Package 'edgeR'

September 24, 2012

Version 2.6.12

Date 2012/08/27

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>, 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 (>= 2.3.0), methods, limma

Suggests MASS, statmod, splines

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 LGPL (>= 2)

R topics documented:

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

Description

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

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

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

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

Author(s)

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

References

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

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

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

`adjustedProfileLik` *Compute Cox-Reid Adjusted Profile Likelihood for Negative Binomial GLMs*

Description

Compute the Cox-Reid Adjusted Profile-likelihood for many negative binomial (NB) GLMs.

Usage

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

Arguments

dispersion	numeric scalar or vector giving the dispersion(s) towards which the tagwise dispersion parameters are shrunk.
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.
adjust	logical, if TRUE then Cox-Reid adjustment is made to the log-likelihood, if FALSE then the log-likelihood is returned without adjustment. Default is TRUE.

Details

In the edgeR context, adjustedProfileLik is a low-level function necessary for estimating dispersion parameters for NB GLMs.

Value

adjustedProfileLik produces a vector of Cox-Reid adjusted profile likelihoods for the given counts, dispersion value, offset and design matrix (i.e. the APL for each gene/tag), which has the same length as the number of rows of the count datamatrix y.

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.

See Also

dispCoxReidInterpolateTagwise, estimateGLMTagwiseDisp, maximizeInterpolant

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
```

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

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Mark Robinson

See Also

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

Examples

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

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.

---

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

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Davis McCarthy

References

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

See Also

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

Examples

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

`bin.dispersion` *Estimate Common Dispersion for Negative Binomial GLMs in Bins of Genes Sorted by Overall Abundance*

Description

Estimates the common dispersion parameter for each of a number of bins of data for a DGE dataset. Genes are sorted into bins based on overall expression level. For multiple-group (one-way layout) experimental designs, conditional maximum likelihood (CML) methods can be used. For general experimental designs the binned common dispersions we can use Cox-Reid approximate conditional inference, Pearson or deviance estimators for a negative binomial generalized linear model.

Usage

```
binCMLDispersion(y, nbins=50)
binGLMDispersion(y, design, min.n=500, offset=NULL,  method="CoxReid", ...)
```

Arguments

`y`	an object that contains the raw counts for each library (the measure of expression level); it can either be a matrix of counts, or a `DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`nbins`	scalar, the number of bins for which to compute common dispersions. Default is 50 bins.
`design`	numeric matrix giving the design matrix for the GLM that is to be fit.

min.n	scalar, the minimum number of genes to be included in each bin.
offset	(optional) 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. Default is NULL. If NULL, then offset is log(lib.size) if y is a matrix or log(y$samples$lib.size * y$samples$norm.factors) if y is a DGEList object.
method	method used to estimated the dispersion. Argument passed to estimateGLMCommonDisp, which calls the functions to do the computations. Possible values are "CoxReid", "Pearson" or "deviance".
...	other arguments are passed to lower-level functions.

Details

To obtain estimates of the common dispersion parameters conditional maximum likelihood (estimateCommonDisp) is used for binCMLDispersion and one of Cox-Reid approximate conditional inference (dispCoxReid), the deviance (dispDeviance) or Pearson (dispPearson) estimates are used for binGLMDispersion. Genes are assigned to bins using the cutWithMinN function to obtain bins spread over the abundance range of the genes while ensuring that each bin has a minimum number of genes, thus permitting reliable estimation of the common dispersion for each bin.

Value

Returns a list with two components:

dispersion	numeric vector providing the common dispersion for each bin
abundance	numeric vector providing the average abundance (expression level) of genes in each bin

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.

See Also

estimateGLMCommonDisp, dispCoxReid, dispPearson, dispDeviance

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
bindisp.CML <- binCMLDispersion(d, nbins=50)
bindisp.GLM <- binGLMDispersion(d, design, min.n=10)
```

`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"), refColumn = NULL, logratioTrim =
```

Arguments

object	either a matrix of raw (read) counts or a DGEList object
method	method to use to calculate the scale factors
refColumn	column to use as reference for method="TMM"
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.

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.

Value

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

Author(s)

Mark Robinson, 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)
```

---

commonCondLogLikDerDelta

Conditional Log-Likelihoods in Terms of Delta

Description

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

Usage

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

Arguments

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

Details

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

Value

numeric scalar of function/derivative evaluated at given delta

Author(s)

Davis McCarthy

See Also

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

Examples

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

`condLogLikDerDelta` *Conditional Log-Likelihood in Terms of Delta*

Description

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

Usage

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

Arguments

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

Details

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

Value

vector or matrix of function/derivative evaluations

Author(s)

Mark Robinson, Davis McCarthy

See Also

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

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Details

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

Value

vector of function/derivative evaluations, one for each transcript

Author(s)

Mark Robinson, Davis McCarthy

Examples

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

---

cpm *Calculate Counts per Million from DGEList or Matrix Object*

Description

Returns counts per million from a DGEList or matrix object by dividing raw counts by library size (which can be normalized) and multiplying by one million.

Usage

```
cpm(x, normalized.lib.sizes=FALSE)
```

Arguments

x	either 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)
normalized.lib.sizes	logical, should the library sizes (total sum of counts for each library) be normalized using the norm.factors component of the DGEList object? Ignored (with a warning) if x is a count matrix.

Details

A convenience function to compute the counts per million for plotting and comparing libraries on a convenient scale. Essentially just does the calculation `1e06*t(t(x)/lib.size)` to produce counts per million, where x is a matrix of counts and the `lib.size` can be the total sum of counts in each library or a normalized version of this using TMM normalization or equivalent method.

Value

`getPriorN` returns a numeric scalar

Author(s)

Davis McCarthy, Gordon Smyth

See Also

DGEList for more information about the DGEList class.

Examples

```
# generate raw counts from NB, create list object
y<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
cpm(y)
d<-DGEList(counts=y,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
# When applied to a DGEList object, x$samples$lib.size is used
cpm(d)
# As x$samples$lib.size here is very different from colSums(y), cpm(y) and cpm(d) give very different resul
```

`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.

Usage

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

Arguments

`x`	numeric vector.
`intervals`	number of intervals (greater than or equal to 2).
`min.n`	minimum number of values in any interval.

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.

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

Slots/List Components

Objects of this class contain the following list components:

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

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

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

Methods

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

Author(s)

Mark Robinson, Davis McCarthy

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

Description

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

Slots/List Components

Objects of this class contain the following list components:

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

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

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

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

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

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

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

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

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

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

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

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

Methods

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

Author(s)

Davis McCarthy

DGEList *DGEList Constructor*

Description

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

Usage

```
DGEList(counts = matrix(0, 0, 0), lib.size = NULL, norm.factors = NULL, group = rep.int(1,ncol(c
```

Arguments

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

Details

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

The optional genes argument supplies a data.frame of annotation for each row or feature.

Value

a DGEList object

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

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), lib.size=colSums(y))
```

DGEList-class *Digital Gene Expression data - class*

Description

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

Slots/List Components

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

`counts`: numeric matrix containing the read counts.

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

Methods

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

Author(s)

Mark Robinson

See Also

DGEList

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

Description

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

Slots/List Components

Objects of this class contain the following list components:

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

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

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

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

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

Methods

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

Author(s)

Davis McCarthy

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

Description

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

Usage

```
dglmStdResid(y, design, dispersion=0, offset=0, nbins=100, make.plot=TRUE, xlab="Mean", ylab="Av
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 DGEList Object*

Description

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

Usage

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

Arguments

x	an object of class DGEList, DGEExact, DGEGLM or TopTags
value	a possible value for dimnames(x): see dimnames

Details

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

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

Value

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

Author(s)

Gordon Smyth

See Also

dimnames in the base package.

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

dispBinTrend *Estimate Dispersions with an Abundance-Dependent Trend for Negative Binomial GLMs*

Description

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 abundance and then using splines or a loess fit to interpolate a dispersion value for each gene, dependent on overall abundance of the gene.

Usage

```
dispBinTrend(y, design, offset=NULL, df = 5, span=2/3, min.n=500, method.bin="CoxReid", method.t
```

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	scalar, the degrees of freedom for the natural cubic splines fit, used to determine the placement of the knots (number of knots is df - 1. Default is 5.
span	scalar, passed to loess to determine the amount of smoothing for the loess fit. Default is 2/3.
min.n	scalar, minimim number of genes in each of the bins into which genes are sorted to form the basis for interpolating the dispersions. Setting a minimum value ensures that there will be sufficient genes in each bin to allow reliable estimation of the common dispersion for each bin.
method.bin	character, passed to binGLMDispersion, to specify the method used to compute the common dispersion within each bin of genes. Default is "CoxReid", other options are "Pearson" and "deviance".
method.trend	character, specifies method to produce a smooth fit through the binned common dispersions in order to interpolate the trended dispersions. Default is "spline" to use natural cubic splines, other option is "loess" to use a loess fit.
trace	logical, should iteration information be output?
...	option arguments to be passed to lower-level function binGLMDispersion.

Details

This function takes the binned common dispersion and abundance from binGLMDispersion 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. This function is called by estimateGLMTrendedDisp.

Value

list with the following components:

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

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

binGLMDispersion, 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))
lib.size <- colSums(y)
design <- model.matrix(~group) # Define the design matrix for the full model
disp <- dispBinTrend(y, design, offset=log(lib.size), min.n=100, span=0.3)
plot(disp$abundance, disp$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, offset=NULL, interval=c(0,4), tol=1e-5, min.row.sum=5, subset=10000)
dispDeviance(y, design, offset=NULL, interval=c(0,4), tol=1e-5, min.row.sum=5, subset=10000, rob
dispPearson(y, design, offset=NULL, interval=c(0,4), tol=1e-5, min.row.sum=5, subset=10000, robu
```

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 abundance.
trace	logical, should iteration information be output?
robust	logical, should a robust estimator be used?

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, while dispDeviance and dispPearson use 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.

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) # Define the design matrix for the full model
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, abundance=NULL, mi
```

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.
abundance	numeric scalar or vector giving the tagwise log-abundance measure for each tag. If null, the abundance is then evaluated by mglmOneGroup
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.n	numeric scalar, smoothing 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.

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, method.optim="Nelder-Mead",
dispCoxReidPowerTrend(y, design, offset=NULL, subset=10000, 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 abundance using cutWithMinN.
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(abundance,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` *Quantile Adjustment to Equalize Library Sizes for a Fixed Value of the Dispersion Parameter*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list with elements

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

Author(s)

Mark Robinson, Davis McCarthy

Examples

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

---

estimateCommonDisp — *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 for the unadjusted counts provided.

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

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

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

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

Author(s)

Mark Robinson, Davis McCarthy, 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

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

Examples

```
# 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))
cmdisp <- estimateCommonDisp(d, verbose=TRUE)
```

---

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", verbose=FALSE, ...)
## Default S3 method:
estimateGLMCommonDisp(y, design=NULL, offset=NULL, method="CoxReid", verbose=FALSE, ...)
```

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".
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*. http://nar.oxfordjournals.org/content/early/2012/02/06/nar.gks042 (Published online 28 January 2012)

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` *Estimate Empirical Bayes Tagwise Dispersions for Negative Binomial GLMs*

Description

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

Usage

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

Arguments

`y`	an object that contains the raw counts for each library (the measure of expression level); it can either be a matrix of counts, or a `DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`design`	numeric design matrix, as for `glmFit`.
`trend`	logical, should an abundance trend be applied to the grid of dispersion values over which the tagwise dispersion estimation is done? Generally this should be `TRUE` if a trended dispersion has been estimated and `FALSE` otherwise.
`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. Default is `NULL`; if object is a `DGEList` and offset is `NULL` then offset will be calculated automatically from codey$samples.
`dispersion`	vector or scalar giving the dispersion value(s) to be used to set the grip of points for computation of the tagwise dispersion in `dispCoxReidInterpolateTagwise`.

... other arguments are passed to lower-level functions. See dispCoxReidInterpolateTagwise for details.

Details

This generic function is essentially a wrapper for dispCoxReidInterpolateTagwise. To obtain estimates of the tagwise dispersion parameters for negative binomial GLMs we use Cox-Reid approximate conditional inference as implemented in dispCoxReidInterpolateTagwise. The approach is to maximize the adjusted profile likelihood over the dispersion value, for the tagwise models and use these values as the tagwise dispersion parameters for differential signal testing in downstream analysis.

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*. http://nar.oxfordjournals.org/content/early/2012/02/06/nar.gks042 (Published online 28 January 2012)

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 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.

Usage

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

Arguments

`y`	an object that contains the raw counts for each library (the measure of expression level); it can either be a matrix of counts, or a `DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`design`	numeric 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 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. Default is `NULL`; if object is a `DGEList` and offset is `NULL` then offset will be calculated automatically from codey$samples.
`...`	other arguments are passed to lower-level functions. See `dispBinTrend`, `dispCoxReidPowerTrend` and `dispCoxReidSplineTrend` for details.

Details

This is a wrapper function for the lower-level functions that actually carry out the dispersion estimation calculations. Provide a convenient, object-oriented interface for users.

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*. http://nar.oxfordjournals.org/content/early/2012/02/06/nar.gks042 (Published online 28 January 2012)

See Also

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

estimateGLMCommonDisp for common dispersion and estimateGLMTagwiseDisp for (trended) tagwise dispersion in the context of generalized linear models.

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
disp <- estimateGLMTrendedDisp(d, design, min.n=10)
```

---

estimatePs *Estimate Expression Levels*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list with elements:

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

Author(s)

Mark Robinson, Davis McCarthy

Examples

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

---

estimateSmoothing *Estimate the Prior Weight for Tagwise Dispersions*

Description

This function is no longer recommended or required. Use getPriorN instead.

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

Usage

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

Arguments

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

Details

We are no longer recommending this function, as it produces variable results. prior.n is now set automatically using getPriorN.

Value

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

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

Author(s)

Mark Robinson, Davis McCarthy

See Also

getPriorN

Examples

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

---

estimateTagwiseDisp *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.n=getPriorN(object), trend="movingave", prop.used=0.3, method=
```

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.n	numeric scalar, smoothing 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.
trend	method for allowing the prior distribution for the dispersion to be abundance-dependent. Possible values are "none", "movingave" and "tricube". "none" means no trend. "movingave" applies a moving average smoother to the local likelihood values. "tricube" applies tricube weighting to locally smooth the common likelihood.
prop.used	optional scalar giving the proportion of all tags/genes to be used for the locally weighted estimation of the tagwise dispersion, allowing the dispersion estimates to vary with abundance (expression level). For each tag/gene the estimate of its dispersion is based on the closest prop.used of all of the genes to that gene, where 'closeness' is based on similarity in expression level.
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

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

"tricube" local weighting is similar to that used by lowess. "movingave" is much faster than "tricube" and gives similar results.

"optimize" is very slow if there is a large number of tags/genes to be analysed (i.e., more than 5000).

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 and Gordon Smyth

References

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

See Also

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

Examples

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

---

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, pri
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.total	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 the function binom.test in the stats package, but generalized to overdispersed counts.

The low level functions exactTestDoubleTail, exactTestBetaApprox, exactTestBySmallP and exactTestByDeviance all assume that the libraries have been normalized to have the same size (expected column sum under the null hypothesis). The higher level function exactTest is intended to be called by users. This has a more object-orientated flavor and produces an object containing all the necessary components for downstream analysis. exactTest equalizes the library sizes using equalizeLibSizes before calling one of 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 to binom.test when the dispersion is near 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. This has good statistical properties but is relatively slow to compute. 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.

Value

exactTestDoubleTail and friends produce a numeric vector of genewise p-values, one for each row of y1 and y2.

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

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)
rownames(y) <- paste("Gene",1:nrow(y),sep=".")
group <- factor(c(1,1,2,2))
d <- DGEList(counts=y,group=group,lib.size=rep(1000,4))

# estimate dispersions and find differences in expression
d <- estimateCommonDisp(d)
d <- estimateTagwiseDisp(d)
de <- exactTest(d)
topTags(de)
```

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

---

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 statistical tests based on the fitted models.

Usage

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

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 providing the value for the dispersion parameter that is used in fitting the GLM for each transcript. Can be a common value for all tags, or a vector of values can provide a unique dispersion value for each tag. If `NULL` (default) then dispersion will be detected and extracted from `y`, if possible, with order of precedence: tagwise dispersion, trended dispersions, common dispersion.
`offset`	numeric scalar, vector or matrix giving the offset that is to be included in the NB GLM for the transcripts. Only one of `offset` and `lib.size` should be supplied—if both are supplied then `offset` will be used and `lib.size` will be ignored. If a scalar, then this value will be used as an offset for all transcripts and libraries. If a vector, it should be have length equal to the number of libraries, and the same vector of offsets will be used for each transcript. If a matrix, then each library for each transcript can have a unique offset, if desired. Defaults to the log normalized library sizes.
`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 — optional numeric vector providing the (effective) library size for each library (must have length equal to the number of columns, or libraries, in the matrix of counts). If NULL, then a default is used. If y is a DGEList object then the default for lib.size is the product of the library sizes and the normalization factors (in the samples slot of the object). If y is a simple matrix of counts, then the default for lib.size is the vector of column sums of y.

prior.count.total — the total number of prior counts to be augmented to the data to shrink the estimated log-fold-changes.

start — optional numeric matrix of initial estimates for the fitted 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, the output from glmFit.

coef — scalar or vector indicating the column(s) of design that are to be dropped when creating the null model for the Likelihood Ratio (LR) Test. Can be numeric or character. If character, the string(s) provided to coef must match a column of the design matrix in the glmfit object passed to glmLRT. The glmLRT fits the null model and then conducts an LR test of the model fit provided in glmfit against the null model defined by the choice of coef. By default, the last column of the design matrix is dropped to form the design matrix for the null model.

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

abundance.trend — logical, whether to allow an abundance-dependent trend when estimating the prior values for the quasi-likelihood multiplicative dispersion parameter.

Details

glmFit and glmLRT implement methods developed by McCarthy et al (2012).

glmFit fits genewise negative binomial generalized linear models (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.

glmLRT conducts likelihood ratio tests for one or more coefficients in the linear model.

glmQLFTest conducts quasi-likelihood F-tests for one or more coefficients in the linear model. This function calls the limma function squeezeVar to conduct empirical Bayes smoothing of the genewise multiplicative dispersions.

Value

object of class DGEGLM with the following components:

coefficients — numeric matrix of estimated coefficients from the tagwise linear models, on the natural log scale, of size nrow(y) by ncol(design).

df.residual — numeric vector of giving residual degrees of freedom for each tag.

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

design	design matrix used in the NB model fit for each tag.
offset	numeric matrix of linear model offsets.
samples	data frame of information about the samples (libraries) in the experiment; taken from the object y.
genes	data frame of tag annotatoi information for each tag; taken from the object y.
dispersion	scalar or vector giving the the value of the dispersion parameter used in each tag's NB model fit.
lib.size	vector of library sizes used in the model fit.
weights	matrix of final weights used in the NB model fits for each tag.
fitted.values	matrix of fitted values from the NB model for each tag.
abundance	vector of gene/tag abundances (expression level), on the log2 scale, computed from the mean count for each gene/tag after scaling count by normalized library size.

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

table	data frame (table) containing the abundance of each tag (log-concentration, logConc), the log2-fold change of expression between conditions/contrasts being tested (logFC), the likelihood ratio statistic (LR.statistic) and the p-value from the LR test (p.value), for each tag in the dataset.
coefficients	matrix of coefficients for the full model defined by the design matrix (i.e. for the full model).
dispersion.used	scalar or vector of the dispersion value(s) used in the GLM fits and LR test.

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

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

table	data frame (table) containing the abundance of each tag (log-concentration, logConc), the log2-fold change of expression between conditions/contrasts being tested (logFC), the quasi-likelihood F statistic (F) and the p-value from the QL F test (p.value), for each tag in the dataset.
coefficients	matrix of coefficients for the full model defined by the design matrix (i.e. for the full model).
dispersion.used	scalar or vector of the dispersion value(s) used in the GLM fits and LR test.

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

Author(s)

Davis McCarthy and Gordon Smyth

References

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

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

See topTags for displaying results from glmLRT.

Examples

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

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

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

# Normalize
d <- calcNormFactors(d)

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

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

# 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, plot=FALSE)
goodTuringProportions(counts)
```

Arguments

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

Details

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

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

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
proportion	estimated proportion of species given the count
P0	estimated combined proportion of all undetected species
n0	number of zeros found in x

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

Author(s)

Gordon Smyth

References

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

Examples

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

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

---

logLikDerP *Log-Likelihood for Proportion*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Mark Robinson, Davis McCarthy

See Also

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

Examples

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

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

---

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

Description

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

Usage

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

Arguments

x	vector of counts or concentrations (group 1)
y	vector of counts or concentrations (group 2)
logAbundance	vector providing the abundance of each tag on the log2 scale. Purely optional (default is NULL), but in combination with logFC provides a more direct way to create an MA-plot if the log-abundance and log-fold change are available.
logFC	vector providing the log-fold change for each tag for a given experimental contrast. Default is NULL, only to be used together with logAbundance as both need to be non-null for their values to be used.
normalize	logical, whether to divide x and y vectors by their sum
smearWidth	scalar, width of the smear
col	vector of colours for the points (if NULL, uses allCol and lowCol)
allCol	colour of the non-smeared points
lowCol	colour of the smeared points
deCol	colour of the DE (differentially expressed) points

de.tags	indices for tags identified as being differentially expressed; use exactTest to identify DE genes
smooth.scatter	logical, whether to produce a 'smooth scatter' plot using the KernSmooth::smoothScatter function or just a regular scatter plot; default is FALSE, i.e. produce a regular scatter plot
lowess	logical, indicating whether or not to add a lowess curve to the MA-plot to give an indication of any trend in the log-fold change with log-concentration
...	further arguments passed on to plot

Details

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

Value

a plot to the current device, 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, z, maxit=10, eps=1e-7, plot=FALSE)
```

Arguments

x	numeric vector of the inputs of the function.
z	numeric vector of the values of the function at the inputs given by x.
maxit	numeric scalar giving the maximum number of iterations for the Newton-Raphson algorithm.
eps	numeric scalar giving the convergence tolerance.
plot	logical, whether or not to plot the function on those given points.

Details

`maximizeInterpolant` calls the function `splinefun` to fit cubic spline interpolation given a set of points.

`maximizeInterpolant` uses Newton-Raphson algorithm in finding the maximum of the function performing the interpolation.

Value

`maximizeInterpolant` returns a single value which maximizes the spline interpolation.

Author(s)

Gordon Smyth

See Also

`splinefun`

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.comm
            show.ave.raw.vars=TRUE, scalar=NULL, NBline=FALSE, nbins=100, log.axes="xy", xlab=NU
binMeanVar(x, conc=NULL, 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 CRDisp or estimateCommonDisp and (possibly) estimateTagwiseDisp should be run on the DGEList object before using plotMeanVar. The argument object must be supplied in the function binMeanVar if common dispersion values are to be computed for each bin.
meanvar	list (optional) containing the output from binMeanVar or the returned value of plotMeanVar. Providing this object as an argument will save time in computing the tag/gene means and variances when producing a mean-variance plot.
show.raw.vars	logical, whether or not to display the raw (pooled) gene/tag variances on the mean-variance plot. Default is FALSE.
show.tagwise.vars	logical, whether or not to display the estimated genewise/tagwise variances on the mean-variance plot. Default is FALSE.
show.binned.common.disp.vars	logical, whether or not to compute the common dispersion for each bin of tags and show the variances computed from those binned common dispersions and the mean expression level of the respective bin of tags. Default is 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

conc
: vector (optional) of values for the concentration (i.e. abundance) of each tag

group
: factor giving the experimental group or condition to which each sample (i.e. column of x or element of y) belongs

common.dispersion
: logical, whether or not to compute the common dispersion for each bin of tags.

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 a

## 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
## We could also estimate common/tagwise dispersions using the Cox-Reid methods with an appropriate design
```

---

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, start=NULL, tol=1e-5, maxit=50, trace=FALSE)
mglmOneGroup(y, dispersion=0, offset=0, maxit=50, trace=FALSE)
mglmOneWay(y, design=NULL, dispersion=0, offset=0, maxit=50, trace=FALSE)
mglmSimple(y, design, dispersion=0, offset=0, weights=NULL)
mglmLevenberg(y, design, dispersion=0, offset=0, start=NULL)
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.
start	numeric matrix of starting values for the GLM coefficients. Number of rows should agree with y and number of columns should agree with design.
tol	numeric scalar giving the convergence tolerance.
maxit	scalar giving the maximum number of iterations for the Fisher scoring algorithm.
trace	logical, whether or not to information should be output at each iteration.

Details

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

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

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

mglmSimple is not vectorized, and simply makes tag-wise calls to glm.fit. This has the advantage that it accesses all the usual information generated by glm.fit. Unfortunately, glm.fit does not always converge, and the tag-wise fitting is relatively slow. mglmLevenberg is similar to mglmSimple, but makes tagwise calls to glmnb.fit in the statmod package instead of glm.fit. glmnb.fit implements a Levenberg-Marquardt modification of the scoring algorithm to prevent divergence.

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)

Davis McCarthy, 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*. http://nar.oxfordjournals.org/content/early/2012/02/06/nar.gks042 (Published online 28 January 2012)

See Also

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

Examples

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

---

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

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(object, xlab="logCPM", ylab="Biological coefficient of variation", pch=16, cex=0.2, ...
```

Arguments

`object`	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.
`...`	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=NU
```

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), x
```

Arguments

x	any matrix or DGEList object.
top	number of top genes used to calculate pairwise distances.
labels	character vector of sample names or labels. If x has no column names, then defaults the index of the samples.
col	numeric or character vector of colors for the plotting characters. 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	title for the x-axis
ylab	title for the y-axis
...	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.

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. This is a list containing the following components:

`distance.matrix`	numeric matrix of pairwise distances between columns of `x`
`cmdscale.out`	output from the function `cmdscale` given the distance matrix
`dim.plot`	dimensions plotted
`x`	x-xordinates of plotted points
`y`	y-cordinates of plotted points

Author(s)

Yunshun Chen and Gordon Smyth

See Also

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

Examples

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

y <- matrix(rnbinom(6000, size = 1/2, mu = 10),1000,6)
rownames(y) <- paste("Gene",1:1000)
y[1:300,4:6] <- y[1:300,4:6] + 10
# without labels, indexes of samples are plotted.
mds <- plotMDS(y,  col=c(rep("black",3), rep("red",3)) )
# or labels can be provided, here group indicators:
plotMDS(mds,  col=c(rep("black",3), rep("red",3)), labels= c(rep("Grp1",3), rep("Grp2",3)))
```

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

Description

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

Usage

```
plotSmear(object, pair = NULL, de.tags=NULL, xlab = "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 teh log-fold change with log-concentration
...	further arguments passed on to plot

Details

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

Value

A plot to the current device

Author(s)

Mark Robinson, Davis McCarthy

See Also

maPlot

Examples

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

# find differential expression
de <- exactTest(d)

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

---

predFC *Predictive log fold changes for RNASeq data*

Description

Estimates the predictive log fold changes for a given prior weight using generalised linear models.

Usage

```
## S3 method for class 'DGEList'
predFC(y, design, prior.count.total=0.5, offset=NULL, dispersion=NULL)
## Default S3 method:
predFC(y, design, prior.count.total=0.5, offset=log(colSums(y)), dispersion=0)
```

Arguments

y	a DGEList object
design	the design matrix for the experiment
prior.count.total	the total number of counts to be augmented to the data
offset	numeric vector or matrix giving the offset in the log-linear model predictor. Usually equal to log library sizes
dispersion	the dispersion estimate for the count data

Details

This function estimates the predictive or posterior log-fold-changes for RNASeq or any count-based data. A small count is added to each library in proportion to the library sizes. If there are 2 groups in the experiment, n=2 for each group, the total prior count is 1, and the library sizes are equal, then in effect 0.5 of a count is added to each group, or 0.25 to each library. This prior count is the same for all genes or tags in the data, with the result that genes with low counts will be dampened more severely and genes with a large number of counts in each library will hardly be affected by the addition of a small count to each group.

In order to get the predictive log-fold-changes, a generalised linear model is fitted to the augmented data, and the coefficients outputted in the form of a matrix.

If offset=NULL, the offset used in the glm will be the log of the library sizes.

If dispersion=NULL, the dispersion used for the glm will be dependent on what is in the DGEList object; it is prioritised in the following manner: tagwise, trended, common and finally if no dispersion estimate is found it will set the dispersion to 0.

Value

Numeric matrix of log-fold-changes (natural log). Each column corresponds to the relevant column of the design matrix.

Author(s)

Belinda Phipson, Gordon Smyth

See Also

glmFit, glmLRT for generalised linear model fitting

estimateGLMCommonDisp, estimateGLMTrendedDisp, estimateGLMTagwiseDisp for estimating dispersions in the context of generalised linear models

estimateCommonDisp, estimateTagwiseDisp for estimating dispersions when the design of the experiment is a simple one-way layout.

calcNormFactors for TMM normalisation

Examples

```
# generate counts from a negative binomial distribution for a two group experiment with n=2 in each group
y<-matrix(rnbinom(400,size=1,mu=10),nrow=100)
y<-DGEList(y,group=c(1,1,2,2))
design<-model.matrix(~y$samples$group)
```

```
# apply TMM normalisation
y<-calcNormFactors(y)

# estimate the common dispersion
y <- estimateGLMCommonDisp(y,design)

# fit a glm to find differentially expressed genes
glm<-glmFit(y,design)
results<-glmLRT(y,glm,coef=2)

#estimate the predictive log fold changes
pfc<-predFC(y,design)

#plot predFC's vs logFC's
plot(pfc[,2],results$table$logFC,xlab="Predictive log fold changes",ylab="Raw log fold changes",pch=16,cex=
abline(a=0,b=1)
```

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

Description

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

Usage

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

Arguments

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

Details

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

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

The function calculates the percentiles that the counts in the unadjusted library represent for the normal and gamma distributions with mean and variance defined by the negative binomial rules:

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

Value

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

Author(s)

Gordon Smyth

Examples

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

---

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

Description

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

Usage

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

Arguments

files	character vector of filenames, or alternatively a data.frame with a column containing the file names of the files containing the libraries of counts and, optionally, columns containing the group to which each library belongs, descriptions of the other samples and other information.
path	character string giving the directory containing the files. The default is the current working directory.

columns	numeric vector stating which two columns contain the tag names and counts, respectively
group	vector, or preferably a factor, indicating the experimental group to which each library belongs. If group is not NULL, then this argument overrides any group information included in the files argument.
labels	character vector giving short names to associate with the libraries. Defaults to the file names.
...	other are passed to read.delim

Details

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

Value

DGEList object

Author(s)

Mark Robinson and Gordon Smyth

See Also

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

Examples

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

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

---

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="p.value")
```

Arguments

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

Value

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

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

There is a show method for this class.

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

References

Robinson MD, Smyth GK (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.

`weightedComLik` *Weighted Common Log-Likelihood*

Description

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

Usage

```
weightedComLik(object,l0,prop.used=0.25)
weightedComLikMA(object,l0,prop.used=0.05)
```

Arguments

`object`	`DGEList` object with (at least) elements `counts` (table of unadjusted counts) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size)
`l0`	matrix of the conditional log-likelihood evaluated at a variety of values for the dispersion (on the delta scale, `phi/(1 + phi)`) for each tag/gene. The matrix has number of rows equal to the number of tags/genes and number of columns equal to the number of grid values (between 0 and 1) for the dispersion at which the conditional log-likelihood is evaluated.
`prop.used`	scalar giving the proportion of tags/genes in the whole dataset to use in computing the weighted common log-likelihood for each tag/gene. Default value is `0.25`, i.e. a quarter of the tags/genes in the dataset, for `weightedComLik` and `0.05` for `weightedComLikMA`.

Details

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

Value

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

Author(s)

Davis McCarthy

Examples

```
counts<-matrix(rnbinom(20,size=1,mu=10),nrow=5)
d<-DGEList(counts=counts,group=rep(1:2,each=2),lib.size=rep(c(1000:1001),2))
d<-estimateCommonDisp(d)
ntags<-nrow(d$counts)
y<-splitIntoGroups(new("DGEList",list(counts=d$pseudo.alt,samples=d$samples)))
grid.vals<-seq(0.001,0.999,length.out=10)
l0<-0
for(i in 1:length(y)) {
    l0<-condLogLikDerDelta(y[[i]],grid.vals,der=0,doSum=FALSE)+l0
}
m0 <- ntags*weightedComLik(d,l0,prop.used=0.25) # Weights sum to 1, so need to multiply by number of tags t
# Or use the moving-average method
m1 <- ntags*weightedComLikMA(d,l0,prop.used=0.05)
```

---

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, doSum=FALSE)
```

Arguments

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

Details

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

Value

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

Author(s)

Mark Robinson, Davis McCarthy

Examples

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

---

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