as an argument for `estimateTagwiseDisp` or `estimateGLMTagwiseDisp`—given the experimental design at hand and the chosen prior degrees of freedom.

Value

`getPriorN` returns a numeric scalar

Author(s)

Davis McCarthy, Gordon Smyth

See Also

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

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))
getPriorN(d)
```

`glmFit` *Genewise Negative Binomial Generalized Linear Mdels*

Description

Fit a negative binomial generalized log-linear model to the read counts for each gene or transcript. Conduct genewise statistical tests for a given coefficient or coefficient contrast.

Usage

```
## S3 method for class 'DGEList'
glmFit(y, design=NULL, dispersion=NULL, offset=NULL, weights=NULL, lib.size=NULL,
       prior.count=0.125, start=NULL, method="auto", ...)
glmLRT(glmfit, coef=ncol(glmfit$design), contrast=NULL, test="chisq")
glmQLFTest(y, design=NULL, dispersion=NULL, coef=ncol(glmfit$design), contrast=NULL, abundance.trend=
```

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 tagwise linear models. Must be of full column rank. Defaults to a single column of ones, equivalent to treating the columns as replicate libraries.
`dispersion`	numeric scalar or vector of negative binomial dispersions. Can be a common value for all tags, or a vector of values can provide a unique dispersion value for each tag. If `NULL` will be extracted from `y`, with order of precedence: tagwise dispersion, trended dispersions, common dispersion.
`offset`	numeric matrix of same size as `y` giving offsets for the log-linear models. Can be a scalor or a vector of length `ncol{y}`, in which case it is expanded out to a matrix.
`weights`	optional numeric matrix giving prior weights for the observations (for each library and transcript) to be used in the GLM calculations. Not supported by methods `"linesearch"` or `"levenberg"`.
`lib.size`	numeric vector of length `ncol(y)` giving library sizes. Only used if `offset=NULL`, in which case `offset` is set to `log(lib.size)`. Defaults to `colSums(y)`.
`prior.count`	average prior count to be added to observation to shrink the estimated log-fold-changes towards zero.
`start`	optional numeric matrix of initial estimates for the linear model coefficients.
`method`	which fitting algorithm to use. Possible values are `"auto"`, `"linesearch"`, `"levenberg"` or `"simple"`.
`...`	other arguments are passed to lower-level functions, for example to `mglmLS`.
`glmfit`	a `DGEGLM` object, usually output from `glmFit`.
`coef`	integer or character vector indicating which coefficients of the linear model are to be tested equal to zero. Values must be columns or column names of `design`. Defaults to the last coefficient. Ignored if `contrast` is specified.
`contrast`	numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero. Number of rows must equal to the number of columns of `design`. If specified, then takes precedence over `coef`.
`test`	which test (distribution) to use in calculating the p-values. Possible values are `"F"` or `"chisq"`.
`abundance.trend`	logical, whether to allow an abundance-dependent trend when estimating the prior values for the quasi-likelihood multiplicative dispersion parameter.
`robust`	logical, whether to estimate the prior.df robustly.
`winsor.tail.p`	numeric vector of length 2 giving proportion to trim (Winsorize) from lower and upper tail of the distribution of genewise deviances when estimating the hyperparameters. Positive values produce robust empirical Bayes ignoring outlier small or large deviances. Only used when `robust=TRUE`.
`plot`	logical, whether to plot the quasi-likelihood dispersion estimates vs abundance

Details

glmFit and glmLRT implement generalized linear model (glm) methods developed by McCarthy et al (2012).

glmFit fits genewise negative binomial glms, all with the same design matrix but possibly different dispersions, offsets and weights. When the design matrix defines a one-way layout, or can be re-parametrized to a one-way layout, the glms are fitting very quickly using mglmOneGroup. Otherwise the default fitting method, implemented in mglmLS, is a parallelized line search algorithm described by McCarthy et al (2012). Other possible fitting methods are mglmLevenberg and mglmSimple.

Positive prior.count cause the returned coefficients to be shrunk in such a way that fold-changes between the treatment conditions are decreased. In particular, infinite fold-changes are avoided. Larger values cause more shrinkage. The returned coefficients are affected but not the likelihood ratio tests or p-values.

glmLRT conducts likelihood ratio tests for one or more coefficients in the linear model. If coef is used, the null hypothesis is that all the coefficients indicated by coef are equal to zero. If contrast is non-null, then the null hypothesis is that the specified contrast of the coefficients is equal to zero. For example, a contrast of c(0,1,-1), assuming there are three coefficients, would test the hypothesis that the second and third coefficients are equal.

glmQLFTest implements the quasi-likelihood method of Lund et al (2012). It behaves the same as glmLRT except that it replaces likelihood ratio tests with quasi-likelihood F-tests for coefficients in the linear model. This function calls the limma function squeezeVar to conduct empirical Bayes smoothing of the genewise multiplicative dispersions. Note that the QuasiSeq package provides a alternative implementation of Lund et al (2012), with slightly different glm, trend and FDR methods.

Value

glmFit produces an object of class DGEGLM containing components counts, samples, genes and abundance from y plus the following new components:

design	design matrix as input.
weights	matrix of weights as input.
df.residual	numeric vector of residual degrees of freedom, one for each tag.
offset	numeric matrix of linear model offsets.
dispersion	vector of dispersions used for the fit.
coefficients	numeric matrix of estimated coefficients from the glm fits, on the natural log scale, of size nrow(y) by ncol(design).
fitted.values	matrix of fitted values from glm fits, same number of rows and columns as y.
deviance	numeric vector of deviances, one for each tag.

glmLRT and glmQFTest produce objects of class DGELRT with the same components as for glmfit plus the following:

table	data frame with the same rows as y containing the log2-fold changes, test statistics and p-values, ready to be displayed by topTags..
comparison	character string describing the coefficient or the contrast being tested.

The data frame `table` contains the following columns:

`logFC`	log2-fold change of expression between conditions being tested.
`logCPM`	average log2-counts per million, the average taken over all libraries in `y`.
`LR`	likelihood ratio statistics (only for `glmLRT`).
`F`	F-statistics (only for `glmQFTest`).
`PValue`	p-values.

Author(s)

Davis McCarthy and Gordon Smyth

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. `http://nar.oxfordjournals.org/content/40/10/4288`

Lund, SP, Nettleton, D, McCarthy, DJ, and Smyth, GK (2012). Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates. *Statistical Applications in Genetics and Molecular Biology* Volume 11, Issue 5, Article 8. `http://www.statsci.org/smyth/pubs/QuasiSeqPreprint.pdf`

See Also

Low-level computations are done by `mglmOneGroup`, `mglmLS`, `mglmLevenberg` or `mglmSimple`.

`topTags` displays results from `glmLRT` or `glmQLFTest`.

The `QuasiSeq` package gives an alternative implementation of `glmQLFTest` based on the same statistical ideas.

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(fit, coef=2)
topTags(results)

# Estimate the dispersion (may be unreliable with so few tags)
d <- estimateGLMCommonDisp(d, design, verbose=TRUE)
```

---

gof *Goodness of Fit Tests for Multiple GLM Fits*

Description

Conducts deviance goodness of fit tests for each fit in a `DGEGLM` object

Usage

```
gof(glmfit, pcutoff=0.1, adjust="holm", plot=FALSE, main="qq-plot of genewise goodness of fit", ...)
```

Arguments

`glmfit`	`DGEGLM` object containing results from fitting NB GLMs to genes in a DGE dataset. Output from `glmFit`.
`pcutoff`	scalar giving the cut-off value for the Holm-adjusted p-value. Genes with Holm-adjusted p-values lower than this cutoff value are flagged as 'dispersion outlier' genes.
`adjust`	method used to adjust goodness of fit p-values for multiple testing.
`plot`	logical, if `TRUE` a qq-plot is produced.
`main`	character, title for the plot.
`...`	other arguments are passed to `qqnorm`.

Details

If `plot=TRUE`, produces a plot similar to Figure 2 of McCarthy et al (2012).

Value

This function returns a list with the following components:

`gof.statistics`	numeric vector of deviance statistics, which are the statistics used for the goodness of fit test
`gof.pvalues`	numeric vector of p-values providing evidence of poor fit; computed from the chi-square distribution on the residual degrees of freedom from the GLM fits.

`outlier`	logical vector indicating whether or not each gene is a 'dispersion outlier' (i.e., the model fit is poor for that gene indicating that the dispersion estimate is not good for that gene).
`df`	scalar, the residual degrees of freedom from the GLM fit for which the goodness of fit statistics have been computed. Also the degrees of freedom for the goodness of fit statistics for the LR (chi-quare) test for significance.

Author(s)

Davis McCarthy and Gordon Smyth

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297 `http://nar.oxfordjournals.org/content/40/10/4288`

See Also

`qqnorm`.

`glmFit` for more information on fitting NB GLMs to DGE data.

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)
# Check how good the fit is for each gene
gof(fit)
```

`goodTuring` *Good-Turing Frequency Estimation*

Description

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

Usage

```
goodTuring(x, conf=1.96)
goodTuringPlot(x)
goodTuringProportions(counts)
```

Arguments

`x`	numeric vector of non-negative integers, representing the observed frequency of each species.
`conf`	confidence factor, as a quantile of the standard normal distribution, used to decide for what values the log-linear relationship between frequencies and frequencies of frequencies is acceptable.
`counts`	matrix of counts

Details

Observed counts are assumed to be Poisson distributed. 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 fundamentals 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.

`goodTuringPlot` plots log-probability (i.e., log frequencies of frequencies) versus log-frequency.

`goodTuringProportions` runs `goodTuring` on each column of data, then uses the results to predict the proportion of each tag in each library.

Value

`goodTuring` returns a list with components

`count`	observed frequencies, i.e., the unique positive values of `x`
`n`	frequencies of frequencies
`n0`	frequency of zero, i.e., number of zeros found in `x`

`proportion`	estimated proportion of each species given its count
`P0`	estimated combined proportion of all undetected species

`goodTuringProportions` returns a matrix of proportions of the same size as `counts`.

Author(s)

Aaron Lun and Gordon Smyth, adapted from Sampson's C code from `http://www.grsampson.net/RGoodTur.html`

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)
goodTuringPlot(x)
```

`loessByCol` *Locally Weighted Mean By Column*

Description

Smooth columns of matrix by non-robust loess curves of degree 0.

Usage

```
loessByCol(y, x=NULL, span=0.5)
locfitByCol(y, x=NULL, weights=1, span=0.5, degree=0)
```

Arguments

`y`	numeric matrix of response variables.
`x`	numeric covariate vector of length `nrow(y)`, defaults to equally spaced.
`span`	width of the smoothing window, in terms of proportion of the data set. Larger values produce smoother curves.
`weights`	relative weights of each observation, one for each covariate value.
`degree`	degree of local polynomial fit

Details

Fits a loess curve with degree 0 to each column of the response matrix, using the same covariate vector for each column. The smoothed column values are tricube-weighted means of the original values.

`locfitByCol` uses the `locfit.raw` function of the `locfit` package.

Value

A list containing a numeric matrix with smoothed columns and a vector of leverages for each covariate value.

`locfitByCol` returns a numeric matrix.

Author(s)

Aaron Lun for `loessByCol`, replacing earlier R code by Davis McCarthy. Gordon Smyth for `locfitByCol`.

See Also

`loess`

Examples

```
y <- matrix(rnorm(100*3), nrow=100, ncol=3)
head(y)
out <- loessByCol(y)
head(out$fitted.values)
```

`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, plot.it=TRUE,
     smearWidth=1, col=NULL, allCol="black", lowCol="orange", deCol="red",
     de.tags=NULL, smooth.scatter=FALSE, lowess=FALSE, ...)
```

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
`plot.it`	logical, whether to produce a plot
`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 (if `plot.it=TRUE`), and invisibly returns the `M` (logFC) and `A` (logConc) values used for the plot, plus identifiers `w` and `v` of genes for which `M` and A values, or just `M` values, respectively, were adjusted to make a nicer looking plot.

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])
```

---

maximizeInterpolant *Maximize a function given a table of values by spline interpolation.*

Description

Maximize a function given a table of values by spline interpolation.

Usage

```
maximizeInterpolant(x, y)
```

Arguments

`x`	numeric vector of the inputs of the function.
`y`	numeric matrix of function values at the values of `x`. Columns correspond to `x` values and each row corresponds to a different function to be maximized.

Details

Calculates the cubic spline interpolant for each row the method of Forsythe et al (1977) using the function `fmm_spline` from `splines.c` in the `stats` package). Then calculates the derivatives of the spline segments adjacant to the input with the maximum function value. This allows identification of the maximum of the interpolating spline.

Value

numeric vector of input values at which the function maximums occur.

Author(s)

Aaron Lun, improving on earlier code by Gordon Smyth

References

Forsythe, G. E., Malcolm, M. A. and Moler, C. B. (1977). *Computer Methods for Mathematical Computations*, Prentice-Hall.

Examples

```
x <- seq(0,1,length=10)
y <- rnorm(10,1,1)
maximizeInterpolant(x,y)
```

maximizeQuadratic — *Maximize a function given a table of values by quadratic interpolation.*

Description

Maximize a function given a table of values by quadratic interpolation.

Usage

```
maximizeQuadratic(y, x=1:ncol(y))
```

Arguments

y	numeric matrix of response values.
x	numeric matrix of inputs of the function of same dimension as `y`. If a vector, must be a row vector of length equal to `ncol(y)`.

Details

For each row of `y`, finds the three `x` values bracketing the maximum of `y`, interpolates a quadatric polyonomial through these `y` for these three values and solves for the location of the maximum of the polynomial.

Value

numeric vector of length equal to `nrow(y)` giving the x-value at which `y` is maximized.

Author(s)

Yunshun Chen and Gordon Smyth

See Also

`maximizeInterpolant`

Examples

```
y <- matrix(rnorm(5*9),5,9)
maximizeQuadratic(y)
```

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,
     show.binned.common.disp.vars=FALSE, show.ave.raw.vars=TRUE, scalar=NULL,
     NBline=FALSE, nbins=100, log.axes="xy", xlab=NULL, ylab=NULL,  ...)
binMeanVar(x, group, nbins=100, common.dispersion=FALSE, object=NULL)
```

Arguments

`object`	`DGEList` object containing the raw data and dispersion value. According the method desired for computing the dispersion, either `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 `FALSE`.
`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. Averages are taken on the square root scale as regular arithmetic means are likely to be upwardly biased for count data, whereas averaging on the square scale gives a better summary of the mean-variance relationship in the data. The default is `TRUE`.
`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.axes`	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
`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.

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. The averages are taken on the square-root scale as for count data the arithmetic mean is upwardly biased. Taking averages on the square-root scale provides a useful summary of how the variance of the gene counts change with respect to 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, with the average taken on the square-root scale
`avevars`	vector of the average raw pooled gene-wise variance within each bin of genes, with the average taken on the square-root scale
`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`	vector giving the pooled variance for each gene
`bins`	list giving the indices of the tags in each bin, ordered from lowest expression bin to highest

Author(s)

Davis McCarthy

See Also

`plotMDS.DGEList`, `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 variances shown and save t

## If we want to show estimated tagwise variances on the plot, we must first estimate them!
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) # Use previously saved object to speed up plo
## We could also estimate common/tagwise dispersions using the Cox-Reid methods with an appropriate design matrix
```

mglm *Fit Negative Binomial Generalized Linear Model to Multiple Response Vectors*

Description

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

Usage

```
mglmLS(y, design, dispersion=0, offset=0, coef.start=NULL, tol=1e-5, maxit=50, trace=FALSE)
mglmOneGroup(y, dispersion=0, offset=0, maxit=50, tol=1e-10)
mglmOneWay(y, design=NULL, dispersion=0, offset=0, maxit=50)
mglmSimple(y, design, dispersion=0, offset=0, weights=NULL)
mglmLevenberg(y, design, dispersion=0, offset=0, coef.start=NULL, start.method="null",
         tol=1e-06, maxit=200)
deviances.function(dispersion)
designAsFactor(design)
```

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 scalar, 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`.
`coef.start`	numeric matrix of starting values for the linear model coefficients. Number of rows should agree with `y` and number of columns should agree with `design`.
`start.method`	method used to generate starting values when `coef.stat=NULL`. Possible values are `"null"` to start from the null model of equal expression levels or `"y"` to use the data as starting value for the mean.
`tol`	numeric scalar giving the convergence tolerance. For `mglmOneGroup`, convergence is judged successful when the step size falls below `tol` in absolute size.
`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, especially by `glmFit`.

`mglmOneGroup` fit 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.

`mglmLevenberg` implements a Levenberg-Marquardt modification of the glm scoring algorithm to prevent divergence, and is implemented in C++.

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.values`	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
`error`	logical vector, did the fit fail?
`converged`	local vector, did the fit converge?

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

`designAsFactor` returns a factor of length equal to `nrow(design)`.

Author(s)

Yunshun Chen, Davis McCarthy, Aaron Lun, Gordon Smyth. C++ code by Aaron Lun.

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. http://nar.oxfordjournals.org/content/40/10/4288

See Also

glmFit, for more object-orientated GLM modelling for DGE data.

Examples

```
y <- matrix(rnbinom(1000,mu=10,size=2),ncol=4)
lib.size <- colSums(y)
dispersion <- 0.1

abundance <- mglmOneGroup(y, dispersion=dispersion, offset=log(lib.size))
AveLogCPM <- log1p(exp(1e6*abundance))/log(2)
summary(AveLogCPM)

## Same as above:
AveLogCPM <- aveLogCPM(y, dispersion, offset=log(lib.size))

## 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)
fit <- mglmLS(y, x, dispersion=dispersion, offset=log(lib.size))
head(fit$coefficients)
```

movingAverageByCol *Moving Average Smoother of Matrix Columns*

Description

Apply a moving average smoother to the columns of a matrix.

Usage

```
movingAverageByCol(x, width=5, full.length=TRUE)
```

Arguments

`x`	numeric matrix
`width`	integer, width of window of rows to be averaged
`full.length`	logical value, should output have same number of rows as input?

Details

If `full.length=TRUE`, narrower windows are used at the start and end of each column to make a column of the same length as input. If `FALSE`, all values are averager of `width` input values, so the number of rows is less than input.

Value

Numeric matrix containing smoothed values. If `full.length=TRUE`, of same dimension as `x`. If `full.length=FALSE`, has `width-1` fewer rows than `x`.

Author(s)

Gordon Smyth

Examples

```
x <- matrix(rpois(20,lambda=5),10,2)
movingAverageByCol(x,3)
```

`normalizeChIPtoInput` *Normalize ChIP-Seq Read Counts to Input and Test for Enrichment*

Description

Normalize ChIP-Seq read counts to input control values, then test for significant enrichment relative to the control.

Usage

```
normalizeChIPtoInput(input, response, dispersion=0.01, niter=6, loss="p", plot=FALSE, verbose=FALSE,
calcNormOffsetsforChIP(input, response, dispersion=0.01, niter=6, loss="p", plot=FALSE, verbose=FALSE
```

Arguments

`input`	numeric vector of non-negative input values, not necessarily integer.
`response`	vector of non-negative integer counts of some ChIP-Seq mark for each gene or other genomic feature.
`dispersion`	negative binomial dispersion, must be positive.
`niter`	number of iterations.
`loss`	loss function to be used when fitting the response counts to the input: `"p"` for cumulative probabilities or `"z"` for z-value.
`plot`	if `TRUE`, a plot of the fit is produced.
`verbose`	if `TRUE`, working estimates from each iteration are output.
`...`	other arguments are passed to the `plot` function.

Details

`normalizeChIPtoInput` identifies significant enrichment for a ChIP-Seq mark relative to input values. The ChIP-Seq mark might be for example transcriptional factor binding or an epigenetic mark. The function works on the data from one sample. Replicate libraries are not explicitly accounted for, and would normally be pooled before using this function.

ChIP-Seq counts are assumed to be summarized by gene or similar genomic feature of interest.

This function makes the assumption that a non-negligible proportion of the genes, say 25% or more, are not truly marked by the ChIP-Seq feature of interest. Unmarked genes are further assumed to have counts at a background level proportional to the input. The function aligns the counts to the input so that the counts for the unmarked genes behave like a random sample. The function estimates the proportion of marked genes, and removes marked genes from the fitting process. For this purpose, marked genes are those with a Holm-adjusted mid-p-value less than 0.5.

The read counts are treated as negative binomial. The dispersion parameter is not estimated from the data; instead a reasonable value is assumed to be given.

`calcNormOffsetsforChIP` returns a numeric matrix of offsets, ready for linear modelling.

Value

`normalizeChIPtoInput` returns a list with components

`p.value`	numeric vector of p-values for enrichment.
`scaling.factor`	factor by which input is scaled to align with response counts for unmarked genes.
`prop.enriched`	proportion of marked genes, as internally estimated

`calcNormOffsetsforChIP` returns a numeric matrix of offsets.

Author(s)

Gordon Smyth

`plotBCV` *Plot Biological Coefficient of Variation*

Description

Plot genewise biological coefficient of variation (BCV) against gene abundance (in log2 counts per million).

Usage

```
plotBCV(y, xlab="Average log CPM", ylab="Biological coefficient of variation",
   pch=16, cex=0.2, col.common="red", col.trend="blue", col.tagwise="black", ...)
```

Arguments

`y`	a `DGEList` object.
`xlab`	label for the x-axis.
`ylab`	label for the y-axis.
`pch`	the plotting symbol. See `points` for more details.
`cex`	plot symbol expansion factor. See `points` for more details.
`col.common`	color of line showing common dispersion
`col.trend`	color of line showing dispersion trend
`col.tagwise`	color of points showing tagwise dispersions
`...`	any other arguments are passed to `plot`.

Details

The BCV is the square root of the negative binomial dispersion. This function displays the common, trended and tagwise BCV estimates.

Value

A plot is created on the current graphics device.

Author(s)

Davis McCarthy, Yunshun Chen, Gordon Smyth

Examples

```
BCV.true <- 0.1
y <- DGEList(matrix(rnbinom(6000, size = 1/BCV.true^2, mu = 10),1000,6))
y <- estimateCommonDisp(y)
y <- estimateTrendedDisp(y)
y <- estimateTagwiseDisp(y)
plotBCV(y)
```

`plotExonUsage` *Create a Plot of Exon Usage from Exon-Level Count Data*

Description

Create a plot of exon usage for a given gene by plotting the (un)transformed counts for each exon, coloured by experimental group.

Usage

```
plotExonUsage(y, geneID, group=NULL, transform="none", counts.per.million=TRUE, legend.coords=NULL, .
```

Arguments

`y`	either a matrix of exon-level counts, a list containing a matrix of counts for each exon or a `DGEList` object with (at least) elements `counts` (table of counts summarized at the exon level) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size). Each row of `y` should represent one exon.
`geneID`	character string giving the name of the gene for which exon usage is to be plotted.
`group`	factor supplying the experimental group/condition to which each sample (column of `y`) belongs. If `NULL` (default) the function will try to extract if from `y`, which only works if `y` is a `DGEList` object.
`transform`	character, supplying the method of transformation to be applied to the exon counts, if any. Options are `"none"` (original counts are preserved), `"sqrt"` (square-root transformation) and `"log2"` (log2 transformation). Default is `"none"`.
`counts.per.million`	logical, if `TRUE` then counts per million (as determined from total library sizes) will be plotted for each exon, if `FALSE` the raw read counts will be plotted. Using counts per million effectively normalizes for different read depth among the different samples, which can make the exon usage plots easier to interpret.
`legend.coords`	optional vector of length 2 giving the x- and y-coordinates of the legend on the plot. If `NULL` (default), the legend will be automatically placed near the top right corner of the plot.
`...`	optional further arguments to be passed on to `plot`.

Details

This function produces a simple plot for comparing exon usage between different experimental conditions for a given gene.

Value

`plotExonUsage` (invisibly) returns the transformed matrix of counts for the gene being plotted and produces a plot to the current device.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

`spliceVariants` for methods to detect genes with evidence for alternative exon usage.

Examples

```
# generate exon counts from NB, create list object
y<-matrix(rnbinom(40,size=1,mu=10),nrow=10)
rownames(y) <- rep(c("gene.1","gene.2"), each=5)
d<-DGEList(counts=y,group=rep(1:2,each=2))
plotExonUsage(d, "gene.1")
```

`plotMDS.DGEList` *Multidimensional scaling plot of digital gene expression profiles*

Description

Calculate distances between RNA-seq or DGE libraries, then produce a multidimensional scaling plot. Distances on the plot represent coefficient of variation of expression between samples for the top genes that best distinguish the samples.

Usage

```
## S3 method for class 'DGEList'
plotMDS(x, top=500, labels=colnames(x), col=NULL, cex=1, dim.plot=c(1,2),
   ndim=max(dim.plot), xlab=NULL, ylab=NULL, method="logFC", prior.count=2, gene.selection="pairwise'
```

Arguments

`x`	an `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. See `text` for possible values.
`cex`	numeric vector of plot symbol expansions. See `text` for possible values.
`dim.plot`	which two dimensions should be plotted, numeric vector of length two.
`ndim`	number of dimensions in which data is to be represented
`xlab`	x-axis label
`ylab`	y-axis label
`method`	how to compute distances. Possible values are "logFC" or `"bcv"`.
`prior.count`	average prior count to be added to observation to shrink the estimated log-fold-changes towards zero. Only used when `method="logFC"`.
`gene.selection`	character, `"pairwise"` to choose the top genes separately for each pairwise comparison between the samples or `"common"` to select the same genes for all comparisons. Only used when `method="logFC"`.
`...`	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. A set of top genes are chosen that have largest biological variation between the libraries (those with largest tagwise dispersion treating all libraries as one group). Then the distance between each pair of libraries (columns) is the biological coefficient of variation (square root of the common dispersion) between those two libraries alone, using the top genes.

If `x` is a `DGEList`, then the library sizes and normalization factors found in the object are used. If `x` is a matrix, then library sizes are computed from the column sums, but no other normalization is done.

The number `top` of top genes chosen for this exercise should roughly correspond to the number of differentially expressed genes with materially large fold-changes. The default setting of 500 genes is widely effective and suitable for routine use, but a smaller value might be chosen for when the samples are distinguished by a specific focused molecular pathway. Very large values (greater than 1000) are not usually so effective.

This function can be slow when there are many libraries.

Value

A plot is created on the current graphics device.

An object of class `MDS` is invisibly returned.

Author(s)

Yunshun Chen, Mark Robinson and Gordon Smyth

See Also

`plotMDS`, `cmdscale`, `as.dist`

Examples

```
# Simulate DGE data for 1000 genes(tags) and 6 samples.
# Samples are in two groups
# First 200 genes are differentially expressed in second group

ngenes <- 1000
nlib <- 6
counts <- matrix(rnbinom(ngenes*nlib, size=1/10, mu=20),ngenes,nlib)
rownames(counts) <- paste("Gene",1:ngenes)
group <- gl(2,3,labels=c("Grp1","Grp2"))
counts[1:200,group=="Grp2"] <- counts[1:200,group=="Grp2"] + 10
y <- DGEList(counts,group=group)
y <- calcNormFactors(y)

# without labels, indexes of samples are plotted.
col <- as.numeric(group)
mds <- plotMDS(y, top=200, col=col)

# or labels can be provided, here group indicators:
plotMDS(mds, col=col, labels=group)
```

`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 experimential 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="Average logCPM", ylab="logFC", pch=19,
   cex=0.2, smearWidth=0.5, panel.first=grid(), smooth.scatter=FALSE, lowess=FALSE, ...)
```

Arguments

`object`	`DGEList`, `DGEExact` 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 the 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)
```

`predFC` *Predictive log-fold changes*

Description

Computes estimated coefficients for a NB glm in such a way that the log-fold-changes are shrunk towards zero.

Usage

```
## S3 method for class 'DGEList'
predFC(y, design=NULL, prior.count=0.125, offset=NULL, dispersion=NULL)
## Default S3 method:
predFC(y, design=NULL, prior.count=0.125, offset=NULL, dispersion=0)
```

Arguments

`y`	a matrix of counts or a `DGEList` object
`design`	the design matrix for the experiment
`prior.count`	the average prior count to be added to each observation. Larger values produce more shrinkage.
`offset`	numeric vector or matrix giving the offset in the log-linear model predictor, as for `glmFit`. Usually equal to log library sizes.
`dispersion`	the negative binomial dispersion

Details

This function computes predictive log-fold changes (pfc) for a NB glm. The pfc are posterior Bayesian estimators of the true log-fold-changes. They are predictive of values that might be replicated in a future experiment.

Specifically the function adds a small prior count to each observation before estimating the glm. The actual prior count that is added is proportion to the library size. This has the effect that any log-fold-change that was zero prior to augmentation remains zero and non-zero log-fold-changes are shrunk towards zero.

The prior counts can be viewed as equivalent to a prior belief that the log-fold changes are small, and the output can be viewed as posterior log-fold-changes from this Bayesian viewpoint. The output coefficients are called *predictive* log fold-changes because, depending on the prior, they may be a better prediction of the true log fold-changes than the raw estimates.

Log-fold changes for transcripts with low counts are shrunk more than transcript with high counts. In particular, infinite log-fold-changes arising from zero counts are avoided. The exact degree to which this is done depends on the negative binomail dispersion.

If `design=NULL`, then the function returns a matrix of the same size as `y` containing log2 counts-per-million, with zero values for the counts avoided. This equivalent to choosing `design` to be the identity matrix with the same number of columns as `y`.

Value

Numeric matrix of linear model coefficients (if `design` is given) or logCPM (if `design=NULL`) on the log2 scale.

Author(s)

Belinda Phipson and Gordon Smyth

See Also

`glmFit`, `exactTest`

Examples

```
# generate counts for a two group experiment with n=2 in each group and 100 genes
dispersion <- 0.1
y <- matrix(rnbinom(400,size=1/dispersion,mu=4),nrow=100)
y <- DGEList(y,group=c(1,1,2,2))
design <- model.matrix(~group, data=y$samples)

#estimate the predictive log fold changes
predlfc<-predFC(y,design,dispersion=dispersion,prior.count=1)
logfc <- predFC(y,design,dispersion=dispersion,prior.count=0)
logfc.truncated <- pmax(pmin(logfc,100),-100)

#plot predFC's vs logFC's
plot(predlfc[,2],logfc.truncated[,2],xlab="Predictive log fold changes",ylab="Raw log fold changes")
abline(a=0,b=1)
```

---

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

Description

Interpolated 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 counts.
`input.mean`	numeric matrix of population means for `x`. If a vector, then of the same length as `nrow(x)`.
`output.mean`	numeric matrix of population means for the output values. If a vector, then of the same length as `nrow(x)`.
`dispersion`	numeric scalar, vector or matrix giving negative binomial dispersion values.

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`.

In principle, `q2qnbinom` gives similar results to calling `pnbinom` followed by `qnbinom` as in the example below. However this function avoids infinite values arising from rounding errors and does appropriate interpolation to return continuous values.

`q2qnbinom` is called by `equalizeLibSizes` to perform quantile-to-quantile normalization.

Value

numeric matrix of same dimensions as x, with output.mean as the new nominal population mean.

Author(s)

Gordon Smyth

See Also

equalizeLibSizes

Examples

```
x <- 15
input.mean <- 10
output.mean <- 20
dispersion <- 0.1
q2qnbinom(x,input.mean,output.mean,dispersion)

# Similar in principle:
qnbinom(pnbinom(x,mu=input.mean,size=1/dispersion),mu=output.mean,size=1/dispersion)
```

---

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)
```

`roast.DGEList` *Rotation Gene Set Tests for Digital Gene Expression Data*

Description

Rotation gene set testing for Negative Binomial generalized linear models.

Usage

```
## S3 method for class 'DGEList'
roast(y, index=NULL, design=NULL, contrast=ncol(design), set.statistic="mean",
   gene.weights=NULL, array.weights=NULL, weights=NULL, block=NULL, correlation,
     var.prior=NULL, df.prior=NULL, trend.var=FALSE, nrot=999)
## S3 method for class 'DGEList'
mroast(y, index=NULL, design=NULL, contrast=ncol(design), set.statistic="mean",
   gene.weights=NULL, array.weights=NULL, weights=NULL, block=NULL, correlation,
   var.prior=NULL, df.prior=NULL, trend.var=FALSE, nrot=999, adjust.method="BH", midp=TRUE, sort="dir
```

Arguments

`y`	`DGEList` object.
`index`	index vector specifying which rows (genes) of `y` are in the test set. This can be a vector of indices, or a logical vector of the same length as `statistics`, or any vector such as `y[iset,]` contains the values for the gene set to be tested.
`design`	design matrix
`contrast`	contrast for which the test is required. Can be an integer specifying a column of `design`, or else a contrast vector of length equal to the number of columns of `design`.
`set.statistic`	summary set statistic. Possibilities are `"mean"`,`"floormean"`,`"mean50"` or `"msq"`.
`gene.weights`	optional numeric vector of weights for genes in the set. Can be positive or negative. For `mroast.DGEList` this vector must have length equal to `nrow(y)`. For `roast.DGEList`, can be of length `nrow(y)` or of length equal to the number of genes in the test set.
`array.weights`	optional numeric vector of array weights.
`weights`	optional matrix of observation weights. If supplied, should be of same dimensions as `y` and all values should be positive.
`block`	optional vector of blocks.
`correlation`	correlation between blocks.
`var.prior`	prior value for residual variances. If not provided, this is estimated from all the data using `squeezeVar`.
`df.prior`	prior degrees of freedom for residual variances. If not provided, this is estimated using `squeezeVar`.
`trend.var`	logical, should a trend be estimated for `var.prior`? See `eBayes` for details. Only used if `var.prior` or `df.prior` are `NULL`.
`nrot`	number of rotations used to estimate the p-values.
`adjust.method`	method used to adjust the p-values for multiple testing. See `p.adjust` for possible values.
`midp`	logical, should mid-p-values be used in instead of ordinary p-values when adjusting for multiple testing?
`sort`	character, whether to sort output table by directional p-values (`"directional"`), non-directional p-value (`"mixed"`), or not at all (`"none"`).

Details

This function implements a method for the ROAST gene set test from Wu et al (2010) for the digital gene expression data, eg. RNA-Seq data. Basically, the Negative Binomial generalized linear models are fitted for count data. The fitted values are converted into z-scores, and then it calls the `roast` function in `limma` package to conduct the gene set test. It tests whether any of the genes in the set are differentially expressed. This allows users to focus on differential expression for any coefficient or contrast in a generalized linear model. If `contrast` is not specified, the last coefficient in the model will be tested. The arguments `array.weights`, `block` and `correlation` have the same meaning as they for for the `lmFit` function.

The arguments `df.prior` and `var.prior` have the same meaning as in the output of the `eBayes` function. If these arguments are not supplied, they are estimated exactly as is done by `eBayes`.

The argument `gene.weights` allows directions or weights to be set for individual genes in the set.

The gene set statistics `"mean"`, `"floormean"`, `"mean50"` and `msq` are defined by Wu et al (2010). The different gene set statistics have different sensitivities to small number of genes. If `set.statistic="mean"` then the set will be statistically significantly only when the majority of the genes are differentially expressed. `"floormean"` and `"mean50"` will detect as few as 25% differentially expressed. `"msq"` is sensitive to even smaller proportions of differentially expressed genes, if the effects are reasonably large.

The output gives p-values three possible alternative hypotheses, `"Up"` to test whether the genes in the set tend to be up-regulated, with positive t-statistics, `"Down"` to test whether the genes in the set tend to be down-regulated, with negative t-statistics, and `"Mixed"` to test whether the genes in the set tend to be differentially expressed, without regard for direction.

`roast` estimates p-values by simulation, specifically by random rotations of the orthogonalized residuals (Langsrud, 2005), so p-values will vary slightly from run to run. To get more precise p-values, increase the number of rotations `nrot`. The p-value is computed as `(b+1)/(nrot+1)` where `b` is the number of rotations giving a more extreme statistic than that observed (Phipson and Smyth, 2010). This means that the smallest possible p-value is `1/(nrot+1)`.

`mroast` does roast tests for multiple sets, including adjustment for multiple testing. By default, `mroast` reports ordinary p-values but uses mid-p-values (Routledge, 1994) at the multiple testing stage. Mid-p-values are probably a good choice when using false discovery rates (`adjust.method="BH"`) but not when controlling the family-wise type I error rate (`adjust.method="holm"`).

`roast` performs a *self-contained* test in the sense defined by Goeman and Buhlmann (2007). For a *competitive* gene set test, see `camera.DGEList`.

Value

`roast` produces an object of class `Roast`. See `roast` for details.

`mroast` produces a data.frame. See `mroast` for details.

Author(s)

Yunshun Chen and Gordon Smyth

References

Goeman, JJ, and Buhlmann, P (2007). Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics* 23, 980-987.

Langsrud, O (2005). Rotation tests. *Statistics and Computing* 15, 53-60.

Phipson B, and Smyth GK (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology*, Volume 9, Article 39.

Routledge, RD (1994). Practicing safe statistics with the mid-p. *Canadian Journal of Statistics* 22, 103-110.

Wu, D, Lim, E, Francois Vaillant, F, Asselin-Labat, M-L, Visvader, JE, and Smyth, GK (2010). ROAST: rotation gene set tests for complex microarray experiments. *Bioinformatics* 26, 2176-2182. http://bioinformatics.oxfordjournals.org/content/26/17/2176

See Also

roast, camera.DGEList

Examples

```
mu <- matrix(10, 100, 4)
group <- factor(c(0,0,1,1))
design <- model.matrix(~group)

# First set of 10 genes that are genuinely differentially expressed
iset1 <- 1:10
mu[iset1,3:4] <- mu[iset1,3:4]+10

# Second set of 10 genes are not DE
iset2 <- 11:20

# Generate counts and create a DGEList object
y <- matrix(rnbinom(100*4, mu=mu, size=10),100,4)
y <- DGEList(counts=y, group=group)

# Estimate dispersions
y <- estimateDisp(y, design)

roast(y, iset1, design, contrast=2)
mroast(y, iset1, design, contrast=2)
mroast(y, list(set1=iset1, set2=iset2), design, contrast=2)
```

---

spliceVariants *Identify Genes with Splice Variants*

Description

Identify genes exhibiting evidence for splice variants (alternative exon usage/transcript isoforms) from exon-level count data using negative binomial generalized linear models.

Usage

```
spliceVariants(y, geneID, dispersion=NULL, group=NULL, estimate.genewise.disp=TRUE, trace=FALSE)
```

Arguments

`y`	either a matrix of exon-level counts or a `DGEList` object with (at least) elements `counts` (table of counts summarized at the exon level) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size). Each row of `y` should represent one exon.
`geneID`	vector of length equal to the number of rows of `y`, which provides the gene identifier for each exon in `y`. These identifiers are used to group the relevant exons into genes for the gene-level analysis of splice variation.
`dispersion`	scalar (in future a vector will also be allowed) supplying the negative binomial dispersion parameter to be used in the negative binomial generalized linear model.
`group`	factor supplying the experimental group/condition to which each sample (column of `y`) belongs. If `NULL` (default) the function will try to extract if from `y`, which only works if `y` is a `DGEList` object.
`estimate.genewise.disp`	logical, should genewise dispersions (as opposed to a common dispersion value) be computed if the `dispersion` argument is `NULL`?
`trace`	logical, whether or not verbose comments should be printed as function is run. Default is `FALSE`.

Details

This function can be used to identify genes showing evidence of splice variation (i.e. alternative splicing, alternative exon usage, transcript isoforms). A negative binomial generalized linear model is used to assess evidence, for each gene, given the counts for the exons for each gene, by fitting a model with an interaction between exon and experimental group and comparing this model (using a likelihood ratio test) to a null model which does not contain the interaction. Genes that show significant evidence for an interaction between exon and experimental group by definition show evidence for splice variation, as this indicates that the observed differences between the exon counts between the different experimental groups cannot be explained by consistent differential expression of the gene across all exons. The function `topTags` can be used to display the results of `spliceVariants` with genes ranked by evidence for splice variation.

Value

`spliceVariants` returns a `DGEExact` object, which contains a table of results for the test of differential splicing between experimental groups (alternative exon usage), a data frame containing the gene identifiers for which results were obtained and the dispersion estimate(s) used in the statistical models and testing.

Author(s)

Davis McCarthy, Gordon Smyth

See Also

estimateExonGenewiseDisp for more information about estimating genewise dispersion values from exon-level counts. DGEList for more information about the DGEList class. topTags for more information on displaying ranked results from spliceVariants. estimateCommonDisp and related functions for estimating the dispersion parameter for the negative binomial model.

Examples

```
# generate exon counts from NB, create list object
y<-matrix(rnbinom(40,size=1,mu=10),nrow=10)
d<-DGEList(counts=y,group=rep(1:2,each=2))
genes <- rep(c("gene.1","gene.2"), each=5)
disp <- 0.2
spliceVariants(d, genes, disp)
```

---

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, DGEGLM, DGEExact and DGELRT Objects*

Description

Extract a subset of a `DGEList`, `DGEGLM`, `DGEExact` or `DGELRT` object.

Usage

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

Arguments

`object`	object of class `DGEList`, `DGEGLM`, `DGEExact` or `DGELRT`, respectively
`i,j`	elements to extract. `i` subsets the tags or genes while `j` subsets the libraries. Note, columns of `DGEGLM`, `DGEExact` and `DGELRT` 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 `DGEGLM`, `DGEExact` and `DGELRT` objects, only rows (i.e. `i`) may be subsetted.

Value

An object of class `DGEList`, `DGEGLM`, `DGEExact` or `DGELRT` 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
```

`systematicSubset` *Take a systematic subset of indices.*

Description

Take a systematic subset of indices stratified by a ranking variable.

Usage

```
systematicSubset(n, order.by)
```

Arguments

`n`	integer giving the size of the subset.
`order.by`	numeric vector of the values by which the indices are ordered.

Value

`systematicSubset` returns a vector of size `n`.

Author(s)

Gordon Smyth

See Also

`order`

Examples

```
y <- rnorm(100, 1, 1)
systematicSubset(20, y)
```

---

thinCounts — *Binomial or Multinomial Thinning of Counts*

Description

Reduce the size of Poisson-like counts by binomial thinning.

Usage

```
thinCounts(x, prob=NULL, target.size=min(colSums(x)))
```

Arguments

`x`	numeric vector or array of non-negative integers.
`prob`	numeric scalar or vector of same length as `x`, the expected proportion of the events to keep.
`target.size`	integer scale or vector of the same length as `NCOL{x}`, the desired total column counts. Must be not greater than column sum of `x`. Ignored if `prob` is not `NULL`.

Details

If `prob` is not `NULL`, then this function calls `rbinom` with `size=x` and `prob=prob` to generate the new counts. This is classic binomial thinning. The new column sums are random, with expected values determined by `prob`.

If `prob` is `NULL`, then this function does multinomial thinning of the counts to achieve specified column totals. The default behavior is to thin the columns to have the same column sum, equal to the smallest column sum of `x`.

If the elements of `x` are Poisson, then binomial thinning produces new Poisson random variables with expected values reduced by factor `prob`. If the elements of each column of `x` are multinomial, then multinomial thinning produces a new multinomial observation with a reduced sum.

Value

A vector or array of the same dimensions as `x`, with thinned counts.

Author(s)

Gordon Smyth

Examples

```
x <- rpois(10,lambda=10)
thinCounts(x,prob=0.5)
```

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="PValue")
```

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, should the top tags be sorted by p-value (`"PValue"`), by absolute log-fold change (`"logFC"`), or not sorted (`"none"`).

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 `logFC`, the log-abundance ratio, i.e. fold change, for each tag in the two groups being compared, `logCPM`, the log-average concentration/abundance for each tag in the two groups being compared, `PValue`, exact p-value for differential expression using the NB model, `FDR`, the p-value adjusted for multiple testing as found using `p.adjust` using the method specified.
`adjust.method`	character string stating the method used to adjust p-values for multiple testing.
`comparison`	a vector giving the names of the two groups being compared.
`test`	character string stating the name of the test.

The dimensions, row names and column names of a `TopTags` object are defined by those of `table`, see `dim.TopTags` or `dimnames.TopTags`.

`TopTags` objects also have a `show` method so that printing produces a compact summary of their contents.

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

References

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

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

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.

---

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)

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)
```

`WLEB` *Calculate Weighted Likelihood Empirical Bayes Estimates*

Description

Estimates the parameters which maximize the given log-likelihood matrix using empirical Bayes method.

Usage

```
WLEB(theta, loglik, prior.n, covariate, trend.method="locfit", span=NULL, overall=TRUE, trend=TRUE, in
```

Arguments

`theta`	numeric vector of values of the parameter at which the log-likelihoods are calculated.
`loglik`	numeric matrix of log-likelihood of all the candidates at those values of parameter.
`prior.n`	numeric scaler, estimate of the prior weight, i.e. the smoothing parameter that indicates the weight to put on the common likelihood compared to the individual's likelihood.
`covariate`	numeric vector of values across which a parameter trend is fitted
`trend.method`	method for estimating the parameter trend. Possible values are `"none"`, `"movingave"` and `"loess"`.
`span`	width of the smoothing window, as a proportion of the data set.
`overall`	logical, should a single value of the parameter which maximizes the sum of all the log-likelihoods be estimated?
`trend`	logical, should a parameter trend (against the covariate) which maximizes the local shared log-likelihoods be estimated?

`individual`	logical, should individual estimates of all the candidates after applying empirical Bayes method along the trend be estimated?
`m0`	numeric matrix of local shared log-likelihoods. If `Null`, they will be calculated using the method selected by `trend.method`.
`m0.out`	logical, should local shared log-likelihoods be included in the output?

Details

This function is a generic function that calculates an overall estimate, trend estimates and individual estimates for each candidate given the values of the log-likelihood of all the candidates at some specified parameter values.

Value

A list with the following:

`overall`	the parameter estimate that maximizes the sum of all the log-likelihoods.
`trend`	the estimated trended parameters against the covariate.
`individual`	the individual estimates of all the candidates after applying empirical Bayes method along the trend.
`shared.loglik`	the estimated numeric matrix of local shared log-likelihoods

Author(s)

Yunshun Chen, Gordon Smyth

See Also

`locfitByCol`, `movingAverageByCol` and `loessByCol` implement the local fit, moving average or loess smoothers.

Examples

```
y <- matrix(rpois(100, lambda=10), ncol=4)
theta <- 7:14
loglik <- matrix(0,nrow=nrow(y),ncol=length(theta))
for(i in 1:nrow(y))
for(j in 1:length(theta))
loglik[i,j] <- sum(dpois(y[i,], theta[j] ,log=TRUE))
covariate <- log(rowSums(y))
out <- WLEB(theta, loglik, prior.n=3, covariate)
out
```

`zscoreNBinom` *Z-score Equivalents of Negative Binomial Deviate*

Description

Compute z-score equivalents of negative binomial random deviates.

Usage

```
zscoreNBinom(q, size, mu)
```

Arguments

`q`	numeric vector or matrix giving negative binomial random values.
`size`	negative binomial size parameter (>0).
`mu`	mean of negative binomial distribution (>0).

Details

This function computes the mid-p value of `q`, then converts to the standard normal deviate with the same cumulative probability distribution value.

Care is taken to do the computations accurately in both tails of the distributions.

Value

Numeric vector or matrix giving equivalent deviates from a standard normal distribution.

Author(s)

Gordon Smyth

See Also

`pnbinom`, `qnorm` in the stats package.

Examples

```
zscoreNBinom(c(0,10,100), mu=10, size=1/10)
```

Index