Package 'edgeR'

October 1, 2026

Version 4.99.6

Date 2026-09-12

Title Empirical Analysis of Digital Gene Expression Data in R

Description Differential expression analysis of sequence count data. Implements a range of statistical methodology based on the negative binomial distributions, including empirical Bayes estimation, exact tests, generalized linear models, quasi-likelihood, and gene set enrichment. Can perform differential analyses of any type of omics data that produces read counts, including RNA-seq, ChIP-seq, ATAC-seq, Bisulfite-seq, SAGE, CAGE, metabolomics, or proteomics spectral counts. RNA-seq analyses can be conducted at the gene or isoform level, and tests can be conducted for differential exon or transcript usage.

Author Yunshun Chen, Lizhong Chen, Aaron TL Lun, Davis J McCarthy, Pedro Baldoni, Matthew E Ritchie, Belinda Phipson, Yifang Hu, Xiaobei Zhou, Mark D Robinson, Gordon K Smyth

Maintainer Yunshun Chen <yuchen@wehi.edu.au>, Gordon Smyth <smyth@wehi.edu.au>

License GPL (>=2)

Depends R (>= 3.6.0), limma (>= 3.63.6)

Imports methods, graphics, stats, utils, locfit

Suggests jsonlite, knitr, Matrix, nanoparquet, readr, rhdf5, SeuratObject, splines, AnnotationDbi, Biobase, BiocStyle, org.Hs.eg.db, S4Vectors, SummarizedExperiment

VignetteBuilder knitr

URL https://bioinf.wehi.edu.au/edgeR/, https://bioconductor.org/packages/edgeR

biocViews AlternativeSplicing, BatchEffect, Bayesian, BiomedicalInformatics, CellBiology, ChIPSeq, Clustering, Coverage, DifferentialExpression, DifferentialMethylation, DifferentialSplicing, DNAMethylation, Epigenetics, FunctionalGenomics, GeneExpression, GeneSetEnrichment, Genetics, Genetics, ImmunoOncology, MultipleComparison, Normalization, Pathways, Proteomics, QualityControl, Regression, RNASeq, SAGE, Sequencing, SingleCell, SystemsBiology, TimeCourse, Transcription, Transcriptomics

NeedsCompilation yes

git_url https://git.bioconductor.org/packages/edgeR

git_branch devel

git_last_commit f47ecb9

git_last_commit_date 2026-09-12

Repository Bioconductor 3.24

Date/Publication 2026-10-01

Contents

edgeR-package . . . 4
addPriorCount . . . 6
adjustedProfileLik . . . 7
as.data.frame . . . 9
as.matrix . . . 10
aveLogCPM . . . 10
binFit . . . 12
binomTest . . . 14
binQLFit . . . 15
binQLFTest . . . 17
camera . . . 19
catchSalmon . . . 21
catchSalmonGene . . . 24
cbind . . . 27
commonCondLogLikDerDelta . . . 28
condLogLikDerSize . . . 29
cpm . . . 30
cutWithMinN . . . 32
decideTests . . . 33
DGEBIN-class . . . 35
DGEExact-class . . . 36
DGEGLM-class . . . 37
DGEList . . . 38
DGEList-class . . . 40
DGEListFromTximport . . . 41
DGELRT-class . . . 43
diffSplice.DGEGLM . . . 44
diffSpliceDGE . . . 47
dim . . . 49
dimnames . . . 50
dispBinTrend . . . 51
dispCoxReid . . . 53
dispCoxReidInterpolateTagwise . . . 55
dispCoxReidSplineTrend . . . 57
dropEmptyLevels . . . 58
edgeRUsersGuide . . . 59
equalizeLibSizes . . . 60
estimateCommonDisp . . . 61
estimateDisp . . . 63
estimateExonGenewiseDisp . . . 65
estimateGLMCommonDisp . . . 66
estimateGLMRobustDisp . . . 68
estimateGLMTagwiseDisp . . . 69
estimateGLMTrendedDisp . . . 71

estimateTagwiseDisp . . . 73
estimateTrendedDisp . . . 75
exactTest . . . 76
expandAsMatrix . . . 78
featureCounts2DGEList . . . 79
filterByExpr . . . 80
getCounts . . . 82
getNormLibSizes . . . 83
getPriorN . . . 84
gini . . . 85
glmFit . . . 86
glmLRT . . . 88
glmQLFit . . . 90
glmQLFTest . . . 93
glmTreat . . . 95
goana.DGELRT . . . 98
gof . . . 100
goodTuring . . . 101
head . . . 103
loessByCol . . . 104
makeCompressedMatrix . . . 105
maPlot . . . 107
maximizeInterpolant . . . 109
maximizeQuadratic . . . 110
mbin . . . 111
meanvar . . . 112
mglm . . . 115
modelMatrixMeth . . . 118
movingAverageByCol . . . 119
nbinomDeviance . . . 120
nbinomUnitDeviance . . . 121
nearestReftoX . . . 122
nearestTSS . . . 123
normalizeBetweenArrays.DGEList . . . 124
normalizeChIPtoInput . . . 125
normLibSizes . . . 126
PCList . . . 129
PCList-class . . . 130
plotBCV . . . 131
plotExonUsage . . . 132
plotMD.DGEList . . . 133
plotMDS.DGEList . . . 135
plotMDS.PCList . . . 137
plotMeanVar2 . . . 138
plotQLDisp . . . 140
plotSmear . . . 141
plotSpliceDGE . . . 142
predFC . . . 143
processAmplicons . . . 145
q2qnbinom . . . 147
read10X . . . 148
readBismark2DGE . . . 150

readBismark2PC . . . 151
readDGE . . . 152
roast.DGEGLM . . . 153
roast.DGEList . . . 155
romer.DGEGLM . . . 158
romer.DGEList . . . 159
rowsum . . . 161
sampleWeights . . . 162
scaleOffset . . . 163
SE2DGEList . . . 164
Seurat2PB . . . 165
spliceVariants . . . 166
splitGencodeTxNames . . . 167
splitIntoGroups . . . 168
subsetting . . . 169
sumTechReps . . . 170
systematicSubset . . . 171
thinCounts . . . 172
topSpliceDGE . . . 173
topTags . . . 174
tpm . . . 176
tpmProp . . . 177
validDGEList . . . 178
weightedCondLogLikDerDelta . . . 179
WLEB . . . 180
zscoreNBinom . . . 182

Index **184**

`edgeR-package` *Empirical Analysis of Digital Gene Expression Data in R*

Description

edgeR is a package for the analysis of digital gene expression data arising from RNA sequencing technologies such as SAGE, CAGE, Tag-seq or RNA-seq, with emphasis on testing for differential expression. It can also be used for other sequencing technologies from which read counts are produced, such as ChIP-seq, Hi-C or CRISPR.

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.

The supplied counts are assumed to be those of genes in an RNA-seq experiment. However, counts can be supplied for any genomic feature of interest, e.g., tags, transcripts, exons, or even arbitrary intervals of the genome.

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

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

Author(s)

Yunshun Chen, Aaron TL Lun, Davis J McCarthy, Lizhong Chen, Pedro Baldoni, Matthew E Ritchie, Belinda Phipson, Yifang Hu, Xiaobei Zhou, Mark D Robinson, Gordon K Smyth

References

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

Baldoni PL, Chen L, Smyth GK (2024). Faster and more accurate assessment of differential transcript expression with Gibbs sampling and edgeR v4. *NAR Genomics and Bioinformatics* 6(4), lqae151. doi:10.1093/nargab/lqae151.

Baldoni PL#, Chen Y#, Hediyeh-zadeh S, Liao Y, Dong X, Ritchie ME, Shi W, Smyth GK (2024). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. *Nucleic Acids Research* 52(3), e13. doi:10.1093/nar/gkad1167.

Chen Y, Pal B, Visvader JE, Smyth GK (2017). Differential methylation analysis of reduced representation bisulfite sequencing experiments using edgeR. *F1000Research* 6, 2055.

Lun AT, Smyth GK (2017). No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data. *Statistical Applications in Genetics and Molecular Biology* 16(2), 83-93.

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Lun ATL, Chen Y, Smyth GK (2016). It's DE-licious: a recipe for differential expression analyses of RNA-seq experiments using quasi-likelihood methods in edgeR. *Methods in Molecular Biology* 1418, 391-416. https://gksmyth.github.io/pubs/QLedgeRPreprint.pdf.

Dai Z, Sheridan JM, Gearing LJ, Moore DL, Su S, Wormald S, Wilcox S, O'Connor L, Dickins RA, Blewitt ME, Ritchie ME (2014). edgeR: a versatile tool for the analysis of shRNA-seq and CRISPR-Cas9 genetic screens. *F1000Research* 3, 95.

Chen Y, Lun ATL, Smyth GK (2014). Differential expression analysis of complex RNA-seq experiments using edgeR. In: *Statistical Analysis of Next Generation Sequence Data*, Somnath Datta and Daniel S Nettleton (eds), Springer, New York, pages 51-74. doi:10.1007/9783319072128_3. https://gksmyth.github.io/pubs/edgeRChapterPreprint.pdf.

Zhou X, Lindsay H, Robinson MD (2014). Robustly detecting differential expression in RNA sequencing data using observation weights. *Nucleic Acids Research* 42, e91.

Anders S, McCarthy DJ, Chen Y, Okoniewski M, Smyth GK, Huber W, Robinson MD (2013). Count-based differential expression analysis of RNA sequencing data using R and Bioconductor. *Nature Protocols* 8, 1765-1786.

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.

Robinson MD, Oshlack A (2010). A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, 11, R25.

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

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)

addPriorCount *Add a Prior Count*

Description

Add a library size-adjusted prior count to each observation.

Usage

```
addPriorCount(y, lib.size=NULL, offset=NULL, prior.count=1, nthreads=1L)
```

Arguments

y	a numeric count matrix, with rows corresponding to genes and columns to libraries.
lib.size	a numeric vector of library sizes.
offset	a numeric vector or matrix of offsets.
prior.count	a numeric scalar or vector of prior counts to be added to each gene.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).

Details

This function adds a positive prior count to each observation, often useful for avoiding zeroes during calculation of log-values. For example, predFC will call this function to calculate shrunken log-fold changes. aveLogCPM and cpm also use the same underlying code to calculate (average) log-counts per million.

The actual value added to the counts for each library is scaled according to the library size. This ensures that the relative contribution of the prior is the same for each library. Otherwise, a fixed prior would have little effect on a large library, but a big effect on a small library.

The library sizes are also modified, with twice the scaled prior being added to the library size for each library. To understand the motivation for this, consider that each observation is, effectively, a proportion of the total count in the library. The addition scheme implemented here represents an empirical logistic transform and ensures that the proportion can never be zero or one.

If offset is supplied, this is used in favor of lib.size where exp(offset) is defined as the vector/matrix of library sizes. If an offset matrix is supplied, this will lead to gene-specific scaling of the prior as described above.

Most use cases of this function will involve supplying a constant value to prior.count for all genes. However, it is also possible to use gene-specific values by supplying a vector of length equal to the number of rows in y.

Value

A list is returned containing y, a matrix of counts with the added priors; and offset, a CompressedMatrix containing the (log-transformed) modified library sizes.

Author(s)

Aaron Lun

See Also

aveLogCPM, cpm, predFC

Examples

```
original <- matrix(rnbinom(1000, mu=20, size=10), nrow=200)
head(original)

out <- addPriorCount(original)
head(out$y)
head(out$offset)
```

---

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

Description

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

Usage

```
adjustedProfileLik(dispersion, y, design, offset, weights=NULL, adjust=TRUE,
            start=NULL, get.coef=FALSE, nthreads=1L)
```

Arguments

dispersion	numeric scalar or vector of dispersions.
y	numeric matrix of counts.
design	numeric matrix giving the design matrix.
offset	numeric matrix of same size as y giving offsets for the log-linear models. Can also be a scalar (i.e., a single value) or a vector of length ncol(y), in which case it is expanded out to a matrix.
weights	optional numeric matrix giving observation weights.
adjust	logical, if TRUE then Cox-Reid adjustment is made to the log-likelihood, if FALSE then the log-likelihood is returned without adjustment.
start	numeric matrix of starting values for the GLM coefficients, to be passed to glmFit.
get.coef	logical, specifying whether fitted GLM coefficients should be returned.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).

Details

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

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

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

The purpose of `start` and `get.coef` is to allow hot-starting for multiple calls to `adjustedProfileLik`, when only the `dispersion` is altered. Specifically, the returned GLM coefficients from one call with `get.coef==TRUE` can be used as the `start` values for the next call.

The `weights` argument is interpreted in terms of averages. Each value of `y` is assumed to be the average of `n` independent and identically distributed NB counts, where `n` is given by the weight. This assumption can be generalized to fractional weights.

Value

If `get.coef==FALSE`, a vector of adjusted profile log-likelihood values is returned containing one element for each row of `y`.

Otherwise, a list is returned containing `apl`, the aforementioned vector of adjusted profile likelihoods, and `beta`, the numeric matrix of fitted GLM coefficients.

Author(s)

Yunshun Chen, Gordon Smyth, Aaron Lun

References

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

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

`glmFit`

Examples

```
y <- matrix(rnbinom(30, mu=10, size=20), 10, 3)
design <- matrix(1, 3, 1)
dispersion <- 0.05
adjustedProfileLik(dispersion, y, design, offset=0)
```

as.data.frame *Collapse TopTags or Other edgeR Object to a Data Frame*

Description

Turn a TopTags object into a data.frame.

Usage

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

Arguments

x	an object of class TopTags, DGEList, PCList, DGEGLM, DGEExact or DGELRT.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
...	other arguments are not currently used.

Details

Convert edgeR objects into data.frames. This method returns the table component of a TopTags object. For DGEExact and DGELRT objects, the genes and table components are combined into a data.frame, similar to what is done by topTags but without sorting or p-value adjustment. For DGEList, the genes and counts components are combined into a data.frame. For PCList, the genes, counts and counts2 components are combined into a data.frame.

Among other things, this functionality allows edgeR objects to be written to files using write.table or write.csv.

Value

A data.frame.

Author(s)

Gordon Smyth

See Also

as.data.frame in the base package.

as.matrix *Collapse a DGEList or DGEGLM Object to a Matrix*

Description

Coerce 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 or DGEGLM.
...	additional arguments, not used for these methods.

Details

This method extracts the matrix of counts from a DGEList or the matrix of coefficients from a DGEGLM fit.

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 in the limma package.

aveLogCPM *Average Log Counts per Million*

Description

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

Usage

```
## S3 method for class 'DGEList'
aveLogCPM(y, normalized.lib.sizes=TRUE, prior.count=2, dispersion=NULL, ...)
## S3 method for class 'PCList'
aveLogCPM(y, prior.count=2, ...)
## S3 method for class 'SummarizedExperiment'
aveLogCPM(y, normalized.lib.sizes=TRUE, prior.count=2,
          dispersion=NULL, ...)
## Default S3 method:
aveLogCPM(y, lib.size=NULL, offset=NULL, prior.count=2, dispersion=NULL,
          weights=NULL, nthreads=1L, ...)
```

Arguments

y	numeric matrix containing counts, or a DGEList or PCList object. Rows for genes and columns for libraries.
normalized.lib.sizes	logical, use normalized library sizes?
prior.count	numeric scalar or vector of length nrow(y), containing the average value(s) to be added to each count to avoid infinite values on the log-scale.
dispersion	numeric scalar or vector of negative-binomial dispersions. Defaults to 0.05.
lib.size	numeric vector of library sizes. Defaults to colSums(y). Ignored if offset is not NULL.
offset	numeric matrix of offsets for the log-linear models. Alternatively can be a vector of length equal to the number of samples.
weights	optional numeric matrix of observation weights.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).
...	other arguments are not currently used.

Details

This function uses mglmOneGroup to compute average counts-per-million (AveCPM) for each row of counts, and returns log2(AveCPM). An average value of prior.count is added to the counts before running mglmOneGroup. If prior.count is a vector, each entry will be added to all counts in the corresponding row of y, as described in addPriorCount.

This function is similar to

log2(rowMeans(cpm(y, ...))),

but with the refinement that larger library sizes are given more weight in the average. The two versions will agree for large values of the dispersion.

Value

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

Author(s)

Gordon Smyth. C++ version by Aaron Lun. C version by Lizhong Chen.

See Also

See cpm for individual logCPM values, rather than genewise averages.

Addition of the prior count is performed using the strategy described in addPriorCount.

The computations for aveLogCPM are done by mglmOneGroup.

Examples

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

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

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

# The function works perfectly with prior.count or dispersion vectors:
aveLogCPM(y, prior.count=runif(nrow(y), 1, 5))
aveLogCPM(y, dispersion=runif(nrow(y), 0, 0.2))
```

---

binFit *Genewise Binomial Generalized Linear Models*

Description

Fit a binomial generalized log-linear model to the paired counts for each gene.

Usage

```
## Default S3 method:
binFit(y, z, design=NULL, weights=NULL, offset=NULL, prior.count=1,
                        nthreads=1L, ...)
## S3 method for class 'PCList'
binFit(y, design=NULL, ...)
```

Arguments

y	a matrix of counts or a PCList object. Rows represent genes and columns represent samples.
z	a matrix of counts when y is a matrix. Rows represent genes and columns represent samples.
design	numeric matrix giving the design matrix for the genewise linear models. Must be of full column rank. Defaults to a single column of ones, equivalent to treating the columns as replicate libraries.
weights	positive prior weights for the GLM fits. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, will be set to unity for all observations.

offset	offsets for the log-linear models. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, it will be set to zero for all observations.
prior.count	average prior count to be added to each observation to shrink the estimated log-odds-changes towards zero.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).
...	other arguments are passed to lower level fitting functions.

Details

Implements binomial generalized linear model (GLM) methods developed by Lizhong Chen. Specifically, binFit fits genewise binomial GLMs, all with the same design matrix but possibly different 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 fit very quickly using mbinOneWay. Otherwise the default fitting method, implemented in mbinIWLS, uses a standard iterated weighted least square algorithm.

Value

An object of class DGEBIN (see DGEBIN-class) containing the following components:

coefficients	numeric matrix of estimated coefficients from the glm fits. Coefficients are on the natural log scale. Matrix has dimensions nrow(y) by ncol(design).
unshrunk.coefficients	numeric matrix of estimated coefficients from the unshrunk glm fits. Present only when prior.count>0.
fitted.values	matrix of fitted values from glm fits, same number of rows and columns as y.
deviance	numeric vector of deviances, one for each gene.
df.residual	numeric vector of residual degrees of freedom, one for each gene.
counts	the first matrix of counts as input.
counts2	the second matrix of counts as input.
design	design matrix as input.
weights	matrix of weights as input.
nthreads	integer giving the number of OpenMP threads used for the fit, as specified by the nthreads argument. It is stored on the fitted object so that downstream tests reuse the same number of threads.

When y is a PCList, the components samples, genes and AveLogCPM are also attached.

Author(s)

Lizhong Chen, Gordon Smyth

See Also

Low-level computations are done by mbinOneWay or mbinIWLS.

binQLFit adds empirical Bayes quasi-dispersion moderation to the binomial fit, and binQLFTest performs quasi-likelihood F-tests on the fitted model. topTags displays the results.

Examples

```
ngenes <- 100
Counts1 <- matrix(rpois(ngenes*4,lambda=20),ngenes,4)
Counts2 <- matrix(rpois(ngenes*4,lambda=20),ngenes,4)
y <- PCList(Counts1, Counts2, group=gl(2,2))
fit <- binFit(y)
```

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

Description

Computes p-values for differential abundance for each gene between two digital libraries, conditioning on the total count for each gene. 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 the count for each gene in the first library. Non-integer values are rounded to the nearest integer.
`y2`	integer vector giving the count for each gene in the second library. Of same length as `y1`. Non-integer values are rounded to the nearest integer.
`n1`	total number of counts in the first library, across all genes. Non-integer values are rounded to the nearest integer. Not required if `p` is supplied.
`n2`	total number of counts in the second library, across all genes. Non-integer values are rounded to the nearest integer. Not required if `p` is supplied.
`p`	expected proportion of `y1` to the total for each gene 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 gene. 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 gene. 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 package, but is much faster.

Value

Numeric vector of p-values.

Author(s)

Gordon Smyth

References

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

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

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

https://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
```

---

binQLFit *Quasi-Likelihood Binomial Generalized Linear Models*

Description

Fit a binomial generalized log-linear model to the paired counts for each gene and estimate the empirical Bayes moderated quasi-dispersions.

Usage

```
## Default S3 method:
binQLFit(y, z, design=NULL, weights=NULL, offset=NULL, AveLogCPM=NULL,
                        abundance.trend=TRUE, covariate.trend=NULL, robust=TRUE,
                            prior.count=1, nthreads=1L, ...)
## S3 method for class 'PCList'
binQLFit(y, design=NULL, abundance.trend=TRUE, covariate.trend=NULL, robust=TRUE, ...)
```

Arguments

y	a matrix of counts or a PCList object. Rows represent genes and columns represent samples.
z	a matrix of counts when y is a matrix. Rows represent genes and columns represent samples.
design	numeric matrix giving the design matrix for the genewise linear models. Must be of full column rank. Defaults to a single column of ones, equivalent to treating the columns as replicate libraries.

weights	positive prior weights for the GLM fits. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, will be set to unity for all observations.
offset	offsets for the log-linear models. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, it will be set to zero for all observations.
AveLogCPM	numeric vector giving average log2-counts per million for each row of y. If NULL, then will be computed by aveLogCPM(y).
abundance.trend	logical, whether to allow an abundance trend for the quasi-dispersion prior.
covariate.trend	numeric vector of length nrow(y). If non-NULL, then will be used instead of AveLogCPM as the covariate for the trended quasi-dispersion prior.
robust	logical, whether to estimate the prior distribution for the quasi-dispersions robustly.
prior.count	average prior count to be added to each observation to shrink the estimated log-odds-changes towards zero.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).
...	other arguments are passed to lower level fitting functions.

Details

binQLFit is the binomial analogue of glmQLFit. It first fits genewise binomial GLMs (as done by binFit), all with the same design matrix but possibly different offsets and weights, then estimates the genewise quasi-dispersions and moderates them towards a common prior by empirical Bayes using squeezeVar. When the design matrix defines a one-way layout, or can be re-parametrized to a one-way layout, the GLMs are fit very quickly using mbinOneWay. Otherwise the default fitting method, implemented in mbinIWLS, uses a standard iterated weighted least square algorithm.

The moderated quasi-dispersions (s2.post) and adjusted degrees of freedom are required by binQLFTest to conduct the quasi-likelihood F-tests.

Value

An object of class DGEBIN (see DGEBIN-class) with the raw-fit components produced by binFit plus the following:

df.residual.adj	numeric vector of adjusted residual degrees of freedom for each gene.
deviance.adj	numeric vector of adjusted deviances, one for each gene.
df.prior	numeric vector of prior degrees of freedom for the quasi-dispersions. Has length nrow(y) if robust=TRUE, otherwise length 1.
s2.prior	numeric vector giving prior value for the quasi-dispersions. Has length nrow(y) if abundance.trend=TRUE, otherwise length 1.
s2.post	numeric vector of posterior genewise quasi-dispersions.
average.ql.dispersion	average quasi-dispersion, the overall overdispersion for binomial models.
AveLogCPM	numeric vector giving average log2-counts per million for each gene.

Author(s)

Lizhong Chen, Gordon Smyth

See Also

binFit performs the underlying binomial GLM fit without the quasi-dispersion moderation.

binQLFTest performs quasi-likelihood F-tests on the fitted model and topTags displays the results.

glmQLFit performs the analogous quasi-likelihood fit for negative binomial models.

Examples

```
ngenes <- 100
Counts1 <- matrix(rpois(ngenes*4, lambda=20), ngenes, 4)
Counts2 <- matrix(rpois(ngenes*4, lambda=20), ngenes, 4)
y <- PCList(Counts1, Counts2, group=gl(2,2))
fit <- binQLFit(y)
res <- binQLFTest(fit, coef=2)
topTags(res)
```

---

binQLFTest *Quasi-Likelihood F-Tests for Binomial Models*

Description

Conduct genewise quasi-likelihood F-tests for a given coefficient or coefficient contrast, for binomial models fitted by binQLFit.

Usage

```
binQLFTest(glmfit, coef = ncol(glmfit$design), contrast = NULL, or = 1, lor = NULL,
        upshot = TRUE)
```

Arguments

glmfit	a DGEBIN object, usually output from binQLFit. Must contain the empirical Bayes quasi-dispersions (s2.post).
coef	integer or character index vector indicating which coefficients of the binomial linear model are to be tested equal to zero. Ignored if contrast is not NULL.
contrast	numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero.
or	scalar or numeric vector of minimum odds ratio below which changes are not considered scientifically meaningful.
lor	scalar or numeric vector of minimum absolute log2 odds ratio below which changes are not considered scientifically meaningful. Defaults to log2(or). If specified then takes precedence over or.
upshot	logical. If TRUE, then a more powerful test is conducted assuming that the true log2 odds ratios are uniformly distributed over the null interval from -lor to lor. If FALSE, then the standard TREAT p-values are computed assuming the worst-case scenario of true log2 odds ratio equal to lor.

Details

binQLFTest is the binomial analogue of glmQLFTest and is typically used after binQLFit. It conducts empirical Bayes quasi-likelihood F-tests for the binomial generalized linear models fitted genewise to paired counts (for example methylated and unmethylated read counts) by binQLFit.

The recommended workflow is to build a PCList object, fit the models with binQLFit, then test with binQLFTest: binQLFit attaches the average log2-CPM (AveLogCPM) and the moderated quasi-dispersions that binQLFTest requires.

The reported logOR is the log2 odds ratio for the tested coefficient or contrast, equal to the estimated coefficient divided by log(2). When binQLFit is run with prior.count>0 (the default), logOR is reported on the shrunk coefficient scale, whereas the F-test and its deviances are computed from the unshrunk maximum-likelihood fit.

When a nonzero threshold is specified via or or lor, a TREAT-style test relative to the threshold is performed. With upshot=TRUE the more powerful UPSHOT p-values are computed by Gauss quadrature over the null interval.

Value

An object of class DGELRT with the same components as glmfit plus the following:

table	data frame with the same rows as glmfit and the following columns: logOR, the log2 odds ratio for the tested coefficient or contrast; logCPM, the average log2-counts per million; F, the quasi-likelihood F-statistic; PValue, the associated p-value.
comparison	character string describing the coefficient or contrast being tested.
df.test	numerator degrees of freedom for the F-test, equal to the number of coefficients tested.
df.total	numeric vector of denominator (total) degrees of freedom for the F-test.

Author(s)

Lizhong Chen, Gordon Smyth

See Also

binQLFit fits the binomial models that are provided to binQLFTest, and PCList constructs the paired-count data object.

topTags displays the results in a convenient form.

glmQLFTest performs the analogous quasi-likelihood F-tests for negative binomial models.

Examples

```
ngenes <- 100
Counts1 <- matrix(rpois(ngenes*4, lambda=20), ngenes, 4)
Counts2 <- matrix(rpois(ngenes*4, lambda=20), ngenes, 4)
y <- PCList(Counts1, Counts2, group=gl(2,2))
fit <- binQLFit(y)
res <- binQLFTest(fit, coef=2)
topTags(res)
```

camera *Competitive Gene Set Tests for Digital Gene Expression Data*

Description

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

Usage

```
## S3 method for class 'DGEList'
camera(y, index, design = NULL, contrast = ncol(design), weights = NULL,
     use.ranks = FALSE, allow.neg.cor=FALSE, inter.gene.cor=0.01, sort = TRUE, ...)
## S3 method for class 'DGEGLM'
camera(y, index, design = NULL, contrast = ncol(design), weights = NULL,
     use.ranks = FALSE, allow.neg.cor=FALSE, inter.gene.cor=0.01, sort = TRUE, ...)
## S3 method for class 'DGELRT'
cameraPR(statistic, index, use.ranks = FALSE, inter.gene.cor=0.01, sort = TRUE, ...)
```

Arguments

y	a DGEList object containing dispersion estimates or a DGEGLM object.
index	an index vector or a list of index vectors. Can be any vector such that y[index,] selects the rows corresponding to the test set. The list can be made using ids2indices.
design	design matrix. Defaults to y$design or, if that is NULL, to model.matrix(~y$samples$group).
contrast	contrast of the linear model coefficients for which the test is required. Can be an integer specifying a column of design, or else a numeric vector of same length as the number of columns of design.
weights	numeric matrix of observation weights of same size as y, or a numeric vector of array weights with length equal to ncol(y), or a numeric vector of gene weights with length equal to nrow(y).
use.ranks	do a rank-based test (TRUE) or a parametric test (FALSE)?
allow.neg.cor	should reduced variance inflation factors be allowed for negative correlations?
inter.gene.cor	numeric, optional preset value for the inter-gene correlation within tested sets. If NA or NULL, then an inter-gene correlation will be estimated for each tested set.
sort	logical, should the results be sorted by p-value?
statistic	a DGELRT object.
...	other arguments are not currently used.

Details

Camera is a competitive gene set test proposed by Wu and Smyth (2012) for microarray data and camera is implemented as an S3 generic function in the limma package. It is often used for gene set enrichment analyses (GSEA) together with a database of gene sets such as the MSigDB. Here we provide camera methods for DGEList and DGEGLM class objects. The negative binomial count data is converted to approximate normal deviates by computing mid-p quantile residuals (Dunn and

Smyth, 1996; Routledge, 1994), under the null hypothesis that the contrast is zero, and the normal deviates are then passed to limma's camera function.

The cameraPR function is a variation of the camera method for pre-ranked differential expression statistics. We provide here a cameraPR method for DGELRT objects.

Value

A data.frame giving the gene set results, with a row for each gene set defined by index. If sort=TRUE then results are sorted in decreasing order of significance. See camera for details.

Author(s)

Yunshun Chen, Gordon Smyth

References

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *Journal of Computational and Graphical Statistics* 5, 236-244. doi:10.1080/10618600.1996.10474708. https://gksmyth.github.io/pubs/residual.html.

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

Wu D, Smyth GK (2012). Camera: a competitive gene set test accounting for inter-gene correlation. *Nucleic Acids Research* 40, e133. doi:10.1093/nar/gks461.

See Also

camera.

Examples

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

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

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

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

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

# Gene set tests
camera(y, iset1, design)
camera(y, iset2, design)
camera(y, list(set1=iset1,set2=iset2), design)

# Alternative pre-ranked version
fit <- glmQLFit(y, design)
```

```
q <- glmQLFTest(fit)
cameraPR(q, list(set1=iset1,set2=iset2))
```

`catchSalmon` *Process kallisto, Salmon, Oarfish, or RSEM Transcript Output*

Description

Read transcript counts from RSEM, kallisto, Salmon or Oarfish output for a series of biological samples and use bootstrap or posterior distribution samples to estimate the read-to-transcript-ambiguity for each transcript.

Usage

```
catchKallisto(parent.dir = NULL, sample.dirs = NULL,
          DGEList = TRUE, divide = FALSE,
          impute.eff.len = TRUE, offset.prior = TRUE, verbose = TRUE)
catchSalmon(parent.dir = NULL, sample.dirs = NULL,
          DGEList = TRUE, divide = FALSE,
          impute.eff.len = TRUE, offset.prior = TRUE, verbose = TRUE)
catchOarfish(parent.dir = NULL, prefixes = NULL,
          DGEList = TRUE, divide = FALSE, verbose = TRUE)
catchRSEM(parent.dir = NULL, files = NULL, ngibbs = 100,
          DGEList = TRUE, divide = FALSE, offset.prior = TRUE, verbose = TRUE)
```

Arguments

`parent.dir`	name of parent directory containing the output directories from kallisto or Salmon or the output files from Oarfish or RSEM.
`sample.dirs`	character vector giving names of or paths to the sample directories created by kallisto or Salmon. Each entry corresponds to one RNA-seq sample. If `parent.dir` and `sample.dirs` are both provided, then the `sample.dirs` are relative to `parent.dir`. If `sample.dirs` is `NULL`, then all the kallisto or Salmon directories found in `parent.dir` will be read.
`prefixes`	character vector giving the sample-specific file prefixes created by Oarfish. Each entry corresponds to one RNA-seq sample. Can contain a directory separator. By default, all files in the `parent.dir` directory with names ending in `.quant`, `.infreps.pq` and `.meta_info.json` will be read.
`files`	character vector specifying the `isoforms.results` files output by RSEM. Each entry corresponds to one RNA-seq sample. By default, all files in the `parent.dir` directory with names ending in `isoforms.results` will be read.
`DGEList`	logical. If `TRUE`, the function will output a `DGEList` instead of a plain list.
`divide`	logical. If `TRUE`, counts are divided by the RTA overdispersions.
`impute.eff.len`	whether to impute missing effective transcript lengths from Salmon.
`offset.prior`	logical. If `TRUE`, a prior offset matrix representing differences in effective transcript length will be included in the DGEList.
`verbose`	logical. If `TRUE`, progress information will be sent to standard output as each sample is processed.
`ngibbs`	number of Gibbs samples used to generate the RSEM results files.

Details

Reading transcript counts. `catchSalmon` assumes that Salmon (Patro et al 2017; Zakeri et al 2017) has been run to estimate transcript counts for one or more RNA samples and that either bootstrap resamples or Gibbs posterior samples are included in the Salmon output. `catchSalmon` will automatically detect whether bootstrap or Gibbs samples are available. The number of technical resamples does not need to be the same for each RNA sample, but the resamples should always be of the same type, either bootstrap or Gibbs. We recommend that at least 200 bootstrap or Gibbs resamples are generated in total across all the RNA samples (Baldoni et al 2024b).

`catchKallisto` assumes that kallisto (Bray et al 2016) has been run to estimate transcript counts for one or more RNA samples and that bootstrap samples have also been generated. The number of bootstrap resamples does not need to be the same for each RNA sample. `catchKallisto` uses the rhdf5 package to read the HDF5 format files output by kallisto.

`catchRSEM` reads `*.isoforms.results` files created by RSEM (Li and Dewey, 2011), which should contain means and standard deviations from Gibbs posterior samples as well as the estimated transcript counts. `catchRSEM` cannot detect the number of Gibbs samples performed for each RNA sample, so this must be specified via the `ngibbs` argument. The number of Gibbs samples is assumed to be the same for each RNA sample.

`catchOarfish` assumes that Oarfish (Jousheghani et al 2025) has been run to estimate transcript counts for one or more RNA samples and that bootstrap resamples are included in the output.

RTA overdispersion. These functions read the transcript counts and use the technical (bootstrap or Gibbs) resamples to estimate an overdispersion parameter for each transcript. The overdispersion represents the variance inflation that occurs from ambiguity in assigning sequence reads to transcripts, a phenomenon that is called *read to transcript ambiguity* (RTA) overdispersion by Baldoni et al (2024ab). Transcripts that overlap other transcripts and have greater quantification uncertainty will have larger RTA overdispersions. The RTA overdispersions are greater than or equal to 1, with 1 representing no variance inflation.

Divided counts. To assess differential transcript expression, the transcript counts can be divided by the overdispersion parameters, after which the divided-counts can be input into standard differential expression pipelines designed for gene-level counts (Baldoni et al 2024a). The edgeR quasi pipeline has been found to perform well with the divided counts (Baldoni et al 2024ab). The divided-counts behave much like negative binomial counts and show the same mean-variance trends as for gene-level RNA-seq counts. While bootstrap and Gibbs resamples both perform well, Baldoni et al (2024b) show that Gibbs resamples are computationally faster than bootstrap resamples and give slightly better performance in the downstream differential expression analyses.

Imputing missing transcript lengths. For paired-end RNA-seq, Salmon uses sample-specific empirical fragment distributions to estimate the effective transcript lengths. For very short transcripts, there may be too few entries in the empirical distribution to estimate the conditional fragment length. In such cases, Salmon falls back to setting the effective transcript length equal to the actual length even though the effective transcript length should in principle be much shorter. If `impute.eff.len` is `TRUE`, then any effective transcript length equal to the transcript's actual length will be reset to the minimum effective transcript length for that transript across all the samples in the series.

Gencode imbedded annotation. Gencode packs transcript and gene annotation into composite transcript IDs, with fields separated by vertical bar `|` characters. All the functions above will detect Gencode-style transcript names and will unpack the annotation fields into separate annotation columns using `splitGencodeTxNames`.

Value

If `DGEList=TRUE`, a `DGEList` object with components:

`counts` matrix of transcript counts, with rows for transcripts and columns for RNA samples.

`genes` data.frame of transcript information with columns `Length`, `AveEffLen`, `Max2MinEffLen`, and `Overdispersion` corresponding to transcript length, effective transcript length averaged over samples, ratio of largest to smallest length for different samples, and RTA overdispersion respectively.

`overdispersion.prior` median overdispersion, used to moderate the transcript-wise overdispersion values.

`resample.type` character vector giving type of resampling (`"bootstrap"` or `"gibbs"`) for each sample.

`divided.counts` a logical indicating whether the counts have been divided by the RTA overdispersions.

`offset.prior` matrix of prior offsets representing relative differences in effective lengths for each transcript on the log-scale (only if `offset.prior=TRUE`). Rows add to zero.

`other$effective.length` matrix of effective transcript lengths.

If Gencode-style composite rownames are detected, then the rownames will be reset to plain Ensembl transcript IDs, without veresion number suffixes, and the `genes` data.frame will include columns:

`EnsG` Ensembl gene ID, without version number suffix.

`HavG` Havana gene ID.

`HavT` Havana transcript ID.

`TxName` Transcript name.

`GeneName` Gene name.

`Type` Type of transcript.

If `DGEList=FALSE`, then a plain list is created. The components are as documented above for DGELists except that the annotation data.frame is called `annotation` instead of `genes`, and the `effective.length` matrix is a top-level component.

For `catchOarfish`, the `annotation` data.frame does not contain the `AveEffLen` and `Max2MinEffLen` columns.

Author(s)

Gordon Smyth and Pedro Baldoni

References

Baldoni PL, Chen Y, Hediyeh-zadeh S, Liao Y, Dong X, Ritchie ME, Shi W, Smyth GK (2024a). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. *Nucleic Acids Research* 52(3), e13. doi:10.1093/nar/gkad1167.

Baldoni PL, Chen L, Smyth GK (2024b). Faster and more accurate assessment of differential transcript expression with Gibbs sampling and edgeR v4. *NAR Genomics and Bioinformatics* 6(4), lqae151. doi:10.1093/nargab/lqae151.

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

Bray NL, Pimentel H, Melsted P, Pachter L (2016). Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology* 34(5), 525-527.

Jousheghani ZZ, Singh NP, Patro R (2025). Oarfish: enhanced probabilistic modeling leads to improved accuracy in long read transcriptome quantification. *Bioinformatics* 41, i304-i313. doi:10.1093/bioinformatics/btaf240.

Li B, Dewey CN (2011). RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics* 12, 323.

Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C (2017). Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods* 14(4), 417-419.

Zakeri M, Srivastava A, Almodaresi F, Patro R (2017). Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics* 33(14), i142-i151.

See Also

See `catchSalmonGene` for gene-level counts and overdispersions.

Examples

```
## Not run:
# Read Salmon ouput and estimate overdispersion for each transcript
s <- catchSalmon(paths)

# Divide the transcript counts by the overdispersions ready for a standard edgeR DE analysis
dge <- DGEList(counts=s$counts/s$annotation$Overdispersion, genes=s$annotation)

## End(Not run)
```

`catchSalmonGene` *Process Kallisto or Salmon Output to Produce Gene Counts*

Description

Produce a gene count object from kallisto or Salmon output for a series of biological samples, optionally with Gencode annotation.

Usage

```
catchKallistoGene(parent.dir = NULL, sample.dirs = NULL,
        tx2gene = NULL, remove.version.numbers = TRUE,
        DGEList = TRUE, divide = FALSE, impute.eff.len = TRUE,
        offset.prior = TRUE, gene.length = "moderate", verbose = TRUE)
catchSalmonGene(parent.dir = NULL, sample.dirs = NULL,
        tx2gene = NULL, remove.version.numbers = TRUE,
        DGEList = TRUE, divide = FALSE, impute.eff.len = TRUE,
        offset.prior = TRUE, gene.length = "moderate", verbose = TRUE)
```

Arguments

parent.dir	name of parent directory containing the Salmon output directories.
sample.dirs	character vector giving paths to the sample directories created by Salmon. If parent.dir and sample.dirs are both provided, then the sample.dirs are relative to parent.dir. If sample.dirs is NULL, then all the Salmon directories found in parent.dir will be read.
tx2gene	data.frame mapping transcript IDs to gene IDs, with first column for transcript IDs and the second for the corresponding gene IDs. Not needed if Gencode annotation was used for the quantifications.
remove.version.numbers	logical, whether to remove version numbers from transcript and gene IDs.
DGEList	logical, whether to output a DGEList instead of a plain list.
divide	logical, whether to divide the gene counts by the RTA overdispersions.
impute.eff.len	logical, whether to impute missing effective transcript lengths.
offset.prior	logical, whether to include a prior offset matrix representing differences in effective gene length.
gene.length	how to compute effective gene length. Possibilities are "moderate", "tximport" and "simple".
verbose	logical, whether to message progress information to standard output as each sample is processed.

Details

catchSalmonGene() assumes that Salmon (Patro et al 2017; Zakeri et al 2017) has been run to estimate transcript counts for one or more RNA samples, and that either bootstrap resamples or Gibbs posterior samples are included in the Salmon output. The function reads the Salmon quantifications and summarizes them at the gene level. The function will automatically detect whether bootstrap or Gibbs samples are available.

catchKallistoGene() assumes that kallisto (Bray et al 2016) has been run to estimate transcript counts for one or more RNA samples and that bootstrap samples have also been generated. The number of bootstrap resamples does not need to be the same for each RNA sample. catchKallistoGene() uses the rhdf5 package to read the HDF5 format files output by kallisto.

The transcript-level counts will be collapsed to gene-level counts and RTA overdispersions will be estimated at the gene level, using a method similar at Baldoni et al (2024ab) but applied to gene level counts instead of transcript counts.

The function needs to know which gene each transcript belongs to. If Salmon was run with a Gencode transcript FASTA file, then catchSalmonGene will detect that automatically. It will then unpack the Gencode transcript names using splitGencodeTxNames to obtain gene and transcript IDs. Otherwise, a data.frame tx2gene needs to be provided to map transcript to gene IDs.

Average transcript lengths are computed for each gene and optionally used to create a prior offset matrix. The average lengths are computed by three possible methods as specified by gene.length, either a simple average of the transcript lengths ("simple"), or the method of Soneson et al (2016) ("tximport"), or by a method that moderates the lengths weakly towards the gene average and, even more weakly, towards the simple average of transcript lengths ("moderate"). The moderated method is recommended.

For paired-end RNA-seq, kallisto and Salmon use sample-specific empirical fragment distributions to estimate the effective transcript lengths. For very short transcripts, there may be too few entries in the empirical distribution to estimate the conditional fragment length. In such cases, kallisto and

Salmon fall back to setting the effective transcript length equal to the actual length even though the effective transcript length should in principle be much shorter. If `impute.eff.len` is `TRUE`, then any effective transcript length equal to the transcript's actual length will be reset to the minimum effective transcript length for that transcript across all the samples in the series.

Value

If `DGEList=TRUE`, a `DGEList` object with gene IDs as row.names and the following components:

`counts`	matrix of gene counts, with rows for genes and columns for RNA samples.
`genes`	data.frame of gene annotation.
`genes$GeneName`	gene name (if Gencode annotation detected).
`genes$NTx`	number of transcripts per gene.
`genes$MaxTxLen`	maximum transcript length per gene.
`genes$AveEffLen`	effective gene length, averaged over samples.
`genes$Max2MinEffLen`	ratio of largest to smallest effective gene length across samples.
`genes$Overdispersion`	genewise RTA overdispersion.
`overdispersion.prior`	median of genewise overdispersions, used to moderate the overdispersion values.
`resample.type`	character vector giving the resampling type (`"bootstrap"` or `"gibbs"`) for each sample.
`divided.counts`	logical, indicating whether the counts have been divided by the RTA overdispersions.
`offset.prior`	matrix of prior offsets representing relative differences in effective lengths for each gene on the log-scale (only if `offset.prior=TRUE`). Rows add to zero.
`other$effective.length`	matrix of effective gene lengths.
`other$tpm`	matrix of genewise TPM values.

If `DGEList=FALSE`, then a plain list is created. The components are as documented above for DGELists except that the annotation data.frame is called `annotation` instead of `genes`, and the `effective.length` and `tpm` matrices are top-level components.

Author(s)

Gordon Smyth and Pedro Baldoni

References

Baldoni PL, Chen Y, Hediyeh-zadeh S, Liao Y, Dong X, Ritchie ME, Shi W, Smyth GK (2024a). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. *Nucleic Acids Research* 52(3), e13. doi:10.1093/nar/gkad1167.

Baldoni PL, Chen L, Smyth GK (2024b). Faster and more accurate assessment of differential transcript expression with Gibbs sampling and edgeR v4. *NAR Genomics and Bioinformatics* 6(4), lqae151. doi:10.1093/nargab/lqae151.

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

Bray NL, Pimentel H, Melsted P, Pachter L (2016). Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology* 34(5), 525-527.

Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C (2017). Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods* 14(4), 417-419.

Soneson C, Love MI, Robinson MD (2016). Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research* 4, 1521. doi:10.12688/f1000research.7563.2.

Zakeri M, Srivastava A, Almodaresi F, Patro R (2017). Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics* 33(14), i142-i151.

See Also

`catchKallisto`, `catchSalmon`, `splitGencodeTxNames`.

Examples

```
## Not run: dge <- DGEList(mydir)
```

`cbind` *Combine DGEList Objects*

Description

Combine a set of `DGEList` objects.

Usage

```
## S3 method for class 'DGEList'
cbind(..., deparse.level=1)
## S3 method for class 'DGEList'
rbind(..., deparse.level=1)
```

Arguments

`...`	`DGEList` objects.
`deparse.level`	not currently used, see `cbind` in the base package.

Details

`cbind` combines data objects assuming the same genes in the same order but different samples. `rbind` combines data objects assuming equivalent samples, i.e., the same RNA targets, but different genes.

For `cbind`, the matrices of count data from the individual objects are cbinded. The data.frames of samples information, if they exist, are rbinded. The combined data object will preserve any additional components or attributes found in the first object to be combined. For `rbind`, the matrices of count data are rbinded while the sample information is unchanged.

Value

An DGEList object holding data from all samples and all genes from the individual objects.

Author(s)

Gordon Smyth

See Also

cbind in the base package.

Examples

```
## Not run:
dge <- cbind(dge1,dge2,dge3)

## End(Not run)
```

---

commonCondLogLikDerDelta

Conditional Log-Likelihoods in Terms of Delta

Description

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

Usage

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

Arguments

y	list with elements comprising the matrices of count data (or pseudocounts) for the different groups.
delta	delta (phi / (phi+1)) parameter of negative binomial.
der	derivative, either 0 (the function), 1 (first derivative) or 2 (second derivative).

Details

The common conditional log-likelihood is constructed by summing over all of the individual genewise 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 genes.

Value

numeric scalar of function/derivative evaluated at given delta

Author(s)

Davis McCarthy

See Also

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

Examples

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

---

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

Description

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

Usage

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

Arguments

y	matrix of counts, all counts in each row having the same population mean.
r	numeric vector or scalar, size parameter of negative binomial distribution, equal to 1/dispersion.
delta	numeric vector or scalar, delta parameter of negative binomial, equal to dispersion/(1+dispersion).
der	integer specifying derivative required, either 0 (the function), 1 (first derivative) or 2 (second derivative).

Details

The library sizes must be equalized before running this function. This function carries out the actual mathematical computations for the conditional log-likelihood and its derivatives, calculating the conditional log-likelihood for each gene. Derivatives are with respect to either the size (r) or the delta parametrization (delta) of the dispersion.

Value

vector of row-wise derivatives with respect to r or delta

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

Examples

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

---

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

Description

Compute counts per million (CPM) or reads per kilobase per million (RPKM).

Usage

```
## S3 method for class 'DGEList'
cpm(y, normalized.lib.sizes = TRUE,
       log = FALSE, prior.count = 2, ...)
## S3 method for class 'SummarizedExperiment'
cpm(y, normalized.lib.sizes = TRUE,
       log = FALSE, prior.count = 2, ...)
## S3 method for class 'DGEGLM'
cpm(y, log = FALSE, shrunk = TRUE, ...)
## Default S3 method:
cpm(y, lib.size = NULL, offset = NULL, offset.prior = NULL,
       log = FALSE, prior.count = 2, nthreads = 1L, ...)
## S3 method for class 'DGEList'
rpkm(y, gene.length = NULL, normalized.lib.sizes = TRUE,
       log = FALSE, prior.count = 2, ...)
## S3 method for class 'SummarizedExperiment'
rpkm(y, gene.length = NULL, normalized.lib.sizes = TRUE,
       log = FALSE, prior.count = 2, ...)
## S3 method for class 'DGEGLM'
rpkm(y, gene.length, log = FALSE, shrunk = TRUE, ...)
## Default S3 method:
rpkm(y, gene.length, lib.size = NULL, offset = NULL, offset.prior = NULL,
       log = FALSE, prior.count = 2, ...)
## S3 method for class 'DGEList'
cpmByGroup(y, group = NULL, dispersion = NULL, ...)
## S3 method for class 'SummarizedExperiment'
cpmByGroup(y, group = NULL, dispersion = NULL, ...)
## Default S3 method:
cpmByGroup(y, group = NULL, dispersion = 0.05,
       offset = NULL, weights = NULL, log = FALSE, prior.count = 2, ...)
## S3 method for class 'DGEList'
rpkmByGroup(y, group = NULL, gene.length = NULL, dispersion = NULL, ...)
## S3 method for class 'SummarizedExperiment'
rpkmByGroup(y, group = NULL, gene.length = NULL, dispersion = NULL, ...)
## Default S3 method:
rpkmByGroup(y, group = NULL, gene.length, dispersion = 0.05,
       offset = NULL, weights = NULL, log = FALSE, prior.count = 2, ...)
```

Arguments

y	a matrix-like object containing counts. Can be a numeric matrix, a DGEList object, a SummarizedExperiment object with a "counts" assay, or any object that can be coerced to a matrix by as.matrix. For cpm and rpkm, it can also be a limma or edgeR fitted model object (MArrayLM, DGEGLM, or DGELRT).
normalized.lib.sizes	logical, use normalized library sizes?
lib.size	library size, defaults to colSums(y). Ignored if offset is specified.
offset	alternative name for offset.prior for compatibility with older code.
offset.prior	offset matrix of same dimensions as y with row means equal to zero, containing observation-specific normalization factors for the library sizes on the log-scale. Used to implement observation-specific normalization strategies based on effective transcript lengths or GC content. This type of normalization is independent of library size normalization based on count distributions as done by normLibSizes(), and the two are treated cumulatively in edgeR.
log	logical, if TRUE then log2 values are returned.
prior.count	average count to be added to each observation to avoid taking log of zero. Used only if log=TRUE.
shrunk	logical, when to use shrunk coefficients. If TRUE then the usual coefficients from the fitted object will be used. If FALSE then the unshrunk coefficients will be used.
gene.length	vector of length nrow(y) giving gene length in bases, or the name of the column y$genes containing the gene lengths.
group	factor giving group membership for columns of y. Defaults to y$sample$group for the DGEList method and to a single level factor for the default method.
dispersion	numeric vector of negative binomial dispersions.
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.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).
...	other arguments can be passed down to cpm.default from other functions.

Details

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

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

The rpkm methods for DGEList, DGEGLM or DGELRT objects will try to find the gene lengths in a column of y$genes called Length or length. Failing that, it will look for any column name containing "length" in any capitalization.

The cpm and rpkm methods for DGEGLM and DGELRT fitted model objects return fitted CPM or RPKM values. If shrunk=TRUE, then the CPM or RPKM values will reflect the prior.count input to the original linear model fit. If shrunk=FALSE, then the CPM or RPKM values will be computed with prior.count=0. Note that the latter could result in taking the log of near-zero values if log=TRUE.

If a prior offset matrix is supplied, either as an argument or as a component of the DGEList, then observation-specific effective library sizes will be obtained by adding `offset.prior` to the log library sizes. If an ordinary offset matrix is supplied (`offset`), either as an argument or in the DGEList, then it will be row-mean corrected and treated as a prior offset matrix.

`cpmByGroup` and `rpkmByGroup` compute group average values on the unlogged scale.

Value

A numeric matrix of CPM or RPKM values, on the log2 scale if `log=TRUE`. `cpm` and `rpkm` produce matrices of the same size as `y`. If `y` was a data object, then observed values are returned. If `y` was a fitted model object, then fitted values are returned.

`cpmByGroup` and `rpkmByGroup` produce matrices with a column for each level of `group`.

Note

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

Author(s)

Davis McCarthy, Gordon Smyth, Yunshun Chen, Aaron Lun

See Also

`aveLogCPM`

Examples

```
y <- matrix(rnbinom(20,size=1,mu=10),5,4)
cpm(y)

d <- DGEList(counts=y, lib.size=1001:1004)
cpm(d)
cpm(d,log=TRUE)

d$genes <- data.frame(Length=c(1000,2000,500,1500,3000))
rpkm(d)

cpmByGroup(d, group=c(1,1,2,2))

rpkmByGroup(d, group=c(1,1,2,2))
```

---

cutWithMinN *Cut Numeric Vector into Non-empty Intervals*

Description

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

Usage

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

Arguments

x	numeric vector.
intervals	number of intervals required.
min.n	minimum number of values in any interval. Must be less than or equal to length(x)/intervals.

Details

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

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

Value

A list with components:

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

Author(s)

Gordon Smyth

See Also

cut, quantile.

Examples

```
x <- c(1,2,3,4,5,6,7,100)
cutWithMinN(x,intervals=3,min.n=1)
```

---

decideTests *Multiple Testing Across Genes and Contrasts*

Description

Identify which genes are significantly differentially expressed from an edgeR test object containing p-values and test statistics.

Usage

```
## S3 method for class 'DGELRT'
decideTests(object, adjust.method="BH", p.value=0.05, lfc=0, ...)
```

Arguments

object	an object of class DGEExact, DGELRT or glmQLFTest from which p-values and log-fold-changes can be extracted.
adjust.method	character string specifying p-value adjustment method. Possible values are "none", "BH", "fdr", "BY" and "holm". See p.adjust for details.
p.value	numeric value between 0 and 1 giving the required family-wise error or false discovery rate.
lfc	numeric, minimum absolute log2-fold-change required.
...	other arguments are not used.

Details

This function applies a multiple testing procedure and significance level cutoff to the genewise tests contained in an edgeR test object and collates the results in a data.frame table.

The function can optionally apply a logFC cutoff as well as a p-value or FDR cutoff (although logFC cutoffs are not recommended, see note below). If the statistical tests are on 1 degree of freedom, then the logFC cutoff is applied to the absolute coefficient or contrast. If the statistical tests are on more than 1 degree of freedom, then the logFC cutoff will be satisfied if any of the coefficients or contrasts that define the test are greater than the cutoff.

Value

An object of class TestResults. This is essentially a single-column integer matrix with elements -1, 0 or 1 indicating whether each gene is classified as significantly down-regulated, not significant or significant up-regulated for the comparison contained in object. To be considered significant, genes need to have adjusted p-value below p.value and log2-fold-change greater than lfc.

If object contains F-tests or LRTs for multiple contrasts, then the genes are simply classified as significant (1) or not significant. In this case, the log2-fold-change threshold lfc has to be achieved by at least one of the contrasts for a gene to be significant.

Note

Although this function enables users to set p-value and logFC cutoffs simultaneously, this combination criterion is not recommended. logFC cutoffs tend to favor low expressed genes and thereby reduce rather than increase biological significance. Unless the fold changes and p-values are very highly correlated, the addition of a fold change cutoff can also increase the family-wise error rate or false discovery rate above the nominal level. Users wanting to use fold change thresholding should consider using glmTreat instead and leaving lfc at the default value when using decideTests.

Author(s)

Davis McCarthy, Gordon Smyth and the edgeR team

See Also

decideTests and TestResults-class in the limma package.

Examples

```
ngenes <- 100
x1 <- rnorm(6)
x2 <- rnorm(6)
design <- cbind(Intercept=1,x1,x2)
beta <- matrix(0,ngenes,3)
beta[,1] <- 4
beta[1:20,2] <- rnorm(20)
mu <- 2^(beta %*% t(design))
y <- matrix(rnbinom(ngenes*6,mu=mu,size=10),ngenes,6)
fit <- glmFit(y,design,dispersion=0.1)
lrt <- glmLRT(fit,coef=2:3)
res <- decideTests(lrt,p.value=0.1)
summary(res)
lrt <- glmLRT(fit,coef=2)
res <- decideTests(lrt,p.value=0.1)
summary(res)
```

DGEBIN-class *Digital Gene Expression Binomial Model Results - Class*

Description

A list-based S4 class for storing results of a binomial generalized linear model fit to each gene in a DGE dataset, as produced by binFit or binQLFit. It is a sibling of DGEGLM-class (the negative-binomial GLM fit): a binomial fit is deliberately *not* a DGEGLM, so that binomial-specific tests such as binQLFTest accept only a DGEBIN and the negative-binomial tests (glmLRT, glmQLFTest, glmTreat) do not.

List Components

For objects of this class, rows correspond to genomic features and columns to coefficients in the linear model.

Objects of this class contain the following list components:

coefficients: matrix of estimated coefficients from the binomial GLM fits, on the natural log scale.

unshrunk.coefficients: matrix of coefficients from the unshrunk fits; present only when prior.count>0.

fitted.values: matrix of fitted (expected) proportions from the binomial GLM for each gene.

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

df.residual: vector of residual degrees of freedom for the model fit to each gene.

counts: the first matrix of counts (successes) as input.

counts2: the second matrix of counts (failures) as input.

design: design matrix for the full model.

weights: matrix of prior weights used in the GLM fitting.

nthreads: integer giving the number of OpenMP threads used for the fit.

AveLogCPM: numeric vector giving average log2 counts per million of the total coverage for each gene.

samples, genes: sample and gene annotation data frames, attached when the input is a PCList-class object.

Objects produced by binQLFit additionally carry the leverage-adjusted and empirical Bayes quasi-dispersion components: df.residual.adj, deviance.adj, df.prior, s2.post, s2.prior and average.ql.dispersion.

Methods

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

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

DGEBIN objects can be subsetted by row, see subsetting.

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

Author(s)

Lizhong Chen, Gordon Smyth

See Also

binFit and binQLFit produce DGEBIN objects; binQLFTest tests them.

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

DGEExact-class *Differential Expression of Digital Gene Expression Data - Class*

Description

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

List Components

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

Objects of this class contain the following list components:

table: data frame containing columns for the log2-fold-change, logFC, the average log2-counts-per-million, logCPM, and the two-sided p-value PValue.

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

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

Methods

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

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

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

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

Author(s)

edgeR team. First created by Mark Robinson and Davis McCarthy

See Also

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

`DGEGLM-class` *Digital Gene Expression Generalized Linear Model Results - Class*

Description

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

List Components

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

Objects of this class contain the following list components:

`coefficients`: matrix containing the coefficients computed from fitting the model defined by the design matrix to each gene in the dataset. Coefficients are on the natural log scale.

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

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

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

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

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

`genes`: data frame containing information about the 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 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 gene.

`fitted.values`: the fitted (expected) values from the GLM for each gene.

`AveLogCPM`: numeric vector giving average log2 counts per million for each gene.

Methods

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

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

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

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

Author(s)

edgeR team. First created by Davis McCarthy.

See Also

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

`DGEList` *Create a DGEList Object*

Description

Assembles a `DGEList` object from its components, especially the table of counts as a matrix or data.frame.

Usage

```
## Default S3 method:
DGEList(counts, lib.size = NULL, norm.factors = NULL,
    samples = NULL, group = NULL, genes = NULL,
    remove.zeros = FALSE, ...)
## S3 method for class 'data.frame'
DGEList(counts, lib.size = NULL, norm.factors = NULL,
    samples = NULL, group = NULL, genes = NULL,
    remove.zeros = FALSE, annotation.columns = NULL, ...)
```

Arguments

`counts`	numeric matrix or data.frame containing sequence read counts, with rows corresponding to genes (genomic features) and columns to samples. Negative values or NAs are not allowed.
`lib.size`	numeric vector of library sizes (sequencing depths) for the samples. Defaults to `colSums(counts)`.
`norm.factors`	numeric vector of normalization factors that modify the library sizes. Defaults to a vector of ones.
`samples`	data.frame containing sample information, with a row for each sample. This data.frame will be appended to the `samples` component of the DGEList object.

group	vector or factor giving the experimental group or treatment condition for each sample. Defaults to a single group.
genes	data.frame containing gene annotation.
remove.zeros	logical, whether to remove rows that have 0 total count.
annotation.columns	specify columns of counts that contain gene annotation rather than counts. Can be a vector of column numbers, or a vector of column names, or a logical vector.
...	other arguments are not currently used.

Details

Assembles a DGEList object from its components. The only compulsory argument is the table of counts.

Normally, counts is a numeric matrix of counts but a data.frame is also allowed. If the counts is a data.frame, then the columns of the data.frame containing gene IDs or other gene annotation can be specified by annotation.columns, and the other columns are assumed to contain sequence read counts. If annotation.columns is not specified, then the function will check for non-numeric columns of counts and will attempt to set the leading columns up to the last non-numeric column as annotation.

Value

A DGEList object.

Author(s)

edgeR team. Originally created by Mark Robinson.

See Also

DGEList-class

Examples

```
ngenes <- 100
nsamples <- 4
Counts <- matrix(rnbinom(ngenes*nsamples,mu=5,size=10),ngenes,nsamples)
rownames(Counts) <- 1:ngenes
colnames(Counts) <- paste0("S",1:4)
Group <- gl(2,2)
Genes <- data.frame(Symbol=paste0("Gene",1:ngenes))
y <- DGEList(counts=Counts, group=Group, genes=Genes)
dim(y)
colnames(y)
y$samples
show(y)
```

DGEList-class *Digital Gene Expression Data - Class*

Description

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

List Components

For objects of this class, rows correspond to genomic features and columns to samples. In the interests of brevity, the genomic features are called genes in the documentation below, but in reality the rows can correspond to any genomic feature for which read counts are meaningful such as transcripts, tags, exons, exon-junctions, or genomic intervals. Objects of this class contain the following essential list components:

`counts`: numeric matrix of read counts, one row for each gene and one column for each sample.

`samples`: data.frame with a row for each sample and columns `group`, `lib.size` and `norm.factors` containing the group labels, library sizes and normalization factors. Other columns can be optionally added to give more detailed sample information.

Optional components include:

`genes`: data.frame giving annotation information for each gene. Same number of rows as `counts`.

`AveLogCPM`: numeric vector giving average log2 counts per million for each gene.

`common.dispersion`: numeric scalar giving the overall dispersion estimate.

`trended.dispersion`: numeric vector giving trended dispersion estimates for each gene.

`tagwise.dispersion`: numeric vector giving tagwise dispersion estimates for each gene (note that 'tag' and 'gene' are synonymous here).

`offset`: numeric matrix of same size as `counts` giving offsets for use in log-linear models.

`offset.prior`: numeric matrix of same size as `counts` giving prior offsets arising from observation-specific normalization not related to library size.

Methods

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

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

`DGEList` objects can be subsetted analogously to matrices, see `subsetting`.

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

Author(s)

edgeR team. First created by Mark Robinson.

See Also

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

---

DGEListFromTximport *Create a DGEList Object from tximport or tximeta Output*

Description

Assembles a DGEList object from the list object created by either tximport::tximport or tximeta::tximeta.

Usage

```
DGEListFromTximport(txi, samples = NULL, group = NULL, genes = NULL,
    remove.zeros = FALSE, divide = FALSE)
DGEListFromTximeta(txm, samples = NULL, group = NULL,
    remove.zeros = FALSE, divide = FALSE)
```

Arguments

txi	list output from tximport().
txm	SummarizedExperiment data object from tximeta().
samples	data.frame containing sample information, with a row for each sample. This data.frame will be appended to the samples component of the DGEList object.
group	vector or factor giving the experimental group or treatment condition for each sample. Defaults to a single group.
genes	data.frame containing gene annotation.
remove.zeros	logical, whether to remove rows that have 0 total count.
divide	logical, whether to divide the counts by the RAA overdispersions.

Details

The functions DGEListFromTximport and DGEListFromTximeta import output from the tximport::tximport and tximeta::tximeta functions, respectively, to the edgeR package. In many cases, tximport and tximeta will have been run with txOut=FALSE to aggregate counts at the gene level. In that case, the length matrix (txi$length for tximport and assay(txi,"length") for tximeta) will contain the average transcript length for each gene in each sample. Alternatively, txi or txm will contain results at the original transcript level, if tximport and tximeta were run with txOut=TRUE, in which case the length matrix will contain effective transcript lengths for the individual transcripts in each sample.

The count matrix, txi$counts for tximport and assay(txi,"counts") for tximeta, becomes the count matrix for the DGEList object. If either tximport or tximeta were run with the option countsFromAbundance="no", then a prior offset matrix is formed from the average transcript lengths found in the respective transcript length matrix. This implements the "original counts and offset" method described by the tximport documentation. If countsFromAbundance has been set to any other value, then the counts have already been bias corrected and no offset matrix will be formed. This implements the "bias corrected counts without an offset" method described by the tximport documentation.

If inferential replicates (bootstrap or Gibbs resampling) exist in the txi or txm data objects then a read-assignment-ambiguity (RAA) overdispersion is estimated for each row of data, using the approach described by Baldoni et al (2024a). If divide=TRUE, then divided counts will be formed by dividing the original counts by the RAA overdispersions.

Note that transcript level counts from various quantification tools can also be read directly by the native edgeR functions catchSalmon, catchKallisto, catchRSEM, or catchOarfish, without the need for tximport as an intermediary, and these will generally give the same results as DGEListFromTximport at the transcript level. In the case of RSEM, using catchRSEM is preferable, because RSEM does not output inferential replicates but catchRSEM can nevertheless estimate the assignment over-dispersion from the inferential replicate means and variances. At the gene level, tximport and DGEListFromTximport implement transcript length normalization, which the direct edgeR functions do not.

Value

A DGEList object.

If txi or txm contains raw counts, then a prior offset matrix offset.prior will be included to account for variation in average transcript length between samples.

The genes component of the object will contain the following columns:

AveLength	average transcript or gene length, over all samples and all transcripts.
Max2MinLength	ratio of largest to smallest length for different samples. If counts are at the gene level, then the lengths are averages over transcripts for each sample, as computed by tximport.
Overdispersion	quasi-dispersion arising from read assignment ambiguity. Equal to 1 for rows with no ambiguity. Larger values indicate more ambiguity.

As well as standard DGEList components, the output component divided.counts indicates whether the counts have been divided and the component txi.counts indicates whether the tximported counts were direct from the quantifier or whether they were scaled by TPMs.

References

Baldoni PL, Chen Y, Hediyeh-zadeh S, Liao Y, Dong X, Ritchie ME, Shi W, Smyth GK (2024a). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. *Nucleic Acids Research* 52(3), e13. doi:10.1093/nar/gkad1167.

Baldoni PL, Chen L, Smyth GK (2024b). Faster and more accurate assessment of differential transcript expression with Gibbs sampling and edgeR v4. *NAR Genomics and Bioinformatics* 6(4), lqae151. doi:10.1093/nargab/lqae151.

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

See Also

catchSalmon, catchKallisto, catchRSEM, catchOarfish, DGEList-class.

The input objects txi or txm are created by either tximport or tximeta in the tximport and tximeta packages, respectively.

DGELRT-class *Digital Gene Expression Likelihood Ratio Test Data and Results - Class*

Description

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

List Components

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

Objects of this class contain the following list components:

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

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

`coefficients.null`: matrix containing the coefficients computed from fitting the null model to each gene 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 `DGELRT` objects can be manipulated as if they were ordinary lists. However they can also be treated as if they were matrices for the purposes of subsetting.

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

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

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

Author(s)

edgeR team. First created by Davis McCarthy

See Also

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

`diffSplice.DGEGLM` *Differential Transcript or Exon Usage*

Description

Given an edgeR fit object with multiple features per gene, test for differential usage of those features within genes between experimental conditions. The features may be any set of isoform-identifying features such as transcripts, exons or exon-junctions.

Usage

```
## S3 method for class 'DGEGLM'
diffSplice(fit, coef=ncol(fit$design), contrast=NULL, geneid, exonid = NULL,
     robust = NULL, nexons.approx = 10, verbose = TRUE, ...)
```

Arguments

`fit`	a `DGEGLM` fitted model object produced by `glmQLFit`. Rows should correspond to transcripts, for a DTU analysis, or to exons and exon-exon junctions for a DEU analysis.
`coef`	subscript indicating which coefficient of the generalized linear model is to be tested for differential usage. Defaults to the last coefficient. Can be an integer or a coefficient name.
`contrast`	numeric vector specifying the contrast of the linear model coefficients to be tested for differential usage. Length must equal to the number of columns of `design`. If specified, then takes precedence over `coef`.
`geneid`	gene identifiers. Either a vector of length `nrow(fit)` or the name of the column of `fit$genes` containing the gene identifiers. Rows with the same ID are assumed to belong to the same gene.
`exonid`	feature or exon identifiers. Either a vector of length `nrow(fit)` or the name of the column of `fit$genes` containing the feature identifiers.
`robust`	logical, should the estimation of the empirical Bayes prior parameters be robustified against outlier sample variances? By default, the same setting will be used as for the `glmQLFit` call as used to create `fit`.
`nexons.approx`	exact test statistics are computed for all genes with up to this number of features (or exons). For genes with more features, a more computationally fast approximation is used.
`verbose`	logical, if `TRUE` some diagnostic information about the number of genes and exons is output.
`...`	other arguments are not currently used.

Details

Given an edgeR fit QL object with multiple features per gene, test for differential usage of those features within genes between experimental conditions. The features may be any set of isoform-identifying features such as transcripts, exons or exon-junctions. The features can be transcripts for a differential transcript usage (DTU) analysis (Baldoni et al 2025), or can be a combination of exons and exon-exon junctions for a differential exon usage (DEU) analysis. Another possibility is for the features to be transcript-overlap equivalence classes (Cmero et al 2019).

diffSplice is an S3 generic function defined in the limma package. This help page describes the diffSplice method for DGEGLM fitted model objects as produced by glmQLFit in edgeR. The output object can be further explored using the topSplice and plotSplice functions of the limma package.

Testing for differential usage is equivalent to testing whether the log-fold-changes in the fit differ between features for the same gene. Two different tests are provided, one at the gene level and one at the feature level. In both cases, the tests are conducted using quasi-F test statistics based on deviance differences. The gene-level test is a quasi-F-test for differences between the log-fold-changes, equivalent to testing for interaction between the contrast comparison and the transcripts for each gene. This produces one F-statistic for each gene, where the numerator df is one less than the number of exons and the denominator is the residual df pooled over all exons for that gene. The null hypothesis is that all the exons have the same log-fold-change and the alternative is that the log-fold-changes are not all equal. The feature-level tests compare each feature to the other features for the same gene. Each feature-level test compares the model in which all the features have the same log-fold-change vs the alternative model that just the specified feature has a different log-fold-change. The null hypothesis is that the log-fold-change for the specified feature is equal to the consensus log-fold-change for the other features, and the alternative is that it is not. This produces one quasi F-statistic for each feature, where the numerator df is equal to one and the denominator is the same as for the gene-level test. The feature-level F-statistics are then converted into t-tests by taking square-roots and multiplying by the sign of log-fold-change difference.

For genes with more than nexons.approx features, a fast approximation is used for the feature-level test, assuming that the consensus log-fold-change for the other features should be close to the consensus log-fold-change for all features for that gene. The null hypothesis for these genes is that the log-fold-change for the specified feature is equal to the consensus log-fold-change for all other features, and the alternative is that it is not. The approximation should be very slightly conservative.

The fit object should be created using glmQLFit. diffSplice will automatically detect whether glmQLFit was run with edgeR v4 bias-adjusted deviances (legacy=FALSE, recommended) or with ordinary deviances (legacy=TRUE). The number of OpenMP threads used for the genewise computations is taken from the nthreads value stored in fit by glmQLFit, so diffSplice has no separate nthreads argument.

The feature-level tests are converted into genewise tests by adjusting the p-values for the same gene by Simes method. As an alternative, the feature-level tests are also converted into genewise tests by adjusting the smallest p-value for each gene by Bonferroni's method.

This function can be used on transcript level RNA-seq counts from Salmon or kallisto, after pre-processing by catchSalmon() (or catchKallisto or catchRSEM) and voomLmFit(), as described by Baldoni et al (2025). It can also be used on equivalence-class counts from Salmon or kallisto, after pre-processing by voomLmFit(), as described by Cmero et al (2019). It can also be used on exon-level read counts.

Value

An object of class MArrayLM containing both exon level and gene level tests. Results are sorted by geneid and by exonid within gene.

coefficients	a single-column numeric matrix containing differential usage coefficients for the coefficient or contrast specified by coef or contrast. The coefficients give the difference between the log-fold-change for that genomic feature versus the average log-fold-change for all other features for the same gene.
t	a single-column numeric matrix of quasi t-statistics.
p.value	a single-column numeric vector of p-values corresponding to the t-statistics.
genes	data.frame of annotation for the low-level features.

genecolname — character string giving the name of the column of genes containing gene IDs.

gene.F — single-column numeric matrix of quasi F-statistics, one row for each gene.

gene.F.p.value — single-column numeric matrix of p-values corresponding to gene.F

gene.simes.p.value — single-column numeric matrix of Simes adjusted p-values, one row for each gene.

gene.bonferroni.p.value — single-column numeric matrix of Bonferroni adjusted p-values, one row for each gene.

gene.genes — data.frame of gene annotation.

Note

diffSpliceDGE is a legacy function for the same purpose as diffSplice.DGEGLM, and diffSplice.DGEGLM is intended to replace it. diffSplice.DGEGLM has better statistical power and receiver operating curve (ROC), while still controlling the FDR. In terms of speed, the new function is slightly slower but the difference should not be enough to have a material effect on analyses.

Author(s)

Lizhong Chen, Yunshun Chen and Gordon Smyth

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *Nucleic Acids Research* 53(22), gkaf1305. doi:10.1093/nar/gkaf1305.

Cmero M, Davidson NM, Oshlack A (2019). Using equivalence class counts for fast and accurate testing of differential transcript usage. *F1000Research* 8, 265. doi:10.12688/f1000research.18276.2.

See Also

topSplice and plotSplice are downstream functions that operate on the output from diffSplice.

diffSplice is the generic function in the limma package. spliceVariants and diffSpliceDGE are older edgeR functions with a similar purpose, both replaced by this function.

Examples

```
# Gene and exon annotation
Gene <- paste("Gene", 1:100, sep="")
Gene <- rep(Gene, each=10)
Exon <- paste("Ex", 1:10, sep="")
Gene.Exon <- paste(Gene, Exon, sep=".")
genes <- data.frame(GeneID=Gene, Gene.Exon=Gene.Exon)

# Two groups with n=3 replicates in each group
group <- factor(rep(1:2, each=3))
design <- model.matrix(~group)

# Generate exon counts.
# Knock-out the first exon of Gene1 by 90%
mu <- matrix(100, nrow=1000, ncol=6)
mu[1,4:6] <- 10
```

```
counts <- matrix(rnbinom(6000,mu=mu,size=20),1000,6)
y <- DGEList(counts=counts, lib.size=rep(1e6,6), genes=genes)

# Exon level fit
fit <- glmQLFit(y, design)

# Differential usage analysis
ds <- diffSplice(fit, geneid="GeneID")
topSplice(ds)
plotSplice(ds)
```

`diffSpliceDGE` *Test for Differential Exon Usage*

Description

Given a negative binomial generalized log-linear model fit at the exon (or transcript) level, test for differential exon (or transcript) usage between experimental conditions.

Usage

```
diffSpliceDGE(glmfit, coef=ncol(glmfit$design), contrast=NULL, geneid, exonid=NULL,
              prior.count=0.125, robust=NULL, verbose=TRUE)
```

Arguments

`glmfit`	a `DGEGLM` fitted model object produced by `glmFit` or `glmQLFit`. Rows should correspond to exons.
`coef`	integer indicating which coefficient of the generalized linear model is to be tested for differential exon usage. Defaults to the last coefficient.
`contrast`	numeric vector specifying the contrast of the linear model coefficients to be tested for differential exon usage. Length must equal to the number of columns of `design`. If specified, then takes precedence over `coef`.
`geneid`	gene identifiers. Either a vector of length `nrow(glmfit)` or the name of the column of `glmfit$genes` containing the gene identifiers. Rows with the same ID are assumed to belong to the same gene.
`exonid`	exon identifiers. Either a vector of length `nrow(glmfit)` or the name of the column of `glmfit$genes` containing the exon identifiers.
`robust`	logical, should the estimation of the empirical Bayes prior parameters be robustified against outlier sample? If `NULL` will be extracted from `glmfit`.
`prior.count`	average prior count to be added to observation to shrink the estimated log-fold-changes towards zero.
`verbose`	logical, if `TRUE` some diagnostic information about the number of genes and exons is output.

Details

This function tests for differential exon usage for each gene for a given coefficient of the generalized linear model.

Testing for differential exon usage is equivalent to testing whether the exons in each gene have the same log-fold-changes as the other exons in the same gene. At exon-level, the log-fold-change of each exon is compared to the log-fold-change of the entire gene which contains that exon. At gene-level, two different tests are provided. One is converting exon-level p-values to gene-level p-values by the Simes method. The other is using exon-level test statistics to conduct gene-level tests.

Value

`diffSpliceDGE` produces an object of class `DGELRT` containing the component `design` from `glmfit` plus the following new components:

`comparison`	character string describing the coefficient being tested.
`coefficients`	numeric vector of coefficients on the natural log scale. Each coefficient is the difference between the log-fold-change for that exon versus the log-fold-change for the entire gene which contains that exon.
`genes`	data.frame of exon annotation.
`genecolname`	character string giving the name of the column of `genes` containing gene IDs.
`exoncolname`	character string giving the name of the column of `genes` containing exon IDs.
`exon.df.test`	numeric vector of testing degrees of freedom for exons.
`exon.p.value`	numeric vector of p-values for exons.
`gene.df.test`	numeric vector of testing degrees of freedom for genes.
`gene.p.value`	numeric vector of gene-level testing p-values.
`gene.Simes.p.value`	numeric vector of Simes' p-values for genes.
`gene.genes`	data.frame of gene annotation.

Some components of the output depend on whether `glmfit` is produced by `glmFit` or `glmQLFit`. If `glmfit` is produced by `glmFit`, then the following components are returned in the output object:

`exon.LR`	numeric vector of LR-statistics for exons.
`gene.LR`	numeric vector of LR-statistics for gene-level test.

If `glmfit` is produced by `glmQLFit`, then the following components are returned in the output object:

`exon.F`	numeric vector of F-statistics for exons.
`gene.df.prior`	numeric vector of prior degrees of freedom for genes.
`gene.df.residual`	numeric vector of residual degrees of freedom for genes.
`gene.F`	numeric vector of F-statistics for gene-level test.

The information and testing results for both exons and genes are sorted by geneid and by exonid within gene.

Note

This function was updated in edgeR 4.6.0 to provide more rigorous control of the FDR. The output `gene.df.test` is increased compared to the previous version, and full Simes adjustment is used instead a modified version.

Despite the improvement to this function, the new function `diffSplice.DGEGLM` is now recommended over `diffSpliceDGE`, and `diffSpliceDGE` is scheduled for removal in a future version of edgeR. The newer function has better statistical power and receiver operating curve (ROC) while still controlling the FDR. In terms of speed, the new function is slightly slower but the difference should not be enough to have a material effect on analyses.

Author(s)

Lizhong Chen, Yunshun Chen, Gordon Smyth

See Also

`diffSplice.DGEGLM` is a newer function with better performance for the same purpose.

Examples

```
# Gene exon annotation
Gene <- paste("Gene", 1:100, sep="")
Gene <- rep(Gene, each=10)
Exon <- paste("Ex", 1:10, sep="")
Gene.Exon <- paste(Gene, Exon, sep=".")
genes <- data.frame(GeneID=Gene, Gene.Exon=Gene.Exon)

group <- factor(rep(1:2, each=3))
design <- model.matrix(~group)
mu <- matrix(100, nrow=1000, ncol=6)
# knock-out the first exon of Gene1 by 90%
mu[1,4:6] <- 10
# generate exon counts
counts <- matrix(rnbinom(6000,mu=mu,size=20),1000,6)

y <- DGEList(counts=counts, lib.size=rep(1e6,6), genes=genes)
gfit <- glmFit(y, design, dispersion=0.05)

ds <- diffSpliceDGE(gfit, geneid="GeneID")
topSpliceDGE(ds)
plotSpliceDGE(ds)
```

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

Description

Retrieve the number of rows (genes) and columns (libraries) for a DGEList, PCList, DGEExact or TopTags Object.

Usage

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

Arguments

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

Details

Digital gene expression data objects share many analogies with ordinary matrices in which the rows correspond to 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 (libraries).

Author(s)

Gordon Smyth, Davis McCarthy

See Also

dim in the base package.

DGEList-class and PCList-class describe the data classes used in edgeR.

Examples

```
Counts <- matrix(rpois(40,lambda=10),20,2)
y <- DGEList(Counts)
dim(y)
nrow(y)
ncol(y)
```

---

dimnames *Retrieve the Dimension Names of a DGE Object*

Description

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

Usage

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

Arguments

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

Details

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

Setting dimension names is supported for DGEList, PCList, DGEExact, DGEGLM and DGELRT objects.

A consequence of these methods is that rownames, colnames, rownames<- and colnames<- will also work as expected on any of the above object classes.

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.

Examples

```
Counts1 <- matrix(rpois(20,lambda=10),10,2)
Counts2 <- matrix(rpois(20,lambda=10),10,2)
y <- PCList(Counts1, Counts2)
rownames(y) <- paste0("Locus",1:10)
colnames(y) <- c("A","B")
dimnames(y)
```

---

dispBinTrend *Estimate Dispersion Trend by Binning for NB GLMs*

Description

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

Usage

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

Arguments

y	numeric matrix of counts.
design	numeric matrix giving the design matrix for the GLM that is to be fit.
offset	numeric scalar, vector or matrix giving the offset (in addition to the log of the effective library size) that is to be included in the NB GLM for the genes. If a scalar, then this value will be used as an offset for all genes and libraries. If a vector, it should have length equal to the number of libraries, and the same vector of offsets will be used for each gene. If a matrix, then each library for each gene can have a unique offset, if desired. In adjustedProfileLik the offset must be a matrix with the same dimension as the table of counts.
df	degrees of freedom for spline curve.
span	span used for loess curve.
min.n	minimum number of genes in a bin.
method.bin	method used to estimate the dispersion in each bin. Possible values are "CoxReid", "Pearson" or "deviance".
method.trend	type of curve to smooth the bins. Possible values are "spline" for a natural cubic regression spline or "loess" for a linear lowess curve.
AveLogCPM	numeric vector giving average log2 counts per million for each gene.
weights	optional numeric matrix giving observation weights.
...	other arguments are passed to estimateGLMCommonDisp.

Details

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

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

Value

list with the following components:

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

Author(s)

Davis McCarthy and Gordon Smyth

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

estimateGLMTrendedDisp

Examples

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

---

dispCoxReid *Estimate Common Dispersion for Negative Binomial GLMs*

Description

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

Usage

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

Arguments

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

`initial.dispersion`
: starting value for the dispersion.

`nthreads`
: integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).

Details

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

`dispCoxReid` maximizes the Cox-Reid adjusted profile likelihood (Cox and Reid, 1987). `dispPearson` sets the average Pearson goodness of fit statistic 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 value. This is also known as the *quasi-likelihood* estimator.

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

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

Value

Numeric vector of length one giving the estimated common dispersion.

Author(s)

Gordon Smyth

References

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

Robinson MD, 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*. doi:10.1093/nar/gks042. (Published online 28 January 2012)

See Also

`estimateGLMCommonDisp`, `optimize`, `uniroot`

Examples

```
ngenes <- 100
nlibs <- 4
y <- matrix(rnbinom(ngenes*nlibs,mu=10,size=10),nrow=ngenes,ncol=nlibs)
group <- factor(c(1,1,2,2))
```

```
lib.size <- rowSums(y)
design <- model.matrix(~group)
disp <- dispCoxReid(y, design, offset=log(lib.size), subset=100)
```

`dispCoxReidInterpolateTagwise` *Estimate Genewise Dispersion for Negative Binomial GLMs by Cox-Reid Adjusted Profile Likelihood*

Description

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

Usage

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

Arguments

`y`	numeric matrix of counts.
`design`	numeric matrix giving the design matrix for the GLM that is to be fit.
`offset`	numeric scalar, vector or matrix giving the offset (in addition to the log of the effective library size) that is to be included in the NB GLM for the genes. If a scalar, then this value will be used as an offset for all genes and libraries. If a vector, it should have length equal to the number of libraries, and the same vector of offsets will be used for each gene. If a matrix, then each library for each gene 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 genewise dispersion parameters are shrunk.
`trend`	logical, whether abundance-dispersion trend is used for smoothing.
`AveLogCPM`	numeric vector giving average log2 counts per million for each gene.
`min.row.sum`	numeric scalar giving a value for the filtering out of low abundance genes. Only genes with total sum of counts above this value are used. Low abundance genes can adversely affect the estimation of the common dispersion, so this argument allows the user to select an appropriate filter threshold for the gene abundance.
`prior.df`	numeric scalar, prior degrees-of-freedom smoothing parameter that indicates the weight to give to the common likelihood compared to the individual gene'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 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 genewise dispersion estimates.
grid.range	numeric vector of length 2, giving relative range, in terms of log2(dispersion), on either side of trendline for each gene for spline grid points.
weights	optional numeric matrix giving observation weights.

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 genewise grid.

Note that the terms 'tag' and 'gene' are synonymous here. The function is only named 'Tagwise' for historical reasons.

Value

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

Author(s)

Yunshun Chen, Gordon Smyth

References

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

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

estimateGLMTagwiseDisp, maximizeInterpolant

Examples

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

dispCoxReidSplineTrend *Estimate Dispersion Trend for Negative Binomial GLMs*

Description

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

Usage

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

Arguments

y	numeric matrix of counts.
design	numeric matrix giving the design matrix for the GLM that is to be fit.
offset	numeric scalar, vector or matrix giving the offset (in addition to the log of the effective library size) that is to be included in the NB GLM for the genes. If a scalar, then this value will be used as an offset for all genes and libraries. If a vector, it should have length equal to the number of libraries, and the same vector of offsets will be used for each gene. If a matrix, then each library for each gene can have a unique offset, if desired. In adjustedProfileLik the offset must be a matrix with the same dimension as the table of counts.
df	integer giving the degrees of freedom of the spline function, see ns in the splines package.
subset	integer, number of rows to use in the calculation. Rows used are chosen evenly spaced by AveLogCPM using cutWithMinN.
AveLogCPM	numeric vector giving average log2 counts per million for each gene.
method.optim	the method to be used in optim. See optim for more detail.
trace	logical, should iteration information be output?

Details

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

dispCoxReidSplineTrend and dispCoxReidPowerTrend fit abundance trends to the genewise 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 gene. The vectors are of the same length as nrow(y).

Author(s)

Yunshun Chen, Davis McCarthy, Gordon Smyth

References

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

See Also

estimateGLMTrendedDisp

Examples

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

---

dropEmptyLevels *Drop Levels of a Factor that Never Occur*

Description

Reform a factor so that only necessary levels are kept.

Usage

```
dropEmptyLevels(x)
```

Arguments

x	a factor or a vector to be converted to a factor.

Details

In general, the levels of a factor, `levels(x)`, may include values that never actually occur. This function drops any levels that do not occur.

If `x` is not a factor, then the function returns `factor(x)`. If `x` is a factor, then the function returns the same value as `factor(x)` or `x[,drop=TRUE]` but somewhat more efficiently.

Value

A factor with the same values as `x` but with a possibly reduced set of levels.

Author(s)

Gordon Smyth

See Also

factor.

Examples

```
x <- factor(c("a","b"), levels=c("c","b","a"))
x
dropEmptyLevels(x)
```

---

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 edgeR User's Guide. The User's Guide is not itself a true vignette because it is not automatically generated using Sweave during the package build process. This means that it cannot be found using `vignette`, hence the need for this special function.

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

Value

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

Author(s)

Gordon Smyth

See Also

system

Examples

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

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

Description

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

Usage

```
## S3 method for class 'DGEList'
equalizeLibSizes(y, dispersion=NULL, ...)
## Default S3 method:
equalizeLibSizes(y, group=NULL, dispersion=NULL,
            lib.size=NULL, ...)
```

Arguments

`y`	matrix of counts or a `DGEList` object.
`dispersion`	numeric scalar or vector of dispersion parameters. By default, is extracted from `y` or, if `y` contains no dispersion information, is set to `0.05`.
`group`	vector or factor giving the experimental group/condition for each library.
`lib.size`	numeric vector giving the total count (sequence depth) for each library.
`...`	other arguments that are not currently used.

Details

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

The output pseudo-counts are the counts that would have theoretically arisen had the effective library sizes been equal for all samples. The pseudo-counts are computed in such a way as to preserve fold-change differences between the groups defined by `y$samples$group` as well as biological variability within each group. Consequently, the results will depend on how the groups are defined.

Note that the column sums of the `pseudo.counts` matrix will not generally be equal, because the effective library sizes are not necessarily the same as actual library sizes and because the normalized pseudo counts are not equal to expected counts.

Value

`equalizeLibSizes.default` returns a list with components:

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

`equalizeLibSizes.DGEList` returns a `DGEList` object with the above two components added.

Note

This function is intended mainly for internal edgeR use. It is not normally called directly by users.

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. doi:10.1093/biostatistics/kxm030.

See Also

q2qnbinom

Examples

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

---

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

Description

Maximizes the negative binomial conditional common likelihood to estimate a common dispersion value across all genes.

Usage

```
## S3 method for class 'DGEList'
estimateCommonDisp(y, tol=1e-06, rowsum.filter=5, verbose=FALSE, ...)
## Default S3 method:
estimateCommonDisp(y, group=NULL, lib.size=NULL, tol=1e-06,
          rowsum.filter=5, verbose=FALSE, ...)
```

Arguments

y	matrix of counts or a DGEList object.
tol	the desired accuracy, passed to optimize.
rowsum.filter	genes with total count (across all samples) below this value will be filtered out before estimating the dispersion.
verbose	logical, if TRUE then the estimated dispersion and BCV will be printed to standard output.
group	vector or factor giving the experimental group/condition for each library.
lib.size	numeric vector giving the total count (sequence depth) for each library.
...	other arguments that are not currently used.

Details

Implements the conditional maximum likelihood (CML) method proposed by Robinson and Smyth (2008) for estimating a common dispersion parameter. This method proves to be accurate and nearly unbiased even for small counts and small numbers of replicates.

The CML method involves computing a matrix of quantile-quantile normalized counts, called pseudo-counts. The pseudo-counts are adjusted in such a way that the library sizes are equal for all samples, while preserving differences between groups and variability within each group. The pseudo-counts are included in the output of the function, but are intended mainly for internal edgeR use.

Value

`estimateCommonDisp.DGEList` adds the following components to the input `DGEList` object:

`common.dispersion`
: estimate of the common dispersion.

`pseudo.counts`
: numeric matrix of pseudo-counts.

`pseudo.lib.size`
: the common library size to which the pseudo-counts have been adjusted.

`AveLogCPM`
: numeric vector giving log2(AveCPM) for each row of `y`.

`estimateCommonDisp.default` returns a numeric scalar of the common dispersion estimate.

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. doi:10.1093/biostatistics/kxm030.

See Also

`equalizeLibSizes`, `estimateTrendedDisp`, `estimateTagwiseDisp`

Examples

```
# True dispersion is 1/5=0.2
y <- matrix(rnbinom(250*4,mu=20,size=5),nrow=250,ncol=4)
dge <- DGEList(counts=y,group=c(1,1,2,2))
dge <- estimateCommonDisp(dge, verbose=TRUE)
```

`estimateDisp` *Estimate Common, Trended and Tagwise Negative Binomial Dispersions by Weighted Likelihood Empirical Bayes*

Description

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

Usage

```
## S3 method for class 'DGEList'
estimateDisp(y, design=NULL, prior.df=NULL, trend.method="locfit", tagwise=TRUE,
   span=NULL, legacy.span=FALSE, min.row.sum=5, grid.length=21, grid.range=c(-10,10),
     robust=FALSE, winsor.tail.p=c(0.05,0.1), tol=1e-06, ...)
## S3 method for class 'SummarizedExperiment'
estimateDisp(y, design=NULL, prior.df=NULL, trend.method="locfit", tagwise=TRUE,
   span=NULL, legacy.span=FALSE, min.row.sum=5, grid.length=21, grid.range=c(-10,10),
     robust=FALSE, winsor.tail.p=c(0.05,0.1), tol=1e-06, ...)
## Default S3 method:
estimateDisp(y, design=NULL, group=NULL, lib.size=NULL, offset=NULL,
   prior.df=NULL, trend.method="locfit", tagwise=TRUE, span=NULL, legacy.span=FALSE,
     min.row.sum=5, grid.length=21, grid.range=c(-10,10),
     robust=FALSE, winsor.tail.p=c(0.05,0.1), tol=1e-06, weights=NULL, ...)
```

Arguments

`y`	matrix of counts, or a `DGEList` object, or a `SummarizedExperiment` object.
`design`	numeric design matrix. Defaults to `model.matrix(~group)` if `group` is specified and otherwise to a single column of ones.
`prior.df`	prior degrees of freedom. It is used in calculating `prior.n`.
`trend.method`	method for estimating dispersion trend. Possible values are `"locfit"` (default), `"none"`, `"movingave"`, `"loess"` and `"locfit.mixed"`, which uses a polynomial of degree 1 for lowly expressed genes.
`tagwise`	logical, should the tagwise dispersions be estimated?
`span`	width of the smoothing window, as a proportion of the data set. Defaults to value chosen by `chooseLowessSpan()`.
`legacy.span`	logical, if `TRUE` the old default for `span` will be used.
`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 dispersion estimation, so this argument allows the user to select an appropriate filter threshold for the tag abundance.
`grid.length`	the number of points on which the interpolation is applied for each tag.
`grid.range`	the range of the grid points around the trend on a log2 scale.
`robust`	logical, should the estimation of `prior.df` be robustified against hypervariable genes?
`winsor.tail.p`	numeric vector of length 1 or 2, giving left and right tail proportions of the deviances to Winsorize when estimating `prior.df`.

tol	the desired accuracy, passed to optimize.
group	vector or factor giving the experimental group/condition for each library. Defaults to a vector of ones with length equal to the number of libraries.
lib.size	numeric vector giving the total count (sequence depth) for each library.
offset	offset matrix for the log-linear model, as for glmFit. Defaults to the log-effective library sizes.
weights	optional numeric matrix giving observation weights.
...	other arguments that are not currently used.

Details

This function calculates a matrix of likelihoods for each tag at a set of dispersion grid points, and then applies weighted likelihood empirical Bayes method to obtain posterior dispersion estimates. If there is no design matrix, it calculates the quantile conditional likelihood for each tag and then maximizes it. In this case, it is similar to the function estimateCommonDisp and estimateTagwiseDisp. If a design matrix is given, it calculates the adjusted profile log-likelihood for each tag and then maximizes it. In this case, it is similar to the functions estimateGLMCommonDisp, estimateGLMTrendedDisp and estimateGLMTagwiseDisp.

Note that the terms 'tag' and 'gene' are synonymous here.

Value

estimateDisp.DGEList adds the following components to the input DGEList object:

design	the design matrix.
common.dispersion	estimate of the common dispersion.
trended.dispersion	estimates of the trended dispersions.
tagwise.dispersion	tagwise estimates of the dispersion parameter if tagwise=TRUE.
AveLogCPM	numeric vector giving log2(AveCPM) for each row of y.
trend.method	method for estimating dispersion trend as given in the input.
prior.df	prior degrees of freedom. If robust=TRUE then prior.df is a vector with smaller values assigned to hypervariable outlier genes.
prior.n	estimate of the prior weight, i.e. the smoothing parameter that indicates the weight to put on the common likelihood compared to the individual tag's likelihood.
span	width of the smoothing window used in estimating dispersions.

estimateDisp.SummarizedExperiment converts the input SummarizedExperiment object into a DGEList object, and then calls estimateDisp.DGEList. The output is a DGEList object.

estimateDisp.default returns a list containing common.dispersion, trended.dispersion, tagwise.dispersion (if tagwise=TRUE), span, prior.df and prior.n.

Note

The default value of span was changed in August 2025 from chooseLowessSpan(length(theta), small.n=50, min.span=0.25, power=0.5) to chooseLowessSpan(length(theta), small.n=50, min.span=0.3, power=1/3). Setting legacy.span=TRUE will cause the old default to be used instead of the new, for backward compatibility.

Author(s)

Yunshun Chen, Gordon Smyth

References

Chen Y, Lun ATL, Smyth GK (2014). Differential expression analysis of complex RNA-seq experiments using edgeR. In: *Statistical Analysis of Next Generation Sequence Data*, Somnath Datta and Daniel S. Nettleton (eds), Springer, New York, pages 51-74. https://gksmyth.github.io/pubs/edgeRChapterPreprint.pdf.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10, 946-963. doi:10.1214/16AOAS920.

See Also

estimateCommonDisp, estimateTagwiseDisp, estimateGLMCommonDisp, estimateGLMTrendedDisp, estimateGLMTagwiseDisp

Examples

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

---

estimateExonGenewiseDisp

Estimate Genewise Dispersions from Exon-Level Count Data

Description

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

Usage

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

Arguments

y	either a matrix of exon-level counts or a DGEList object with (at least) elements counts (table of counts summarized at the exon level) and samples (data frame containing information about experimental group, library size and normalization factor for the library size). Each row of y should represent one exon.
geneID	vector of length equal to the number of rows of y, which provides the gene identifier for each exon in y. These identifiers are used to group the relevant exons into genes for the gene-level analysis of splice variation.
group	factor supplying the experimental group/condition to which each sample (column of y) belongs. If NULL (default) the function will try to extract it 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, method="CoxReid",
                      subset=10000, verbose=FALSE, ...)
## Default S3 method:
estimateGLMCommonDisp(y, design=NULL, offset=NULL,
                      method="CoxReid", subset=10000, AveLogCPM=NULL,
                      verbose=FALSE, weights=NULL,...)
```

Arguments

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

The DGEList method returns the same DGEList y as input but with common.dispersion as an added component. The output object will also contain a component AveLogCPM if it was not already present in y.

Author(s)

Gordon Smyth, Davis McCarthy, Yunshun Chen

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

dispCoxReid, dispPearson, dispDeviance

estimateGLMTrendedDisp for trended dispersions or estimateGLMTagwiseDisp for genewise dispersions in the context of a generalized linear model.

estimateCommonDisp for the common dispersion or estimateTagwiseDisp for genewise dispersions 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
```

`estimateGLMRobustDisp` *Empirical Robust Bayes Tagwise Dispersions for Negative Binomial GLMs Using Observation Weights*

Description

Compute a robust estimate of the negative binomial dispersion parameter for each gene, with expression levels specified by a log-linear model, using observation weights. These observation weights will be stored and used later for estimating regression parameters.

Usage

```
estimateGLMRobustDisp(y, design = NULL, prior.df = 10, update.trend = TRUE,
                      trend.method = "bin.loess", maxit = 6, k = 1.345,
                      residual.type = "pearson", verbose = FALSE,
                      record = FALSE)
```

Arguments

`y`	a `DGEList` object.
`design`	numeric design matrix, as for `glmFit`.
`prior.df`	prior degrees of freedom.
`update.trend`	logical. Should the trended dispersion be re-estimated at each iteration?
`trend.method`	method (low-level function) used to estimate the trended dispersions. `estimateGLMTrendedDisp`.
`maxit`	maximum number of iterations for weighted `estimateGLMTagwiseDisp`.
`k`	the tuning constant for Huber estimator. If the absolute value of residual (r) is less than k, its observation weight is 1, otherwise `k/abs(r)`.
`residual.type`	type of residual (r) used for estimating the observation weight.
`verbose`	logical. Should verbose comments be printed?
`record`	logical. Should information for each iteration be recorded (and returned as a list)?

Details

Moderation of dispersion estimates towards a trend can be sensitive to outliers, resulting in an increase in false positives. That is, since the dispersion estimates are moderated downwards toward the trend and because the regression parameter estimates may be affected by the outliers, some genes are incorrectly deemed to be significantly differentially expressed. This function uses an iterative procedure where weights are calculated from residuals and estimates are made after re-weighting.

The robustly computed genewise estimates are reported in the `tagwise.dispersion` vector of the returned `DGEList`. The terms 'tag' and 'gene' are synonymous in this context.

Note: it is not necessary to first calculate the common, trended and genewise dispersion estimates. If these are not available, the function will first calculate these in an unweighted fashion.

Value

`estimateGLMRobustDisp` produces a `DGEList` object, which contains the (robust) genewise dispersion parameter estimate for each gene for the negative binomial model that maximizes the weighted Cox-Reid adjusted profile likelihood, as well as the observation weights. The observation weights are calculated using residuals and the Huber function.

Note that when `record=TRUE`, a simple list of `DGEList` objects is returned, one for each iteration (this is for debugging or tracking purposes).

Author(s)

Xiaobei Zhou, Mark D. Robinson

References

Zhou X, Lindsay H, Robinson MD (2014). Robustly detecting differential expression in RNA sequencing data using observation weights. Nucleic Acids Research, 42(11), e91.

See Also

This function calls `estimateGLMTrendedDisp` and `estimateGLMTagwiseDisp`.

Examples

```
y <- matrix(rnbinom(100*6,mu=10,size=1/0.1),ncol=6)
d <- DGEList(counts=y,group=c(1,1,1,2,2,2),lib.size=c(1000:1005))
d <- normLibSizes(d)
design <- model.matrix(~group, data=d$samples) # Define the design matrix for the full model
d <- estimateGLMRobustDisp(d, design)
summary(d$tagwise.dispersion)
```

---

estimateGLMTagwiseDisp

Empirical Bayes Tagwise Dispersions for Negative Binomial GLMs

---

Description

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

Usage

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

Arguments

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

Details

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

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

Note that the terms 'tag' and 'gene' are synonymous here. The function is only named 'Tagwise' for historical reasons.

This function calls the lower-level function dispCoxReidInterpolateTagwise.

Value

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

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

Author(s)

Gordon Smyth, Davis McCarthy

References

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

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

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

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

Examples

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

---

estimateGLMTrendedDisp

Estimate Trended Dispersion for Negative Binomial GLMs

Description

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

Usage

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

Arguments

y	a matrix of counts or a DGEList object.
design	numeric design matrix, as for `glmFit`.
method	method (low-level function) used to estimate the trended dispersions. Possible values are "auto" (default, switch to "bin.spline" method if the number of genes is greater than 200 and "power" method otherwise),"bin.spline", "bin.loess" (which both result in a call to dispBinTrend), "power" (call to dispCoxReidPowerTrend), or "spline" (call to dispCoxReidSplineTrend).

offset	numeric scalar, vector or matrix giving the linear model offsets, as for glmFit.
AveLogCPM	numeric vector giving average log2 counts per million for each gene.
weights	optional numeric matrix giving observation weights.
...	other arguments are passed to lower-level functions dispBinTrend, dispCoxReidPowerTrend or dispCoxReidSplineTrend.

Details

Estimates the dispersion parameter for each gene with a trend that depends on the overall level of expression for that gene. This is done for a DGE dataset for general experimental designs by using Cox-Reid approximate conditional inference for a negative binomial generalized linear model for each gene with the unadjusted counts and design matrix provided.

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

Value

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

When the input object is a numeric matrix, it returns a vector of trended dispersion estimates calculated by one of the lower-level functions dispBinTrend, dispCoxReidPowerTrend and dispCoxReidSplineTrend.

Author(s)

Gordon Smyth, Davis McCarthy, Yunshun Chen

References

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

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

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

Examples

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

estimateTagwiseDisp *Estimate Empirical Bayes Tagwise Dispersion Values*

Description

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

Usage

```
## S3 method for class 'DGEList'
estimateTagwiseDisp(y, prior.df=10, trend="movingave", span=NULL, method="grid",
           grid.length=11, grid.range=c(-6,6), tol=1e-06, verbose=FALSE, ...)
## Default S3 method:
estimateTagwiseDisp(y, group=NULL, lib.size=NULL, dispersion, AveLogCPM=NULL,
       prior.df=10, trend="movingave", span=NULL, method="grid", grid.length=11,
           grid.range=c(-6,6), tol=1e-06, verbose=FALSE, ...)
```

Arguments

y	matrix of counts or a DGEList object.
prior.df	prior degrees of freedom.
trend	method for estimating dispersion trend. Possible values are "movingave" (default), "loess" and "none".
span	width of the smoothing window, as a proportion of the data set.
method	method for maximizing the posterior likelihood. Possible values are "grid" (default) 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-Raphson iterations.
verbose	logical, if TRUE then diagnostic output is produced during the estimation process.
group	vector or factor giving the experimental group/condition for each library.
lib.size	numeric vector giving the total count (sequence depth) for each library.
dispersion	common dispersion estimate, used as an initial estimate for the tagwise estimates.
AveLogCPM	numeric vector giving average log2 counts per million for each tag.
...	other arguments that are not currently used.

Details

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

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

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

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

Note that the terms 'tag' and 'gene' are synonymous here. The function is only named 'Tagwise' for historical reasons.

Value

`estimateTagwiseDisp.DGEList` adds the following components to the input `DGEList` object:

`prior.df`	prior degrees of freedom.
`prior.n`	estimate of the prior weight.
`tagwise.dispersion`	numeric vector of the tagwise dispersion estimates.
`span`	width of the smoothing window, in terms of proportion of the data set.

`estimateTagwiseDisp.default` returns a numeric vector of the tagwise dispersion estimates.

Author(s)

Mark Robinson, Davis McCarthy, Yunshun Chen and Gordon Smyth

References

Robinson MD, Smyth GK (2007). Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics* 23, 2881-2887. doi:10.1093/bioinformatics/btm453.

See Also

`estimateCommonDisp` is usually run before `estimateTagwiseDisp`.

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

Examples

```
# True dispersion is 1/5=0.2
y <- matrix(rnbinom(250*4,mu=20,size=5),nrow=250,ncol=4)
dge <- DGEList(counts=y,group=c(1,1,2,2))
dge <- estimateCommonDisp(dge)
dge <- estimateTagwiseDisp(dge)
```

`estimateTrendedDisp` *Estimate Empirical Bayes Trended Dispersion Values*

Description

Estimates trended dispersion values by an empirical Bayes method.

Usage

```
## S3 method for class 'DGEList'
estimateTrendedDisp(y, method="bin.spline", df=5, span=2/3, ...)
## Default S3 method:
estimateTrendedDisp(y, group=NULL, lib.size=NULL, AveLogCPM=NULL,
            method="bin.spline", df=5, span=2/3, ...)
```

Arguments

`y`	matrix of counts or a `DGEList` object.
`method`	method used to estimate the trended dispersions. Possible values are `"bin.spline"` and `"bin.loess"`.
`df`	integer giving the degrees of freedom of the spline function if `"bin.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`.
`group`	vector or factor giving the experimental group/condition for each library.
`lib.size`	numeric vector giving the total count (sequence depth) for each library.
`AveLogCPM`	numeric vector giving average log2 counts per million for each tag.
`...`	other arguments that are not currently used.

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.

Author(s)

Yunshun Chen and Gordon Smyth

See Also

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

Examples

```
ngenes <- 1000
nlib <- 4
log2cpm <- seq(from=0,to=16,length=ngenes)
lib.size <- 1e7
mu <- 2^log2cpm * lib.size * 1e-6
dispersion <- 1/sqrt(mu) + 0.1
counts <- rnbinom(ngenes*nlib, mu=mu, size=1/dispersion)
counts <- matrix(counts,ngenes,nlib)
y <- DGEList(counts,lib.size=rep(lib.size,nlib))
y <- estimateCommonDisp(y)
y <- estimateTrendedDisp(y)
```

---

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

Description

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

Usage

```
exactTest(object, pair=1:2, dispersion="auto", rejection.region="doubletail",
          big.count=900, prior.count=0.125)
exactTestDoubleTail(y1, y2, dispersion=0, big.count=900)
exactTestBySmallP(y1, y2, dispersion=0)
exactTestByDeviance(y1, y2, dispersion=0)
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 genes. Allowable character values are `"common"`, `"trended"`, `"tagwise"` or `"auto"`. Default behavior (`"auto"`) is to use the most complex dispersions found in the data object.
`rejection.region`	type of rejection region for two-sided exact test. Possible values are `"doubletail"`, `"smallp"` or `"deviance"`.

big.count	count size above which asymptotic beta approximation will be used.
prior.count	average prior count used to shrink log-fold-changes. Larger values produce more shrinkage.
y1	numeric matrix of counts for the first of the two experimental groups to be tested for differences. Rows correspond to genes 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 and columns to libraries. Libraries are assumed to be equal in size - e.g. adjusted pseudocounts from the output of equalizeLibSizes. Must have the same number of rows as y1.

Details

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

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

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

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

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

Value

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

table	data frame containing columns for the log2-fold-change (logFC), the average log2-counts-per-million (logCPM), and the two-sided p-value (PValue).
comparison	character vector giving the names of the two groups being compared.

genes data frame containing annotation for each gene; taken from object.

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

Author(s)

Mark Robinson, Davis McCarthy, Gordon Smyth

References

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

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

See Also

equalizeLibSizes, binomTest

Examples

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

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

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

---

expandAsMatrix *Expand to a Matrix*

Description

Expand scalar or vector to a matrix.

Usage

```
expandAsMatrix(x, dim=NULL, byrow=TRUE)
```

Arguments

x	scalar, vector, matrix or CompressedMatrix.
dim	integer vector of length 2 specifying the required dimensions of the output matrix.
byrow	logical scalar specifying if the matrix should be filled by columns or rows for a vector x.

Details

This function expands a scalar, row/column vector or CompressedMatrix to be a matrix of dimensions `dim`. It is used internally in edgeR to convert offsets, weights and other values to a matrix for consistent handling. If `dim` is `NULL`, the function is equivalent to calling `as.matrix(x)`.

If `x` is a vector, its length must match one of the output dimensions. The matrix will then be filled by repeating the matrix across the matching dimension. For example, if `length(x)==dim[1]`, the matrix will be filled such that each row contains `x`. If both dimensions match, filling is determined by `byrow`, with filling by rows as the default.

If `x` is a CompressedMatrix object, the size of any non-repeated dimensions must be consistent with the corresponding output dimension. The `byrow` argument will be ignored as the repeat specifications will dictate how expansion should be performed. See ?`CompressedMatrix` for more details.

Value

Numeric matrix of dimension `dim`.

Author(s)

Gordon Smyth

Examples

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

`featureCounts2DGEList` *Convert featureCounts Object to a DGEList*

Description

Converts the list output by `Rsubread::featureCounts` to a DGEList object.

Usage

```
featureCounts2DGEList(x, verbose = TRUE)
```

Arguments

`x`	a list produced by `Rsubread::featureCounts`.
`verbose`	if `TRUE`, will give a message to report on the number of junctions that can be assigned to a gene.

Details

Rsubread's `featureCounts` function counts reads by features (typically exons or genomic intervals) or meta-features (typically genes). It can also count reads crossing exon-exon junctions or reads internal to exons. This function assembles all the read counts output by `featureCounts` into a DGEList, ensuring unique row.names where appropriate. The proportion of reads assigned to features is also stored.

If x contains exon and junction counts, then exons and junctions are numbered within genes, and rownames are of the form GeneID.e1 for exons and GeneID.j1 for junctions. Junctions are assigned length 1 in the output.

This function is designed for gene-oriented analyses, including differential usage analyses of exons and junctions. If x contains junction reads, then only junctions for which both the donor and acceptor sites can be assigned to the same gene are included in the output object. The function also drops the columns Transcript and Status from x$counts_junction. Users who want to retain all junctions and annotation, including chimeric junctions, can use DGEList(x$counts_junction).

Value

A DGEList data object. The samples component of the object includes a column PropAssigned recording the proportion of reads (or read-pairs) successfully assigned to features. The genes component of the object includes columns GeneID, Chr, Start, End, Strand, Length, and Exon. The Exon column gives the numbering of the exons and junctions within genes.

Author(s)

Gordon Smyth.

See Also

DGEList-class defines DGEList objects.

See the featureCounts function in the Rsubread package for creation of x.

---

filterByExpr *Filter Genes by Expression Level*

Description

Determine which genes have sufficiently large counts to be retained in a statistical analysis.

Usage

```
## S3 method for class 'DGEList'
filterByExpr(y, design = NULL, group = NULL, lib.size = NULL, ...)
## S3 method for class 'SummarizedExperiment'
filterByExpr(y, design = NULL, group = NULL, lib.size = NULL, ...)
## Default S3 method:
filterByExpr(y, design = NULL, group = NULL, lib.size = NULL,
          min.count = 10, min.total.count = 15, large.n = 10, min.prop = 0.7, ...)
```

Arguments

y	matrix of counts, or a DGEList object, or a SummarizedExperiment object.
design	design matrix. Ignored if group is not NULL. Defaults to y$design if y is a DGEList. Used to compute the minimum group sample size (MinSampleSize).
group	vector or factor giving group membership. Provides an alternative way to specify the design matrix when there is only one explanatory factor (oneway layout). Defaults to y$samples$group if y is a DGEList and design and group are both NULL.

lib.size	library size. Defaults to colSums(y) if y is a matrix or to normLibSizes(y) if y is a DGEList. Used to compute the CPM cutoff value.
min.count	numeric. Minimum count required for at least some samples. Used to compute the CPM cutoff value.
min.total.count	numeric. Minimum total count required across all samples.
large.n	integer. Number of samples that would be considered a "large" group. Used to modify MinSampleSize for large datasets.
min.prop	numeric. Used to modify MinSampleSize for large datasets. See Details below for the exact formula.
...	other arguments are not currently used.

Details

This function implements the filtering strategy that was described informally by Chen et al (2016). Roughly speaking, the strategy keeps genes that have at least min.count reads in a worthwhile number of samples.

More precisely, the filtering keeps genes that have CPM >= CPM.cutoff in MinSampleSize samples, where CPM.cutoff = min.count / median(lib.size) * 1e6 and MinSampleSize is the smallest group sample size or, more generally, the minimum inverse leverage computed from the design matrix.

If all the group sample sizes are large, then MinSampleSize is reduced somewhat. If MinSampleSize > large.n, then genes are kept if CPM >= CPM.cutoff in k samples where k = large.n + (MinSampleSize - large.n) * min.prop. This rule requires that genes are expressed in at least min.prop * MinSampleSize samples, even when MinSampleSize is large.

In addition, each kept gene is required to have at least min.total.count reads across all the samples.

Value

Logical vector of length nrow(y) indicating which rows of y to keep in the analysis.

Note

Note that filterByExpr implements independent filtering in the sense that the filtering would be exactly the same even if the sample group labels or the rows of the design matrix were permuted. The function uses the design and group arguments only to identify the smallest group size (MinSampleSize). Non-independent filtering that used information about which sample belongs to each group, for example keeping genes expressed in a certain number of samples in at least one group, would be invalid because it would potentially cause false discoveries in subsequent differential expression analyses.

Author(s)

Gordon Smyth

References

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Examples

```
## Not run:
keep <- filterByExpr(y, design)
y <- y[keep,]

## End(Not run)
```

---

getCounts *Extract Components of a DGEList Object*

Description

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

`getOffset(y)` returns offsets for the log-linear predictor that account for sequencing depth and possibly other normalization factors. Specifically it returns the matrix `y$offset` if that is non-null. Otherwise, it returns the log effective library sizes (`log(lib.size*norm.factors)`), added columnwise to the prior offset matrix `y$offset.prior` if that exists.

`getDispersion(y)` returns the most complex dispersion estimates (common, trended or genewise) 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, if an offset matrix was found, otherwise a vector of length `ncol(y)`. `getDispersion` returns a vector of dispersion values, either of length 1 or of length `nrow(y)`.

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)
```

---

getNormLibSizes *Effective Library Sizes*

Description

Extract effective (normalized) library sizes.

Usage

```
## Default S3 method:
getNormLibSizes(y, log = FALSE, ...)
```

Arguments

y	an object of class DGEList, DGEGLM or DGELRT. Alternatively a numeric matrix or an object that can be coerced to a numeric matrix.
log	logical, if TRUE then the library sizes are returned on the natural log scale.
...	other arguments are not currently used.

Details

This function extracts normalized library sizes, equal to the original library sizes multiplied by the corresponding normalization factors, from an edgeR data object or fitted model object.

If the object contains row-specific offsets (i.e., a non-sparse matrix of offsets), then the offsets for the first row are returned.

Value

A numeric matrix of effective (normalized) library sizes. If log=TRUE, then natural log values are returned, equal to library size offsets for a NB log-linear model.

Author(s)

Gordon Smyth

See Also

normLibSizes

Examples

```
ngenes <- 100
nsamples <- 4
y <- DGEList(counts=matrix(rnbinom(ngenes*nsamples,size=1,mu=10),ngenes,nsamples))
y <- normLibSizes(y)
data.frame(y$samples, eff.lib.size=getNormLibSizes(y))
```

---

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

Description

Returns the `lib.size` component of the `samples` component of a 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 genewise dispersion estimates.

Details

When estimating genewise 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 genewise 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)
```

---

gini *Gini Dispersion Index*

Description

Gini index for each column of a matrix.

Usage

```
gini(x)
```

Arguments

x	a non-negative numeric matrix, or an object that can be coerced to such a matrix by as.matrix.

Details

The Gini coefficient or index is a measure of inequality or diversity. It is zero if all the values of x are equal. It reaches a maximum value of 1/nrow(x) when all values are zero except for one.

The Gini index is only interpretable for non-negative quantities. It is not meaningful if x contains negative values.

Value

Numeric vector of length ncol(x).

Author(s)

Gordon Smyth

References

https://en.wikipedia.org/wiki/Gini_coefficient.

Examples

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

glmFit *Genewise Negative Binomial Generalized Linear Models*

Description

Fit a negative binomial generalized log-linear model to the read counts for each gene.

Usage

```
## Default S3 method:
glmFit(y, design=NULL, dispersion=NULL, offset=NULL, lib.size=NULL, weights=NULL,
       prior.count=0.125, start=NULL, nthreads=1L, ...)
## S3 method for class 'DGEList'
glmFit(y, design=NULL, dispersion=NULL, prior.count=0.125, start=NULL, ...)
## S3 method for class 'SummarizedExperiment'
glmFit(y, design=NULL, dispersion=NULL, prior.count=0.125, start=NULL, ...)
```

Arguments

y	a matrix of counts or a DGEList object or a SummarizedExperiment containing counts. Rows represent genes and columns represent samples.
design	numeric matrix giving the design matrix for the genewise linear models. Must be of full column rank. Defaults to a single column of ones, equivalent to treating the columns as replicate libraries.
dispersion	negative binomial dispersions. Can be a single value, or a vector of length nrow(y), or a matrix of the same size as y. If NULL, will be extracted from y in this order of precedence: genewise dispersion, trended dispersions, common dispersion.
offset	offsets for the log-linear models containing log effective library sizes. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, then will be computed by getOffset(y).
lib.size	numeric vector of length ncol(y) giving library sizes. Only used if offset=NULL, in which case offset is set to log(lib.size). Defaults to colSums(y).
weights	positive prior weights for the GLM fits. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, will be set to unity for all observations.
prior.count	average prior count to be added to each observation to shrink the estimated log-fold-changes towards zero.
start	optional numeric matrix of initial estimates for the linear model coefficients.
nthreads	integer giving the number of OpenMP threads to use for the genewise GLM fitting. Defaults to 1 (serial).
...	other arguments are passed to lower level fitting functions.

Details

Implements generalized linear model (GLM) methods developed by McCarthy et al (2012). Specifically, glmFit fits genewise negative binomial GLMs, all with the same design matrix but possibly different dispersions, offsets and weights. When the design matrix defines a one-way layout, or

can be re-parametrized to a one-way layout, the GLMs are fitted very quickly using mglmOneGroup. Otherwise the default fitting method, implemented in mglmLevenberg, uses a Fisher scoring algorithm with Levenberg-style damping.

Positive prior.count values cause the returned coefficients to be shrunk in such a way that fold-changes between the treatment conditions are decreased and infinite fold-changes are avoided (Phipson, 2013). Larger prior.count values cause more shrinkage. Coefficient shrinkage does not affect the likelihood ratio tests or p-values.

Value

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

design	design matrix as input.
weights	matrix of weights as input.
df.residual	numeric vector of residual degrees of freedom, one for each gene.
offset	numeric matrix of linear model offsets.
dispersion	vector of dispersions used for the fit.
coefficients	numeric matrix of estimated coefficients from the glm fits. Coefficients are on the natural log scale. Matrix has dimensions nrow(y) by ncol(design).
unshrunk.coefficients	numeric matrix of estimated coefficients from the glm fits when no log-fold-changes shrinkage is applied, on the natural log scale, of size nrow(y) by ncol(design). It exists only when prior.count is not 0.
fitted.values	matrix of fitted values from glm fits, same number of rows and columns as y.
deviance	numeric vector of deviances, one for each gene.
nthreads	integer giving the number of OpenMP threads used for the fit, as specified by the nthreads argument. It is stored on the fitted object so that downstream tests reuse the same number of threads.

Author(s)

Davis McCarthy, Gordon Smyth, Yunshun Chen, Aaron Lun, Lizhong Chen

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

Phipson B (2013). Empirical Bayes modelling of expression profiles and their associations. Ph.D. thesis, Department of Mathematics and Statistics, The University of Melbourne. http://hdl.handle.net/11343/38162.

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

See Also

Low-level computations are done by mglmOneGroup or mglmLevenberg.

topTags displays results from glmLRT.

Examples

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

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

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

# Normalize
d <- normLibSizes(d)

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

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

`glmLRT` *Genewise Likelihood Ratio Tests*

Description

Given genewise generalized linear model fits, conduct likelihood ratio tests for a given coefficient or coefficient contrast.

Usage

```
glmLRT(glmfit, coef=ncol(glmfit$design), contrast=NULL)
```

Arguments

`glmfit`	a `DGEGLM` object, usually output from `glmFit`.
`coef`	integer or character vector indicating which coefficients of the linear model are to be tested equal to zero. Values must be columns or column names of `design`. Defaults to the last coefficient. Ignored if `contrast` is specified.
`contrast`	numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero. If a matrix, then rows correspond to the columns of `design` and columns are contrasts. If non-NULL, then takes precedence over `coef`.

Details

Using genewise GLMs from `glmFit`, `glmLRT` conducts likelihood ratio tests for one or more coefficients in the linear model. If `contrast` is NULL, the null hypothesis is that all the coefficients specified by `coef` are equal to zero. If `contrast` is non-NULL, then the null hypothesis is that the specified contrasts of the coefficients are equal to zero. For example, `contrast = c(0,1,-1)`, assuming there are three coefficients, would test the hypothesis that the second and third coefficients are equal. If `contrast` is a matrix, then each column is a contrast and the null hypothesis is that all the contrasts are equal to zero.

Value

An object of class `DGELRT` with the same components as for `glmFit` plus the following:

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

The data frame `table` contains the following columns:

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

Author(s)

Gordon Smyth, Davis McCarthy, Yunshun Chen

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

`glmFit` fits the genewise GLMs that are input to `glmLRT`.

`topTags` displays results from `glmLRT`.

Examples

```
# See glmFit for an example that includes glmLRT
```

glmQLFit *Genewise Negative Binomial Generalized Linear Models with Quasi-Dispersion Estimation*

Description

Fit a negative binomial generalized log-linear model to the read counts for each gene. Estimate the genewise quasi-dispersions with empirical Bayes moderation.

Usage

```
## Default S3 method:
glmQLFit(y, design = NULL, dispersion = NULL, offset = NULL, lib.size = NULL,
        weights = NULL, abundance.trend = TRUE, AveLogCPM = NULL,
        covariate.trend = NULL, robust = FALSE, winsor.tail.p = c(0.05, 0.1),
        legacy = FALSE, nthreads = 1L, ...)
## S3 method for class 'DGEList'
glmQLFit(y, design = NULL, dispersion = NULL, abundance.trend = TRUE,
        robust = FALSE, winsor.tail.p = c(0.05, 0.1),
        legacy = FALSE, ...)
## S3 method for class 'SummarizedExperiment'
glmQLFit(y, design = NULL, dispersion = NULL, abundance.trend = TRUE,
        robust = FALSE, winsor.tail.p = c(0.05, 0.1),
        legacy = FALSE, ...)
```

Arguments

y	a matrix of counts or a DGEList object or a SummarizedExperiment containing counts. Rows represent genes and columns represent samples.
design	numeric matrix giving the design matrix for the genewise linear models. Must be of full column rank. Defaults to a single column of ones, equivalent to treating the columns as replicate libraries.
dispersion	negative binomial dispersions. Can be a single value, or a vector of length nrow(y), or a matrix of the same size as y. In the new edgeR v4 pipeline (legacy=FALSE), is usually left NULL. Only preset dispersion if you know what you are doing! In this pipeline, any dispersion values greater than 4 will be capped at 4. In the older edgeR v3 pipeline (legacy=TRUE), will be extracted from the DGEList object y. The trended dispersions will be extracted if present, otherwise common dispersion, or will be set to 0.05 if neither is present.
offset	offsets for the log-linear models containing log effective library sizes. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, then will be computed by getOffset(y).
lib.size	numeric vector of length ncol(y) giving library sizes. Only used if offset=NULL, in which case offset is set to log(lib.size). Defaults to colSums(y).
weights	positive prior weights for the GLM fits. Can be a single value, or a vector of length ncol(y), or a matrix of the same size as y. If NULL, will be set to unity for all observations.
abundance.trend	logical, whether to allow an abundance trend for the quasi-dispersion prior.

AveLogCPM	numeric vector giving average log2-counts per million for each row of y. If NULL, then will be computed by aveLogCPM(y).
covariate.trend	numeric vector of length nrow(y). If non-NULL, then will be used instead of AveLogCPM as the covariate for the trended quasi-dispersion prior.
robust	logical, whether to estimate the prior distribution for the quasi-dispersions robustly.
winsor.tail.p	numeric vector of length 2 giving proportion to trim (Winsorize) from lower and upper tail of the distribution of genewise deviances when estimating the hyperparameters of the quasi-dispersion prior. Used as input to squeezeVar when robust=TRUE.
legacy	logical, if TRUE then produce legacy results as for Bioconductor 3.16 and earlier. If FALSE, then use the new quasi-dispersion estimation method with adjusted deviances.
nthreads	integer giving the number of OpenMP threads to use for the genewise computations. Defaults to 1 (serial).
...	other arguments are passed to glmFit.

Details

glmQLFit and glmQLFTest implement the quasi-likelihood (QL) methods of Lund et al (2012) with some enhancements and with slightly different GLM, trend and FDR methods. See Lun et al (2016) or Chen et al (2016) for tutorials describing the use of glmQLFit and glmQLFit as part of a complete analysis pipeline. Another case study using glmQLFit and glmQLFTest is given in Section 4.7 of the edgeR User's Guide.

glmQLFit is similar to glmFit except that it also estimates a quasi-dispersion (quasi-dispersion) for each gene. It calls the limma function squeezeVar to conduct empirical Bayes moderation of the genewise quasi-dispersions. If robust=TRUE, then the robust hyperparameter estimation features of squeezeVar are used (Phipson et al, 2016). If abundance.trend=TRUE, then a prior trend is estimated based on the average logCPMs. If covariate.trend is not NULL, then a prior trend is estimated using the covariate.trend values as predictors.

glmQLFit gives special attention to handling of small counts and zero counts. When legacy=TRUE, the function uses the method of Lun and Smyth (2017) to adjust the residual degrees of freedom when fitted values of zero provide no useful residual degrees of freedom for estimating the quasi-dispersion. The usual residual degrees of freedom are returned as df.residual while the adjusted residual degrees of freedom are returned as df.residuals.zeros.

If legacy=FALSE, then a more comprehensive adjustment for small counts is used. The new method adjusts both the residual deviances and the residual degrees of freedom to improve the accuracy of the quasi-dispersion estimates even for very small counts (Chen et al 2024). With this new method, the residual deviance is no longer equal to the usual residual deviance from generalized linear model theory and the residual degrees of freedom are no longer integers. With the new method, the glmQLFTest function should give good error rate control even for data with many small counts. The legacy=FALSE method was introduced in edgeR 4.0.0 with the Bioconductor 3.18 release. Setting legacy=TRUE will reproduce earlier results as for edgeR 3.8.0 and Bioconductor 3.16.

glmQLFit requires the NB dispersion to be pre-specified when legacy=TRUE but not when legacy=FALSE. In the new pipeline, the NB dispersion will be automatically estimated by glmQLFit whenever dispersion=NULL. In that case, the NB-dispersion will be estimated as the common dispersion of the most highly expressed genes by AveLogCPM. The number of top genes used is min(ceiling(p*ngenes), 10000), where p = chooseLowessSpan(ngenes*sqrt(df.residual),small.n=20,min.span=0.02), ngenes is the number of genes (rows of y) and df.residual is the residual degrees of freedom, equal to ncol(y)-ncol(design).

Value

`glmQLFit` with `legacy=TRUE` produces an object of class `DGEGLM` with the same components as produced by `glmFit`, plus:

`df.residual.zeros`
: numeric vector of effective residual degrees of freedom for each gene, after accounting for treatment groups with all zero counts.

`df.prior`
: numeric vector of prior degrees of freedom for the quasi-dispersions. Has length `nrow(y)` if `robust=TRUE`, otherwise length 1.

`s2.prior`
: numeric vector giving prior value for the quasi-dispersions. Has length `nrow(y)` if `abundance.trend=TRUE`, otherwise length 1.

`s2.post`
: numeric vector of posterior genewise quasi-dispersions.

`glmQLFit` with `legacy=FALSE` produces an object of class `DGEGLM` with the same components as produced by `glmFit`, plus:

`df.residual.adj`
: numeric vector of adjusted residual degrees of freedom for each gene.

`df.prior`
: numeric vector of prior degrees of freedom for the quasi-dispersions. Has length `nrow(y)` if `robust=TRUE`, otherwise length 1.

`s2.prior`
: numeric vector giving prior value for the quasi-dispersions. Has length `nrow(y)` if `abundance.trend=TRUE`, otherwise length 1.

`s2.post`
: numeric vector of posterior genewise quasi-dispersions.

`average.ql.dispersion`
: average quasi-dispersion, used to scale the NB dispersion after estimating the quasi-dispersions.

Note

The negative binomial dispersions `dispersion` supplied to `glmQLFit` must be based on a global model, that is, they must be either trended or common dispersions. It is not correct to supply genewise dispersions because `glmQLFit` estimates gene-specific variability using the quasi-dispersions.

Author(s)

Yunshun Chen, Aaron Lun, Davis McCarthy, Lizhong Chen and Gordon Smyth

References

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Lun ATL, Chen Y, Smyth GK (2016). It's DE-licious: a recipe for differential expression analyses of RNA-seq experiments using quasi-likelihood methods in edgeR. *Methods in Molecular Biology* 1418, 391-416. doi:10.1007/9781493935789_19. See also the preprint https://gksmyth.github.io/pubs/QLedgeRPreprint.pdf.

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

Genetics and Molecular Biology 11(5), Article 8. doi:10.1515/15446115.1826. https://gksmyth.github.io/pubs/QuasiSeqPreprint.pdf.

Lun ATL, Smyth GK (2017). No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data. *Statistical Applications in Genetics and Molecular Biology* 16(2), 83-93. doi:10.1515/sagmb20170010.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10, 946-963. doi:10.1214/16AOAS920.

See Also

`glmQLFTest` performs F-tests using the fit from `glmQLFit`.

`plotQLDisp` can be used to visualize the distribution of quasi-dispersions after EB shrinkage from `glmQLFit`.

The `QuasiSeq` package gives an alternative implementation of the Lund et al (2012) methods.

Examples

```
nlibs <- 4
ngenes <- 1000
dispersion.true <- 1/rchisq(ngenes, df=10)
design <- model.matrix(~factor(c(1,1,2,2)))

# Generate count data
y <- rnbinom(ngenes*nlibs,mu=20,size=1/dispersion.true)
y <- matrix(y,ngenes,nlibs)
d <- DGEList(y)
d <- normLibSizes(d)

# Fit the NB GLMs with QL methods
fit <- glmQLFit(d, design)
results <- glmQLFTest(fit)
topTags(results)
fit <- glmQLFit(d, design, robust=TRUE)
results <- glmQLFTest(fit)
topTags(results)
fit <- glmQLFit(d, design, abundance.trend=FALSE)
results <- glmQLFTest(fit)
topTags(results)
```

`glmQLFTest` *Quasi-Likelihood F-Tests*

Description

Conduct genewise quasi F-tests for a given coefficient or coefficient contrast.

Usage

```
glmQLFTest(glmfit, coef = ncol(glmfit$design), contrast = NULL, fc=1, lfc=NULL,
        poisson.bound = TRUE, upshot=TRUE)
```

Arguments

glmfit	a DGEGLM object, usually output from glmQLFit.
coef	integer or character index vector indicating which coefficients of the linear model are to be tested equal to zero. Ignored if contrast is not NULL.
contrast	numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero.
fc	scalar or numeric vector of minimum fold-change below which changes are not considered scientifically meaningful.
lfc	scalar or numeric vector of minimum log2-fold-change below which changes are not considered scientifically meaningful. Defaults to log2(fc). If specified then takes precedence over fc.
poisson.bound	logical, if TRUE then the p-value returned will never be less than would be obtained for a likelihood ratio test with NB-dispersion equal to 0 and quasi-dispersion equal to 1. This argument is no longer relevant for the edgeR v4 QL pipeline and will be ignored if glmfit was created by glmQLFit with legacy=FALSE.
upshot	logical. If TRUE, then a more powerful test is conducted assuming that the true logFC are uniformly distributed over the null interval from -lfc to lfc. If FALSE, then the standard TREAT p-values are computed assuming the worst-case scenario of true logFC equal to lfc.

Details

glmQLFTest is typically used after glmQLFit. The two functions implement the quasi-likelihood (QL) methods of Lund et al (2012) with some enhancements and with slightly different GLM, trend and FDR methods. See Lun et al (2016) or Chen et al (2016) for tutorials describing the use of glmQLFit and glmQLFit as part of a complete analysis pipeline. Another case study using glmQLFit and glmQLFTest is given in Section 4.7 of the edgeR User's Guide.

glmQLFTest is similar to glmLRT except that it replaces likelihood ratio tests with empirical Bayes quasi-likelihood F-tests. The p-values from glmQLFTest are always greater than or equal to those that would be obtained from glmLRT using the same negative binomial dispersions.

Value

An object of class DGELRT with the same components as produced by glmLRT, except that the table$LR column becomes table$F and contains quasi-likelihood F-statistics. It also stores df.total, a numeric vector containing the denominator degrees of freedom for the F-test, equal to df.prior + df.residual.zeros.

Author(s)

Yunshun Chen, Aaron Lun, Davis McCarthy, Lizhong Chen and Gordon Smyth

References

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Lun ATL, Chen Y, Smyth GK (2016). It's DE-licious: a recipe for differential expression analyses of RNA-seq experiments using quasi-likelihood methods in edgeR. *Methods in Molecular Biology* 1418, 391-416. doi:10.1007/9781493935789_19. See also the preprint https://gksmyth.github.io/pubs/QLedgeRPreprint.pdf.

Lund SP, Nettleton D, McCarthy DJ, Smyth GK (2012). Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates. *Statistical Applications in Genetics and Molecular Biology* 11(5), Article 8. doi:10.1515/15446115.1826. See also the preprint https://gksmyth.github.io/pubs/QuasiSeqPreprint.pdf.

Lun ATL, Smyth GK (2017). No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data. *Statistical Applications in Genetics and Molecular Biology* 16(2), 83-93. doi:10.1515/sagmb20170010.

Phipson B, Lee S, Majewski IJ, Alexander WS, Smyth GK (2016). Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Annals of Applied Statistics* 10, 946-963. doi:10.1214/16AOAS920.

See Also

glmQLFit provides estimated GLMs to glmQLFTest.

topTags displays results from glmQLFTest.

plotQLDisp can be used to visualize the distribution of quasi-dispersions after EB shrinkage from glmQLFit.

The QuasiSeq package gives an alternative implementation of the Lund et al (2012) methods.

Examples

```
# See glmQLFit for an example using glmQLFTest
```

---

glmTreat *Test for Differential Expression Relative to a Threshold*

Description

Conduct genewise statistical tests for a given coefficient or contrast relative to a specified fold-change threshold.

Usage

```
glmTreat(glmfit, coef = ncol(glmfit$design), contrast = NULL, lfc = log2(1.2),
         null = "interval", legacy=FALSE)
```

Arguments

glmfit	a DGEGLM object, usually output from glmFit or glmQLFit.
coef	integer or character vector indicating which coefficients of the linear model are to be tested equal to zero. Values must be columns or column names of design. Defaults to the last coefficient. Ignored if contrast is specified.
contrast	numeric vector specifying the contrast of the linear model coefficients to be tested against the log2-fold-change threshold. Length must equal to the number of columns of design. If specified, then takes precedence over coef.

lfc	numeric scalar specifying the absolute value of the log2-fold change threshold above which differential expression is to be considered.
null	character string, choices are "worst.case" or "interval". If "worst.case", then the null hypothesis assumes that the true logFC is on the boundary of the possible values, either at lfc or -lfc, whichever gives the largest p-value. This gives the most conservative results. If "interval", then the null hypothesis assumes the true logFC to belong to a bounded interval of possible values.
legacy	logical, if TRUE then produce legacy results as for Bioconductor 3.22 and earlier. If FALSE, then use the improved z-score approach.

Details

glmTreat implements a test for differential expression relative to a minimum required fold-change threshold. Instead of testing for genes which have log-fold-changes different from zero, it tests whether the log2-fold-change is greater than lfc in absolute value. glmTreat is analogous to the TREAT approach developed by McCarthy and Smyth (2009) for microarrays.

Note that the lfc testing threshold used to define the null hypothesis is not the same as a log2-fold-change cutoff, as the observed log2-fold-change needs to be substantially larger than lfc for the gene to be called as significant. In practice, modest values for lfc such as log2(1.1), log2(1.2) or log2(1.5) are usually the most useful. In practice, setting lfc=log2(1.2) or lfc=log2(1.5) will usually cause most differentially expressed genes to have estimated fold-changes of 2-fold or greater, depending on the sample size and precision of the experiment.

Note also that glmTreat constructs test statistics using the unshrunk log2-fold-changes(unshrunk.logFC) rather than the log2-fold-changes that are usually reported (logFC). If no shrinkage has been applied to the log-fold-changes, i.e., the glms were fitted with prior.count=0, then unshrunk.logFC and logFC are the same and the former is omitted from the output object.

glmTreat detects whether glmfit was produced by glmFit or glmQLFit. In the former case, it conducts a modified likelihood ratio test (LRT) against the fold-change threshold. In the latter case, it conducts a quasi-likelihood (QL) F-test against the threshold.

If lfc=0, then glmTreat is equivalent to glmLRT or glmQLFTest, depending on whether likelihood or quasi-likelihood is being used.

glmTreat with positive lfc gives larger p-values than would be obtained with lfc=0. If null="worst.case", then glmTreat conducts a test closely analogous to the treat function in the limma package. This conducts a test in which the null hypothesis puts the true logFC on the boundary of the [-lfc,lfc] interval closest to the observed logFC. If null="interval", then the null hypothesis assumes an interval of possible values for the true logFC. This approach is somewhat less conservative.

Note that, unlike other edgeR functions such as glmLRT and glmQLFTest, glmTreat can only accept a single contrast. If contrast is a matrix with multiple columns, then only the first column will be used.

Value

glmTreat produces an object of class DGELRT with the same components as for glmfit plus the following:

lfc	absolute value of the specified log2-fold-change threshold.
table	data frame with the same rows as glmfit containing the log2-fold-changes, average log2-counts per million and p-values, ready to be displayed by topTags.
comparison	character string describing the coefficient or the contrast being tested.

The data frame table contains the following columns:

logFC	shrunk log2-fold-change of expression between conditions being tested.
unshrunk.logFC	unshrunk log2-fold-change of expression between conditions being tested. Exists only when prior.count is not equal to 0 for glmfit.
logCPM	average log2-counts per million, the average taken over all libraries.
PValue	p-values.

Note

glmTreat was previously called treatDGE in edgeR versions 3.9.10 and earlier.

Author(s)

Yunshun Chen, Lizhong Chen and Gordon Smyth

References

McCarthy DJ, Smyth GK (2009). Testing significance relative to a fold-change threshold is a TREAT. *Bioinformatics* 25, 765-771. doi:10.1093/bioinformatics/btp053.

See Also

topTags displays results from glmTreat.

treat is the corresponding function in the limma package, designed for use with normally distributed log-expression data rather than for negative binomial counts.

Examples

```
ngenes <- 100
n1 <- 3
n2 <- 3
nlibs <- n1+n2
mu <- 100
phi <- 0.1
group <- c(rep(1,n1), rep(2,n2))
design <- model.matrix(~as.factor(group))

### 4-fold change for the first 5 genes
i <- 1:5
fc <- 4
mu <- matrix(mu, ngenes, nlibs)
mu[i, 1:n1] <- mu[i, 1:n1]*fc

counts <- matrix(rnbinom(ngenes*nlibs, mu=mu, size=1/phi), ngenes, nlibs)
d <- DGEList(counts=counts,lib.size=rep(1e6, nlibs), group=group)

gfit <- glmFit(d, design, dispersion=phi)
tr <- glmTreat(gfit, coef=2, lfc=1)
topTags(tr)
```

goana.DGELRT *Gene Ontology or KEGG Analysis of Differentially Expressed Genes*

Description

Test for over-representation of gene ontology (GO) terms or KEGG pathways in the up and down differentially expressed genes from a linear model fit.

Usage

```
## S3 method for class 'DGELRT'
goana(de, geneid = rownames(de), FDR = 0.05, trend = FALSE, ...)
## S3 method for class 'DGELRT'
kegga(de, geneid = rownames(de), FDR = 0.05, trend = FALSE, ...)
```

Arguments

de	an DGELRT or DGEExact object.
geneid	gene IDs. Either a character vector of length nrow(de) or the name of the column of de$genes containing the Gene IDs.
FDR	false discovery rate cutoff for differentially expressed genes. Numeric value between 0 and 1.
trend	adjust analysis for gene length or abundance? Can be logical, or a numeric vector of covariate values, or the name of the column of de$genes containing the covariate values. If TRUE, then de$AveLogCPM is used as the covariate.
...	any other arguments are passed to goana.default or kegga.default.

Details

goana performs Gene Ontology enrichment analyses for the up and down differentially expressed genes from a linear model analysis. kegga performs the corresponding analysis for KEGG pathways.

The argument de should be a fitted model object created by glmLRT, glmTreat, glmQLFTest or exactTest.

For goana, the gene IDs must be Entrez Gene IDs. These can be supplied either as row.names of de or as a column of de$genes. In the latter case, the column name containing the Entrez IDs is given by geneid. Alternatively, if the Entrez IDs are not part of the de object, then they can be supplied as a vector argument to geneid.

For kegga, gene IDs other than Entrez Gene IDs are supported for some species. See kegga.default for more information.

If trend=FALSE, the function computes one-sided hypergeometric tests equivalent to Fisher's exact test.

If trend=TRUE or a covariate is supplied, then a trend is fitted to the differential expression results and the method of Young et al (2010) is used to adjust for this trend. The adjusted test uses Wallenius' noncentral hypergeometric distribution.

Value

goana produces a data.frame with a row for each GO term and the following columns:

Term	GO term.
Ont	ontology that the GO term belongs to. Possible values are "BP", "CC" and "MF".
N	Number of genes in the GO term.
Up	number of up-regulated differentially expressed genes.
Down	number of down-regulated differentially expressed genes.
P.Up	p-value for over-representation of GO term in up-regulated genes.
P.Down	p-value for over-representation of GO term in down-regulated genes.

The row names of the data frame give the GO term IDs.

kegga produces a data.frame as above except that the rownames are KEGG pathway IDs, Term becomes Path and there is no Ont column.

Note

This is the help page for goana when de is an edgeR fitted model object. See ?goana for other possible input types.

Author(s)

Yunshun Chen and Gordon Smyth

References

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Young MD, Wakefield MJ, Smyth GK, Oshlack A (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology* 11, R14. doi:10.1186/gb2010112r14.

See Also

goana, topGO, kegga, topKEGG

Examples

```
## Not run:
fit <- glmFit(y, design)
lrt <- glmLRT(fit)
go <- goana(lrt, species="Hs")
topGO(go, ont="BP", sort="up")
topGO(go, ont="BP", sort="down")
## End(Not run)
```

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 residual deviances", ...)
```

Arguments

glmfit	a DGEGLM object containing results from fitting NB GLMs to genes in a DGE dataset with a global dispersion model. Usually this is 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

This function is useful for evaluating the adequacy of a global dispersion model, such as a constant or trended dispersion. If plot=TRUE, then it produces a qq-plot similar to those in Figure 2 of McCarthy et al (2012).

Value

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-square) test for significance.

If plot=TRUE, then a plot is also produced on the current graphics device.

Note

This function should not be used with tagwise estimated dispersions such as those from estimateGLMTagwiseDisp or estimateDisp. glmfit should contain trended or constant dispersions.

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 doi:10.1093/nar/gks042.

See Also

qqnorm.

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

Examples

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

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

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

# Normalize
d <- normLibSizes(d)

# Fit the NB GLMs
fit <- glmFit(d, design, dispersion=dispersion.true)
# Check how good the fit is for each gene
gof(fit)
```

---

goodTuring *Good-Turing Frequency Estimation*

Description

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

Usage

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

Arguments

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

Details

Observed counts are assumed to be Poisson distributed. Using a non-parametric empirical Bayes strategy, the algorithm evaluates the posterior expectation of each species mean given its observed count. The posterior means are then converted to proportions. In the empirical Bayes step, the counts are smoothed by assuming a log-linear relationship between frequencies and frequencies of frequencies. The fundamentals of the algorithm are from Good (1953). Gale and Sampson (1995) proposed a simplified 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 is not used as part of the algorithm, but is returned by this function.

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

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

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

Value

goodTuring returns a list with components

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

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

Author(s)

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

References

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

Good IJ (1953). The population frequencies of species and the estimation of population parameters. *Biometrika* 40, 237-264.

Examples

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

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

---

head *Return the First to Last Part of a Data Object*

Description

Retrieve the first or last parts of a DGEList, PCList, DGEExact, DGEGLM, DGELRT or TopTags object.

Usage

```
## S3 method for class 'DGEList'
head(x, n = 6L, ...)
## S3 method for class 'DGEList'
tail(x, n = 6L, ...)
```

Arguments

x	an object of class DGEList, PCList, DGEExact, DGEGLM, DGELRT or TopTags.
n	a single integer. If positive or zero, gives the number of rows of the resulting object. If negative, all but the n last/first rows of x.
...	other arguments are not currently used.

Details

head (tail) returns the first (last) n rows when n >= 0 or all but the last (first) n rows when n < 0.

Value

An object like x but generally with fewer rows.

Author(s)

Gordon Smyth

See Also

head in the utils package or head.EList in the limma package.

Examples

```
Counts <- matrix(rpois(40,lambda=10),20,2)
rownames(Counts) <- paste0("Gene",1:20)
colnames(Counts) <- c("A","B")
y <- DGEList(Counts)
head(y)
tail(y)
```

`loessByCol` *Locally Weighted Mean by Column*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

`locfitByCol` returns a numeric matrix.

Author(s)

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

See Also

`loess`

Examples

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

`makeCompressedMatrix` *Make a CompressedMatrix*

Description

Construct a CompressedMatrix object from a scalar, vector or matrix.

Usage

```
makeCompressedMatrix(x, dims = NULL, byrow = TRUE)

## S3 method for class 'CompressedMatrix'
dim(x)
## S3 method for class 'CompressedMatrix'
length(x)

## S3 method for class 'CompressedMatrix'
x[i, j, drop=TRUE]
## S3 replacement method for class 'CompressedMatrix'
x[i, j] <- value

## S3 method for class 'CompressedMatrix'
Ops(e1, e2)

## S3 method for class 'CompressedMatrix'
rbind(...)
## S3 method for class 'CompressedMatrix'
cbind(...)

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

Arguments

`x`	For `makeCompressedMatrix`, a scalar, vector, matrix or CompressedMatrix object. For the S3 methods, a CompressedMatrix object.
`dims`	integer vector of length 2 giving matrix dimensions, ignored if `x` is already a matrix.
`byrow`	logical. If `x` is a vector, should it be repeated across rows (default) or across columns?
`i, j`	subset indices to apply to `x`, which behave the same as indices for `matrix` subsetting.
`drop`	logical, indicating whether or not to drop dimensions when subsetting to a single row/column.

value	an array-like object or vector to be used to replace values in x.
e1, e2	a CompressedMatrix object.
...	multiple CompressedMatrix objects for rbind and cbind. Otherwise additional arguments that are ignored in as.matrix.

Details

CompressedMatrix objects are used throughout edgeR to save space in storing offsets and (to a lesser extent) weights. This is because, for routine analyses, offsets are the same for all genes so it makes little sense to expand it to the full dimensions of the count matrix. Most functions will accept a CompressedMatrix as input to offset or weights arguments.

Value

An object of class CompressedMatrix, containing x and the additional attributes repeat.row and repeat.col.

Class construction

The makeCompressedMatrix function creates a CompressedMatrix object from x. The CompressedMatrix class inherits from a matrix and holds two logical scalar attributes repeat.row and repeat.col. Each attribute specifies whether the values are to be repeated across rows and/or across columns. This avoids the need to store redundant values in a full-sized matrix of dimensions dim, as would be done with expandAsMatrix.

To illustrate, consider that rows usually correspond to genes while columns usually correspond to libraries. If we have a vector of library sizes, this will hold one unique value per library that is the same for all genes. Thus, we should use byrow=TRUE, which will construct a CompressedMatrix object storing one row containing this vector. Here, repeat.row=TRUE and repeat.col=FALSE, indicating that the row is to be repeated for all genes.

On the other hand, we may have a vector of gene-specific values that is the same for all libraries (e.g., dispersions). In this case, we should use byrow=FALSE to construct the CompressedMatrix object. This will store one column with repeat.row=FALSE and repeat.col=TRUE, indicating that the column should be repeated across libraries.

In cases where x is a scalar, byrow is ignored and both repeat.row and repeat.col will be TRUE by default. If x is a matrix, both attributes will be FALSE. If x is a CompressedMatrix, it will be returned without modification.

Class methods

Subsetting of a CompressedMatrix object depends on the values of repeat.row and repeat.col. If the rows are repeated, any subsetting by row will be effectively ignored, only altering the stored dimensions of x without changing the values. Similarly, if the columns are repeated, any subsetting by column will be ignored. If neither are repeated, subsetting behaves as it would for a normal matrix.

Combining of a CompressedMatrix object will also make use of the repeat structure. If rows are repeated in all objects to be combined, the output of cbind will also have repeated rows. Similarly, if columns are repeated, the output of rbind will also have repeated columns. Otherwise, all objects are expanded to their full size prior to combining.

Binary operators work on pairs of CompressedMatrix objects, again preserving the repeat structure whenever possible. Extracting dimensions uses a second Dims field in the attributes, bypassing the dim for a base matrix. Calling as.matrix on a CompressedMatrix object will return the ordinary (uncompressed) matrix.

Author(s)

Aaron Lun

References

Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R (2020). Array programming with NumPy. *Nature* 585(7825), 357-362. doi:10.1038/s4158602026492.

Hillis WD, Steele GL Jr (1986). Data parallel algorithms. *Communications of the ACM* 9(12), 1170-1183. doi:10.1145/7902.7903.

See Also

`as.matrix`, `expandAsMatrix`

Examples

```
# Repeated rows:
library.sizes <- runif(4, 1e6, 2e6)
lib.mat <- makeCompressedMatrix(library.sizes, c(10, 4), byrow=TRUE)
lib.mat

lib.mat[,1:2] # subset by column works as expected
lib.mat[1:10,] # subset by row has no effect (see Details)
as.matrix(lib.mat)

# Repeated columns:
gene.disp <- runif(10, 0.01, 0.1)
disp.mat <- makeCompressedMatrix(gene.disp, c(10, 4), byrow=FALSE)
disp.mat

disp.mat[,1:2] # subset by column has no effect
disp.mat[1:5,] # subset by row works as expected
as.matrix(disp.mat)

# Scalar:
weights <- makeCompressedMatrix(1, c(10, 4))
weights[1:10,] # subsetting has no effect
weights[,1:10]
as.matrix(weights)

# Matrix:
offsets <- makeCompressedMatrix(matrix(runif(40), 10, 4))
offsets[1:5,]
offsets[,1:2]
as.matrix(offsets)
```

`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 a low A value is presented.

Usage

```
maPlot(x, y, logAbundance=NULL, logFC=NULL, normalize=FALSE, plot.it=TRUE,
     smearWidth=1, col=NULL, allCol="black", lowCol="orange", deCol="red",
     de.tags=NULL, smooth.scatter=FALSE, lowess=FALSE, ...)
```

Arguments

x	vector of counts or concentrations (group 1).
y	vector of counts or concentrations (group 2).
logAbundance	vector providing the abundance of each gene 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 gene for a given experimental contrast. Default is NULL, only to be used together with logAbundance as both need to be non-null for their values to be used.
normalize	logical, whether to divide x and y vectors by their sum.
plot.it	logical, whether to produce a plot.
smearWidth	scalar, width of the smear.
col	vector of colors for the points (if NULL, uses allCol and lowCol).
allCol	color of the non-smeared points.
lowCol	color of the smeared points.
deCol	color of the DE (differentially expressed) points.
de.tags	indices for genes identified as being differentially expressed; use exactTest or glmLRT to identify DE genes. Note that 'tag' and 'gene' are synonymous here.
smooth.scatter	logical, whether to produce a 'smooth scatter' plot using the graphics::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

If plot.it=TRUE, a scatterplot is made on the current graphics device. The function also invisibly returns a list containing the M (logFC) and A (logConc) values used for the plot, plus identifiers w and v of genes for which M and A values, or just M values, respectively, were adjusted to make a nicer looking plot.

Author(s)

Mark Robinson, Davis McCarthy

See Also

plotSmear

Examples

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

---

maximizeInterpolant *Maximize a Function given a Table of Values by Spline Interpolation*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Aaron Lun, improving on earlier code by Gordon Smyth

References

Forsythe GE, Malcolm MA, Moler CB (1977). *Computer Methods for Mathematical Computations*, Prentice-Hall.

Examples

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

`maximizeQuadratic` *Maximize a Function given a Table of Values by Quadratic Interpolation*

Description

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

Usage

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

Arguments

`y`	numeric matrix of response values.
`x`	numeric matrix of inputs of the function of the 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 quadratic polynomial through these `y` values for these three `x` 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)
```

mbin *Fit Binomial Generalized Linear Models to Multiple Response Vectors: Low Level Functions*

Description

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

Usage

```
mbinOneWay(y, z, design = NULL, group = NULL, offset = NULL, weights = NULL,
        maxit = 50, tol = 1e-8, nthreads = 1L)
mbinIWLS(y, z, design, offset = NULL, weights = NULL,
        maxit = 100, tol = 1e-06, nthreads = 1L)
```

Arguments

y	numeric matrix containing the counts. Rows for genes and columns for libraries.
z	numeric matrix containing the paired counts. Rows for genes and columns for libraries.
design	numeric matrix giving the design matrix of the GLM. Assumed to be full column rank. This is a required argument for mbinIWLS and code{designAsFactor}. For mBinOneWay, it defaults to model.matrix(~0+group) if group is specified and otherwise to model.matrix(~1).
group	factor giving group membership for oneway layout. If both design and group are specified, then they must agree in terms of designAsFactor. If design=NULL, then a group-means design matrix is implied.
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.
tol	numeric scalar giving the convergence tolerance. For mbinOneWay, convergence is judged successful when the step size falls below tol in absolute size.
maxit	integer giving the maximum number of iterations for the iterated weighted least square algorithm. The iteration will be stopped when this limit is reached even if the convergence criterion hasn't been satisfied.
nthreads	integer giving the number of OpenMP threads to use for the genewise fitting. Defaults to 1 (serial).

Details

These functions are low-level work-horses used by higher-level functions in the edgeR package, especially by binFit. Most users will not need to call these functions directly.

The functions mbinOneWay and mbinIWLS all fit a binomial GLM to each row of y and z. The row-wise GLMS all have the same design matrix but possibly different offsets and weights. These functions are all low-level in that they operate on atomic objects (numeric matrices and vectors).

mbinOneWay fits a oneway layout to each response vector. It treats the libraries as belonging to a number of groups and fits each group.

mbinIWLS fits an arbitrary binomial model to each response vector. It implements a standard iterated weighted least square algorithm. The main computation is implemented in C.

Value

mbinOneWay produces a list with the following components:

coefficients	matrix of estimated coefficients for the models. Rows correspond to rows of y and columns to columns of design.
fitted.values	matrix of fitted values. Of same dimensions as y.
deviance	numeric vector of residual deviances.

mbinIWLS produces a list with the following components:

coefficients	matrix of estimated coefficients for the linear models.
fitted.values	matrix of fitted values.
deviance	numeric vector of residual deviances.
iter	number of iterations used.
fail	logical vector indicating genes for which the maximum damping was exceeded before convergence was achieved.

Author(s)

Gordon Smyth, Lizhong Chen.

See Also

Most users will call either binFit, the higher-level function offering more object-oriented GLM modeling of Paired Counts data.

---

meanvar *Explore the Mean-Variance Relationship for DGE Data*

Description

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

Usage

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

Arguments

object	DGEList object containing the raw data and dispersion value. Depending on the method desired for computing the dispersion, 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 gene means and variances when producing a mean-variance plot.
show.raw.vars	logical, whether or not to display the raw (pooled) genewise variances on the mean-variance plot. Default is FALSE.
show.tagwise.vars	logical, whether or not to display the estimated genewise variances on the mean-variance plot (note that 'tag' and 'gene' are synonymous). Default is FALSE.
show.binned.common.disp.vars	logical, whether or not to compute the common dispersion for each bin of genes and show the variances computed from those binned common dispersions and the mean expression level of the respective bin of genes. Default is FALSE.
show.ave.raw.vars	logical, whether or not to show the average of the raw variances for each bin of genes plotted against the average expression level of the genes 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. This would be expected to have 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 genes and columns representing samples.
group	factor giving the experimental group or condition to which each sample (i.e. column of x or element of y) belongs.
common.dispersion	logical, whether or not to compute the common dispersion for each bin of genes.

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 genes 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 changes 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 genewise variances can also be plotted. Estimated genewise variances can be calculated using either qCML estimates of the genewise 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 genes in each bin, ordered from lowest expression bin to highest

Author(s)

Davis McCarthy

See Also

`plotMeanVar2` is related in purpose but works from raw counts and uses standardized residuals.

`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
# Produce a mean-variance plot with the raw variances shown and save the means
# and variances for later use
meanvar <- plotMeanVar(d, show.raw.vars=TRUE)
## If we want to show estimated genewise variances on the plot, we must first estimate them!
d <- estimateCommonDisp(d) # Obtain an estimate of the dispersion parameter
d <- estimateTagwiseDisp(d)  # Obtain genewise dispersion estimates
```

```
# Use previously saved object to speed up plotting
plotMeanVar(d, meanvar=meanvar, show.tagwise.vars=TRUE, NBline=TRUE)
## We could also estimate common/genewise dispersions using the Cox-Reid methods with an
## appropriate design matrix
```

---

mglm *Fit Negative Binomial Generalized Linear Models to Multiple Response Vectors: Low Level Functions*

Description

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

Usage

```
mglmOneGroup(y, dispersion = 0, offset = 0, weights = NULL,
             coef.start = NULL, maxit = 50, tol = 1e-10, verbose = FALSE,
             nthreads = 1L)
mglmOneWay(y, design = NULL, group = NULL, dispersion = 0, offset = 0, weights = NULL,
             coef.start = NULL, maxit = 50, tol = 1e-10,
             nthreads = 1L)
mglmLevenberg(y, design, dispersion = 0, offset = 0, weights = NULL,
             coef.start = NULL, start.method = "null", maxit = 200, tol = 1e-06,
             nthreads = 1L)
designAsFactor(design)
```

Arguments

y	numeric matrix containing the negative binomial counts. Rows for genes and columns for libraries.
design	numeric matrix giving the design matrix of the GLM. Assumed to be full column rank. This is a required argument for mglmLevenberg and designAsFactor. For mglmOneWay, it defaults to model.matrix(~0+group) if group is specified and otherwise to model.matrix(~1).
group	factor giving group membership for oneway layout. If both design and group are specified, then they must agree in terms of designAsFactor. If design=NULL, then a group-means design matrix is implied.
dispersion	numeric scalar or vector giving the dispersion parameter for each GLM. Can be a scalar giving one value for all genes, or a vector of length equal to the number of genes giving genewise dispersions.
offset	numeric vector or matrix giving the offset that is to be included in the log-linear model predictor. Can be a scalar, a vector of length equal to the number of libraries, or a matrix of the same size as y.
weights	numeric vector or matrix of non-negative quantitative weights. Can be a vector of length equal to the number of libraries, or a matrix of the same size as y.
coef.start	numeric matrix of starting values for the linear model coefficients. Number of rows should agree with y and number of columns should agree with design. For mglmOneGroup, a numeric vector or a matrix with one column. This argument does not usually need to be set as the automatic starting values perform well.

start.method	method used to generate starting values when coef.stat=NULL. Possible values are "null" to start from the null model of equal expression levels or "y" to use the data as starting value for the mean.
tol	numeric scalar giving the convergence tolerance. For mglmOneGroup, convergence is judged successful when the step size falls below tol in absolute size.
maxit	integer giving the maximum number of iterations for the Fisher scoring algorithm. The iteration will be stopped when this limit is reached even if the convergence criterion hasn't been satisfied.
verbose	logical. If TRUE, warnings will be issued when maxit iterations are exceeded before convergence is achieved.
nthreads	integer giving the number of OpenMP threads to use for the genewise fitting. Defaults to 1 (serial).

Details

These functions are low-level work-horses used by higher-level functions in the edgeR package, especially by glmFit. Most users will not need to call these functions directly.

The functions mglmOneGroup, mglmOneWay and mglmLevenberg all fit a negative binomial GLM to each row of y. The row-wise GLMs all have the same design matrix but possibly different dispersions, offsets and weights. These functions are all low-level in that they operate on atomic objects (numeric matrices and vectors).

mglmOneGroup fits an intercept only model to each response vector. In other words, it treats all the libraries as belonging to one group. It implements Fisher scoring with a score-statistic stopping criterion for each gene. 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 gene.

mglmOneWay fits a oneway layout to each response vector. It treats the libraries as belonging to a number of groups and calls mglmOneGroup for each group.

mglmLevenberg fits an arbitrary log-linear model to each response vector. It implements a Levenberg-Marquardt modification of the GLM scoring algorithm to prevent divergence. The main computation is implemented in C++.

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

designAsFactor is used to convert a general design matrix into a oneway layout if that is possible. It determines how many distinct row values the design matrix is capable of computing and returns a factor with a level for each possible distinct value.

Value

mglmOneGroup produces a numeric vector of coefficients, which estimate the log-nucleotide-fraction for each gene.

mglmOneWay produces a list with the following components:

coefficients	matrix of estimated coefficients for the linear models. Rows correspond to rows of y and columns to columns of design.
fitted.values	matrix of fitted values. Of same dimensions as y.

mglmLevenberg produces a list with the following components:

coefficients	matrix of estimated coefficients for the linear models.

`fitted.values`	matrix of fitted values.
`deviance`	numeric vector of residual deviances.
`iter`	number of iterations used.
`fail`	logical vector indicating genes for which the maximum damping was exceeded before convergence was achieved.

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

Author(s)

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

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

Chen Y, Chen L, Lun ATL, Baldoni PL, Smyth GK (2025). edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. *Nucleic Acids Research* 53(2), gkaf018. doi:10.1093/nar/gkaf018.

See Also

Most users will call either `glmFit`, the higher-level function offering more object-oriented GLM modeling of DGE data, or else `exactTest`, which is designed for oneway layouts.

Examples

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

## Compute intercept for each row
beta <- mglmOneGroup(y, dispersion = dispersion, offset = log(lib.size))

## Unlogged intercepts add to one:
sum(exp(beta))

## Fit the NB GLM to the counts with a given design matrix
f1 <- factor(c(1,1,2,2))
f2 <- factor(c(1,2,1,2))
X <- model.matrix(~ f1 + f2)
fit <- mglmLevenberg(y, X, dispersion = dispersion, offset = log(lib.size))
head(fit$coefficients)
```

`modelMatrixMeth` *Construct Design Matrix for edgeR Analysis of Methylation Count Data*

Description

Construct design matrix (aka model matrix) for edgeR analysis of methylation count data from sample level information.

Usage

```
modelMatrixMeth(object, ...)
```

Arguments

`object`	a sample-level design matrix or model formula or terms object.
`...`	any other arguments are passed to `model.matrix`.

Details

This function computes a design matrix for modeling methylated and unmethylated counts. The resulting design matrix can be input to `glmFit` when analyzing BS-seq methylation data using edgeR.

In BS-seq methylation analysis, each DNA sample generates two counts, a count of methylated reads and a count of unmethylated reads, for each genomic locus. The function converts sample-level information about the treatment conditions to make an appropriate design matrix with two rows for each sample. Counts are assumed to be ordered as methylated and then unmethylated by sample.

If `design.treatments <- model.matrix(object,...)` has `nsamples` rows and `p` columns, then `modelMatrixMeth(object, ...)` has `2*nsamples` rows and `nsamples+p` columns. See Chen et al (2017) for more information.

Value

A numeric design matrix. It has 2 rows for each sample and a column for each sample in addition to the columns generated by `model.matrix(object, ...)`.

Author(s)

Gordon Smyth

References

Chen Y, Pal B, Visvader JE, Smyth GK (2017). Differential methylation analysis of reduced representation bisulfite sequencing experiments using edgeR. *F1000Research* 6, 2055. doi:10.12688/f1000research.13196.2.

See Also

`model.matrix` in the stats package.

Examples

```
Treatments <- gl(3,2,labels=c("A","B","C"))
modelMatrixMeth(~Treatments)

# Equivalent calling sequence:
design.treatments <- model.matrix(~Treatments)
modelMatrixMeth(design.treatments)
```

---

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 averages 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)
```

nbinomDeviance *Negative Binomial Deviance*

Description

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

Usage

```
nbinomDeviance(y, mean, dispersion=0, weights=NULL)
```

Arguments

y	numeric matrix containing the negative binomial counts, with rows for genes and columns for libraries. A vector will be treated as a matrix with one row.
mean	numeric matrix of expected values, of same dimension as y. A vector will be treated as a matrix with one row.
dispersion	numeric vector or matrix of negative binomial dispersions, as for glmFit. Can be a scalar, a vector of length equal to nrow(y), or a matrix of same dimensions as y.
weights	numeric vector or matrix of non-negative weights, as for glmFit. Can be a scalar, a vector of length equal to ncol(y), or a matrix of same dimensions as y.

Details

Computes the total residual deviance for each row of y, i.e., weighted row sums of the unit deviances.

Care is taken to ensure accurate computation in limiting cases when the dispersion is near zero or mean*dispersion is very large.

Value

nbinomDeviance returns a numeric vector of length equal to the number of rows of y.

Author(s)

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

References

Dunn PK, Smyth GK (2018). *Generalized linear models with examples in R*. Springer, New York, NY. doi:10.1007/9781441901187.

Jorgensen B (2013). Generalized linear models. *Encyclopedia of Environmetrics* 3, Wiley. doi:10.1002/9780470057339.vag010.pub2.

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. doi:10.1093/nar/gks042.

See Also

nbinomUnitDeviance

Examples

```
y <- matrix(1:6,3,2)
mu <- matrix(3,3,2)
nbinomDeviance(y,mu,dispersion=0.2)
```

---

nbinomUnitDeviance *Negative Binomial Unit Deviance*

Description

Compute unit deviances for the negative binomial distribution.

Usage

```
nbinomUnitDeviance(y, mean, dispersion = 0)
```

Arguments

y	vector or matrix of negative binomial counts.
mean	vector or matrix of means (expected values). If a matrix, then of same dimensions as y.
dispersion	negative binomial dispersions. Can be a single value, a vector of length nrow(y), or a matrix of same dimensions as y.

Details

The unit deviance of the negative binomial distribution is a measure of the distance between y and mean. If mean and dispersion are the true mean and dispersion of the negative binomial distribution, then the unit deviance follows an approximate chisquare distribution on 1 degree of freedom.

This function computes the unit deviance for each y observation. Care is taken to ensure accurate computation in limiting cases when the dispersion is near zero or mean*dispersion is very large.

Value

Numeric vector or matrix of the same size as y containing unit deviances.

Author(s)

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

References

Dunn PK, Smyth GK (2018). *Generalized linear models with examples in R*. Springer, New York, NY. doi:10.1007/9781441901187. ISBN: 978-1-4419-0118-7.

Jorgensen B (2013). Generalized linear models. *Encyclopedia of Environmetrics* 3, Wiley. doi:10.1002/9780470057339.vag010.pub2.

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. doi:10.1093/nar/gks042.

Examples

```
y <- 1:4
names(y) <- letters[1:4]
nbinomUnitDeviance(y,mean=2.5,dispersion=0.2)
```

`nearestReftoX` *Find Nearest Element of Reference for Each Element of X*

Description

Find nearest element of a sorted reference vector to each element of `x`.

Usage

```
nearestReftoX(x, reference, ...)
```

Arguments

`x`	numeric vector.
`reference`	numeric vector, sorted in increasing order.
`...`	other arguments as passed to `findInterval`.

Details

This function finds the element of a reference table (`reference`) that is closest to each element of an incoming vector (`x`).

The function is a simple wrapper for `findInterval` in the base package. It calls `findInterval` with `vec` equal to the mid-points between the reference values.

Value

Integer vector giving indices of elements of `reference`.

Author(s)

Gordon Smyth

See Also

`findInterval`

Examples

```
nearestReftoX(c(-10,0.5,0.6,2,3), reference = c(-1,0,2))
```

`nearestTSS` *Find Nearest Transcriptional Start Site*

Description

Find nearest TSS and distance to nearest TSS for a vector of chromosome loci.

Usage

```
nearestTSS(chr, locus, species="Hs")
```

Arguments

`chr`	character vector of chromosome names.
`locus`	integer or numeric vector of genomic loci, of same length as `chr`.
`species`	character string specifying the species. Possible values are `"Hs"` (human hg38), `"Mm"` (mouse mm10), `"Rn"` (rat), `"Dm"` (fly), `"Dr"` (zebra fish), `"Ce"` (worm), `"Bt"` (bovine), `"Gg"` (chicken), `"Mmu"` (rhesus), `"Cf"` (canine) or `"Pt"` (chimpanzee).

Details

This function takes a series of genomic loci, defined by a vector of chromosome names and a vector of genomic positions within the chromosomes, and finds the nearest transcriptional start site (TSS) for each locus. The chromosome names can be in the format `"1","2","X"` or can be `"chr1","chr2","chrX"`.

For genes with more than one annotated TSS, only the most 5' (upstream) of the alternative TSSs is reported.

This function uses the Bioconductor organism package named "org.XX.eg.db" where XX is `species`. Note that each organism package supports only a particular build of the genome for that species. For human (`species="Hs"`), the results are for the hg38 genome build. For mouse (`species="Mm"`), the results are for the mm10 genome build.

Value

A data.frame with the following columns:

`gene_id`	character vector giving the Entrez Gene ID of the nearest TSS for each element of `chr` and `locus`.
`symbol`	character vector of gene symbols.
`strand`	character vector with `"+"` for positive strand genes and `"-"` for negative strand genes.
`tss`	integer vector giving TSS.
`width`	integer vector giving genomic width of the gene.
`distance`	integer vector giving distance to nearest TSS. Positive values mean that the TSS is downstream of the locus, negative values mean that it is upstream. Gene body loci will therefore have negative distances and promoter loci will have positive.

Author(s)

Gordon Smyth

See Also

nearestReftoX

Examples

```
nearestTSS(chr = c("1","1"), locus = c(1000000,2000000))
```

---

normalizeBetweenArrays.DGEList

Apply Microarray Normalization to DGEList

Description

Applies microarray-style normalization to a DGEList by modifying the offset matrix.

Usage

```
normalizeBetweenArrays.DGEList(object,
     method = "cyclicloess", cyclic.method = "affy", ...)
```

Arguments

object	DGEList object.
method	character string specifying the normalization method to be used. Choices are "none", "scale", "quantile" or "cyclicloess".
cyclic.method	character string indicating the variant of normalizeCyclicLoess to be used if method=="cyclicloess", see normalizeCyclicLoess for possible values.
...	other arguments are passed to normalizeQuantiles or normalizeCyclicLoess.

Details

This function applies microarray-style normalization methods to a DGEList object by setting the offset matrix appropriately. The original counts remain unchanged.

The function uses cpm() to compute log2CPM values, then uses limma::normalizeBetweenArrays to apply the specified normalization method to the log2CPM values, then the difference between the normalized and unnormalized log2CPM matrices is transferred to the offset matrix. As usual, the offset entries represent observation-specific normalized loge library sizes.

This function is an alternative to normLibSizes. If normLibSizes has been called previously on the same object, then the results will be ignored.

Value

The input object is returned with an appropriately normalized offset component.

Author(s)

Gordon Smyth

See Also

normalizeBetweenArrays.

Examples

```
ngenes <- 100
nsamples <- 4
Counts <- matrix(rnbinom(ngenes*nsamples,mu=5,size=10),ngenes,nsamples)
rownames(Counts) <- 1:ngenes
colnames(Counts) <- paste0("S",1:4)
y <- DGEList(counts=Counts)
y <- normalizeBetweenArrays.DGEList(y)
head(y$offset)
```

---

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

Description

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

Usage

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

Arguments

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

Details

normalizeChIPtoInput identifies significant enrichment for a ChIP-Seq mark relative to input values. The ChIP-Seq mark might be for example transcriptional factor binding or an epigenetic mark. The function works on the data from one sample. Replicate libraries are not explicitly accounted for; this function can either be run on each sample individually or on a pool of replicates.

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.

When `plot=TRUE`, the genes shown in red are the marked genes (with Holm mid-p-value < 0.5) that have been removed as probably enriched during the fitting process. The normalization line has been fitted to the non-marked genes plotted in black.

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 modeling.

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

`normLibSizes` *Normalize Library Sizes*

Description

Calculate scaling factors to convert the raw library sizes for a set of sequenced samples into normalized effective library sizes.

Usage

```
## S3 method for class 'DGEList'
normLibSizes(object,
                method = c("TMM", "TMMwsp", "RLE", "upperquartile", "none"),
          refColumn = NULL, logratioTrim = .3, sumTrim = 0.05, doWeighting = TRUE,
                Acutoff = -1e10, p = 0.75, ...)
## S3 method for class 'SummarizedExperiment'
normLibSizes(object,
                method = c("TMM","TMMwsp","RLE","upperquartile","none"),
          refColumn = NULL, logratioTrim = .3, sumTrim = 0.05, doWeighting = TRUE,
                Acutoff = -1e10, p = 0.75, ...)
## Default S3 method:
normLibSizes(object, lib.size = NULL,
                method = c("TMM","TMMwsp","RLE","upperquartile","none"),
          refColumn = NULL, logratioTrim = .3, sumTrim = 0.05, doWeighting = TRUE,
                Acutoff = -1e10, p = 0.75, ...)
```

Arguments

object	a matrix of raw read counts or a DGEList object or a SummarizedExperiment object.
lib.size	numeric vector of library sizes corresponding to the columns of the matrix object.
method	normalization method to be used.
refColumn	column to use as reference for method="TMM". Can be a column number or a numeric vector of length nrow(object).
logratioTrim	the fraction (0 to 0.5) of observations to be trimmed from each tail of the distribution of log-ratios (M-values) before computing the mean. Used by method="TMM" for each pair of samples.
sumTrim	the fraction (0 to 0.5) of observations to be trimmed from each tail of the distribution of A-values before computing the mean. Used by method="TMM" for each pair of samples.
doWeighting	logical, whether to use (asymptotic binomial precision) weights when computing the mean M-values. Used by method="TMM" for each pair of samples.
Acutoff	minimum cutoff applied to A-values. Count pairs with lower A-values are ignored. Used by method="TMM" for each pair of samples.
p	numeric value between 0 and 1 specifying which quantile of the counts should be used by method="upperquartile".
...	other arguments are not currently used.

Details

This function computes scaling factors to convert observed library sizes into normalized library sizes, also called "effective library sizes". The effective library sizes for use in downstream analysis are lib.size * norm.factors where lib.size contains the original library sizes and norm.factors is the vector of scaling factors computed by this function.

The TMM method implements the trimmed mean of M-values method proposed by Robinson and Oshlack (2010). By default, the M-values are weighted according to inverse variances, as computed by the delta method for logarithms of binomial random variables. If refColumn is unspecified, then the column whose count-per-million upper quartile is closest to the mean upper quartile is set as the reference library.

The TMMwsp method stands for "TMM with singleton pairing". This is a variant of TMM that is intended to have more stable performance when the counts have a high proportion of zeros. In the TMM method, genes that have zero count in either library are ignored when comparing pairs of libraries. In the TMMwsp method, the positive counts from such genes are reused to increase the number of features by which the libraries are compared. The singleton positive counts are paired up between the libraries in decreasing order of size and then a slightly modified TMM method is applied to the re-ordered libraries. If refColumn is unspecified, then the column with largest sum of square-root counts is used as the reference library.

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

The upperquartile method 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 genes that are zero in all libraries. The idea is generalized here to allow normalization by any quantile of the count distributions.

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

For symmetry, normalization factors are adjusted to multiply to 1. Rows of object that have zero counts for all columns are removed before normalization factors are computed. The number of such rows does not affect the estimated normalization factors.

If object is a SummarizedExperiment object, then it is converted to a DGEList using SE2DGEList and the DGEList method applied.

Value

If object is a matrix, then the output is a vector with length ncol(object) giving the library scaling factors.

If object is a DGEList or SummarizedExperiment object, then the output is a DGEList the same as input with the library scaling factors stored as object$samples$norm.factors.

Note

normLibSizes become the new name for calcNormFactors in edgeR 3.42.0. The two functions are equivalent but calcNormFactors will eventually be retired.

Author(s)

Mark Robinson, Gordon Smyth, Yunshun Chen.

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.

Robinson MD, Oshlack A (2010). A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology* 11, R25.

See Also

getNormLibSizes, SE2DGEList.

Examples

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

# The TMM and TMMwsp methods give zero weight to the genes with the largest fold-changes:
y <- cbind(1,c(1,1,1,1,1,1,1,1,1,100))
normLibSizes(y, lib.size=c(1e6,1e6))

# normLibSizes makes the fold-changes for the majority of genes as close to 1 as possible:
# In this example, computing CPMs with raw library sizes makes the two samples look quite different.
dge <- DGEList(counts=y)
cpm(dge)
# By contrast, normalizing the library sizes makes most of the CPMs equal in the two samples:
dge <- normLibSizes(dge)
cpm(dge)
getNormLibSizes(dge)
```

PCList *Create a PCList Object*

Description

Assembles a PCList object from its components, especially the paired counts as a couple of matrices.

Usage

```
PCList(counts, counts2, samples = NULL, group = NULL, genes = NULL, ...)
```

Arguments

counts	numeric matrix containing sequence read counts, with rows corresponding to genes (genomic features) and columns to samples. Negative values or NAs are not allowed.
counts2	numeric matrix containing sequence read counts, with rows corresponding to genes (genomic features) and columns to samples. Negative values or NAs are not allowed.
samples	data.frame containing sample information, with a row for each sample. This data.frame will be appended to the samples component of the PCList object.
group	vector or factor giving the experimental group or treatment condition for each sample. Defaults to a single group.
genes	data.frame containing gene annotation.
...	other arguments are not currently used.

Details

Assembles a PCList object from its components. The only compulsory argument is the paired counts.

Value

A PCList object.

Author(s)

edgeR team. Originally created by Lizhong Chen.

See Also

PCList-class

Examples

```
ngenes <- 100
nsamples <- 4
Counts1 <- matrix(rnbinom(ngenes*nsamples,mu=5,size=10),ngenes,nsamples)
Counts2 <- matrix(rnbinom(ngenes*nsamples,mu=5,size=10),ngenes,nsamples)
rownames(Counts1) <- rownames(Counts2) <- 1:ngenes
colnames(Counts1) <- colnames(Counts2) <- paste0("S",1:4)
Group <- gl(2,2)
Genes <- data.frame(Symbol=paste0("Gene",1:ngenes))
y <- PCList(counts=Counts1, counts2=Counts2, group=Group, genes=Genes)
dim(y)
colnames(y)
y$samples
show(y)
```

PCList-class *Paired Counts Data Object - Class*

Description

A list-based S4 class for storing paired counts and associated information from sequencing technologies.

List Components

For objects of this class, rows correspond to genomic features and columns to samples. The genomic features can be genes, genomic loci, or any feature for which paired read counts are meaningful. Objects of this class contain the following essential list components:

`counts`: numeric matrix of read counts, one row for each feature and one column for each sample.

`counts2`: numeric matrix of paired read counts, one row for each feature and one column for each sample.

`samples`: data.frame with a row for each sample and column `group` containing the group labels. Other columns can be optionally added to give more detailed sample information.

Optional components include:

`genes`: data.frame giving annotation information for each feature. Same number of rows as `counts`.

`AveLogCPM`: numeric vector giving average log2 total counts per million for each feature.

Methods

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

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

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

`PCList` objects inherit a compact `show` method from the `LargeDataObject` class, so that printing produces a compact summary of their contents.

Author(s)

edgeR team. First created by Lizhong Chen.

See Also

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

---

plotBCV *Plot Biological Coefficient of Variation*

Description

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

Usage

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

Arguments

y	a DGEList object.
xlab	label for the x-axis.
ylab	label for the y-axis.
pch	the plotting symbol. See points for more details.
cex	plot symbol expansion factor. See points for more details.
col.common	color of line showing common dispersion.
col.trend	color of line showing dispersion trend.
col.tagwise	color of points showing genewise dispersions. Note that 'tag' and 'gene' are synonymous here.
...	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 genewise 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, colored by experimental group.

Usage

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

Arguments

`y`	either a matrix of exon-level counts, a list containing a matrix of counts for each exon or a `DGEList` object with (at least) elements `counts` (table of counts summarized at the exon level) and `samples` (data frame containing information about experimental group, library size and normalization factor for the library size). Each row of `y` should represent one exon.
`geneID`	character string giving the name of the gene for which exon usage is to be plotted.
`group`	factor supplying the experimental group/condition to which each sample (column of `y`) belongs. If `NULL` (default) the function will try to extract it 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 on 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")
```

---

plotMD.DGEList *Mean-Difference Plot of Count Data*

Description

Creates a mean-difference plot (aka MA plot) with color coding for highlighted points.

Usage

```
## S3 method for class 'DGEList'
plotMD(object, column=1, xlab="Average log CPM (this sample and others)",
       ylab="log-ratio (this sample vs others)",
       main=colnames(object)[column], status=object$genes$Status,
       zero.weights=FALSE, prior.count=3, ...)
## S3 method for class 'SummarizedExperiment'
plotMD(object, column=1, xlab="Average log CPM (this sample and others)",
     ylab="log-ratio (this sample vs others)", zero.weights=FALSE, prior.count=3, ...)
## S3 method for class 'DGEGLM'
plotMD(object, column=ncol(object), coef=NULL, xlab="Average log CPM",
       ylab="log-fold-change", main=colnames(object)[column],
       status=object$genes$Status, zero.weights=FALSE, ...)
## S3 method for class 'DGEBIN'
plotMD(object, column=ncol(object), coef=NULL, xlab="Average log CPM",
       ylab="log-odds-ratio", main=colnames(object)[column],
       status=object$genes$Status, zero.weights=FALSE, ...)
## S3 method for class 'DGELRT'
plotMD(object, xlab="Average log CPM", ylab="log-fold-change",
       main=object$comparison, status=object$genes$Status, contrast=1,
       adjust.method="BH", p.value=0.05, ...)
## S3 method for class 'DGEExact'
plotMD(object, xlab="Average log CPM", ylab="log-fold-change",
       main=NULL, status=object$genes$Status,
       adjust.method="BH", p.value=0.05, ...)
```

Arguments

object	an object of class DGEList, SummarizedExperiment, DGEGLM, DGEBIN, DGELRT or DGEExact.
column	integer, column of object to be plotted.
coef	alternative to column for fitted model objects. If specified, then column is ignored.
xlab	character string, label for x-axis.
ylab	character string, label for y-axis.
main	character string, title for plot.
status	vector giving the control status of each spot on the array, of same length as the number of rows of object. If NULL under the DGEList, SummarizedExperiment, DGEGLM or DGEBIN method, then all points are plotted in the default color, symbol and size. If NULL under the DGELRT or DGEExact method, then decideTests is run to determine the status of all the genes. The up-regulated DE genes are highlighted in red and down-regulated in blue.
zero.weights	logical, should spots with zero or negative weights be plotted?
prior.count	the average prior count to be added to each observation. Larger values produce more shrinkage.
contrast	integer specifying which log-fold-change to be plotted in the case of testing multiple contrasts. Only used for the DGELRT method with multiple contrasts.
adjust.method	character string passed to decideTests specifying p-value adjustment method. Only used when status is NULL. See decideTests for details.
p.value	numeric value between 0 and 1 giving the desired size of the test. Only used and passed to decideTests when status is NULL.
...	other arguments are passed to plotWithHighlights.

Details

A mean-difference plot (MD-plot) is a plot of log fold changes (differences) versus average log values (means). The history of mean-difference plots and MA-plots is reviewed in Ritchie et al (2015).

For DGEList and SummarizedExperiment objects, a between-sample MD-plot is produced. Counts are first converted to log2-CPM values. An artificial array is produced by averaging all the samples other than the sample specified. A mean-difference plot is then produced from the specified sample and the artificial sample. This procedure reduces to an ordinary mean-difference plot when there are just two arrays total.

If object is a DGEGLM object, then the plot is a fitted model MD-plot in which the estimated coefficient is on the y-axis and the average logCPM value is on the x-axis. If object is a DGEBIN object (a binomial fit from binFit or binQLFit), then the fitted log-odds-ratio coefficient is on the y-axis and the average logCPM of the total coverage is on the x-axis. If object is a DGEExact or DGELRT object, then the MD-plot displays the logFC vs the logCPM values from the results table.

The status vector can correspond to any grouping of the probes that is of interest. If object is a fitted model object, then the status vector is often used to indicate statistical significance, so that differentially expressed points are highlighted.

The status can be included as the component object$genes$Status instead of being passed as an argument to plotMD.

See plotWithHighlights for how to set colors and graphics parameters for the highlighted and non-highlighted points.

Value

A plot is created on the current graphics device.

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* Volume 43, e47. doi:10.1093/nar/gkv007.

See Also

`plotSmear`

The driver function for `plotMD` is `plotWithHighlights`.

`plotMDS.DGEList` *Multidimensional Scaling Plot of Distances between Digital Gene Expression Profiles*

Description

Plot samples on a two-dimensional scatterplot so that distances on the plot approximate the expression differences between the samples.

Usage

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

Arguments

`x`	a `DGEList` or `SummarizedExperiment` 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 it defaults to the index of the samples.
`pch`	plotting symbol or symbols. See `points` for possible values. Ignored if `labels` is non-`NULL`.
`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.
`gene.selection`	character, `"pairwise"` to choose the top genes separately for each pairwise comparison between the samples, or `"common"` to select the same genes for all comparisons. Only used when `method="logFC"`.
`xlab`	x-axis label.

ylab	y-axis label.
method	method used to compute distances. Possible values are "logFC" or "bcv". Note the "bcv" method is scheduled to be removed in a future version of edgeR.
prior.count	average prior count to be added to observation to shrink the estimated log-fold-changes towards zero. Only used when method="logFC".
plot	logical. If TRUE then a plot is created on the current graphics device.
var.explained	logical. If TRUE then the percentage variation explained is included in the axis labels.
...	any other arguments are passed to plot.

Details

The default method (method="logFC") is to convert the counts to log-counts-per-million using cpm and to pass these to the limma plotMDS function. This method calculates distances between samples based on log2 fold changes. See the plotMDS help page for details.

The alternative method (method="bcv") calculates distances based on biological coefficient of variation. A set of top genes is chosen that have the largest biological variation between the libraries (those with largest genewise 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. Beware that the "bcv" method is slow when the number of samples is large. The "bcv" method is in general much less used than "logFC" and is scheduled to be removed in a future version of edgeR.

The number of genes (top) 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 when the samples are distinguished by a specific focused molecular pathway. Very large values (greater than 1000) are not usually so effective.

Value

An object of class MDS is invisibly returned and (if plot=TRUE) a plot is created on the current graphics device.

Author(s)

Yunshun Chen, Mark Robinson and Gordon Smyth

See Also

plotMDS, cmdscale, as.dist

Examples

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

ngenes <- 1000
nlib <- 6
counts <- matrix(rnbinom(ngenes*nlib, size=1/10, mu=20),ngenes,nlib)
rownames(counts) <- paste("gene",1:ngenes, sep=".")
group <- gl(2,3,labels=c("Grp1","Grp2"))
```

```
counts[1:200,group=="Grp2"] <- counts[1:200,group=="Grp2"] + 10
y <- DGEList(counts,group=group)
y <- normLibSizes(y)

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

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

`plotMDS.PCList` *Multidimensional Scaling Plot of Distances between M-values from the Paired Count Data*

Description

Plot samples on a two-dimensional scatterplot so that distances on the plot approximate the M-values differences between the samples.

Usage

```
## S3 method for class 'PCList'
plotMDS(x, top = 500, labels = NULL, pch = NULL, cex = 1,
       dim.plot = c(1,2), gene.selection = "pairwise", xlab = NULL, ylab = NULL,
       prior.count = 2, plot = TRUE, var.explained = TRUE, ...)
```

Arguments

`x`	a `PCList` object.
`top`	number of top features used to calculate pairwise distances.
`labels`	character vector of sample names or labels. If `x` has no column names, then it defaults to the index of the samples.
`pch`	plotting symbol or symbols. See `points` for possible values. Ignored if `labels` is non-`NULL`.
`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.
`gene.selection`	character, `"pairwise"` to choose the top genes separately for each pairwise comparison between the samples, or `"common"` to select the same genes for all comparisons.
`xlab`	x-axis label.
`ylab`	y-axis label.
`prior.count`	average prior count added to each count before forming the M-values (log2 odds), to shrink the values for low counts towards zero.
`plot`	logical. If `TRUE` then a plot is created on the current graphics device.
`var.explained`	logical. If `TRUE` then the percentage variation explained is included in the axis labels.
`...`	any other arguments are passed to `plot`.

Details

The paired count data is converted to M-values (log2 odds) and passed to the limma plotMDS function. This method calculates distances between samples based on log2 odds ratios. See the plotMDS help page for details.

The number of features (top) chosen for this exercise should roughly correspond to the number of features with large log2 odds ratios. The default setting of 500 features is widely effective and suitable for routine use, but a smaller value might be chosen when the samples are distinguished by a specific focused molecular pathway. Very large values (greater than 1000) are not usually so effective.

Value

An object of class MDS is invisibly returned and (if plot=TRUE) a plot is created on the current graphics device.

Author(s)

Lizhong Chen and Gordon Smyth

See Also

plotMDS, cmdscale, as.dist

Examples

```
ngenes <- 100
Counts1 <- matrix(rpois(ngenes*4,lambda=10),ngenes,4)
Counts2 <- matrix(rpois(ngenes*4,lambda=10),ngenes,4)
y <- PCList(Counts1, Counts2, group=gl(2,2))
plotMDS(y)
```

---

plotMeanVar2 *Plot Mean-Variance Relationship in DGE Data Using Standardized Residuals*

Description

Group observations by size of the fitted value and plot average squared residual vs average fitted value.

Usage

```
plotMeanVar2(y, design = NULL, dispersion = 0, offset = 0, nbins = 100, make.plot = TRUE,
          xlab = "Mean", ylab = "Ave. binned standardized residual", ...)
```

Arguments

y	numeric matrix of counts, each row represents one gene, each column represents one DGE library.
design	numeric matrix giving the design matrix of the GLM. Assumed to be full column rank. Defaults to an intercept column.
dispersion	numeric scalar or vector giving the dispersion parameter for each GLM. Can be a scalar giving one value for all genes, or a vector of length equal to the number of genes giving genewise dispersions.
offset	numeric vector or matrix giving the offset that is to be included in the log-linear model predictor. Can be a vector of length equal to the number of libraries, or a matrix of the same size as y.
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".
...	other arguments are passed to plot.

Details

This function explores the mean-variance relationship in count data. The function fits a Poisson or NB GLM model to each gene using the appropriate design matrix and computes squared ordinary residuals. The residuals are standardized by the leverages but not by model variances. The fitted values are divided into nbins bins and the mean fitted value and mean squared residual are computed for each bin.

This function is similar in purpose to plotMeanVar but uses standardized residuals instead of pooled variances.

Value

Produces a mean-variance plot and returns a list with the following component:

mean	numeric vector of average fitted values for bins
var	numeric vector of the average squared residuals for bins

Author(s)

Davis McCarthy and Gordon Smyth

See Also

plotMeanVar.

plotMDS.DGEList, plotSmear, plotMD.DGEList and plotBCV provide other ways to visualize DGE data.

Examples

```
# Example with Poisson data
log2mu <- seq(from=0,to=10,length=1000)
y <- matrix(rpois(4*1000,lambda=2^log2mu),1000,4)
binned <- plotMeanVar2(y)
abline(0,1)
```

`plotQLDisp` *Plot the Quasi-Dispersion*

Description

Plot the genewise quasi-dispersion against the gene abundance (in log2 counts per million).

Usage

```
plotQLDisp(glmfit, xlab="Average Log2 CPM", ylab="Quarter-Root Mean Deviance", pch=16,
       cex=0.2, col.shrunk="red", col.trend="blue", col.raw="black", ...)
```

Arguments

`glmfit`	a `DGEGLM` object produced by `glmQLFit`.
`xlab`	label for the x-axis.
`ylab`	label for the y-axis.
`pch`	the plotting symbol. See `points` for more details.
`cex`	plot symbol expansion factor. See `points` for more details.
`col.shrunk`	color of the points representing the squeezed quasi-dispersions.
`col.trend`	color of line showing dispersion trend.
`col.raw`	color of points showing the unshrunk dispersions.
`...`	any other arguments are passed to `plot`.

Details

This function displays the quarter-root of the quasi-dispersions for all genes, before and after shrinkage towards a trend. If `glmfit` was constructed without an abundance trend, the function instead plots a horizontal line (of color `col.trend`) at the common value towards which dispersions are shrunk. The quarter-root transformation is applied to improve visibility for dispersions around unity.

Value

A plot is created on the current graphics device.

Author(s)

Aaron Lun, Davis McCarthy, Gordon Smyth, Yunshun Chen.

References

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

Examples

```
nbdisp <- 1/rchisq(1000, df=10)
y <- DGEList(matrix(rnbinom(6000, size = 1/nbdisp, mu = 10),1000,6))
design <- model.matrix(~factor(c(1,1,1,2,2,2)))
y <- estimateDisp(y, design)

fit <- glmQLFit(y, design)
plotQLDisp(fit)

fit <- glmQLFit(y, design, abundance.trend=FALSE)
plotQLDisp(fit)
```

---

plotSmear *Smear Plot*

Description

Make a mean-difference plot of two libraries of count data with smearing of points with very low counts, especially those that are zero for one of the columns.

Usage

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

Arguments

object	DGEList, DGEExact or DGELRT object containing data to produce an MA-plot.
pair	pair of experimental conditions to plot (if NULL, the first two conditions are used). Ignored if object is a DGELRT object.
de.tags	rownames for genes identified as being differentially expressed; use exactTest or glmLRT to identify DE genes. Note that 'tag' and 'gene' are synonymous here.
xlab	x-label of plot.
ylab	y-label of plot.
pch	plotting character. Can be a single value or a vector of length nrow(object). Default value of 19 gives a round point.
cex	character expansion factor, numerical value giving the amount by which plotting text and symbols should be magnified relative to the default; default cex=0.2 to make the plotted points smaller.
smearWidth	width of the smear.
panel.first	an expression to be evaluated after the plot axes are set up but before any plotting takes place; the default grid() draws a background grid to aid interpretation of the plot.

smooth.scatter	logical, whether to produce a 'smooth scatter' plot using the KernSmooth::smoothScatter function or just a regular scatter plot; default is FALSE, i.e. produce a regular scatter plot.
lowess	logical, indicating whether or not to add a lowess curve to the MA-plot to give an indication of any trend in the log-fold change with log-concentration.
...	further arguments passed on to plot.

Details

plotSmear produces a type of mean-difference plot (or MA plot) with a special representation (smearing) of log-ratios that are infinite. plotSmear resolves the problem of plotting genes that have a total count of zero for one of the groups by adding the 'smear' of points at a 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) genes.

Value

Invisibly returns the x and y coordinates of the plotted points, and a plot is created on the current device.

Author(s)

Mark Robinson created the original concept of smearing the infinite log-fold-changes.

See Also

maPlot, plotMD.DGEList

Examples

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

# find differential expression
de <- exactTest(d)

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

---

plotSpliceDGE *Differential Splicing Plot*

Description

Plot relative log-fold changes by exons for the specified gene and highlight the significantly spliced exons.

Usage

```
plotSpliceDGE(lrt, geneid=NULL, genecolname=NULL, rank=1L, FDR=0.05)
```

Arguments

lrt	DGELRT object produced by diffSpliceDGE.
geneid	character string, ID of the gene to plot.
genecolname	column name of lrt$genes containing gene IDs. Defaults to lrt$genecolname.
rank	integer, if geneid=NULL then this ranked gene will be plotted.
FDR	numeric, mark exons with false discovery rate less than this cutoff.

Details

Plot relative log2-fold-changes by exon for the specified gene. The relative logFC is the difference between the exon's logFC and the overall logFC for the gene, as computed by diffSpliceDGE. The significantly spliced individual exons are highlighted as red dots. The size of the red dots is weighted by their significance.

Value

A plot is created on the current graphics device.

Author(s)

Yunshun Chen, Yifang Hu and Gordon Smyth

See Also

diffSpliceDGE, topSpliceDGE.

---

predFC *Predictive Log-Fold Changes*

Description

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

Usage

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

Arguments

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

Details

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

Specifically, the function adds a small prior count to each observation before fitting the GLM (see addPriorCount for details). The actual prior count that is added is proportional to the library size. This has the effect that any log-fold-change that was zero prior to augmentation remains zero and non-zero log-fold-changes are shrunk towards zero.

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

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

Value

Numeric matrix of (shrunk) linear model coefficients on the log2 scale.

Author(s)

Belinda Phipson and Gordon Smyth

References

Phipson B (2013). *Empirical Bayes modelling of expression profiles and their associations*. PhD Thesis. University of Melbourne, Australia. http://hdl.handle.net/11343/38162.

See Also

glmFit, exactTest, addPriorCount

Examples

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

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

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

---

processAmplicons *Process FASTQ Files from Pooled Genetic Sequencing Screens*

Description

Given a list of sample-specific index (barcode) sequences and hairpin/sgRNA-specific sequences from an amplicon sequencing screen, generate a DGEList of counts from the raw FASTQ files containing the sequence reads. The position of the index sequences and hairpin/sgRNA sequences is considered variable, with the hairpin/sgRNA sequences assumed to be located after the index sequences in the read.

Usage

```
processAmplicons(readfile, readfile2 = NULL, barcodefile, hairpinfile,
                 allowMismatch = FALSE, barcodeMismatchBase = 1,
                 hairpinMismatchBase = 2, dualIndexForwardRead = FALSE,
                 verbose = FALSE, barcodesInHeader = FALSE,
                 hairpinBeforeBarcode = FALSE, plotPositions = FALSE)
```

Arguments

readfile	character vector giving one or more FASTQ filenames. Must be uncompressed text files.
readfile2	optional character vector giving one or more FASTQ filenames for reverse read. Must be uncompressed text files.
barcodefile	filename containing sample-specific barcode IDs and sequences. File may be gzipped.
hairpinfile	filename containing hairpin/sgRNA-specific IDs and sequences. File may be gzipped.
allowMismatch	logical, indicates whether sequence mismatch is allowed.
barcodeMismatchBase	maximum number of base sequence mismatches allowed in a barcode sequence when allowMismatch is TRUE.

`hairpinMismatchBase`
: maximum number of base sequence mismatches allowed in a hairpin/sgRNA sequence when `allowMismatch` is `TRUE`.

`dualIndexForwardRead`
: logical, indicates if forward reads contain a second barcode sequence (must be present in `barcodefile`) that should be matched.

`verbose`
: if `TRUE`, output program progress.

`barcodesInHeader`
: logical, indicates if barcode sequences should be matched in the header (sequence identifier) of each read (i.e. the first of every group of four lines in the FASTQ files).

`hairpinBeforeBarcode`
: logical, indicates that hairpin sequences appear before barcode sequences in each read. Setting this to TRUE will allow hairpins to exist anywhere in a read irrespective of barcode position. Leaving as FALSE will improve performance if hairpins are positioned after barcode sequences.

`plotPositions`
: logical, indicates if a density plot displaying the position of each barcode and hairpin/sgRNA sequence in the reads should be created. If `dualIndexForwardRead` is `TRUE` or `readfile2` is not `NULL`, plotPositions will generate two density plots, side by side, indicating the positions of the first barcodes and hairpins in the first plot, and second barcodes in the second.

Details

The `processAmplicons` function allows for hairpins/sgRNAs/sample index sequences to be in variable positions within each read.

The input barcode file and hairpin/sgRNA files are tab-separated text files with at least two columns (named 'ID' and 'Sequences') containing the sample or hairpin/sgRNA IDs and a second column indicating the sample index or hairpin/sgRNA sequences to be matched. If `dualIndexForwardRead` is `TRUE`, a third column 'Sequences2' is expected in the barcode file. If `readfile2` is specified, another column 'SequencesReverse' is expected in the barcode file. The barcode file may also contain a 'group' column that indicates which experimental group a sample belongs to. Additional columns in each file will be included in the respective `$samples` or `$genes` data.frames of the final `DGEList` object. These files, along with the FASTQ files, are assumed to be in the current working directory.

To compute the count matrix, matching to the given barcodes and hairpins/sgRNAs is conducted in two rounds. The first round looks for an exact sequence match for the given barcode sequences and hairpin/sgRNA sequences through the entire read, returning the first match found. If a match isn't found, the program performs a second round of matching which allows for sequence mismatches if `allowMismatch` is set to `TRUE`. The maximum number of mismatch bases in barcode and hairpin/sgRNA is specified by the parameters `barcodeMismatchBase` and `hairpinMismatchBase` respectively.

The program outputs a `DGEList` object, with a count matrix indicating the number of times each barcode and hairpin/sgRNA combination could be matched in reads from the input FASTQ files.

For further examples and data, refer to the case studies available from https://bioinf.wehi.edu.au/shRNAseq/.

The argument `hairpinBeforeBarcode` was introduced in edgeR release version 3.38.2 and developmental version 3.39.3. Previously the function expected the sequences in the FASTQ files to have a fixed structure as per Figure 1A of Dai et al (2014). The revised function can process reads where the hairpins/sgRNAs/sample index sequences are in variable positions within each read. When

`plotPositions=TRUE` a density plot of the match positions is created to allow the user to assess whether they occur in the expected positions.

Value

A `DGEList` object with following components:

`counts`	read count matrix tallying up the number of reads with particular barcode and hairpin/sgRNA matches. Each row is a hairpin/sgRNA and each column is a sample.
`genes`	In this case, hairpin/sgRNA-specific information (ID, sequences, corresponding target gene) may be recorded in this data.frame.
`lib.size`	auto-calculated column sum of the counts matrix.

Note

This function replaced the earlier function `processHairpinReads` in edgeR 3.7.17.

Author(s)

Oliver Voogd, Zhiyin Dai, Shian Su and Matthew Ritchie

References

Dai Z, Sheridan JM, Gearing LJ, Moore DL, Su S, Wormald S, Wilcox S, O'Connor L, Dickins RA, Blewitt ME, Ritchie ME (2014). edgeR: a versatile tool for the analysis of shRNA-seq and CRISPR-Cas9 genetic screens. *F1000Research* 3, 95. doi:10.12688/f1000research.3928.2.

---

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

Description

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

Usage

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

Arguments

`x`	numeric matrix of counts.
`input.mean`	population means for `x`. Can be a vector of length `nrow(x)` or a matrix of same dimensions as `x`.
`output.mean`	population means for the output values. Can be a vector of length `nrow(x)` or a matrix of same dimensions as `x`.
`dispersion`	negative binomial dispersion values. Can be a single value, or a vector of length `nrow(x)`, or a matrix of same dimensions as `x`.

Details

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

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

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

Value

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

Author(s)

Gordon Smyth

See Also

equalizeLibSizes

Examples

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

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

---

read10X *Read 10X Genomics Files*

Description

Reads 10X Genomics files containing single-cell RNA-seq UMI counts in Matrix Market format.

Usage

```
read10X(mtx = NULL, genes = NULL, barcodes = NULL, path = ".", DGEList = TRUE)
```

Arguments

mtx	name of mtx file containing counts in Matrix Exchange Format. Defaults to matrix.mtx or matrix.mtx.gz.
genes	name of file containing gene IDs and names. Defaults to features.tsv or genes.tsv or gzipped versions of the same.
barcodes	optional name of file containing barcodes. Defaults to "barcodes.tsv" or barcodes.tsv.gz.

path	character string giving the directory containing the files. Defaults to the current working directory.
DGEList	logical. If TRUE, a DGEList will be returned, otherwise an unclassed list is returned.

Details

This function reads output files created by the 10X Genomics Cellranger pipeline, see https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/matrices. The UMI counts are assembled into an integer matrix in R with accompanying gene IDs and gene symbols. The results are returned as either a DGEList or an ordinary list.

The files mtx, genes and barcodes can be provided in either gzipped or unzipped versions.

This function creates an ordinary matrix of counts. To read the counts instead into a sparse matrix format, the read10xResults function in the scater package is an alternative.

Value

Either a DGEList object (if DGEList=TRUE) or an ordinary list with the following components:

counts	matrix of counts.
genes	data.frame containing gene symbols.
samples	data.frame containing information about each cell. This will be omitted if barcodes=NULL and DGEList=FALSE.

The only difference between the DGEList and list formats is that the DGEList adds some extra columns to the samples data.frame.

Author(s)

Gordon Smyth

See Also

read10xResults in the scater package.

Examples

```
## Not run:
GEO <- "ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2510nnn/GSM2510617/suppl/"
GEOmtx <- paste0(GEO,"GSM2510617_P7-matrix.mtx.gz")
GEOgenes <- paste0(GEO,"GSM2510617_P7-genes.tsv.gz")
download.file(GEOmtx,"matrix.mtx.gz")
download.file(GEOgenes,"genes.tsv.gz")
y <- read10X("matrix.mtx.gz", "genes.tsv.gz")

## End(Not run)
```

`readBismark2DGE` *Read Bismark Coverage Files*

Description

Read Bismark coverage files containing methylated and unmethylated read counts for CpG loci and create DGEList.

Usage

```
readBismark2DGE(files, sample.names=NULL, readr=TRUE, verbose=TRUE)
```

Arguments

`files`	character vector of file names.
`sample.names`	character vector of sample names. If `NULL`, sample names will be extracted from the file names.
`readr`	logical. If `TRUE`, `readr` package is used to read the coverage files, otherwise `read.delim` is used.
`verbose`	logical. If `TRUE`, read progress messages are sent to standard output.

Details

This function reads tab-delimited coverage files output by Bismark software. Counts from multiple files are collated into a `DGEList` object.

Value

A `DGEList` object with a row for each unique genomic locus found in the files and two columns (containing methylated and unmethylated counts) for each sample.

Note

This function represents genomic loci as integers, so the largest locus position must be less than the maximum integer in R (about `2e9`). The number of chromosomes times the largest locus position must be less than `1e16`.

Author(s)

Gordon Smyth

References

Chen Y, Pal B, Visvader JE, Smyth GK (2017). Differential methylation analysis of reduced representation bisulfite sequencing experiments using edgeR. *F1000Research* 6, 2055. doi:10.12688/f1000research.13196.2.

`readBismark2PC` *Read Bismark Coverage Files*

Description

Read Bismark coverage files containing methylated and unmethylated read counts for CpG loci and create a PCList.

Usage

```
readBismark2PC(files, sample.names=NULL, readr=TRUE, verbose=TRUE)
```

Arguments

`files`	character vector of file names.
`sample.names`	character vector of sample names. If `NULL`, sample names will be extracted from the file names.
`readr`	logical. If `TRUE`, `readr` package is used to read the coverage files, otherwise `read.delim` is used.
`verbose`	logical. If `TRUE`, read progress messages are sent to standard output.

Details

This function reads tab-delimited coverage files output by Bismark software. Counts from multiple files are collated into a `PCList` object.

Value

A `PCList` object with a row for each unique genomic locus found in the files and two count matrices (containing methylated and unmethylated counts) for each sample.

Note

This function represents genomic loci as integers, so the largest locus position must be less than the maximum integer in R (about `2e9`). The number of chromosomes times the largest locus position must be less than `1e16`.

Author(s)

Gordon Smyth

References

Chen Y, Pal B, Visvader JE, Smyth GK (2017). Differential methylation analysis of reduced representation bisulfite sequencing experiments using edgeR. *F1000Research* 6, 2055. doi:10.12688/f1000research.13196.2.

Examples

```
# Two small Bismark coverage files: chr, start, end, meth percentage, count_meth, count_unmeth
tab <- intToUtf8(9)
f1 <- tempfile(fileext=".cov")
f2 <- tempfile(fileext=".cov")
writeLines(paste("chr1", c(10,20), c(10,20), c(50,80), c(5,8), c(5,2), sep=tab), f1)
writeLines(paste("chr1", c(10,30), c(10,30), c(60,70), c(6,7), c(4,3), sep=tab), f2)
y <- readBismark2PC(c(f1,f2), readr=FALSE, verbose=FALSE)
dim(y)
y$counts
```

---

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

Description

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

Usage

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

Arguments

files	character vector of filenames, or a data.frame of sample information containing a column called files.
path	character string giving the directory containing the files. Defaults to the current working directory.
columns	numeric vector stating which columns of the input files contain the gene names and counts respectively.
group	optional vector or factor indicating the experimental group to which each file belongs.
labels	character vector giving short names to associate with the files. Defaults to the file names.
...	other arguments are passed to read.delim.

Details

Each file is assumed to contain digital gene expression data for one genomic sample or count library, with gene identifiers in the first column and counts in the second column. Gene identifiers are assumed to be unique and not repeated in any one file. The function creates a combined table of counts with rows for genes and columns for samples. A count of zero will be entered for any gene that was not found in any particular sample.

By default, the files are assumed to be tab-delimited and to contain column headings. Other file formats can be handled by adding arguments to be passed to read.delim. For example, use header=FALSE if there are no column headings and use sep="," to read a comma-separated file.

Instead of being a vector, the argument files can be a data.frame containing all the necessary sample information. In that case, the filenames and group identifiers can be given as columns files and group respectively, and the labels can be given as the row.names of the data.frame.

Value

A DGEList object containing a matrix of counts, with a row for each unique tag found in the input files and a column for each input file.

Author(s)

Mark Robinson and Gordon Smyth

See Also

See read.delim for other possible arguments that can be accepted.

DGEList-class, DGEList.

Examples

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

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

---

roast.DGEGLM *Self-contained Gene Set Tests for Digital Gene Expression Data*

Description

Rotation gene set testing for Negative Binomial generalized linear models.

Usage

```
## S3 method for class 'DGEGLM'
fry(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
      sort = "directional", ...)

## S3 method for class 'DGEGLM'
roast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
      set.statistic = "mean", gene.weights = NULL, nrot = 1999, ...)

## S3 method for class 'DGEGLM'
mroast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
       set.statistic = "mean", gene.weights = NULL, nrot = 1999,
       adjust.method = "BH", midp = TRUE, sort = "directional", ...)
```

Arguments

y	DGEGLM object.
index	index vector specifying which rows (probes) of y are in the test set. Can be a vector of integer indices, or a logical vector of length nrow(y), or a vector of gene IDs corresponding to entries in geneid. Alternatively it can be a data.frame with the first column containing the index vector and the second column containing directional gene weights. For mroast or fry, index is a list of index vectors or a list of data.frames.

design	the design matrix. Defaults to y$design or, if that is NULL, to model.matrix(~y$samples$group).
contrast	contrast for which the test is required. Can be an integer specifying a column of design, or the name of a column of design, or a numeric contrast vector of length equal to the number of columns of design.
geneid	gene identifiers corresponding to the rows of y. Can be either a vector of length nrow(y) or the name of the column of y$genes containing the gene identifiers. Defaults to rownames(y).
set.statistic	summary set statistic. Possibilities are "mean","floormean","mean50" or "msq".
gene.weights	numeric vector of directional (positive or negative) genewise weights. For mroast or fry, this vector must have length equal to nrow(y). For roast, can be of length nrow(y) or of length equal to the number of genes in the test set.
nrot	number of rotations used to compute the p-values.
adjust.method	method used to adjust the p-values for multiple testing. See p.adjust for possible values.
midp	logical, should mid-p-values be used instead of ordinary p-values when adjusting for multiple testing?
sort	character, whether to sort output table by directional p-value ("directional"), non-directional p-value ("mixed"), or not at all ("none").
...	other arguments are currently ignored.

Details

These functions perform self-contained gene set tests against the null hypothesis that none of the genes in the set are differentially expressed. fry is the recommended function in the edgeR context.

The roast gene set test was proposed by Wu et al (2010) for microarray data and the roast and mroast methods documented here extend the test to digital gene expression data. The roast method uses residual space rotations instead of permutations to obtain p-values, a technique that takes advantage of the full generality of linear models. The negative binomial count data is converted to approximate normal deviates by computing mid-p quantile residuals (Dunn and Smyth, 1996; Routledge, 1994) under the null hypothesis that the contrast is zero, and the normal deviates are then passed to the limma roast function. See roast for more description of the test and for a complete list of possible arguments. mroast is similar but performs roast tests for multiple gene sets instead of just one.

The fry method documented here similarly generalizes the fry gene set test for microarray data. fry is recommended over roast or mroast for count data because, in this context, it is equivalent to mroast but with an infinite number of rotations.

Value

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

mroast and fry produce a data.frame. See mroast for details.

Author(s)

Yunshun Chen and Gordon Smyth

References

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *J. Comput. Graph. Statist.*, 5, 236-244. https://gksmyth.github.io/pubs/residual.html.

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

Wu D, Lim E, Vaillant F, Asselin-Labat ML, Visvader JE, Smyth GK (2010). ROAST: rotation gene set tests for complex microarray experiments. *Bioinformatics* 26, 2176-2182. doi:10.1093/bioinformatics/btq401.

See Also

roast, camera.DGEGLM

Examples

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

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

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

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

# Estimate dispersions
fit <- glmQLFit(y, design, legacy=FALSE)

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

---

roast.DGEList *Self-contained Gene Set Tests for Digital Gene Expression Data*

Description

Rotation gene set testing for Negative Binomial generalized linear models.

Usage

```
## S3 method for class 'DGEList'
fry(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
      sort = "directional", ...)

## S3 method for class 'DGEList'
roast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
```

```
      set.statistic = "mean", gene.weights = NULL, nrot = 1999, ...)

## S3 method for class 'DGEList'
mroast(y, index = NULL, design = NULL, contrast = ncol(design), geneid = NULL,
       set.statistic = "mean", gene.weights = NULL, nrot = 1999,
       adjust.method = "BH", midp = TRUE, sort = "directional", ...)
```

Arguments

y	DGEList object.
index	index vector specifying which rows (probes) of y are in the test set. Can be a vector of integer indices, or a logical vector of length nrow(y), or a vector of gene IDs corresponding to entries in geneid. Alternatively it can be a data.frame with the first column containing the index vector and the second column containing directional gene weights. For mroast or fry, index is a list of index vectors or a list of data.frames.
design	the design matrix. Defaults to y$design or, failing that, to model.matrix(~y$samples$group).
contrast	contrast for which the test is required. Can be an integer specifying a column of design, or the name of a column of design, or a numeric contrast vector of length equal to the number of columns of design.
geneid	gene identifiers corresponding to the rows of y. Can be either a vector of length nrow(y) or the name of the column of y$genes containing the gene identifiers. Defaults to rownames(y).
set.statistic	summary set statistic. Possibilities are "mean","floormean","mean50" or "msq".
gene.weights	numeric vector of directional (positive or negative) genewise weights. For mroast or fry, this vector must have length equal to nrow(y). For roast, can be of length nrow(y) or of length equal to the number of genes in the test set.
nrot	number of rotations used to compute the p-values.
adjust.method	method used to adjust the p-values for multiple testing. See p.adjust for possible values.
midp	logical, should mid-p-values be used instead of ordinary p-values when adjusting for multiple testing?
sort	character, whether to sort output table by directional p-value ("directional"), non-directional p-value ("mixed"), or not at all ("none").
...	other arguments are currently ignored.

Details

These functions perform self-contained gene set tests against the null hypothesis that none of the genes in the set are differentially expressed. fry is the recommended function in the edgeR context.

The roast gene set test was proposed by Wu et al (2010) for microarray data and the roast and mroast methods documented here extend the test to digital gene expression data. The roast method uses residual space rotations instead of permutations to obtain p-values, a technique that takes advantage of the full generality of linear models. The negative binomial count data is converted to approximate normal deviates by computing mid-p quantile residuals (Dunn and Smyth, 1996; Routledge, 1994) under the null hypothesis that the contrast is zero, and the normal deviates are then passed to the limma roast function. See roast for more description of the test and for a complete list of possible arguments. mroast is similar but performs roast tests for multiple gene sets instead of just one.

The fry method documented here similarly generalizes the fry gene set test for microarray data. fry is recommended over roast or mroast for count data because, in this context, it is equivalent to mroast but with an infinite number of rotations.

Value

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

mroast and fry produce a data.frame. See mroast for details.

Author(s)

Yunshun Chen and Gordon Smyth

References

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *J. Comput. Graph. Statist.*, 5, 236-244. https://gksmyth.github.io/pubs/residual.html.

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

Wu D, Lim E, Vaillant F, Asselin-Labat ML, Visvader JE, Smyth GK (2010). ROAST: rotation gene set tests for complex microarray experiments. *Bioinformatics* 26, 2176-2182. doi:10.1093/bioinformatics/btq401.

See Also

roast, camera.DGEList

Examples

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

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

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

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

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

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

romer.DGEGLM *Rotation Gene Set Enrichment for Digital Gene Expression Data*

Description

Romer gene set enrichment tests for Negative Binomial generalized linear models.

Usage

```
## S3 method for class 'DGEGLM'
romer(y, index, design=NULL, contrast=ncol(design), ...)
```

Arguments

y	DGEGLM object.
index	list of indices specifying the rows of y in the gene sets. The list can be made using ids2indices.
design	design matrix. Defaults to y$design.
contrast	contrast for which the test is required. Can be an integer specifying a column of design, or the name of a column of design, or else a contrast vector of length equal to the number of columns of design.
...	other arguments are passed to romer.default. For example, the number of rotations nrot can be increased from the default of 9999 to increase the resolution of the p-values.

Details

The ROMER procedure described by Majewski et al (2010) is implemented in romer in the limma package. This romer method for DGEGLM objects makes the romer procedure available for count data such as RNA-seq data. The negative binomial count data is converted to approximate normal deviates by computing mid-p quantile residuals (Dunn and Smyth, 1996; Routledge, 1994) under the null hypothesis that the contrast is zero. The normal deviates are then passed to the romer function in limma. See romer for more description of the test and for a complete list of possible arguments.

Value

Numeric matrix giving p-values and the number of matched genes in each gene set. Rows correspond to gene sets. There are four columns giving the number of genes in the set and p-values for the alternative hypotheses up, down or mixed. See romer for details.

Author(s)

Yunshun Chen and Gordon Smyth

References

Majewski IJ, Ritchie ME, Phipson B, Corbin J, Pakusch M, Ebert A, Busslinger M, Koseki H, Hu Y, Smyth GK, Alexander WS, Hilton DJ, Blewitt ME (2010). Opposing roles of polycomb repressive complexes in hematopoietic stem and progenitor cells. *Blood*, 116, 731-719. doi:10.1182/blood-200912260760.

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *J. Comput. Graph. Statist.*, 5, 236-244. https://gksmyth.github.io/pubs/residual.html.

Routledge RD (1994). Practicing safe statistics with the mid-p. *Canadian Journal of Statistics* 22, 103-110. doi:10.2307/3315826.

See Also

romer

Examples

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

# First set of 10 genes that are genuinely differentially expressed
iset1 <- 1:10
mu[iset1,3:4] <- mu[iset1,3:4]+20

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

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

# Estimate dispersions
fit <- glmQLFit(y, design, legacy=FALSE)

romer(fit, iset1, design, contrast=2)
romer(fit, iset2, design, contrast=2)
romer(fit, list(set1=iset1, set2=iset2), design, contrast=2)
```

---

romer.DGEList *Rotation Gene Set Enrichment for Digital Gene Expression Data*

Description

Romer gene set enrichment tests for Negative Binomial generalized linear models.

Usage

```
## S3 method for class 'DGEList'
romer(y, index, design=NULL, contrast=ncol(design), ...)
```

Arguments

y	DGEList object.
index	list of indices specifying the rows of y in the gene sets. The list can be made using ids2indices.
design	design matrix. Defaults to y$design or, failing that, to model.matrix(~y$samples$group).
contrast	contrast for which the test is required. Can be an integer specifying a column of design, or the name of a column of design, or else a contrast vector of length equal to the number of columns of design.
...	other arguments are passed to romer.default. For example, the number of rotations nrot can be increased from the default of 9999 to increase the resolution of the p-values.

Details

The ROMER procedure described by Majewski et al (2010) is implemented in romer in the limma package. This romer method for DGEList objects makes the romer procedure available for count data such as RNA-seq data. The negative binomial count data is converted to approximate normal deviates by computing mid-p quantile residuals (Dunn and Smyth, 1996; Routledge, 1994) under the null hypothesis that the contrast is zero. The normal deviates are then passed to the romer function in limma. See romer for more description of the test and for a complete list of possible arguments.

Value

Numeric matrix giving p-values and the number of matched genes in each gene set. Rows correspond to gene sets. There are four columns giving the number of genes in the set and p-values for the alternative hypotheses up, down or mixed. See romer for details.

Author(s)

Yunshun Chen and Gordon Smyth

References

Majewski IJ, Ritchie ME, Phipson B, Corbin J, Pakusch M, Ebert A, Busslinger M, Koseki H, Hu Y, Smyth GK, Alexander WS, Hilton DJ, Blewitt ME (2010). Opposing roles of polycomb repressive complexes in hematopoietic stem and progenitor cells. *Blood*, 116, 731-719. doi:10.1182/blood-200912260760.

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *J. Comput. Graph. Statist.*, 5, 236-244. https://gksmyth.github.io/pubs/residual.html.

Routledge RD (1994). Practicing safe statistics with the mid-p. *Canadian Journal of Statistics* 22, 103-110. doi:10.2307/3315826.

See Also

romer

Examples

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

# First set of 10 genes that are genuinely differentially expressed
iset1 <- 1:10
mu[iset1,3:4] <- mu[iset1,3:4]+20

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

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

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

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

---

rowsum *Sum over Groups of Genes*

Description

Condense the rows of a DGEList or PCList object so that counts are summed over specified groups of genes.

Usage

```
## S3 method for class 'DGEList'
rowsum(x, group, reorder=FALSE, na.rm=FALSE, ...)
## S3 method for class 'PCList'
rowsum(x, group, reorder=FALSE, na.rm=FALSE, ...)
## S3 method for class 'SummarizedExperiment'
rowsum(x, group, reorder=FALSE, na.rm=FALSE, ...)
```

Arguments

x	a DGEList, a PCList object or a SummarizedExperiment object.
group	a vector or factor giving the grouping, with one element per row of x. Missing values will be treated as another group and a warning will be given.
reorder	if TRUE, then the row.names of the resulting DGEList will be in order of sort(unique(group)); if FALSE, they will be in the order that groups were encountered.
na.rm	logical (TRUE or FALSE). Should NA (including NaN) values be discarded?
...	other arguments are not currently used.

Details

If x is a SummarizedExperiment object, it is first converted into a DGEList object.

A new DGEList or PCList object is computed, with the same columns as x, but for which the rows correspond to the unique values of group. The counts for rows with the same group value are summed.

Columns of x$genes will be retained in the output if they contain group-level annotation. Columns that vary within groups will be dropped.

Value

DGEList or PCList object with the same number of columns as x and rows corresponding to the unique values of group.

Author(s)

Lizhong, Gordon Smyth

See Also

rowsum in the base package.

Examples

```
x <- DGEList(counts=matrix(1:8,4,2))
rowsum(x, group=c("A","A","B","B"))
```

---

sampleWeights *Empirical Sample Quality Weights*

Description

Estimate relative quality weights for each sample from the unit deviances and unit residual df output by glmQLFit.

Usage

```
sampleWeights(glmfit)
```

Arguments

glmfit	a DGEGLM object (from glmQLFit) or a DGEBIN object (from binQLFit). The number of OpenMP threads is taken from glmfit$nthreads.

Details

Estimates an empirical quality weight for each sample from the matrix of adjusted unit deviances and adjusted unit df from glmQLFit (a DGEGLM object) or binQLFit (a DGEBIN object), both with legacy=FALSE. The quasi-dispersion of each observation is assumed to be equal to s2_g / w_i.

The returned sample weights can be entered directly as the weights argument for subsequent glmQLFit or binQLFit fits.

Value

A CompressedMatrix with the same dimensions as glmfit$counts holding the per-sample weights, repeated across all genes (by row). The per-sample weights are the distinct column values.

Author(s)

Gordon Smyth

See Also

This function uses output from glmQLFit and is analogous to arrayWeights in the limma package. The voomWithQualityWeights and voomLmFit functions also estimate sample weights for count data.

Examples

```
# Sample 2 has higher dispersion and lower weight
y <- matrix(rpois(1000*5,lambda=8),1000,5)
y[,2] <- 2 * y[,2]
fit <- glmQLFit(y)
w <- sampleWeights(fit)
barplot(as.matrix(w)[1,], main="Sample weights")
```

---

scaleOffset *Scale Offsets*

Description

Ensures scale of offsets is consistent with library sizes.

Usage

```
## S3 method for class 'DGEList'
scaleOffset(y, offset, ...)
## Default S3 method:
scaleOffset(y, offset, ...)
```

Arguments

y	numeric vector or matrix of counts, or a DGEList object.
offset	numeric vector or matrix of offsets to be scaled. If a vector, its length must equal the length of y or the number of columns of y. If a matrix, its dimension must equal the dimension of y.
...	other arguments that are not currently used.

Details

scaleOffset ensures that the scale of offsets is consistent with library sizes. This is done by ensuring that the mean offset for each gene is the same as the mean log-library size. The length or dimensions of offset should be consistent with the number of libraries in y.

Value

scaleOffset.default returns a numeric vector if offset is a vector, a matrix if offset is a matrix or a CompressedMatrix object if offset is a CompressedMatrix. scaleOffset.DGEList computes the scaled offsets and stores them in the offset component of the input DGEList object.

Author(s)

Aaron Lun, Yunshun Chen

Examples

```
y <- matrix(rnbinom(40,size=1,mu=100),10,4)
offset <- rnorm(4)
scaleOffset(y, offset)
```

SE2DGEList *SummarizedExperiment to DGEList*

Description

Given any SummarizedExperiment data object, extract basic information needed and convert it into a DGEList object.

Usage

```
SE2DGEList(object)
```

Arguments

object	a SummarizedExperiment object. Must have counts in its assay component.

Details

This function takes a SummarizedExperiment data object as input. The counts of the assay component of the input SummarizedExperiment data object is extracted and used as the counts component of the output DGEList object. The rowRanges or rowData of the input is converted into a data.frame and used as genes in the output. The colData of the input is also converted into a data.frame and used as the sample information in the output.

Value

A DGEList object.

Author(s)

Yunshun Chen and Gordon Smyth

Examples

```
## Not run:
library(SummarizedExperiment)
example(SummarizedExperiment)
y <- SE2DGEList(se)

## End(Not run)
```

Seurat2PB *Seurat or SeuratObject Class Object to Pseudo-Bulk DGEList*

Description

Given a Seurat or SeuratObject data object, create pseudo-bulk samples using the sample and cluster information and return a DGEList object.

Usage

```
Seurat2PB(object, sample, cluster="seurat_clusters", assay="RNA")
```

Arguments

object	a Seurat or SeuratObject class data object. Must have counts in its assay.
sample	character specifying the column of object@meta.data that contains sample information of the single cell data. Must be one of the column names of object@meta.data.
cluster	character specifying the column of object@meta.data that contains single cell cluster information. Must be one of the column names of object@meta.data. Defaults to seurat_clusters.
assay	character specifying which assay of the object the counts are extracted from. Defaults to RNA.

Details

This function takes a Seurat or SeuratObject data object as input. It is assumed that the input data object contains raw RNA-seq read counts of multiple samples as well as cell clustering information (usually from a single cell integration analysis). The counts of the RNA assay of the input data object is first extracted. A pseudo-bulk count matrix is formed by aggregating the read counts of all the cells within the same cluster for each sample. The pseudo-bulk count matrix is then used as the counts component of the output DGEList object. The cluster and sample information of the pseudo-bulk counts is stored in the samples component. The row names of the input data object, together with other extra feature information (if any) found in the input, are stored in the genes component.

Value

A DGEList object.

Author(s)

Yunshun Chen

Examples

```
## Not run:
ngenes <- 1e3
ncells <- 1e3
sp <- paste0("sample", sample(1:2, ncells, replace=TRUE))
clst <- sample(1:3, ncells, replace=TRUE)

counts <- matrix(rnbinom(ngenes*ncells, mu=10, size=2), ngenes, ncells)
colnames(counts) <- paste0("Cell",1:ncells)
rownames(counts) <- paste0("Gene",1:ngenes)
so <- CreateSeuratObject(counts = counts)
so@meta.data <- cbind(so@meta.data, sample=sp, cluster=clst)

y <- Seurat2PB(so, sample="sample", cluster="cluster")

## End(Not run)
```

`spliceVariants` *Identify Genes with Splice Variants (Classic Pipeline)*

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`	negative binomial dispersion. Single numeric value.
`group`	factor supplying the experimental group/condition to which each sample (column of `y`) belongs. If `NULL` (default) the function will try to extract it 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 in 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

The newer function `diffSplice.DGEGLM` is now recommended.

`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)
```

---

splitGencodeTxNames *Split Gencode Transcript Names*

Description

Split delimited transcript names provided by Gencode FASTA files into separate columns.

Usage

```
splitGencodeTxNames(x, remove.version.numbers = TRUE)
```

Arguments

x	character vector of Gencode transcript names.
remove.version.numbers	logical, whether to remove version numbers from transcript and gene IDs.

Details

Gencode transcript names consist of different annotations pasted together with | delimiters. Here we split the annotations into separate columns of a matrix.

Value

A character matrix with the following columns:

EnsT	Ensembl transcript ID, without version number suffix.
EnsG	Ensembl gene ID, without version number suffix.
HavG	Havana gene ID.
HavT	Havana transcript ID.
TxName	Transcript name.
GeneName	Gene name.
AnnLength	Transcript length.
Type	Type of transcript.

Author(s)

Gordon Smyth

See Also

catchSalmon, strsplit2.

---

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 group.

Usage

```
## S3 method for class 'DGEList'
splitIntoGroups(y, ...)
## Default S3 method:
splitIntoGroups(y, group=NULL, ...)
splitIntoGroupsPseudo(pseudo, group, pair)
```

Arguments

y	matrix of counts or a DGEList object.
group	vector or factor giving the experimental group/condition for each library.
pseudo	numeric matrix of quantile-adjusted pseudocounts to be split.
pair	vector of length two stating pair of groups to be split for the pseudocounts.
...	other arguments that are not currently used.

Value

splitIntoGroups outputs a list in which each element is a matrix of 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$counts) <- paste("gene", 1:nrow(d$counts), sep=".")
z1 <- splitIntoGroups(d)
z2 <- splitIntoGroupsPseudo(d$counts, d$group, pair=c(1,2))
```

---

subsetting *Subset DGEList, PCList, DGEGLM, DGEExact and DGELRT Objects*

Description

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

Usage

```
## S3 method for class 'DGEList'
object[i, j, keep.lib.sizes=TRUE]
## S3 method for class 'PCList'
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]
## S3 method for class 'TopTags'
object[i, j]
```

Arguments

object	object of class DGEList, PCList, DGEGLM, DGEExact or DGELRT. For subsetListOfArrays, any list of conformal matrices and vectors.
i, j	elements to extract. i subsets the genes while j subsets the libraries. Note that columns of DGEGLM, DGEExact and DGELRT objects cannot be subsetted.
keep.lib.sizes	logical, if TRUE the lib.sizes will be kept unchanged on output, otherwise they will be recomputed as the column sums of the counts of the remaining rows.

Details

i,j may take any values acceptable for the matrix components of object of class DGEList and PCList. 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 the same class as object holding data from the specified subset of rows and columns.

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
```

---

sumTechReps *Sum over Replicate Samples*

Description

Condense the columns of a matrix or DGEList object so that counts are summed over technical replicate samples.

Usage

```
## Default S3 method:
sumTechReps(x, ID=colnames(x), ...)
## S3 method for class 'DGEList'
sumTechReps(x, ID=colnames(x), ...)
## S3 method for class 'SummarizedExperiment'
sumTechReps(x, ID, ...)
```

Arguments

x	a numeric matrix or DGEList object.
ID	sample identifier.
...	other arguments are not currently used.

Details

A new matrix or DGEList object is computed in which the counts for technical replicate samples are replaced by their sums.

Value

A data object of the same class as x with a column for each unique value of ID. For a SummarizedExperiment object, it is converted into a DGEList object. Columns are in the same order as the ID values first occur in the ID vector.

Author(s)

Gordon Smyth and Yifang Hu

See Also

rowsum.

Examples

```
x <- matrix(rpois(8*3,lambda=5),8,3)
colnames(x) <- c("a","a","b")
sumTechReps(x)
```

---

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	the expected proportion of the events to keep. Either a unit vector or of the same length as x.
target.size	the desired total column counts. Either a unit vector or of the same length as NCOL{x}. Must not be greater than the 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)
```

---

topSpliceDGE *Top Table of Differentially Spliced Genes or Exons*

Description

Top table ranking the most differentially spliced genes or exons.

Usage

```
topSpliceDGE(lrt, test="Simes", number=10, FDR=1)
```

Arguments

lrt	DGELRT object produced by diffSpliceDGE.
test	character string, possible values are "Simes", "gene" or "exon". "exon" gives exon-level tests for each exon. "gene" gives gene-level tests for each gene. "Simes" gives genewise p-values derived from the exon-level tests after Simes adjustment for each gene.
number	integer, maximum number of rows to output.
FDR	numeric, only show exons or genes with false discovery rate less than this cutoff.

Details

Ranks genes or exons by evidence for differential splicing. The exon-level tests test for differences between each exon and all the exons for the same gene. The gene-level tests test for any differences in exon usage between experimental conditions.

The Simes method processes the exon-level p-values to give an overall call of differential splicing for each gene. It returns the minimum Simes-adjusted p-value for each gene.

The gene-level tests are likely to be powerful for genes in which several exons are differentially spliced. The Simes p-values are likely to be more powerful when only a minority of the exons for a gene are differentially spliced. The exon-level tests are not recommended for formal error rate control.

Value

A data.frame with any annotation columns found in `lrt` plus the following columns

`NExons`	number of exons if `test="Simes"` or `"gene"`
`Gene.Exon`	exon annotation if `test="exon"`
`logFC`	log-fold change of one exon vs all the exons for the same gene (if `test="exon"`)
`exon.LR`	LR-statistics for exons (if `test="exon"` and the object for `diffSpliceDGE` was produced by `glmFit`)
`exon.F`	F-statistics for exons (if `test="exon"` and the object for `diffSpliceDGE` was produced by `glmQLFit`)
`gene.LR`	LR-statistics for genes (if `test="gene"` and the object for `diffSpliceDGE` was produced by `glmFit`)
`gene.F`	F-statistics for genes (if `test="gene"` and the object for `diffSpliceDGE` was produced by `glmQLFit`)
`P.Value`	p-value
`FDR`	false discovery rate

Author(s)

Yunshun Chen and Gordon Smyth

See Also

`diffSpliceDGE`.

---

topTags *Table of the Top Differentially Expressed Genes/Tags*

Description

Extracts the most differentially expressed genes (or sequence tags) from a test object, ranked either by p-value or by absolute log-fold-change.

Usage

```
topTags(object, n = 10, adjust.method = "BH", sort.by = "PValue", p.value = 1)
```

Arguments

`object`	a `DGEExact` or `DGELRT` object containing test statistics and p-values. Usually created by `exactTest`, `glmLRT`, `glmTreat` or `glmQLFTest`.
`n`	integer, maximum number of genes/tags to return.
`adjust.method`	character string specifying the method used to adjust p-values for multiple testing. See `p.adjust` for possible values.
`sort.by`	character string specifying the sort method. Possibilities are `"PValue"` for p-value, `"logFC"` for absolute log-fold change or `"none"` for no sorting.
`p.value`	numeric cutoff value for adjusted p-values. Only tags with adjusted p-values equal or lower than specified are returned.

Details

This function is closely analogous to the topTable function in the limma package. It accepts a test statistic object created by any of the edgeR functions exactTest, glmLRT, glmTreat or glmQLFTest and extracts a readable data.frame of the most differentially expressed genes. The data.frame collates the annotation and differential expression statistics for the top genes. The data.frame is wrapped in a TopTags output object that records the test statistic used and the multiple testing adjustment method.

TopTags objects will return dimensions and hence functions such as dim, nrow or ncol are defined on them. TopTags objects also have a show method so that printing produces a compact summary of their contents.

topTags permits ranking by fold-change but the authors do not recommend fold-change ranking or fold-change cutoffs for routine RNA-seq analysis. The p-value ranking is intended to be more biologically meaningful, especially if the p-values were computed using glmTreat.

Value

An object of class TopTags, which is a list-based class with the following components:

table	a data.frame containing differential expression results for the top genes in sorted order. The number of rows is the smaller of n and the number of genes with adjusted p-value less than or equal to p.value. The data.frame includes all the annotation columns from object$genes and all statistic columns from object$table plus one of:

FDR:	false discovery rate (only when adjust.method is "BH", "BY" or "fdr")
FWER:	family-wise error rate (only when adjust.method is "holm", "hochberg", "hommel" or "bonferroni").

adjust.method	character string specifying the method used to adjust p-values for multiple testing, same as input argument.
comparison	character vector giving the names of the two groups being compared (for DGEExact objects) or the glm contrast being tested (for DGELRT objects).
test	character string stating the name of the test.

Note

The terms 'tag' and 'gene' are used synonymously on this page and refer to the rows of object. In general, the rows might be genes, sequence tags, transcripts, exons or whatever type of genomic feature is appropriate for the analysis at hand.

Author(s)

Mark Robinson, Davis McCarthy, Yunshun Chen, Gordon Smyth

References

Chen Y, Lun ATL, Smyth GK (2016). From reads to genes to pathways: differential expression analysis of RNA-Seq experiments using Rsubread and the edgeR quasi-likelihood pipeline. *F1000Research* 5, 1438. doi:10.12688/f1000research.8987.2.

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. doi:10.1093/nar/gks042.

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, glmTreat, glmQLFTest, dim.TopTags, p.adjust.

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$counts) <- paste("gene",1:nrow(d$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 genes to view
tp <- topTags(de, n=15)
# Here we view top 15
tp
# Or order by fold change instead
topTags(de,sort.by="logFC")
```

---

tpm *Transcripts per Million*

Description

Compute transcripts per million (TPM) from counts or from a fitted model object.

Usage

```
tpm(y, effective.tx.length, rta.overdispersion=NULL, shrunk=FALSE, ...)
```

Arguments

y	any object that can be input to cpm with rows corresponding to transcripts. Typically a DGEList containing divided transcript counts or a DGEGLM fitted model object fitted to divided transcript counts.
effective.tx.length	numeric vector giving effective transcript lengths.
rta.overdispersion	numeric vector giving overdispersions used to compute divided counts.
shrunk	logical, if y is a DGEGLM fitted model object, should shrunk or unshrunk coefficients be used? Default is unshrunk.
...	other arguments are passed to cpm.

Details

This function computes transcripts per million (TPM) by rescaling counts-per-million at the transcript level by the effective transcript length and by any count scaling factor. The function will typically be run as part of the DTU analysis after running `catchSalmon`, dividing the transcript counts by the RTA-overdispersion, and running `glmQLFit` on the divided counts. The `tpm` function can then extract TPM values from the estimated model coefficients.

Note that the TPM values do not sum exactly to one million for each sample. The total TPM varies from sample to sample because the effective library sizes estimated by `normLibSizes` imply that the total RNA production may differ from sample to sample. The TPM values preserve the same fold-changes between samples and conditions as obtained from `cpm`.

Value

A numeric matrix of TPM values.

Author(s)

Gordon Smyth

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *bioRxiv* doi:10.1101/2025.04.07.647659.

See Also

`cpm`, `diffSplice.DGEGLM`

---

tpmProp *TPM Proportion by Transcript for Each Gene*

Description

Compute the expression proportion from each transcript for each gene.

Usage

```
tpmProp(tpm, geneid)
```

Arguments

`tpm`	numeric matrix of TPM values, with rows corresponding to transcripts.
`geneid`	character vector or factor of Gene IDs.

Details

Convert a matrix of TPMs to expression proportions for each gene.

Value

A numeric matrix of TPM proportions, adding to 1 for each gene in each sample.

Author(s)

Gordon Smyth

References

Baldoni PL, Chen L, Li M, Chen Y, Smyth GK (2025). Dividing out quantification uncertainty enables assessment of differential transcript usage with limma and edgeR. *bioRxiv* doi:10.1101/2025.04.07.647659.

See Also

tpm

`validDGEList` *Check for Valid DGEList Object*

Description

Check for existence of standard components of DGEList object.

Usage

```
validDGEList(y)
```

Arguments

y	DGEList object.

Details

This function checks that the standard `counts` and `samples` components of a `DGEList` object are present.

Value

`DGEList` with missing components added.

Author(s)

Gordon Smyth

See Also

DGEList

Examples

```
counts <- matrix(rpois(4*2,lambda=5),4,2)
dge <- new("DGEList", list(counts=counts))
validDGEList(dge)
```

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 gene, maximized to find the smoothed (moderated) estimate of the dispersion parameter

Usage

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

Arguments

y	list with elements comprising the matrices of count data (or pseudocounts) for the different groups.
delta	delta (phi / (phi+1))parameter of negative binomial.
tag	gene at which the weighted conditional log-likelihood is evaluated.
prior.n	smoothing parameter that indicates the weight to put on the common likelihood compared to the individual gene's likelihood; default 10 means that the common likelihood is given 10 times the weight of the individual gene's likelihood in the estimation of the genewise dispersion.
ntags	number of genes in the dataset to be analyzed.
der	which derivative to evaluate, either 0 (the function), 1 (first derivative) or 2 (second derivative).

Details

This function computes the weighted conditional log-likelihood for a given gene, 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 gene. The delta scale is used for convenience (delta is bounded between 0 and 1). Users should note that 'tag' and 'gene' are synonymous when interpreting the names of the arguments for this function.

Value

numeric value giving the function or derivative evaluated for the given gene and delta.

Author(s)

Mark Robinson, Davis McCarthy

Examples

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

WLEB *Weighted Likelihood Empirical Bayes*

Description

Compute empirical Bayes moderated parameter estimators using a weighted likelihood approach.

Usage

```
WLEB(theta, loglik, prior.n = 5,
     covariate = NULL, trend.method = "locfit", span = NULL, legacy.span = FALSE,
     overall = TRUE, trend = TRUE, individual = TRUE, m0 = NULL, m0.out = FALSE)
```

Arguments

theta	numeric vector of values of the parameter at which the log-likelihoods are calculated.
loglik	numeric matrix of log-likelihood of all the candidates at those values of parameter.
prior.n	the weight put on the common likelihood compared to the individual's likelihood, expressed in terms of the number of observations in a notional prior dataset.
covariate	numeric vector of values across which a parameter trend is fitted.
trend.method	method for estimating the parameter trend. Possible values are "none", "movingave", "loess", "locfit" or "locfit.mixed". The latter options cause movingAverageByCol(), loessBycol() and locfitByCol() respectively to be called to smooth loglik by the covariate values. The "locfit.mixed" method is the same as "locfit" but uses a polynomial of degree 1 for lowly expressed genes.
span	width of the smoothing window, as a proportion of the data set. By default, is chosen by chooseLowessSpan().
legacy.span	logical, if TRUE the old default for span will be used.
overall	logical, should a single value of the parameter which maximizes the sum of all the log-likelihoods be estimated?
trend	logical, should a parameter trend (against the covariate) which maximizes the local shared log-likelihoods be estimated?
individual	logical, should individual estimates of all the candidates after applying empirical Bayes method along the trend be estimated?
m0	numeric matrix of local shared log-likelihoods. If NULL, it will be calculated using the method selected by trend.method.
m0.out	logical, should local shared log-likelihoods be included in the output?

Details

This function implements a very general empirical Bayes strategy outlined by McCarthy et al (2012). McCarthy et al used the method to estimate genewise negative binomial dispersion parameters, but here the method is generalized to apply to any parameter and any likelihood function. The method gives similar results to parametric empirical Bayes with a conjugate prior, when a conjugate prior exists, but does not require the prior distribution to be specified. The prior distribution

is instead inferred from the pooled likelihood, i.e., from the likelihood that would arise from pooling all the cases or from pooling all the cases with similar covariate values.

The function assumes a series of cases. Each case leads to a data set from which a parameter (theta) is to be estimated. For each case, the log-likelihood function has been evaluated over a grid of possible values for theta. The function takes as input the matrix of log-likelihood values where the rows correspond to cases and the columns correspond to putative parameter values.

Each case is associated with a covariate value that might affect theta. The "overall" parameter estimate is the maximum likelihood estimator of theta that arises if the likelihood is averaged over cases and then maximized over theta. The "trend" parameter estimates are estimates for theta that arise if each column of `loglik` is replaced by a smooth trend with respect to the covariate. The "individual" parameter estimate for each case is a compromise between the maximum likelihood estimate for that case alone and a global parameter estimate computed from all the cases, the latter being either the overall estimate (if `trend.method="none"`) or the trend estimate (otherwise).

Value

A list with the following:

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

Note

The default value of span was changed in August 2025 from `chooseLowessSpan(length(theta), small.n=50, min.span=0.25, power=0.5)` to `chooseLowessSpan(length(theta), small.n=50, min.span=0.3, power=1/3)`. Setting `legacy=TRUE` will cause the old default to be used instead of the new, for backward compatibility.

Author(s)

Yunshun Chen, Gordon Smyth

References

McCarthy DJ, Chen Y, Smyth GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* 40, 4288-4297. doi:10.1093/nar/gks042.

See Also

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

Examples

```
y <- matrix(rpois(100, lambda=10), ncol=4)
theta <- 7:14
loglik <- matrix(0,nrow=nrow(y),ncol=length(theta))
for(i in 1:nrow(y))
for(j in 1:length(theta))
```

```
 loglik[i,j] <- sum(dpois(y[i,], theta[j] ,log=TRUE))
covariate <- log(rowSums(y))
out <- WLEB(theta, loglik, prior.n=3, covariate)
out
```

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

Description

Compute z-score equivalents of negative binomial random deviates.

Usage

```
zscoreNBinom(q, size, mu, method = "midp")
```

Arguments

`q`	numeric vector or matrix of non-negative quantiles.
`size`	numeric vector of non-negative size parameters.
`mu`	numeric vector of means.
`method`	method for converting from discrete to continuous distribution. Possible values are `"midp"` or `"random"`.

Details

The mid-p method (`method=="midp"`) applies a continuity correction by splitting the probability mass of each integer in two. It computes the mid-p tail probability 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.

The randomized method (`method=="random"`) computes randomized quantile residuals (Dunn and Smyth, 1996). In this method, the tail probabilities are randomized over the possible values covered by the discrete integer counts. If `q` follows a negative binomial distribution with the size and mean correctly specified, then the z-values generated by the randomized method are exactly standard normal.

Non-integer values of `q` are allowed. The mid-p method handles non-integer values by interpolation while the randomized method rounds `q` to integers.

Value

Numeric vector or matrix giving equivalent quantiles from the standard normal distribution.

Author(s)

Gordon Smyth

References

Berry G, Armitage P (1995). Mid-P confidence intervals: a brief review. *The Statistician*, 417-423.

Dunn PK, Smyth GK (1996). Randomized quantile residuals. *Journal of Computational and Graphical Statistics* **5**, 1-10. https://gksmyth.github.io/pubs/residual.html.

See Also

pnbinom, qnorm in the stats package.

Examples

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

Index

∗ **Data exploration**
cpm, 30
gini, 85
plotMDS.DGEList, 135
plotMDS.PCList, 137
plotSmear, 141
thinCounts, 172
∗ **Differential exon usage**
diffSpliceDGE, 47
plotExonUsage, 132
plotSpliceDGE, 142
spliceVariants, 166
∗ **Differential expression**
binFit, 12
binomTest, 14
binQLFit, 15
binQLFTest, 17
decideTests, 33
exactTest, 76
glmFit, 86
glmLRT, 88
glmQLFit, 90
glmQLFTest, 93
∗ **Differential methylation**
modelMatrixMeth, 118
readBismark2DGE, 150
readBismark2PC, 151
∗ **Differential transcript usage**
diffSplice.DGEGLM, 44
tpm, 176
tpmProp, 177
∗ **Differential usage**
diffSplice.DGEGLM, 44
featureCounts2DGEList, 79
tpm, 176
tpmProp, 177
∗ **Dispersion estimation**
adjustedProfileLik, 7
commonCondLogLikDerDelta, 28
condLogLikDerSize, 29
dispCoxReidInterpolateTagwise, 55
dispCoxReidSplineTrend, 57
estimateCommonDisp, 61
estimateDisp, 63
estimateExonGenewiseDisp, 65
estimateGLMCommonDisp, 66
estimateGLMRobustDisp, 68
estimateGLMTagwiseDisp, 69
estimateGLMTrendedDisp, 71
estimateTagwiseDisp, 73
estimateTrendedDisp, 75
getPriorN, 84
glmQLFit, 90
gof, 100
meanvar, 112
plotBCV, 131
plotMeanVar2, 138
plotQLDisp, 140
weightedCondLogLikDerDelta, 179
WLEB, 180
∗ **Documentation**
edgeR-package, 4
edgeRUsersGuide, 59
∗ **Gene set testing**
goana.DGELRT, 98
roast.DGEGLM, 153
roast.DGEList, 155
romer.DGEGLM, 158
romer.DGEList, 159
∗ **Model fit**
binFit, 12
binQLFit, 15
binQLFTest, 17
glmFit, 86
glmLRT, 88
glmQLFit, 90
glmQLFTest, 93
maPlot, 107
mbin, 111
mglm, 115
nbinomDeviance, 120
nbinomUnitDeviance, 121
plotMD.DGEList, 133
plotSmear, 141
predFC, 143
∗ **Normalization**

equalizeLibSizes, 60
getNormLibSizes, 83
goodTuring, 101
normalizeChIPtoInput, 125
normLibSizes, 126
q2qnbinom, 147
scaleOffset, 163
∗ **Reading data files**
catchSalmon, 21
catchSalmonGene, 24
DGEListFromTximport, 41
read10X, 148
readBismark2DGE, 150
readBismark2PC, 151
readDGE, 152
splitGencodeTxNames, 167
∗ **Transcript expression**
catchSalmon, 21
catchSalmonGene, 24
splitGencodeTxNames, 167
∗ **differential transcript usage**
featureCounts2DGEList, 79
∗ **edgeR classes**
as.data.frame, 9
as.matrix, 10
cbind, 27
DGEBIN-class, 35
DGEExact-class, 36
DGEGLM-class, 37
DGEList, 38
DGEList-class, 40
DGEListFromTximport, 41
DGELRT-class, 43
dim, 49
dimnames, 50
featureCounts2DGEList, 79
getCounts, 82
head, 103
makeCompressedMatrix, 105
PCList, 129
PCList-class, 130
SE2DGEList, 164
Seurat2PB, 165
subsetting, 169
validDGEList, 178
∗ **normalization**
normalizeBetweenArrays.DGEList, 124
∗ **rna-seq**
tpm, 176
tpmProp, 177
[.CompressedMatrix *(makeCompressedMatrix)*, 105
[.DGEBIN *(subsetting)*, 169
[.DGEExact *(subsetting)*, 169
[.DGEGLM *(subsetting)*, 169
[.DGELRT *(subsetting)*, 169
[.DGEList *(subsetting)*, 169
[.PCList *(subsetting)*, 169
[.TopTags *(subsetting)*, 169
[<-.CompressedMatrix *(makeCompressedMatrix)*, 105

addPriorCount, 6, *11, 12*, *144*
adjustedProfileLik, 7
arrayWeights, *163*
as.data.frame, *9*, 9
as.dist, *136*, *138*
as.matrix, *10*, 10, *107*
as.matrix.CompressedMatrix *(makeCompressedMatrix)*, 105
as.matrix.DGEList, *85*
aveLogCPM, *6, 7*, 10, *32*

binFit, 12, *16, 17*, *35, 36*, *112*, *134*
binMeanVar *(meanvar)*, 112
binom.test, *14, 15*
binomTest, 14, *78*
binQLFit, *13*, 15, *17, 18*, *35, 36*, *134*
binQLFTest, *13*, *16, 17*, 17, *35, 36*

calcNormFactors *(normLibSizes)*, 126
calcNormOffsetsforChIP *(normalizeChIPtoInput)*, 125
camera, 19, *20*
camera.DGEGLM, *155*
camera.DGEList, *157*
cameraPR.DGELRT *(camera)*, 19
catchKallisto, *27*, *42*
catchKallisto *(catchSalmon)*, 21
catchKallistoGene *(catchSalmonGene)*, 24
catchOarfish, *42*
catchOarfish *(catchSalmon)*, 21
catchRSEM, *42*
catchRSEM *(catchSalmon)*, 21
catchSalmon, 21, *27*, *42*, *168*
catchSalmonGene, *24*, 24
cbind, *27*, 27, *28*
cbind.CompressedMatrix *(makeCompressedMatrix)*, 105
cmdscale, *136*, *138*
commonCondLogLikDerDelta, 28
CompressedMatrix, *79*
CompressedMatrix *(makeCompressedMatrix)*, 105

CompressedMatrix-class (makeCompressedMatrix), 105
condLogLikDerDelta (condLogLikDerSize), 29
condLogLikDerSize, 29
cpm, *6*, *7*, *12*, 30, *177*
cpmByGroup (cpm), 30
cut, *33*
cutWithMinN, 32, *57*

decideTests, 33, *34*, *134*
designAsFactor (mglm), 115
DGEBIN-class, 35
DGEExact, *174*
DGEExact-class, 36
DGEGLM-class, 37
DGEList, *22*, *26*, *28*, 38, *39*, *41*, *42*, *76*, *80*, *85*, *146*, *147*, *149*, *150*, *153*, *167*, *178*
DGEList-class, 40
DGEListFromTximeta (DGEListFromTximport), 41
DGEListFromTximport, 41
DGELRT, *174*
DGELRT-class, 43
diffSplice, *46*
diffSplice.DGEGLM, 44, *49*, *167*, *177*
diffSpliceDGE, *46*, 47, *143*, *174*
dim, 49, *50*
dim.CompressedMatrix (makeCompressedMatrix), 105
dim.DGEBIN, *36*
dim.DGEExact, *37*
dim.DGEGLM, *38*
dim.DGEList, *40*, *130*
dim.DGELRT, *43*
dim.TopTags, *176*
dimnames, 50, *51*
dimnames.DGEBIN, *36*
dimnames.DGEExact, *37*
dimnames.DGEGLM, *38*
dimnames.DGEList, *40*, *130*
dimnames.DGELRT, *43*
dimnames<-.DGEBIN (dimnames), 50
dimnames<-.DGEExact (dimnames), 50
dimnames<-.DGEGLM (dimnames), 50
dimnames<-.DGEList (dimnames), 50
dimnames<-.DGELRT (dimnames), 50
dimnames<-.PCList (dimnames), 50
dispBinTrend, 51, *72*
dispCoxReid, 53, *67*
dispCoxReidInterpolateTagwise, 55, *70*
dispCoxReidPowerTrend, *72*
dispCoxReidPowerTrend (dispCoxReidSplineTrend), 57
dispCoxReidSplineTrend, 57, *72*
dispDeviance, *67*
dispDeviance (dispCoxReid), 53
dispPearson, *67*
dispPearson (dispCoxReid), 53
dropEmptyLevels, 58

edgeR (edgeR-package), 4
edgeR-package, 4
edgeRUsersGuide, 59
equalizeLibSizes, 60, *62*, *77*, *78*, *148*
estimateCommonDisp, *29*, 61, *65–67*, *71*, *74*, *75*, *167*
estimateDisp, 63, *100*
estimateExonGenewiseDisp, 65, *167*
estimateGLMCommonDisp, *54*, *65*, 66, *71*
estimateGLMRobustDisp, 68
estimateGLMTagwiseDisp, *56*, *65*, *67–69*, 69, *84*, *100*
estimateGLMTrendedDisp, *53*, *58*, *65*, *67–69*, *71*, 71
estimateTagwiseDisp, *62*, *65*, *67*, *71*, 73, *84*
estimateTrendedDisp, *62*, 75
exactTest, 76, *117*, *144*, *176*
exactTestBetaApprox (exactTest), 76
exactTestByDeviance (exactTest), 76
exactTestBySmallP (exactTest), 76
exactTestDoubleTail (exactTest), 76
expandAsMatrix, 78, *106*, *107*
Extract, *170*

factor, *58*
featureCounts2DGEList, 79
filterByExpr, 80
findInterval, *122*
fry.DGEGLM (roast.DGEGLM), 153
fry.DGEList (roast.DGEList), 155

getCounts, 82
getDispersion (getCounts), 82
getNormLibSizes, 83, *128*
getOffset (getCounts), 82
getPriorN, 84
gini, 85
glmFit, *7*, *8*, *53*, *64*, *67*, *68*, *70–72*, 86, *89*, *91*, *92*, *101*, *117*, *120*, *144*
glmLRT, *35*, 88, *94*, *176*
glmQLFit, *16*, *17*, 90, *94*, *95*, *140*, *163*
glmQLFTest, *18*, *35*, *93*, 93, *176*
glmTreat, *35*, 95, *176*
goana, *99*

goana.default, *98*
goana.DGEExact *(*goana.DGELRT*)*, 98
goana.DGELRT, 98
gof, 100
goodTuring, 101
goodTuringPlot *(*goodTuring*)*, 101
goodTuringProportions *(*goodTuring*)*, 101

head, *103*, 103
head.EList, *103*

ids2indices, *19*, *158*, *160*

kegga, *99*
kegga.default, *98*
kegga.DGEExact *(*goana.DGELRT*)*, 98
kegga.DGELRT *(*goana.DGELRT*)*, 98

length.CompressedMatrix
*(*makeCompressedMatrix*)*, 105
locfitByCol, *181*
locfitByCol *(*loessByCol*)*, 104
loess, *104*
loessByCol, *74*, 104, *181*

makeCompressedMatrix, 105
maPlot, 107, *114*, *142*
maximizeInterpolant, *56*, 109, *110*
maximizeQuadratic, 110
mbin, 111
mbinIWLS, *13*, *16*
mbinIWLS *(*mbin*)*, 111
mbinOneWay, *13*, *16*
mbinOneWay *(*mbin*)*, 111
MDS, *136*, *138*
meanvar, 112
mglm, 115
mglmLevenberg, *87*
mglmLevenberg *(*mglm*)*, 115
mglmOneGroup, *12*, *87*
mglmOneGroup *(*mglm*)*, 115
mglmOneWay *(*mglm*)*, 115
model.matrix, *118*
modelMatrixMeth, 118
movingAverageByCol, *74*, 119, *181*
mroast, *154*, *157*
mroast.DGEGLM *(*roast.DGEGLM*)*, 153
mroast.DGEList *(*roast.DGEList*)*, 155

nbinomDeviance, 120
nbinomUnitDeviance, *121*, 121
nearestReftoX, 122, *124*
nearestTSS, 123
normalizeBetweenArrays, *125*
normalizeBetweenArrays.DGEList, 124
normalizeChIPtoInput, 125
normalizeCyclicLoess, *124*
normLibSizes, *83*, 126

Ops.CompressedMatrix
*(*makeCompressedMatrix*)*, 105
optim, *57*
optimize, *54*, *61*, *64*
order, *172*

p.adjust, *34*, *154*, *156*, *174*, *176*
PCList, *18*, *129*, 129, *131*, *151*
PCList-class, 130
plotBCV, 131, *139*
plotExonUsage, 132
plotMD.DGEBIN *(*plotMD.DGEList*)*, 133
plotMD.DGEExact *(*plotMD.DGEList*)*, 133
plotMD.DGEGLM *(*plotMD.DGEList*)*, 133
plotMD.DGEList, 133, *139*, *142*
plotMD.DGELRT *(*plotMD.DGEList*)*, 133
plotMD.SummarizedExperiment
*(*plotMD.DGEList*)*, 133
plotMDS, *136*, *138*
plotMDS.DGEList, *114*, 135, *139*
plotMDS.PCList, 137
plotMDS.SummarizedExperiment
*(*plotMDS.DGEList*)*, 135
plotMeanVar, *139*
plotMeanVar *(*meanvar*)*, 112
plotMeanVar2, *114*, 138
plotQLDisp, *93*, *95*, 140
plotSmear, *109*, *114*, *139*, 141
plotSplice, *46*
plotSpliceDGE, 142
plotWithHighlights, *134, 135*
pnbinom, *183*
points, *131*, *135*, *137*, *140*
predFC, *6, 7*, 143
processAmplicons, 145

q2qnbinom, *61*, 147
q2qpois *(*q2qnbinom*)*, 147
qnorm, *183*
qqnorm, *101*
quantile, *33*

rbind.CompressedMatrix
*(*makeCompressedMatrix*)*, 105
rbind.DGEList *(*cbind*)*, 27
read.delim, *152, 153*
read10X, 148
readBismark2DGE, 150

readBismark2PC, 151
readDGE, 152
Roast, *154*, *157*
roast, *154–157*
roast.DGEGLM, 153
roast.DGEList, 155
romer, *158–160*
romer.default, *158*, *160*
romer.DGEGLM, 158
romer.DGEList, 159
rowsum, 161, *162*, *171*
rpkm *(*cpm*)*, 30
rpkmByGroup *(*cpm*)*, 30

sage.test, *15*
sampleWeights, 162
scaleOffset, 163
SE2DGEList, *128*, 164
Seurat2PB, 165
show,DGEBIN-method *(*DGEBIN-class*)*, 35
show,DGEExact-method *(*DGEExact-class*)*, 36
show,DGEGLM-method *(*DGEGLM-class*)*, 37
show,DGELRT-method *(*DGELRT-class*)*, 43
show,TopTags-method *(*topTags*)*, 174
spliceVariants, *46*, *133*, 166
splitGencodeTxNames, *27*, 167
splitIntoGroups, 168
splitIntoGroupsPseudo *(*splitIntoGroups*)*, 168
squeezeVar, *16*, *91*
strsplit2, *168*
subsetting, *36–38*, *40*, *43*, *130*, 169
sumTechReps, 170
Sweave, *59*
system, *59*
systematicSubset, *67*, 171

tail.DGEBIN *(*head*)*, 103
tail.DGEExact *(*head*)*, 103
tail.DGEGLM *(*head*)*, 103
tail.DGEList *(*head*)*, 103
tail.DGELRT *(*head*)*, 103
tail.PCList *(*head*)*, 103
tail.TopTags *(*head*)*, 103
TestResults, *34*
text, *135*, *137*
thinCounts, 172
topGO, *99*
topKEGG, *99*
topSplice, *46*
topSpliceDGE, *143*, 173
topTable, *175*
topTags, *13*, *17*, *18*, *87*, *89*, *95*, *97*, *167*, 174
TopTags-class *(*topTags*)*, 174
tpm, 176, *178*
tpmProp, 177
treat, *97*

uniroot, *54*

validDGEList, 178
voomLmFit, *163*
voomWithQualityWeights, *163*

weightedCondLogLikDerDelta, 179
WLEB, 180

zscoreNBinom, 182