Package: DRIMSeq (via r-universe)

October 30, 2026

Type Package

Title Differential transcript usage and tuQTL analyses with Dirichlet-multinomial model in RNA-seq

Version 1.41.4

Date 2026-10-29

Description The package provides two frameworks. One for the differential transcript usage analysis between different conditions and one for the tuQTL analysis. Both are based on modeling the counts of genomic features (i.e., transcripts) with the Dirichlet-multinomial distribution. The package also makes available functions for visualization and exploration of the data and results.

biocViews ImmunoOncology, SNP, AlternativeSplicing, DifferentialSplicing, Genetics, RNASeq, Sequencing, WorkflowStep, MultipleComparison, GeneExpression, DifferentialExpression

License GPL (>= 3)

Depends R (>= 3.4.0)

Imports utils, stats, MASS, GenomicRanges, IRanges, S4Vectors, BiocGenerics, methods, BiocParallel, limma, edgeR, ggplot2, reshape2

Suggests grid, BiocStyle, knitr, testthat

LazyData true

ByteCompile false

VignetteBuilder knitr

Collate 'DRIMSeq.R' 'aaa.R' 'class_show_utils.R' 'class_MatrixList.R' 'class_dmDSdata.R' 'class_dmDSprecision.R' 'class_dmDSfit.R' 'class_dmDStest.R' 'class_dmSQTLdata.R' 'class_dmSQTLprecision.R' 'class_dmSQTLfit.R' 'class_dmSQTLtest.R' 'dmDS_CRadjustment.R' 'dmDS_estimateCommonPrecision.R' 'dmDS_estimateTagwisePrecision.R' 'dmDS_filter.R' 'dmDS_fit.R'

'dmDS_profileLik.R' 'dmSQTL_CRadjustment.R'
'dmSQTL_estimateCommonPrecision.R'
'dmSQTL_estimateTagwisePrecision.R' 'dmSQTL_filter.R'
'dmSQTL_fit.R' 'dmSQTL_permutations.R' 'dmSQTL_profileLik.R'
'dm_CRadjustmentManyGroups.R' 'dm_CRadjustmentOneGroup.R'
'dm_CRadjustmentRegression.R' 'dm_LRT.R' 'dm_core_Hessian.R'
'dm_core_colorb.R' 'dm_core_deviance.R' 'dm_core_lik.R'
'dm_core_score.R' 'dm_estimateMeanExpression.R'
'dm_fitManyGroups.R' 'dm_fitOneGroup.R' 'dm_fitRegression.R'
'dm_plotData.R' 'dm_plotPrecision.R' 'dm_plotProportions.R'
'dm_plotPvalues.R' 'dm_profileLikModeration.R'

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libicu-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-10-29 17:56:34 UTC

RemoteUrl https://github.com/bioc/DRIMSeq

RemoteRef HEAD

RemoteSha d7a601b2e9a568e89c3e2ed0fe1207b3456ff075

Contents

dm_plotDataDSInfo . . . 3
dm_plotProportions . . . 3
dmDSdata . . . 4
dmDSdata-class . . . 5
dmDSfit-class . . . 7
dmDSprecision-class . . . 9
dmDStest-class . . . 12
dmFilter . . . 14
dmFit . . . 17
dmPrecision . . . 21
dmSQTLdata . . . 25
dmSQTLdata-class . . . 27
dmSQTLfit-class . . . 29
dmSQTLprecision-class . . . 30
dmSQTLtest-class . . . 32
dmTest . . . 33
MatrixList-class . . . 37
plotData . . . 39
plotPrecision . . . 41
plotProportions . . . 43
plotPValues . . . 46

Index **49**

dm_plotDataDSInfo *Plot the frequency of present features*

Description

Plot the frequency of present features

Usage

```
dm_plotDataDSInfo(info, ds_info)
```

Arguments

info	Data frame with gene_id and feature_id of ALL features
ds_info	Data frame with gene_id and feature_id of ONLY DS features

Value

ggplot object

dm_plotProportions *Plot feature proportions*

Description

Plot observed and/or estimated feature proportions.

Usage

```
dm_plotProportions(
  counts,
  group,
  md = NULL,
  fit_full = NULL,
  main = NULL,
  plot_type = "boxplot1",
  order_features = TRUE,
  order_samples = TRUE,
  group_colors = NULL,
  feature_colors = NULL
)
```

Arguments

counts	Matrix with rows corresponding to features and columns corresponding to samples. Row names are used as labels on the plot.
group	Factor that groups samples into conditions.
md	Data frame with additional sample information.
fit_full	Matrix of estimated proportions with rows corresponding to features and columns corresponding to samples. If NULL, nothing is plotted.
main	Character vector with main title for the plot. If NULL, nothing is plotted.
plot_type	Character defining the type of the plot produced. Possible values "barplot", "boxplot1", "boxplot2", "lineplot", "ribbonplot".
order_features	Logical. Whether to plot the features ordered by their expression.
order_samples	Logical. Whether to plot the samples ordered by the group variable. If FALSE order from the sample(x) is kept.
group_colors	Character vector with colors for each group.
feature_colors	Character vector with colors for each feature.

Value

ggplot object with the observed and/or estimated with Dirichlet-multinomial model feature ratios. Estimated proportions are marked with diamond shapes.

---

dmDSdata *Create dmDSdata object*

Description

Constructor function for a dmDSdata object.

Usage

```
dmDSdata(counts, samples)
```

Arguments

counts	Data frame with counts. Rows correspond to features, for example, transcripts or exons. This data frame has to contain a gene_id column with gene IDs, feature_id column with feature IDs and columns with counts for each sample. Column names corresponding to sample IDs must be the same as in the sample data frame.
samples	Data frame where each row corresponds to one sample. Columns have to contain unique sample IDs in sample_id variable and a grouping variable group.

Value

Returns a dmDSdata object.

Author(s)

Malgorzata Nowicka

See Also

plotData

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

d <- d[names(d) %in% gene_id_subset, ]
```

---

dmDSdata-class *dmDSdata object*

Description

dmDSdata contains expression, in counts, of genomic features such as exons or transcripts and sample information needed for the differential exon/transcript usage (DEU or DTU) analysis. It can be created with function dmDSdata.

Usage

```
## S4 method for signature 'dmDSdata'
counts(object)

samples(x, ...)

## S4 method for signature 'dmDSdata'
samples(x)

## S4 method for signature 'dmDSdata'
```

```
names(x)

## S4 method for signature 'dmDSdata'
length(x)

## S4 method for signature 'dmDSdata,ANY'
x[i, j]
```

Arguments

object, x	dmDSdata object.
...	Other parameters that can be defined by methods using this generic.
i, j	Parameters used for subsetting.

Value

- `counts(object)`: Get a data frame with counts.
- `samples(x)`: Get a data frame with the sample information.
- `names(x)`: Get the gene names.
- `length(x)`: Get the number of genes.
- `x[i, j]`: Get a subset of dmDSdata object that consists of counts for genes i and samples j.

Slots

`counts` MatrixList of expression, in counts, of genomic features. Rows correspond to genomic features, such as exons or transcripts. Columns correspond to samples. MatrixList is partitioned in a way that each of the matrices in a list contains counts for a single gene.

`samples` Data frame with information about samples. It must contain `sample_id` variable with unique sample names and other covariates that desribe samples and are needed for the differential analysis.

Author(s)

Malgorzata Nowicka

See Also

dmDSprecision, dmDSfit, dmDStest

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")
```

```
## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

d <- d[names(d) %in% gene_id_subset, ]
```

`dmDSfit-class` *dmDSfit object*

Description

dmDSfit extends the `dmDSprecision` class by adding the full model Dirichlet-multinomial (DM) and beta-binomial (BB) likelihoods, regression coefficients and feature proportion estimates. Result of calling the `dmFit` function.

Usage

```
## S4 method for signature 'dmDSfit'
design(object, type = "full_model")

proportions(x, ...)

## S4 method for signature 'dmDSfit'
proportions(x)

## S4 method for signature 'dmDSfit'
coefficients(object, level = "gene")
```

Arguments

`type`	Character indicating which design matrix should be returned. Possible values `"precision"`, `"full_model"` or `"null_model"`.
`x, object`	dmDSprecision object.
`...`	Other parameters that can be defined by methods using this generic.
`level`	Character specifying which type of results to return. Possible values `"gene"` or `"feature"`.

Value

- `design(object)`: Get a matrix with the full design.
- `proportions(x)`: Get a data frame with estimated feature ratios for each sample.
- `coefficients(x)`: Get the DM or BB regression coefficients.

Slots

`design_fit_full` Numeric matrix of the design used to fit the full model.

`fit_full` MatrixList containing estimated feature ratios in each sample based on the full Dirichlet-multinomial (DM) model.

`lik_full` Numeric vector of the per gene DM full model likelihoods.

`coef_full` MatrixList with the regression coefficients based on the DM model.

`fit_full_bb` MatrixList containing estimated feature ratios in each sample based on the full beta-binomial (BB) model.

`lik_full_bb` Numeric vector of the per gene BB full model likelihoods.

`coef_full_bb` MatrixList with the regression coefficients based on the BB model.

Author(s)

Malgorzata Nowicka

See Also

dmDSdata, dmDSprecision, dmDStest

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

## Use a subset of genes, which is defined in the following file
gene_id_subset <- readLines(file.path(data_dir, "gene_id_subset.txt"))

d <- d[names(d) %in% gene_id_subset, ]

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)
```

```
d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))

## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")
```

`dmDSprecision-class` *dmDSprecision object*

Description

dmDSprecision extends the `dmDSdata` by adding the precision estimates of the Dirichlet-multinomial distribution used to model the feature (e.g., transcript, exon, exonic bin) counts for each gene in the differential usage analysis. Result of calling the `dmPrecision` function.

Usage

```
## S4 method for signature 'dmDSprecision'
design(object, type = "precision")

mean_expression(x, ...)

## S4 method for signature 'dmDSprecision'
mean_expression(x)
```

```
common_precision(x, ...)

## S4 method for signature 'dmDSprecision'
common_precision(x)

common_precision(x) <- value

## S4 replacement method for signature 'dmDSprecision'
common_precision(x) <- value

genewise_precision(x, ...)

## S4 method for signature 'dmDSprecision'
genewise_precision(x)

genewise_precision(x) <- value

## S4 replacement method for signature 'dmDSprecision'
genewise_precision(x) <- value
```

Arguments

type	Character indicating which design matrix should be returned. Possible values "precision", "full_model" or "null_model".
x, object	dmDSprecision object.
...	Other parameters that can be defined by methods using this generic.
value	Values that replace current attributes.

Details

Normally, in the differential analysis based on RNA-seq data, such as, for example, differential gene expression, dispersion (of negative-binomial model) is estimated. Here, we estimate precision of the Dirichlet-multinomial model as it is more convenient computationally. To obtain dispersion estimates, one can use a formula: dispersion = 1 / (1 + precision).

Value

- `mean_expression(x)`: Get a data frame with mean gene expression.
- `common_precision(x)`, `common_precision(x) <- value`: Get or set common precision. `value` must be numeric of length 1.
- `genewise_precision(x)`, `genewise_precision(x) <- value`: Get a data frame with gene-wise precision or set new gene-wise precision. `value` must be a data frame with "gene_id" and "genewise_precision" columns.

Slots

`mean_expression` Numeric vector of mean gene expression.

`common_precision` Numeric value of estimated common precision.

`genewise_precision` Numeric vector of estimated gene-wise precisions.

`design_precision` Numeric matrix of the design used to estimate precision.

Author(s)

Malgorzata Nowicka

See Also

dmDSdata, dmDSfit, dmDStest

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)
```

```
head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))
```

`dmDStest-class` *dmDStest object*

Description

dmDStest extends the `dmDSfit` class by adding the null model Dirichlet-multinomial (DM) and beta-binomial (BB) likelihoods and the gene-level and feature-level results of testing for differential exon/transcript usage. Result of calling the `dmTest` function.

Usage

```
## S4 method for signature 'dmDStest'
design(object, type = "null_model")

results(x, ...)

## S4 method for signature 'dmDStest'
results(x, level = "gene")
```

Arguments

`type`	Character indicating which design matrix should be returned. Possible values `"precision"`, `"full_model"` or `"null_model"`.
`x, object`	dmDStest object.
`...`	Other parameters that can be defined by methods using this generic.
`level`	Character specifying which type of results to return. Possible values `"gene"` or `"feature"`.

Value

- `results(x)`: get a data frame with gene-level or feature-level results.

Slots

`design_fit_null` Numeric matrix of the design used to fit the null model.

`lik_null` Numeric vector of the per gene DM null model likelihoods.

`lik_null_bb` Numeric vector of the per gene BB null model likelihoods.

`results_gene` Data frame with the gene-level results including: `gene_id` - gene IDs, `lr` - likelihood ratio statistics based on the DM model, `df` - degrees of freedom, `pvalue` - p-values and `adj_pvalue` - Benjamini & Hochberg adjusted p-values.

`results_feature` Data frame with the feature-level results including: `gene_id` - gene IDs, `feature_id` - feature IDs, `lr` - likelihood ratio statistics based on the BB model, `df` - degrees of freedom, `pvalue` - p-values and `adj_pvalue` - Benjamini & Hochberg adjusted p-values.

Author(s)

Malgorzata Nowicka

See Also

dmDSdata, dmDSprecision, dmDSfit

Examples

```
# ------------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)
```

```
head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))

## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")

## Fit null model proportions and perform the LR test to detect DTU
d <- dmTest(d, coef = "groupKD")

## Plot the gene-level p-values
plotPValues(d)

## Get the gene-level results
head(results(d))

## Plot feature proportions for a top DTU gene
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]

top_gene_id <- res$gene_id[1]

plotProportions(d, gene_id = top_gene_id, group_variable = "group")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "lineplot")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "ribbonplot")
```

`dmFilter` *Filtering*

Description

Filtering of genes and features with low expression. Additionally, for the dmSQTLdata object, filtering of genotypes with low frequency.

Usage

```
dmFilter(x, ...)
```

```
## S4 method for signature 'dmDSdata'
dmFilter(
  x,
  min_samps_gene_expr = 0,
  min_samps_feature_expr = 0,
  min_samps_feature_prop = 0,
  min_gene_expr = 0,
  min_feature_expr = 0,
  min_feature_prop = 0,
  run_gene_twice = FALSE
)

## S4 method for signature 'dmSQTLdata'
dmFilter(
  x,
  min_samps_gene_expr = 0,
  min_samps_feature_expr = 0,
  min_samps_feature_prop = 0,
  minor_allele_freq = 0.05 * nrow(samples(x)),
  min_gene_expr = 0,
  min_feature_expr = 0,
  min_feature_prop = 0,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

x	dmDSdata or dmSQTLdata object.
...	Other parameters that can be defined by methods using this generic.
min_samps_gene_expr	Minimal number of samples where genes should be expressed. See Details.
min_samps_feature_expr	Minimal number of samples where features should be expressed. See Details.
min_samps_feature_prop	Minimal number of samples where features should be expressed. See details.
min_gene_expr	Minimal gene expression.
min_feature_expr	Minimal feature expression.
min_feature_prop	Minimal proportion for feature expression. This value should be between 0 and 1.
run_gene_twice	Whether to re-run the gene-level filter after the feature-level filters.
minor_allele_freq	Minimal number of samples where each of the genotypes has to be present.
BPPARAM	Parallelization method used by bplapply.

Details

Filtering parameters should be adjusted according to the sample size of the experiment data and the number of replicates per condition.

`min_samps_gene_expr` defines the minimal number of samples where genes are required to be expressed at the minimal level of `min_gene_expr` in order to be included in the downstream analysis. Ideally, we would like that genes were expressed at some minimal level in all samples because this would lead to better estimates of feature ratios.

Similarly, `min_samps_feature_expr` and `min_samps_feature_prop` defines the minimal number of samples where features are required to be expressed at the minimal levels of counts `min_feature_expr` or proportions `min_feature_prop`. In differential transcript/exon usage analysis, we suggest using `min_samps_feature_expr` and `min_samps_feature_prop` equal to the minimal number of replicates in any of the conditions. For example, in an assay with 3 versus 5 replicates, we would set these parameters to 3, which allows a situation where a feature is expressed in one condition but may not be expressed at all in another one, which is an example of differential transcript/exon usage.

By default, all the filtering parameters equal zero which means that features with zero expression in all samples are removed as well as genes with only one non-zero feature.

In QTL analysis, usually, we deal with data that has many more replicates than data from a standard differential usage assay. Our example data set consists of 91 samples. Requiring that genes are expressed in all samples may be too stringent, especially since there may be missing values in the data and for some genes you may not observe counts in all 91 samples. Slightly lower threshold ensures that we do not eliminate such genes. For example, if `min_samps_gene_expr = 70` and `min_gene_expr = 10`, only genes with expression of at least 10 in at least 70 samples are kept. Samples with expression lower than 10 have `NA`s assigned and are skipped in the analysis of this gene. `minor_allele_freq` indicates the minimal number of samples for the minor allele presence. Usually, it is equal to roughly 5% of total samples.

Value

Returns filtered `dmDSdata` or `dmSQTLdata` object.

Author(s)

Malgorzata Nowicka

See Also

`plotData`

Examples

```
# ------------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")
```

```
## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

d <- d[names(d) %in% gene_id_subset, ]

# ----------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ----------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

# ----------------------------------------------------------------------------
# Create dmSQTLdata object
# ----------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)

# ----------------------------------------------------------------------------
# sQTL analysis - simple group comparison
# ----------------------------------------------------------------------------

## Filtering
d <- dmFilter(d, min_samps_gene_expr = 70, min_samps_feature_expr = 5,
  minor_allele_freq = 5, min_gene_expr = 10, min_feature_expr = 10)

plotData(d)
```

dmFit *Fit the Dirichlet-multinomial and/or the beta-binomial full model regression*

Description

Obtain the maximum likelihood estimates of Dirichlet-multinomial (gene-level) and/or beta-binomial (feature-level) regression coefficients, feature proportions in each sample and corresponding likelihoods. In the differential exon/transcript usage analysis, the regression model is defined by a design matrix. In the exon/transcript usage QTL analysis, regression models are defined by genotypes. Currently, beta-binomial model is implemented only in the differential usage analysis.

Usage

```
dmFit(x, ...)

## S4 method for signature 'dmDSprecision'
dmFit(
  x,
  design,
  one_way = TRUE,
  bb_model = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  add_uniform = FALSE,
  BPPARAM = BiocParallel::SerialParam()
)

## S4 method for signature 'dmSQTLprecision'
dmFit(
  x,
  one_way = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

x	dmDSprecision or dmSQTLprecision object.
...	Other parameters that can be defined by methods using this generic.
design	Numeric matrix defining the full model.

one_way	Logical. Should the shortcut fitting be used when the design corresponds to multiple group comparison. This is a similar approach as in edgeR. If TRUE (the default), then proportions are fitted per group and regression coefficients are recalculated from those fits.
bb_model	Logical. Whether to perform the feature-level analysis using the beta-binomial model.
prop_mode	Optimization method used to estimate proportions. Possible value "constrOptim".
prop_tol	The desired accuracy when estimating proportions.
coef_mode	Optimization method used to estimate regression coefficients. Possible value "optim".
coef_tol	The desired accuracy when estimating regression coefficients.
verbose	Numeric. Definie the level of progress messages displayed. 0 - no messages, 1 - main messages, 2 - message for every gene fitting.
add_uniform	Whether to add a small fractional count to zeros, (adding a uniform random variable between 0 and 0.1). This option allows for the fitting of genewise precision and coefficients for genes with two features having all zero for one group, or the last feature having all zero for one group.
BPPARAM	Parallelization method used by bplapply.

Details

In the regression framework here, we adapt the idea from glmFit in edgeR about using a shortcut algorithm when the design is equivalent to simple group fitting. In such a case, we estimate the DM proportions for each group of samples separately and then recalculate the DM (and/or the BB) regression coefficients corresponding to the design matrix. If the design matrix does not define a simple group fitting, for example, when it contains a column with continuous values, then the regression framework is used to directly estimate the regression coefficients.

Arguments that are used for the proportion estimation in each group when the shortcut fitting can be used start with prop_, and those that are used in the regression framework start with coef_.

In the differential transcript usage analysis, setting one_way = TRUE allows switching to the shortcut algorithm only if the design is equivalent to simple group fitting. one_way = FALSE forces usage of the regression framework.

In the QTL analysis, currently, genotypes are defined as numeric values 0, 1, and 2. When one_way = TRUE, simple multiple group fitting is performed. When one_way = FALSE, a regression framework is used with the design matrix defined by a formula ~ group where group is a continuous (not categorical) variable with values 0, 1, and 2.

Value

Returns a dmDSfit or dmSQTLfit object.

Author(s)

Malgorzata Nowicka

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.

See Also

plotProportions glmFit

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))
```

```
## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")
```

`dmPrecision` *Estimate the precision parameter in the Dirichlet-multinomial model*

Description

Maximum likelihood estimates of the precision parameter in the Dirichlet-multinomial model used for the differential exon/transcript usage or QTL analysis.

Usage

```
dmPrecision(x, ...)

## S4 method for signature 'dmDSdata'
dmPrecision(
  x,
  design,
  mean_expression = TRUE,
  common_precision = TRUE,
  genewise_precision = TRUE,
  prec_adjust = TRUE,
  prec_subset = 0.1,
  prec_interval = c(0, 1000),
  prec_tol = 10,
  prec_init = 100,
  prec_grid_length = 21,
  prec_grid_range = c(-10, 10),
  prec_moderation = "trended",
  prec_prior_df = 0,
  prec_span = 0.1,
  one_way = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  add_uniform = FALSE,
```

```
  BPPARAM = BiocParallel::SerialParam()
)

## S4 method for signature 'dmSQTLdata'
dmPrecision(
  x,
  mean_expression = TRUE,
  common_precision = TRUE,
  genewise_precision = TRUE,
  prec_adjust = TRUE,
  prec_subset = 0.1,
  prec_interval = c(0, 1000),
  prec_tol = 10,
  prec_init = 100,
  prec_grid_length = 21,
  prec_grid_range = c(-10, 10),
  prec_moderation = "none",
  prec_prior_df = 0,
  prec_span = 0.1,
  one_way = TRUE,
  speed = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

x	dmDSdata or dmSQTLdata object.
...	Other parameters that can be defined by methods using this generic.
design	Numeric matrix defining the model that should be used when estimating precision. Normally this should be a full model design used also in dmFit.
mean_expression	Logical. Whether to estimate the mean expression of genes.
common_precision	Logical. Whether to estimate the common precision.
genewise_precision	Logical. Whether to estimate the gene-wise precision.
prec_adjust	Logical. Whether to use the Cox-Reid adjusted or non-adjusted profile likelihood.
prec_subset	Value from 0 to 1 defining the percentage of genes used in common precision estimation. The default is 0.1, which uses 10 randomly selected genes to speed up the precision estimation process. Use set.seed function to make the analysis reproducible. See Examples.

prec_interval — Numeric vector of length 2 defining the interval of possible values for the common precision.

prec_tol — The desired accuracy when estimating common precision.

prec_init — Initial precision. If common_precision is TRUE, then prec_init is overwritten by common precision estimate.

prec_grid_length — Length of the search grid.

prec_grid_range — Vector giving the limits of grid interval.

prec_moderation — Precision moderation method. One can choose to shrink the precision estimates toward the common precision ("common") or toward the (precision versus mean expression) trend ("trended")

prec_prior_df — Degree of moderation (shrinkage) in case when it can not be calculated automaticaly (number of genes on the upper boundary of grid is smaller than 10). By default it is equal to 0.

prec_span — Value from 0 to 1 defining the percentage of genes used in smoothing sliding window when calculating the precision versus mean expression trend.

one_way — Logical. Should the shortcut fitting be used when the design corresponds to multiple group comparison. This is a similar approach as in edgeR. If TRUE (the default), then proportions are fitted per group and regression coefficients are recalculated from those fits.

prop_mode — Optimization method used to estimate proportions. Possible value "constrOptim".

prop_tol — The desired accuracy when estimating proportions.

coef_mode — Optimization method used to estimate regression coefficients. Possible value "optim".

coef_tol — The desired accuracy when estimating regression coefficients.

verbose — Numeric. Definie the level of progress messages displayed. 0 - no messages, 1 - main messages, 2 - message for every gene fitting.

add_uniform — Whether to add a small fractional count to zeros, (adding a uniform random variable between 0 and 0.1). This option allows for the fitting of genewise precision and coefficients for genes with two features having all zero for one group, or the last feature having all zero for one group.

BPPARAM — Parallelization method used by bplapply.

speed — Logical. If FALSE, precision is calculated per each gene-block. Such calculation may take a long time, since there can be hundreds of SNPs/blocks per gene. If TRUE, there will be only one precision calculated per gene and it will be assigned to all the blocks matched with this gene.

Details

Normally, in the differential analysis based on RNA-seq data, such as, for example, differential gene expression, dispersion (of negative-binomial model) is estimated. Here, we estimate precision

of the Dirichlet-multinomial model as it is more convenient computationally. To obtain dispersion estimates, one can use a formula: dispersion = 1 / (1 + precision).

Parameters that are used in the precision (dispersion = 1 / (1 + precision)) estimation start with prefix `prec_`. Those that are used for the proportion estimation in each group when the shortcut fitting `one_way = TRUE` can be used start with `prop_`, and those that are used in the regression framework start with `coef_`.

There are two optimization methods implemented within dmPrecision: `"optimize"` for the common precision and `"grid"` for the gene-wise precision.

Only part of the precision parameters in dmPrecision have an influence on a given optimization method. Here is a list of such active parameters:

`"optimize"`:

- `prec_interval`: Passed as `interval`.
- `prec_tol`: The accuracy defined as `tol`.

`"grid"`, which uses the grid approach from `estimateDisp` in `edgeR`:

- `prec_init`, `prec_grid_length`, `prec_grid_range`: Parameters used to construct the search grid `prec_init * 2^seq(from = prec_grid_range[1], to = prec_grid_range[2], length = prec_grid_length)`.
- `prec_moderation`: Dipsersion shrinkage is available only with `"grid"` method.
- `prec_prior_df`: Used only when precision shrinkage is activated. Moderated likelihood is equal to `loglik + prec_prior_df * moderation`. Higher `prec_prior_df`, more shrinkage toward common or trended precision is applied.
- `prec_span`: Used only when precision moderation toward trend is activated.

Value

Returns a `dmDSprecision` or `dmSQTLprecision` object.

Author(s)

Malgorzata Nowicka

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.

See Also

`plotPrecision` `estimateDisp`

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))
```

`dmSQTLdata` *Create dmSQTLdata object*

Description

Constructor functions for a dmSQTLdata object. dmSQTLdata assignes to a gene all the SNPs that are located in a given surrounding (window) of this gene.

Usage

```
dmSQTLdata(
  counts,
  gene_ranges,
  genotypes,
  snp_ranges,
  samples,
  window = 5000,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

counts	Data frame with counts. Rows correspond to features, for example, transcripts or exons. This data frame has to contain a gene_id column with gene IDs, feature_id column with feature IDs and columns with counts for each sample. Column names corresponding to sample IDs must be the same as in the sample data frame.
gene_ranges	GRanges object with gene location. It must contain gene names when calling names().
genotypes	Data frame with genotypes. Rows correspond to SNPs. This data frame has to contain a snp_id column with SNP IDs and columns with genotypes for each sample. Column names corresponding to sample IDs must be the same as in the sample data frame. The genotype of each sample is coded in the following way: 0 for ref/ref, 1 for ref/not ref, 2 for not ref/not ref, -1 or NA for missing value.
snp_ranges	GRanges object with SNP location. It must contain SNP names when calling names().
samples	Data frame with column sample_id corresponding to unique sample IDs
window	Size of a down and up stream window, which is defining the surrounding for a gene. Only SNPs that are located within a gene or its surrounding are considered in the sQTL analysis.
BPPARAM	Parallelization method used by bplapply.

Details

It is quite common that sample grouping defined by some of the SNPs is identical. Compare dim(genotypes) and dim(unique(genotypes)). In our QTL analysis, we do not repeat tests for the SNPs that define the same grouping of samples. Each grouping is tested only once. SNPs that define such unique groupings are aggregated into blocks. P-values and adjusted p-values are estimated at the block level, but the returned results are extended to a SNP level by repeating the block statistics for each SNP that belongs to a given block.

Value

Returns a dmSQTLdata object.

Author(s)

Malgorzata Nowicka

See Also

plotData

Examples

```
# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)
```

---

dmSQTLdata-class *dmSQTLdata object*

Description

dmSQTLdata contains genomic feature expression (counts), genotypes and sample information needed for the transcript/exon usage QTL analysis. It can be created with function dmSQTLdata.

Usage

```
## S4 method for signature 'dmSQTLdata'
counts(object)

## S4 method for signature 'dmSQTLdata'
samples(x)

## S4 method for signature 'dmSQTLdata'
names(x)
```

```
## S4 method for signature 'dmSQTLdata'
length(x)

## S4 method for signature 'dmSQTLdata,ANY'
x[i, j]
```

Arguments

x, object	dmSQTLdata object.
i, j	Parameters used for subsetting.

Value

- `names(x)`: Get the gene names.
- `length(x)`: Get the number of genes.
- `x[i, j]`: Get a subset of dmDSdata object that consists of counts, genotypes and blocks corresponding to genes i and samples j.

Slots

`counts` MatrixList of expression, in counts, of genomic features. Rows correspond to genomic features, such as exons or transcripts. Columns correspond to samples. MatrixList is partitioned in a way that each of the matrices in a list contains counts for a single gene.

`genotypes` MatrixList of unique genotypes. Rows correspond to blocks, columns to samples. Each matrix in this list is a collection of unique genotypes that are matched with a given gene.

`blocks` MatrixList with two columns `block_id` and `snp_id`. For each gene, it identifies SNPs with identical genotypes across the samples and assigns them to blocks.

`samples` Data frame with information about samples. It must contain variable `sample_id` with unique sample names.

Author(s)

Malgorzata Nowicka

See Also

dmSQTLprecision, dmSQTLfit, dmSQTLtest

Examples

```
# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
```

```
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)
```

---

dmSQTLfit-class *dmSQTLfit object*

Description

dmSQTLfit extends the dmSQTLprecision class by adding the full model Dirichlet-multinomial (DM) likelihoods, regression coefficients and feature proportion estimates needed for the transcript/exon usage QTL analysis. Full model is defined by the genotype of a SNP associated with a gene. Estimation takes place for all the genes and all the SNPs/blocks assigned to the genes. Result of dmFit.

Slots

`fit_full` List of MatrixList objects containing estimated feature ratios in each sample based on the full Dirichlet-multinomial (DM) model.

`lik_full` List of numeric vectors of the per gene DM full model likelihoods.

`coef_full` MatrixList with the regression coefficients based on the DM model.

Author(s)

Malgorzata Nowicka

See Also

dmSQTLdata, dmSQTLprecision, dmSQTLtest

Examples

```
# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)
```

```
# ------------------------------------------------------------------------
# sQTL analysis - simple group comparison
# ------------------------------------------------------------------------

## Filtering
d <- dmFilter(d, min_samps_gene_expr = 70, min_samps_feature_expr = 5,
  minor_allele_freq = 5, min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d)

plotPrecision(d)

## Fit full model proportions
d <- dmFit(d)
```

`dmSQTLprecision-class` *dmSQTLprecision object*

Description

dmSQTLprecision extends the `dmSQTLdata` by adding the precision estimates of Dirichlet-multinomial distribution used to model the feature (e.g., transcript, exon, exonic bin) counts for each gene-SNP pair in the QTL analysis. Result of `dmPrecision`.

Usage

```
## S4 method for signature 'dmSQTLprecision'
mean_expression(x)

## S4 method for signature 'dmSQTLprecision'
common_precision(x)

## S4 method for signature 'dmSQTLprecision'
genewise_precision(x)
```

Arguments

x	dmSQTLprecision object.

Value

- `mean_expression(x)`: Get a data frame with mean gene expression.
- `common_precision(x)`: Get common precision.
- `genewise_precision(x)`: Get a data frame with gene-wise precision.

Slots

`mean_expression` Numeric vector of mean gene expression.

`common_precision` Numeric value of estimated common precision.

`genewise_precision` List of estimated gene-wise precisions. Each element of this list is a vector of precisions estimated for all the genotype blocks assigned to a given gene.

Author(s)

Malgorzata Nowicka

See Also

dmSQTLdata, dmSQTLfit, dmSQTLtest

Examples

```
# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)

# ------------------------------------------------------------------------
# sQTL analysis - simple group comparison
# ------------------------------------------------------------------------

## Filtering
d <- dmFilter(d, min_samps_gene_expr = 70, min_samps_feature_expr = 5,
  minor_allele_freq = 5, min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## To make the analysis reproducible
set.seed(123)
## Calculate precision
```

```
d <- dmPrecision(d)

plotPrecision(d)
```

`dmSQTLtest-class` *dmSQTLtest object*

Description

dmSQTLtest extends the `dmSQTLfit` class by adding the null model Dirichlet-multinomial likelihoods and the gene-level results of testing for differential transcript/exon usage QTLs. Result of `dmTest`.

Usage

```
## S4 method for signature 'dmSQTLtest'
results(x)
```

Arguments

`x`	dmSQTLtest object.
`...`	Other parameters that can be defined by methods using this generic.

Value

- `results(x)`: Get a data frame with gene-level results.

Slots

`lik_null` List of numeric vectors with the per gene-snp DM null model likelihoods.

`results_gene` Data frame with the gene-level results including: `gene_id` - gene IDs, `block_id` - block IDs, `snp_id` - SNP IDs, `lr` - likelihood ratio statistics based on the DM model, `df` - degrees of freedom, `pvalue` - p-values estimated based on permutations and `adj_pvalue` - Benjamini & Hochberg adjusted p-values.

Author(s)

Malgorzata Nowicka

See Also

`dmSQTLdata`, `dmSQTLprecision`, `dmSQTLfit`

Examples

```
# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)

# ------------------------------------------------------------------------
# sQTL analysis - simple group comparison
# ------------------------------------------------------------------------

## Filtering
d <- dmFilter(d, min_samps_gene_expr = 70, min_samps_feature_expr = 5,
  minor_allele_freq = 5, min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d)

plotPrecision(d)

## Fit full model proportions
d <- dmFit(d)

## Fit null model proportions, perform the LR test to detect tuQTLs
## and use the permutation approach to adjust the p-values
d <- dmTest(d)

## Plot the gene-level p-values
plotPValues(d)

## Get the gene-level results
head(results(d))
```

`dmTest` *Likelihood ratio test to detect differential transcript/exon usage*

Description

First, estimate the null Dirichlet-multinomial and beta-binomial model parameters and likelihoods using the null model design. Second, perform the gene-level (DM model) and feature-level (BB model) likelihood ratio tests. In the differential exon/transcript usage analysis, the null model is defined by the null design matrix. In the exon/transcript usage QTL analysis, null models are defined by a design with intercept only. Currently, beta-binomial model is implemented only in the differential usage analysis.

Usage

```
dmTest(x, ...)

## S4 method for signature 'dmDSfit'
dmTest(
  x,
  coef = NULL,
  design = NULL,
  contrast = NULL,
  one_way = TRUE,
  bb_model = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  BPPARAM = BiocParallel::SerialParam()
)

## S4 method for signature 'dmSQTLfit'
dmTest(
  x,
  permutation_mode = "all_genes",
  one_way = TRUE,
  prop_mode = "constrOptim",
  prop_tol = 1e-12,
  coef_mode = "optim",
  coef_tol = 1e-12,
  verbose = 0,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

x	dmDSfit or dmSQTLfit object.
...	Other parameters that can be defined by methods using this generic.
coef	Integer or character vector indicating which coefficients of the linear model are to be tested equal to zero. Values must indicate column numbers or column names of the design used in dmFit.

`design` Numeric matrix defining the null model.

`contrast` Numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero. For a matrix, number of rows (for a vector, its length) must equal to the number of columns of `design` used in `dmFit`.

`one_way` Logical. Should the shortcut fitting be used when the design corresponds to multiple group comparison. This is a similar approach as in `edgeR`. If `TRUE` (the default), then proportions are fitted per group and regression coefficients are recalculated from those fits.

`bb_model` Logical. Whether to perform the feature-level analysis using the beta-binomial model.

`prop_mode` Optimization method used to estimate proportions. Possible value `"constrOptim"`.

`prop_tol` The desired accuracy when estimating proportions.

`coef_mode` Optimization method used to estimate regression coefficients. Possible value `"optim"`.

`coef_tol` The desired accuracy when estimating regression coefficients.

`verbose` Numeric. Definie the level of progress messages displayed. 0 - no messages, 1 - main messages, 2 - message for every gene fitting.

`BPPARAM` Parallelization method used by `bplapply`.

`permutation_mode`
Character specifying which permutation scheme to apply for p-value calculation. When equal to `"all_genes"`, null distribution of p-values is calculated from all genes and the maximum number of permutation cycles is 10. When `permutation_mode = "per_gene"`, null distribution of p-values is calculated for each gene separately based on permutations of this individual gene. The latter approach may take a lot of computational time. We suggest using the first option.

Details

One must specify one of the arguments: `coef`, `design` or `contrast`.

When `contrast` is used to define the null model, the null design matrix is recalculated using the same approach as in `glmLRT` function from `edgeR`.

Value

Returns a `dmDStest` or `dmSQTLtest` object.

Author(s)

Malgorzata Nowicka

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.

See Also

plotPValues glmLRT

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))

## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
```

```
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")

## Fit null model proportions and perform the LR test to detect DTU
d <- dmTest(d, coef = "groupKD")

## Plot the gene-level p-values
plotPValues(d)

## Get the gene-level results
head(results(d))

## Plot feature proportions for a top DTU gene
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]

top_gene_id <- res$gene_id[1]

plotProportions(d, gene_id = top_gene_id, group_variable = "group")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "lineplot")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "ribbonplot")
```

`MatrixList-class` *MatrixList object*

Description

A MatrixList object is a container for a list of matrices which have the same number of columns but can have varying number of rows. Additionally, one can store an extra information corresponding to each of the matrices in `metadata` matrix.

Usage

```
## S4 method for signature 'MatrixList'
names(x)

## S4 replacement method for signature 'MatrixList'
names(x) <- value

## S4 method for signature 'MatrixList'
rownames(x)
```

```
## S4 replacement method for signature 'MatrixList'
rownames(x) <- value

## S4 method for signature 'MatrixList'
colnames(x)

## S4 replacement method for signature 'MatrixList'
colnames(x) <- value

## S4 method for signature 'MatrixList'
length(x)

## S4 method for signature 'MatrixList'
elementNROWS(x)

## S4 method for signature 'MatrixList'
dim(x)

## S4 method for signature 'MatrixList'
nrow(x)

## S4 method for signature 'MatrixList'
ncol(x)

## S4 method for signature 'MatrixList'
x[[i, j]]

## S4 method for signature 'MatrixList'
x$name

## S4 method for signature 'MatrixList,ANY'
x[i, j]
```

Arguments

x	MatrixList object.
value, i, j, name	Parameters used for subsetting and assigning new attributes to x.

Value

- names(x), names(x) <- value: Get or set names of matrices.
- rownames(x), rownames(x) <- value, colnames(x), colnames(x) <- value: Get or set row names or column names of unlistData slot.
- length(x): Get the number of matrices in a list.
- elementNROWS(x): Get the number of rows of each of the matrices.
- dim(x), nrow(x), ncol(x): Get the dimensions, number of rows or number of columns of unlistData slot.

- `x[[i]]`, `x[[i, j]]`: Get the matrix i, and optionally, get only columns j of this matrix.
- `x$name`: Shortcut for `x[["name"]]`.
- `x[i, j]`: Get a subset of MatrixList that consists of matrices i with columns j.

Slots

`unlistData` Matrix which is a row binding of all the matrices in a list.

`partitioning` List of indexes which defines the row partitioning of unlistData matrix into the original matrices.

`metadata` Matrix of additional information where each row corresponds to one of the matrices in a list.

Author(s)

Malgorzata Nowicka

`plotData` *Plot data summary*

Description

Plot data summary

Usage

```
plotData(x, ...)

## S4 method for signature 'dmDSdata'
plotData(x)

## S4 method for signature 'dmSQTLdata'
plotData(x, plot_type = "features")
```

Arguments

`x`	`dmDSdata` or `dmSQTLdata` object.
`...`	Other parameters that can be defined by methods using this generic.
`plot_type`	Character specifying which type of histogram to plot. Possible values `"features"`, `"snps"` or `"blocks"`.

Value

Returns a `ggplot` object and can be further modified, for example, using `theme()`. Plots a histogram of the number of features per gene. Additionally, for `dmSQTLdata` object, plots a histogram of the number of SNPs per gene and a histogram of the number of unique SNPs (blocks) per gene.

Author(s)

Malgorzata Nowicka

See Also

plotPrecision, plotProportions, plotPValues

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

# ------------------------------------------------------------------------
# Create dmSQTLdata object
# ------------------------------------------------------------------------
# Use subsets of data defined in the Geuvadis files on zenodo (formerly: GeuvadisTranscriptExpr package)

DRIMSeq:::.load_data("geuvadis")

colnames(counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(genotypes)[4] <- "snp_id"
samples <- data.frame(sample_id = colnames(counts)[-c(1,2)])

d <- dmSQTLdata(counts = counts, gene_ranges = gene_ranges,
  genotypes = genotypes, snp_ranges = snp_ranges,
  samples = samples, window = 5e3)
```

```
# ------------------------------------------------------------------------
# sQTL analysis - simple group comparison
# ------------------------------------------------------------------------

## Filtering
d <- dmFilter(d, min_samps_gene_expr = 70, min_samps_feature_expr = 5,
  minor_allele_freq = 5, min_gene_expr = 10, min_feature_expr = 10)

plotData(d)
```

`plotPrecision` *Precision versus mean expression plot*

Description

Precision versus mean expression plot

Usage

```
plotPrecision(x, ...)

## S4 method for signature 'dmDSprecision'
plotPrecision(x)

## S4 method for signature 'dmSQTLprecision'
plotPrecision(x)
```

Arguments

x	dmDSprecision or dmSQTLprecision object.
...	Other parameters that can be defined by methods using this generic.

Value

Normally in the differential analysis based on RNA-seq data, such plot has dispersion parameter plotted on the y-axis. Here, the y-axis represents precision since in the Dirichlet-multinomial model this is the parameter that is directly estimated. It is important to keep in mind that the precision parameter (gamma0) is inverse proportional to dispersion (theta): theta = 1 / (1 + gamma0). In RNA-seq data, we can typically observe a trend where the dispersion decreases (here, precision increases) for genes with higher mean expression.

Author(s)

Malgorzata Nowicka

See Also

plotData, plotProportions, plotPValues

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))
```

plotProportions *Plot feature proportions*

Description

This plot is available only for a group design, i.e., a design that is equivalent to multiple group fitting.

Usage

```
plotProportions(x, ...)

## S4 method for signature 'dmDSfit'
plotProportions(
  x,
  gene_id,
  group_variable,
  plot_type = "barplot",
  order_features = TRUE,
  order_samples = TRUE,
  plot_fit = TRUE,
  plot_main = TRUE,
  group_colors = NULL,
  feature_colors = NULL
)

## S4 method for signature 'dmSQTLfit'
plotProportions(
  x,
  gene_id,
  snp_id,
  plot_type = "boxplot1",
  order_features = TRUE,
  order_samples = TRUE,
  plot_fit = FALSE,
  plot_main = TRUE,
  group_colors = NULL,
  feature_colors = NULL
)
```

Arguments

x	dmDSfit, dmDStest or dmSQTLfit, dmSQTLtest object.
...	Other parameters that can be defined by methods using this generic.
gene_id	Character indicating a gene ID to be plotted.

group_variable	Character indicating the grouping variable which is one of the columns in the samples slot of x.
plot_type	Character defining the type of the plot produced. Possible values "barplot", "boxplot1", "boxplot2", "lineplot", "ribbonplot".
order_features	Logical. Whether to plot the features ordered by their expression.
order_samples	Logical. Whether to plot the samples ordered by the group variable. If FALSE order from the sample(x) is kept.
plot_fit	Logical. Whether to plot the proportions estimated by the full model.
plot_main	Logical. Whether to plot a title with the information about the Dirichlet-multinomial estimates.
group_colors	Character vector with colors for each group defined by group_variable.
feature_colors	Character vector with colors for each feature of gene defined by gene_id.
snp_id	Character indicating the ID of a SNP to be plotted.

Details

In the QTL analysis, plotting of fitted proportions is deactivated even when plot_fit = TRUE. It is due to the fact that neither fitted values nor regression coefficients are returned by the dmFit function as they occupy a lot of memory.

Value

For a given gene, plot the observed and estimated with Dirichlet-multinomial model feature proportions in each group. Estimated group proportions are marked with diamond shapes.

Author(s)

Malgorzata Nowicka

See Also

plotData, plotPrecision, plotPValues

Examples

```
# ------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)
```

```
## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# --------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# --------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))

## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")

## Fit null model proportions and perform the LR test to detect DTU
d <- dmTest(d, coef = "groupKD")

## Plot the gene-level p-values
plotPValues(d)

## Get the gene-level results
head(results(d))

## Plot feature proportions for a top DTU gene
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]
```

```
top_gene_id <- res$gene_id[1]

plotProportions(d, gene_id = top_gene_id, group_variable = "group")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "lineplot")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "ribbonplot")
```

`plotPValues` *Plot p-value distribution*

Description

Plot p-value distribution

Usage

```
plotPValues(x, ...)

## S4 method for signature 'dmDStest'
plotPValues(x, level = "gene")

## S4 method for signature 'dmSQTLtest'
plotPValues(x)
```

Arguments

`x`	`dmDStest` or `dmSQTLtest` object.
`...`	Other parameters that can be defined by methods using this generic.
`level`	Character specifying which type of results to return. Possible values `"gene"` or `"feature"`.

Value

Plot a histogram of p-values.

Author(s)

Malgorzata Nowicka

See Also

`plotData`, `plotPrecision`, `plotProportions`

Examples

```
# ------------------------------------------------------------------------------
# Create dmDSdata object
# ------------------------------------------------------------------------------
## Get kallisto transcript counts from zenodo (formerly: 'PasillaTranscriptExpr' package)

## Load data
DRIMSeq:::.load_data("pasilla")

## Create a pasilla_samples data frame
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)

## Create a dmDSdata object
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)

# ------------------------------------------------------------------------------
# Differential transcript usage analysis - simple two group comparison
# ------------------------------------------------------------------------------

## Filtering
## Check what is the minimal number of replicates per condition
table(samples(d)$group)

d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

plotData(d)

## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))

## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)

plotPrecision(d)

head(mean_expression(d))
common_precision(d)
head(genewise_precision(d))

## Fit full model proportions
d <- dmFit(d, design = design_full)

## Get fitted proportions
head(proportions(d))
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
## Get the BB regression coefficients (feature-level)
```

```
head(coefficients(d), level = "feature")

## Fit null model proportions and perform the LR test to detect DTU
d <- dmTest(d, coef = "groupKD")

## Plot the gene-level p-values
plotPValues(d)

## Get the gene-level results
head(results(d))

## Plot feature proportions for a top DTU gene
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]

top_gene_id <- res$gene_id[1]

plotProportions(d, gene_id = top_gene_id, group_variable = "group")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "lineplot")

plotProportions(d, gene_id = top_gene_id, group_variable = "group",
  plot_type = "ribbonplot")
```

Index

[,MatrixList,ANY-method (MatrixList-class), 37
[,MatrixList-method (MatrixList-class), 37
[,dmDSdata,ANY-method (dmDSdata-class), 5
[,dmDSdata-method (dmDSdata-class), 5
[,dmSQTLdata,ANY-method (dmSQTLdata-class), 27
[,dmSQTLdata-method (dmSQTLdata-class), 27
[[,MatrixList-method (MatrixList-class), 37
$,MatrixList-method (MatrixList-class), 37

bplapply, *15*, *19*, *23*, *26*, *35*

coefficients,dmDSfit-method (dmDSfit-class), 7
colnames,MatrixList-method (MatrixList-class), 37
colnames<-,MatrixList-method (MatrixList-class), 37
common_precision (dmDSprecision-class), 9
common_precision,dmDSprecision-method (dmDSprecision-class), 9
common_precision,dmSQTLprecision-method (dmSQTLprecision-class), 30
common_precision<- (dmDSprecision-class), 9
common_precision<-,dmDSprecision-method (dmDSprecision-class), 9
counts,dmDSdata-method (dmDSdata-class), 5
counts,dmSQTLdata-method (dmSQTLdata-class), 27

design,dmDSfit-method (dmDSfit-class), 7
design,dmDSprecision-method (dmDSprecision-class), 9
design,dmDStest-method (dmDStest-class), 12
dim,MatrixList-method (MatrixList-class), 37
dm_plotDataDSInfo, 3
dm_plotProportions, 3
dmDSdata, 4, *4*, *5*, *8*, *9*, *11*, *13*, *15*, *16*, *22*, *39*
dmDSdata-class, 5
dmDSfit, *6*, *11–13*, *19*, *34*, *43*
dmDSfit-class, 7
dmDSprecision, *6–8*, *13*, *18*, *24*, *41*
dmDSprecision-class, 9
dmDStest, *6*, *8*, *11*, *35*, *43*, *46*
dmDStest-class, 12
dmFilter, 14
dmFilter,dmDSdata-method (dmFilter), 14
dmFilter,dmSQTLdata-method (dmFilter), 14
dmFit, *7*, 17, *22*, *29*, *34*, *35*
dmFit,dmDSprecision-method (dmFit), 18
dmFit,dmSQTLprecision-method (dmFit), 18
dmPrecision, *9*, 21, *30*
dmPrecision,dmDSdata-method (dmPrecision), 21
dmPrecision,dmSQTLdata-method (dmPrecision), 21
dmSQTLdata, *15*, *16*, *22*, 25, *26*, *27*, *29–32*, *39*
dmSQTLdata-class, 27
dmSQTLfit, *19*, *28*, *31*, *32*, *34*, *43*
dmSQTLfit-class, 29
dmSQTLprecision, *18*, *24*, *28*, *29*, *32*, *41*
dmSQTLprecision-class, 30
dmSQTLtest, *28*, *29*, *31*, *35*, *43*, *46*
dmSQTLtest-class, 32
dmTest, *12*, *32*, 33
dmTest,dmDSfit-method (dmTest), 33
dmTest,dmSQTLfit-method (dmTest), 33

edgeR, *19*, *23*, *24*, *35*
elementNROWS,MatrixList-method *(MatrixList-class)*, 37
estimateDisp, *24*

genewise_precision *(dmDSprecision-class)*, 9
genewise_precision,dmDSprecision-method *(dmDSprecision-class)*, 9
genewise_precision,dmSQTLprecision-method *(dmSQTLprecision-class)*, 30
genewise_precision<- *(dmDSprecision-class)*, 9
genewise_precision<-,dmDSprecision-method *(dmDSprecision-class)*, 9
glmFit, *19*, *20*
glmLRT, *35*, *36*
GRanges, *26*

length,dmDSdata-method *(dmDSdata-class)*, 5
length,dmSQTLdata-method *(dmSQTLdata-class)*, 27
length,MatrixList-method *(MatrixList-class)*, 37

MatrixList, *6*, *8*, *28*, *29*
MatrixList-class, 37
mean_expression *(dmDSprecision-class)*, 9
mean_expression,dmDSprecision-method *(dmDSprecision-class)*, 9
mean_expression,dmSQTLprecision-method *(dmSQTLprecision-class)*, 30

names,dmDSdata-method *(dmDSdata-class)*, 5
names,dmSQTLdata-method *(dmSQTLdata-class)*, 27
names,MatrixList-method *(MatrixList-class)*, 37
names<-,MatrixList-method *(MatrixList-class)*, 37
ncol,MatrixList-method *(MatrixList-class)*, 37
nrow,MatrixList-method *(MatrixList-class)*, 37

plotData, *5*, *16*, *27*, 39, *42*, *44*, *46*
plotData,dmDSdata-method *(plotData)*, 39
plotData,dmSQTLdata-method *(plotData)*, 39
plotPrecision, *24*, *40*, 41, *44*, *46*
plotPrecision,dmDSprecision-method *(plotPrecision)*, 41
plotPrecision,dmSQTLprecision-method *(plotPrecision)*, 41
plotProportions, *20*, *40*, *42*, 43, *46*
plotProportions,dmDSfit-method *(plotProportions)*, 43
plotProportions,dmSQTLfit-method *(plotProportions)*, 43
plotPValues, *36*, *40*, *42*, *44*, 46
plotPValues,dmDStest-method *(plotPValues)*, 46
plotPValues,dmSQTLtest-method *(plotPValues)*, 46
proportions *(dmDSfit-class)*, 7
proportions,dmDSfit-method *(dmDSfit-class)*, 7

results *(dmDStest-class)*, 12
results,dmDStest-method *(dmDStest-class)*, 12
results,dmSQTLtest-method *(dmSQTLtest-class)*, 32
rownames,MatrixList-method *(MatrixList-class)*, 37
rownames<-,MatrixList-method *(MatrixList-class)*, 37

samples *(dmDSdata-class)*, 5
samples,dmDSdata-method *(dmDSdata-class)*, 5
samples,dmSQTLdata-method *(dmSQTLdata-class)*, 27