DRIMSeq: Dirichlet-multinomial framework for differential transcript usage and transcript usage QTL analyses in RNA-seq

Malgorzata Nowicka*, Mark D. Robinson

March 1, 2017

This vignette describes *DRIMSeq* version 1.3.3.

Contents

*gosia.nowicka@uzh.ch

1 Main changes in the DRIMSeq package

For the full list of changes, type:

```
news(package = "DRIMSeq")
```

- Implementation of the regression framework in differential transcript usage analysis. It allows fitting more complicated than multiple group comparison experimental designs, for example, one can account for the batch effects or the fact that samples are paired or model continuous time course changes. It enables also testing of more complicated contrasts.
- Transcript-level analysis based on the beta-binomial model. In this case, each transcript ratio is modeled separately assuming it follows the beta-binomial distribution which is a one-dimensional version of the Dirichlet-multinomial distribution. Based on the fact that when $(Y_1, \ldots, Y_q) \sim DM(\pi_1, \ldots, \pi_q, \gamma_0)$ then $Y_j \sim BB(\pi_j, \gamma_0)$, we do not need to reestimate the beta-binomial parameters, only the likelihoods are recalculated. *DRIMSeq* returns gene-level and transcript-level p-values which can be used as input to the stage-wise testing procedure [**?**] as screening and confirmation p-values, respectively. Such approach provides increased power to identify transcripts which are actually differentially used in a gene detected as gene with DTU.
- Usage of term 'precision' instead of 'dispersion'. In the differential analysis based on the negative-binomial model, dispersion parameter is estimated. This dispersion parameter captures all sources of the inter-library variation between replicates present in the RNA-seq data. In the DM model, we do not directly estimate dispersion but a parameter called precision which is closely linked to dispersion via the formula: $dispersion = 1/(1 + precision)$. In the previous version of [**?**], we used 'dispersion' as a name for the functions and variables calculating and storing, in fact, the precision estimates. Now, we use the term 'precision'.

2 Overview of the Dirichlet-multinomial model

For the statistical details about the Dirichlet-multinomial model, see the *DRIMSeq* paper [**?**].
In the *DRIMSeq* package we implemented a Dirichlet-multinomial framework that can be used for modeling various multivariate count data with the interest in finding the instances where the ratios of observed features are different between the experimental conditions. Such a model can be applied, for example, in differential transcript usage (DTU) analysis or in the analysis that aim in detecting SNPs that are associated with differential transcript usage (tuQTL analysis). In both cases the multivariate features of a gene are transcripts. The implementation of Dirichlet-multinomial model in *DRIMSeq* package is customized for differential transcript usage and tuQTL analyses, but the data objects used in *DRIMSeq* can contain various types of counts. Therefore, other types of multivariate differential analyses can be performed such as differential methylation analysis or differential polyA usage from polyA-seq data.
In short, the method consists of three statistical steps:

- First, we use the profile likelihood to estimate the precision which is inverse proportional to dispersion, i.e., the variability of transcript ratios between samples (replicates) within conditions. Dispersion is needed in order to find the significant changes in transcript ratios between conditions which should be sufficiently stronger than the changes/variability within conditions.
- Second, we use maximum likelihood to estimate at the gene-level the regression coefficients in the Dirichlet-multinomial (DM) regression, the fitted transcript proportions in each sample and the full model likelihoods. For the analysis at the transcript-level we apply the beta-binomial (BB) regression to each transcript separately. In the differential transcript usage analysis, the full model is defined by

the user with the design matrix. In the QTL analysis, full models are defined by the genotypes of SNPs associated with a given gene.

- Finally, we fit the null model DM and BB regression and use the likelihood ratio statistics to test for the differences in transcript proportions between the full and null models at the gene and transcript level. In the differential transcript usage analysis, the null model is again defined by the user. In the QTL analysis, null models correspond to regression with intercept only.

3 Important notes

- Currently, transcript-level analysis based on the BB model are implemented only in the DTU analysis (`bb_model = TRUE`).
- When the model (full or null) of interest corresponds to multiple (or one) group fitting, then a shortcut algorithm called 'one way' (`one_way = TRUE`), which we adapted from the `glmFit` function in *edgeR*, can be used. Choosing it is equivalent to running the original *DRIMSeq* implementation. In such a case, we use maximum likelihood to estimate the transcript proportions in each group separately and then the regression coefficients are calculated using matrix operations. Otherwise, the regression coefficients are directly estimated with the maximum likelihood approach.

4 Hints for DRIMSeq pipelines

In this vignette, we present how one could perform differential transcript usage analysis and tuQTL analysis with the *DRIMSeq* package. We use small data sets so that the whole pipelines can be run within few minutes in *R* on a single core computer. In practice, the package is designed to take advantage of multicore computing for larger data sets.

In the filtering function `dmFilter`, all the parameters that are influencing transcript count filtering are set to zero. This results in a very relaxed filtering, where transcripts with zero expression in all the samples and genes with only one transcript remained are removed.

Functions `dmPrecision`, `dmFit` and `dmTest`, which perform the actual statistical analyses described above, have many other parameters available for tweaking, but they do have the default values assigned. Those values were chosen based on many real data analyses. Some of the steps are quite time consuming, especially the precision estimation, where proportions of each gene are refitted for different precision parameters. To speed up the calculations, we have parallelized many functions using *BiocParallel*. Thus, if possible, we recommend to increase the number of workers in `BPPARAM`.

In general, tuQTL analyses are more computationally intensive than differential transcript usage analysis because one needs to do the analysis for every SNP in the surrounding region of a gene. Additionally, a permutation scheme is used to compute the p-values. It is indeed feasible to perform tuQTL analysis for small chunks of genome, for example, per chromosome.

5 DRIMSeq differential transcript usage analysis workflow

5.1 Example data

To demonstrate the application of *DRIMSeq* in differential transcript usage analysis, we will use the *pasilla* data set produced by Brooks et al. [**?**]. The aim of their study was to identify exons that are regulated by pasilla protein, the Drosophila melanogaster ortholog of mammalian NOVA1 and NOVA2 (well studied transcript usage factors). In their RNA-seq experiment, the libraries were prepared from 7 biologically independent samples: 4 control samples and 3 samples in which pasilla

was knocked-down. The libraries were sequenced on Illumina Genome Analyzer II using single-end and paired-end sequencing and different read lengths. The RNA-seq data can be downloaded from the NCBI's Gene Expression Omnibus (GEO) under the accession number GSE18508. In the examples below, we use a subset of *kallisto* [**?**] counts available in *PasillaTranscriptExpr* package, where you can find all the steps needed, for preprocessing the GEO data, to get a table with transcript counts.

5.2 Differential transcript usage analysis between two conditions

In this section, we present how to perform the DTU analysis between two conditions control and knock-down without accounting for the batch effect which is the library layout. Analysis where batch effects are included are presented in the next section.
We start the analysis by creating a *dmDSdata* object, which contains transcript counts and information about grouping samples into conditions. With each step of the pipeline, additional elements are added to this object. At the end of the analysis, the object contains results from all the steps, such as precision estimates, regression coefficients, fitted transcript ratios in each sample, likelihood ratio statistics, p-values, adjusted p-values at gene and transcript level. As new elements are added, the object also changes its name *dmDSdata* $\rightarrow$ *dmDSprecision* $\rightarrow$ *dmDSfit* $\rightarrow$ *dmDStest*, but each container inherits slots and methods available for the previous one.

5.2.1 Loading pasilla data into R

The transcript-level counts obtained from *kallisto* and metadata are saved as text files in the `extdata` directory of the *PasillaTranscriptExpr* package.

```
library(PasillaTranscriptExpr)

data_dir  <- system.file("extdata", package = "PasillaTranscriptExpr")

## Load metadata
pasilla_metadata <- read.table(file.path(data_dir, "metadata.txt"),
  header = TRUE, as.is = TRUE)

## Load counts
pasilla_counts <- read.table(file.path(data_dir, "counts.txt"),
  header = TRUE, as.is = TRUE)
```

Load the *DRIMSeq* package.

```
library(DRIMSeq)
```

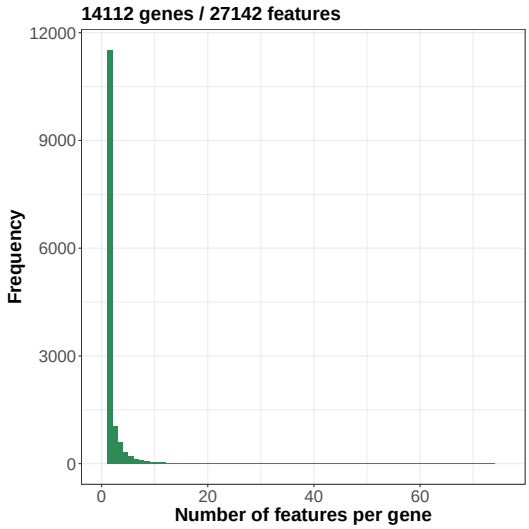
To create a `dmDSdata` object, saved as variable `d`, we need to prepare a data frame containing information about samples and we will call it `pasilla_samples`. It has to have a variable called `sample_id` with unique sample names that are identical to column names in `pasilla_counts` that correspond to samples. Additionally, it has to contain other variables that the user would like to use for the further regression analysis. Here, we are interested in the differential analysis between the control and knock-down condition. This information is stored in `pasilla_metadata$condition`. The data frame with counts called `pasilla_counts` is already formatted in the right way. It contains variables `feature_id` with unique transcript names and `gene_id` with unique gene IDs and columns with counts have the same names as `sample_id` in `pasilla_samples`.
When printing variable `d`, you can see its class, size (number of genes and samples) and which

accessor methods can be applied. For `dmDSdata` object, there are two methods that return data frames with counts and samples.

```
pasilla_samples <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition)
levels(pasilla_samples$group)
## [1] "CTL" "KD"
d <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples)
d
## An object of class dmDSdata
## with 14112 genes and 7 samples
## * data accessors: counts(), samples()
head(counts(d), 3)
##       gene_id  feature_id GSM461176 GSM461177 GSM461178 GSM461179 GSM461180 GSM461181
## 1 FBgn0031208 FBtr0300689   0.00000         0   0.00000  27.04866         0   0.00000
## 2 FBgn0031208 FBtr0300690   5.01688         4   1.00518   0.00000         2   2.00464
## 3 FBgn0031208 FBtr0330654   0.00000         0   0.00000   0.00000         0   0.00000
##   GSM461182
## 1         0
## 2         0
## 3         0
head(samples(d), 3)
##   sample_id group
## 1 GSM461176   CTL
## 2 GSM461177   CTL
## 3 GSM461178   CTL
```

You can also make a data summary plot, which is a histogram of the number of transcripts per gene.

```
plotData(d)
```



To make the analysis runnable within this vignette, we want to keep only a small 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, ]
d
## An object of class dmDSdata
## with 42 genes and 7 samples
## * data accessors: counts(), samples()
```

5.2.2 Filtering

Genes may have many transcripts that are lowly expressed or not expressed at all. You can remove them using the `dmFilter` function. Filtering of lowly expressed transcripts can be done at two levels: minimal *expression* using `min_samps_feature_expr` and `min_feature_expr` parameters or minimal *proportion* with `min_samps_feature_prop` and `min_feature_prop`.
In the *pasilla* experiment we use a filtering based only on the transcript absolute expression and parameters are adjusted according to the number of replicates per condition. Since we have 3 knock-down and 4 control samples, we set `min_samps_feature_expr` equal to 3. In this way, we allow a situation where a transcript is expressed in one condition but not in another, which is a case of differential transcript usage. The level of transcript expression is controlled by `min_feature_expr`. We set it to the value of 10, which means that only the transcripts that have at least 10 estimated counts in at least 3 samples are kept for the downstream analysis.
Filtering at the gene level ensures that the observed transcript ratios have some minimal reliability. Although, Dirichlet-multinomial model works on feature counts, and not on feature ratios, which means that it gives more confidence to the ratios based on 100 versus 500 reads than 1 versus 5, minimal filtering based on gene expression removes the genes with mostly zero counts and reduces the number of tests in multiple test correction. For the *pasilla* data, we want that genes have at least 10 counts in all the samples: `min_samps_gene_expr = 7` and `min_gene_expr = 10`.

```
# Check what is the minimal number of replicates per condition
table(samples(d)$group)
##
## CTL  KD
##   4   3
d <- dmFilter(d, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)
```

5.2.3 Precision estimation

Ideally, we would like to get accurate precision estimates for every gene, which is problematic when analyzing small data sets because precision estimates become inaccurate when the sample size decreases, especially for lowly expressed genes. As an alternative, we could assume that all the genes have the same precision and based on all the data, we could calculate a common precision, but we expect this to be too strong of an assumption. Moderated precision is a trade-off between gene-wise and common precision. The moderated estimates originate from a weighted likelihood which is a combination of common and individual likelihoods. We recommend this approach when analyzing small sample size data sets.
At this step, three values may be calculated: mean expression of genes, common precision and gene-wise precisions. In the default setting, all of them are computed and common precision is used as an initial value in the grid approach to estimate gene-wise precisions, which are shrunk toward

the trended precision.
By default, to estimate the common precision, we use 10% percent (`prec_subset = 0.1`) of randomly selected genes. That is due to the fact that common precision is used only as an initial value, and estimating it based on all the genes takes a substantial part of time. To ensure that the analysis are reproducible, the user should define a random seed `set.seed()` before running the `dmPrecision()` function. Thank to that, each time the same subset of genes is selected.
To estimate precision parameters, the user has to define a design matrix with the full model of interest, which will be also used later in the proportion estimation. Here, the full model is defined by a formula `group` which indicates that samples come from two conditions.
This step of our pipeline is the most time consuming. Thus, for real data analysis, consider using `BPPARAM = BiocParallel::MulticoreParam()` with more than one worker.

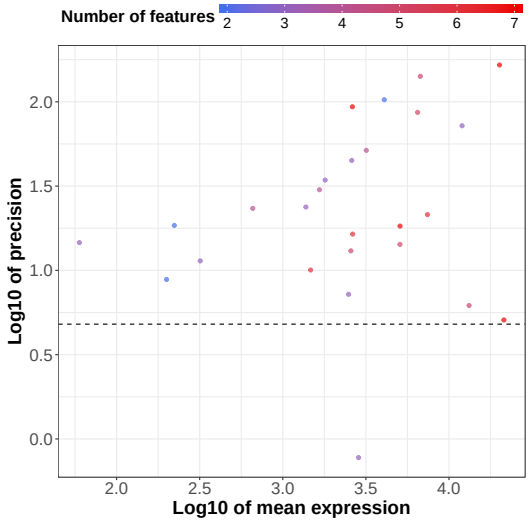
```
## Create the design matrix
design_full <- model.matrix(~ group, data = samples(d))
design_full
##   (Intercept) groupKD
## 1           1       0
## 2           1       0
## 3           1       0
## 4           1       1
## 5           1       1
## 6           1       1
## 7           1       0
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

```
## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d, design = design_full)
## !  Using a subset of 0.1 genes to estimate common precision !
## !  Using common_precision = 4.7973 as prec_init !
## !  Using 0 as a shrinkage factor !
d
## An object of class dmDSprecision
## with 26 genes and 7 samples
## * data accessors: counts(), samples()
##   design()
##   mean_expression(), common_precision(), genewise_precision()
head(mean_expression(d), 3)
##       gene_id mean_expression
## 1 FBgn0000256        2622.286
## 2 FBgn0020309       13217.714
## 3 FBgn0259735       11992.903
common_precision(d)
## [1] 4.797285
```

```
head(genewise_precision(d))
##       gene_id genewise_precision
## 1 FBgn0000256          93.442123
## 2 FBgn0020309           6.187015
## 3 FBgn0259735          72.095637
## 4 FBgn0032785          11.387467
## 5 FBgn0040297          13.056098
## 6 FBgn0032979         102.856815
```

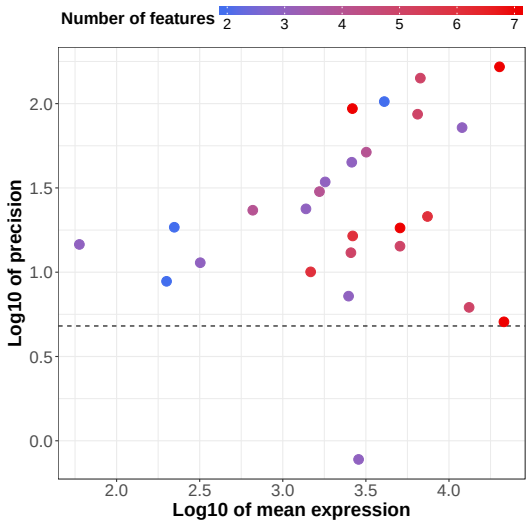
To inspect the behavior of precision estimates, you can plot them against the mean gene expression. Normally in the differential analysis based on RNA-seq data, such plot has dispersion parameter plotted on the y-axis. Here, the y-axis represents precision since in the Dirichlet-multinomial model this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

```
plotPrecision(d)
```



All of the plotting functions from *DRIMSeq* package, return a *ggplot* object which can be further modified using *ggplot2* package. For example, the user can increase the size of points.

```
library(ggplot2)
ggp <- plotPrecision(d)
ggp + geom_point(size = 4)
```



5.2.4 Proportion estimation

In this step, we estimate the full model regression coefficients, fitted proportions and likelihoods. We use the same design matrix as in the precision estimation step.
By default, `one_way = TRUE` which means that whenever the design corresponds to multiple group fitting, the 'one way' shortcut algorithm will be used, which in fact corresponds to the first implementation of *DRIMSeq*. Transcript proportions are estimated for each condition separately and then the regression coefficients are calculated using matrix operations. By setting `verbose = 1`, we can see that the one way approach is used with the current design.
When `bb_model = TRUE` (the default), additionally to the gene-level Dirichlet-multinomial estimates the transcript-level beta-binomial results will be computed.

```
d <- dmFit(d, design = design_full, verbose = 1)
## * Fitting the DM model..
##     Using the one way approach.
## Took 0.4256 seconds.
## * Fitting the BB model..
##     Using the one way approach.
## Took 0.1244 seconds.
d
## An object of class dmDSfit
## with 26 genes and 7 samples
## * data accessors: counts(), samples()
##   design()
##   mean_expression(), common_precision(), genewise_precision()
##   proportions(), coefficients()
## Get fitted proportions
head(proportions(d))
##       gene_id  feature_id   GSM461176   GSM461177   GSM461178   GSM461179   GSM461180
## 1 FBgn0000256 FBtr0290077 0.357378834 0.357378834 0.357378834 0.076551334 0.076551334
## 2 FBgn0000256 FBtr0290078 0.043420022 0.043420022 0.043420022 0.270032340 0.270032340
## 3 FBgn0000256 FBtr0290082 0.006048217 0.006048217 0.006048217 0.001885293 0.001885293
## 4 FBgn0000256 FBtr0077511 0.286681118 0.286681118 0.286681118 0.222610895 0.222610895
```

```
## 5 FBgn0000256 FBtr0290081 0.016850271 0.016850271 0.016850271 0.036979434 0.036979434
## 6 FBgn0000256 FBtr0077513 0.280210368 0.280210368 0.280210368 0.378708732 0.378708732
##     GSM461181   GSM461182
## 1 0.076551334 0.357378834
## 2 0.270032340 0.043420022
## 3 0.001885293 0.006048217
## 4 0.222610895 0.286681118
## 5 0.036979434 0.016850271
## 6 0.378708732 0.280210368
## Get the DM regression coefficients (gene-level)
head(coefficients(d))
##       gene_id  feature_id X.Intercept.     groupKD
## 1 FBgn0000256 FBtr0290077    3.6368991 -1.88157341
## 2 FBgn0000256 FBtr0290078    1.5290234  1.48688248
## 3 FBgn0000256 FBtr0290082   -0.4421337 -1.50641866
## 4 FBgn0000256 FBtr0077511    3.4164732 -0.59368371
## 5 FBgn0000256 FBtr0290081    0.5824694  0.44525661
## 6 FBgn0000256 FBtr0077513    3.3936433 -0.03951182
## Get the BB regression coefficients (feature-level)
head(coefficients(d), level = "feature")
##       gene_id  feature_id X.Intercept.     groupKD
## 1 FBgn0000256 FBtr0290077    3.6368991 -1.88157341
## 2 FBgn0000256 FBtr0290078    1.5290234  1.48688248
## 3 FBgn0000256 FBtr0290082   -0.4421337 -1.50641866
## 4 FBgn0000256 FBtr0077511    3.4164732 -0.59368371
## 5 FBgn0000256 FBtr0290081    0.5824694  0.44525661
## 6 FBgn0000256 FBtr0077513    3.3936433 -0.03951182
```

5.2.5 Testing for differential transcript usage

Calling the `dmTest` function results in two calculations. First, null model is fitted. This null model can be defined by the user via `coef`, `design` or `contrast` parameters. Second, likelihood ratio statistics are used to test for the difference between the full and null model. Both steps are done at the gene and transcript level when `bb_model = TRUE`.

In our example, we would like to test whether there are differences in transcript usage between control (CTL) and knock-down (KD). We can achieve that by using the `coef` parameter which should indicate which columns of the full design should be removed to get the null design. We define it equal to `"groupKD"`. Then the null design has only an intercept column which means that all the samples are treated as if they came from one condition. Note that `one_way = TRUE` and the one way approach is used.

```
d <- dmTest(d, coef = "groupKD", verbose = 1)
## * Fitting the DM model..
##    Using the one way approach.
## Took 0.2102 seconds.
## * Calculating likelihood ratio statistics..
## Took 0.0011 seconds.
## * Fitting the BB model..
```

```
##    Using the one way approach.
## Took 0.0393 seconds.
## * Calculating likelihood ratio statistics..
## Took 5e-04 seconds.
design(d)
##   (Intercept)
## 1           1
## 2           1
## 3           1
## 4           1
## 5           1
## 6           1
## 7           1
head(results(d), 3)
##       gene_id          lr df       pvalue   adj_pvalue
## 1 FBgn0000256 146.2706017  6 4.751618e-29 1.235421e-27
## 2 FBgn0020309  17.8956828  4 1.293394e-03 4.532265e-03
## 3 FBgn0259735   0.9383577  2 6.255157e-01 7.744480e-01
```

The same can be achieved by directly defining the null design matrix with the `design` parameter.

```
design_null <- model.matrix(~ 1, data = samples(d))
design_null
##   (Intercept)
## 1           1
## 2           1
## 3           1
## 4           1
## 5           1
## 6           1
## 7           1
## attr(,"assign")
## [1] 0
d <- dmTest(d, design = design_null)
head(results(d), 3)
##       gene_id          lr df       pvalue   adj_pvalue
## 1 FBgn0000256 146.2706017  6 4.751618e-29 1.235421e-27
## 2 FBgn0020309  17.8956828  4 1.293394e-03 4.532265e-03
## 3 FBgn0259735   0.9383577  2 6.255157e-01 7.744480e-01
```

Or by using the `contrast` parameter.

```
contrast <- c(0, 1)
d <- dmTest(d, contrast = contrast)
design(d)
##   [,1]
## 1   -1
## 2   -1
## 3   -1
## 4   -1
```

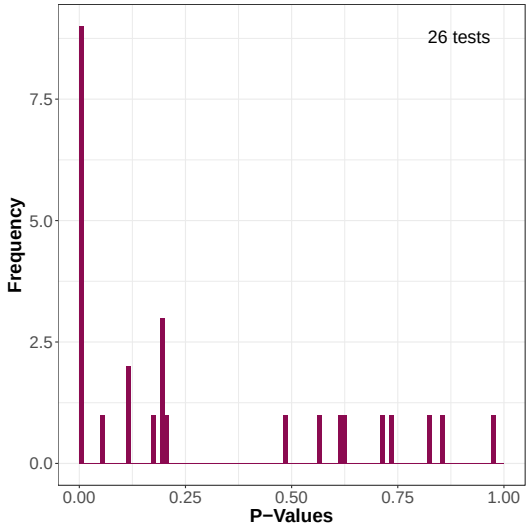
```
## 5   -1
## 6   -1
## 7   -1
head(results(d), 3)
##       gene_id          lr df       pvalue   adj_pvalue
## 1 FBgn0000256 146.2706017  6 4.751618e-29 1.235421e-27
## 2 FBgn0020309  17.8956828  4 1.293394e-03 4.532265e-03
## 3 FBgn0259735   0.9383577  2 6.255157e-01 7.744480e-01
```

To obtain the results of likelihood ratio tests, you have to call the function `results`, which returns a data frame with likelihood ratio statistics, degrees of freedom, p-values and Benjamini and Hochberg (BH) adjusted p-values for each gene by default and for each transcript when `level = "feature"`.

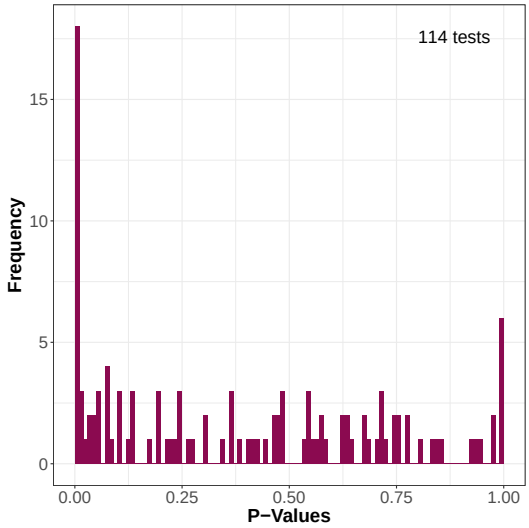
```
head(results(d, level = "feature"), 3)
##       gene_id  feature_id        lr df       pvalue   adj_pvalue
## 1 FBgn0000256 FBtr0290077 87.668156  1 7.741069e-21 4.412409e-19
## 2 FBgn0000256 FBtr0290078 72.369382  1 1.784614e-17 6.781534e-16
## 3 FBgn0000256 FBtr0290082  1.341445  1 2.467793e-01 5.548327e-01
```

You can plot a histogram of gene-level and transcript-level p-values.

```
plotPValues(d)
```

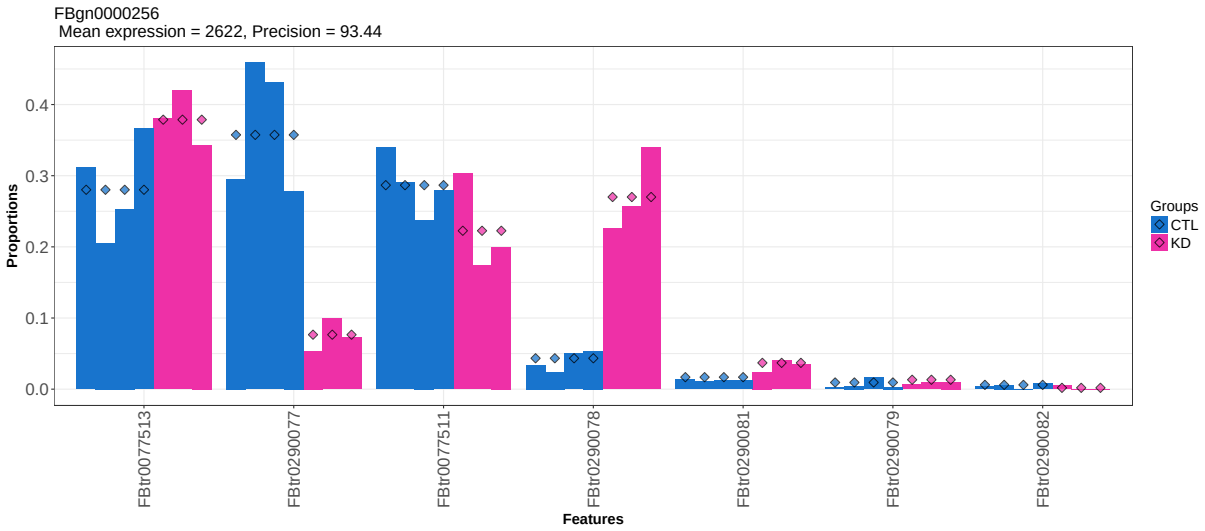



```
plotPValues(d, level = "feature")
```

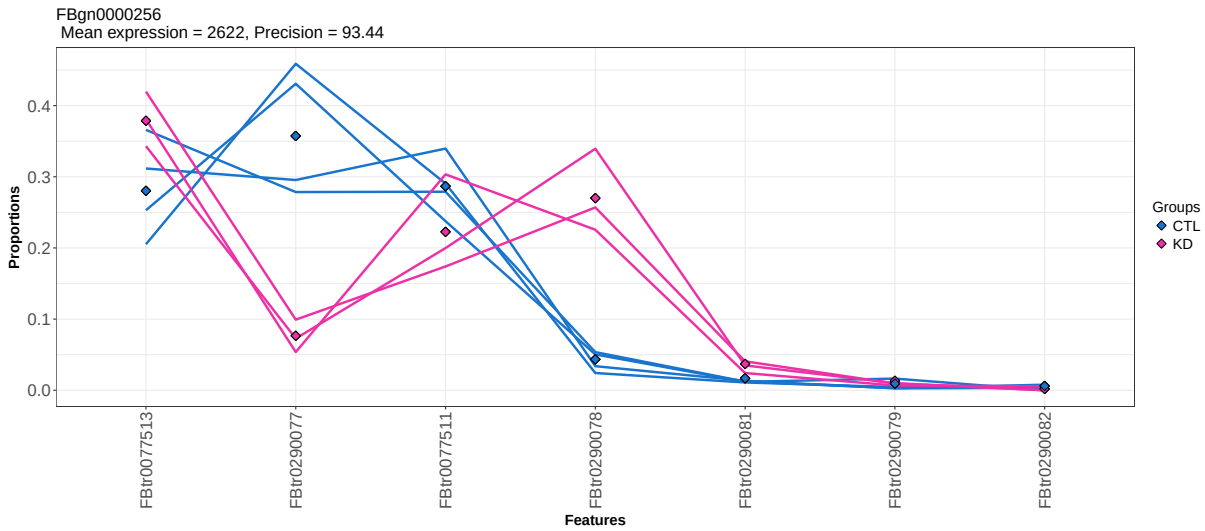


For genes of interest, you can make plots (bar plots, line plots, box plots, ribbon plots) of observed and estimated with Dirichlet-multinomial model transcript ratios. You have to define the `group_variable` parameter which should indicate a variable from `samples(d)`. Currently, plots can be done only for categorical variables. We choose the `"group"` column since it corresponds to the comparison of our interest. Estimated proportions are marked with diamond shapes. As an example, we plot the top significant gene.

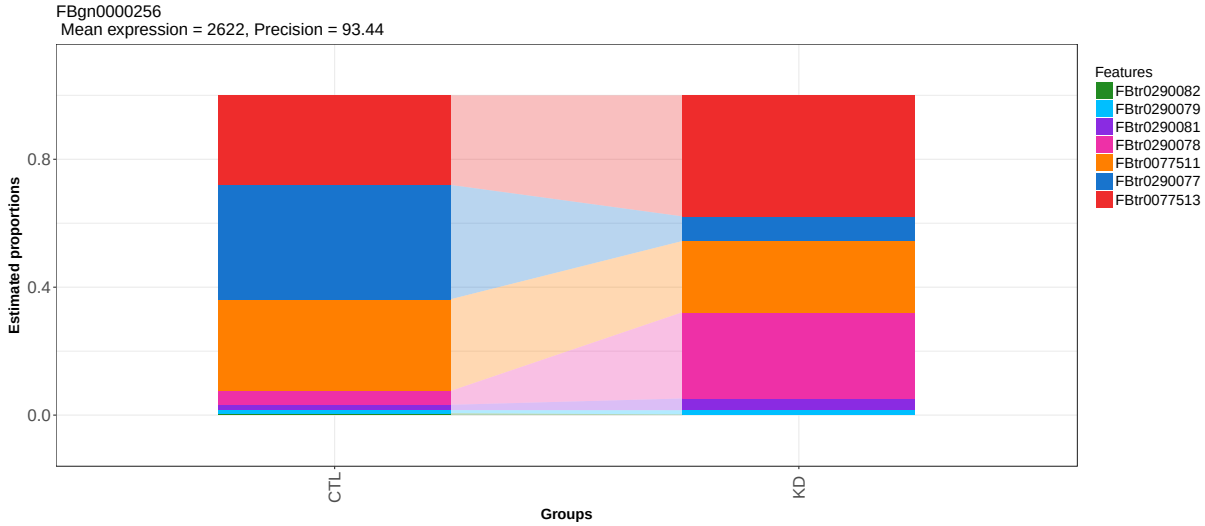
```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]
top_gene_id <- res$gene_id[1]

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")
```



5.2.6 Two-stage test

DRIMSeq returns gene and transcript level p-values which can be used as an input to the stage-wise analysis [**?**] implemented in the `stageR` package, currently available on github https://github.com/statOmics/stageR. As pointed by the authors of `stageR`, interpreting both gene-level and transcript-level adjusted p-values does not provide appropriate FDR control and should be avoided. However, applying a stage-wise testing provides a useful biological interpretation of these results and improved statistical performance.

In short, the procedure consists of a screening stage and a confirmation stage. In the screening stage, gene-level BH-adjusted p-values are screened to detect genes for which the hypothesis of interest is significantly rejected. Only those genes are further considered in the confirmation stage, where for each gene separately, transcript-level p-values are adjusted to control the FWER and BH-adjusted significance level of the screening stage.

It is important to note that transcript-level stage-wise adjusted p-values for genes that do not pass the screening stage are set to `NA`. Also the stage-wise adjusted p-values can not be compared to

significance level other than chosen in the stage-wise analysis. If that is of interest, one has to rerun this analysis with the new significance level.
The following code chunk is not evaluated by this vignette and to run it, user has to make sure that the `stageR` package is installed. It shows how one can use the *DRIMSeq* output in the stage-wise analysis.

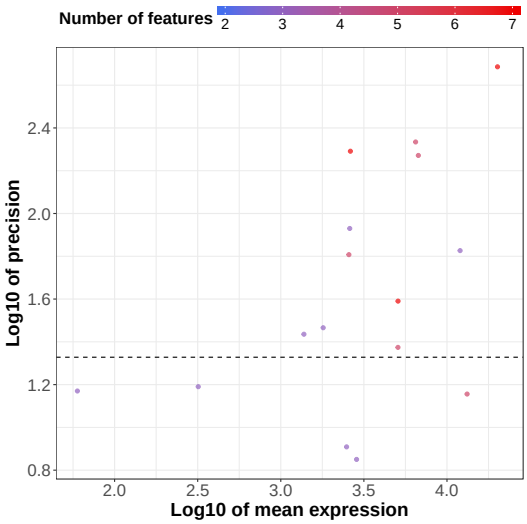
```
library(stageR)

## Assign gene-level pvalues to the screening stage
pScreen <- results(d)$pvalue
names(pScreen) <- results(d)$gene_id

## Assign transcript-level pvalues to the confirmation stage
pConfirmation <- matrix(results(d, level = "feature")$pvalue, ncol = 1)
rownames(pConfirmation) <- results(d, level = "feature")$feature_id

## Create the gene-transcript mapping
tx2gene <- results(d, level = "feature")[, c("feature_id", "gene_id")]

## Create the stageRTx object and perform the stage-wise analysis
stageRObj <- stageRTx(pScreen = pScreen, pConfirmation = pConfirmation,
  pScreenAdjusted = FALSE, tx2gene = tx2gene)

stageRObj <- stageWiseAdjustment(object = stageRObj, method = "dtu",
  alpha = 0.05)

getSignificantGenes(stageRObj)

getSignificantTx(stageRObj)

padj <- getAdjustedPValues(stageRObj, order = TRUE,
  onlySignificantGenes = FALSE)

head(padj)
```

5.3 Differential transcript usage analysis between two conditions with accounting for the batch effects

The regression framework implemented in *DRIMSeq* allows to account for the batch effects. Here, this would be the library layout stored in `pasilla_metadata$LibraryLayout`. The steps of this analysis are the same as described above. The only difference is that we have to include the library layout variable in the `sample` slot in the `dmDSdata` object and define a full model that contains the batch effect.

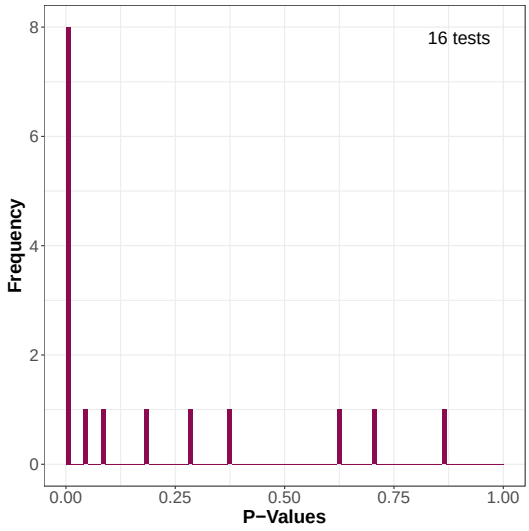
```
pasilla_samples2 <- data.frame(sample_id = pasilla_metadata$SampleName,
  group = pasilla_metadata$condition,
  library_layout = pasilla_metadata$LibraryLayout)

d2 <- dmDSdata(counts = pasilla_counts, samples = pasilla_samples2)
```

```
## Subsetting to a vignette runnable size
d2 <- d2[names(d2) %in% gene_id_subset, ]

## Filtering
d2 <- dmFilter(d2, min_samps_gene_expr = 7, min_samps_feature_expr = 3,
  min_gene_expr = 10, min_feature_expr = 10)

## Create the design matrix
design_full2 <- model.matrix(~ group + library_layout, data = samples(d2))
design_full2
##   (Intercept) groupKD library_layoutSINGLE
## 1           1       0                    1
## 2           1       0                    0
## 3           1       0                    0
## 4           1       1                    1
## 5           1       1                    0
## 6           1       1                    0
## 7           1       0                    1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
##
## attr(,"contrasts")$library_layout
## [1] "contr.treatment"
## To make the analysis reproducible
set.seed(123)

## Calculate precision
d2 <- dmPrecision(d2, design = design_full2)
## !  Using a subset of 0.1 genes to estimate common precision !
## !  Using common_precision = 21.2862 as prec_init !
## !  Using 0 as a shrinkage factor !
common_precision(d)
## [1] 4.797285
head(genewise_precision(d))
##       gene_id genewise_precision
## 1 FBgn0000256          93.442123
## 2 FBgn0020309           6.187015
## 3 FBgn0259735          72.095637
## 4 FBgn0032785          11.387467
## 5 FBgn0040297          13.056098
## 6 FBgn0032979         102.856815
plotPrecision(d2)
```

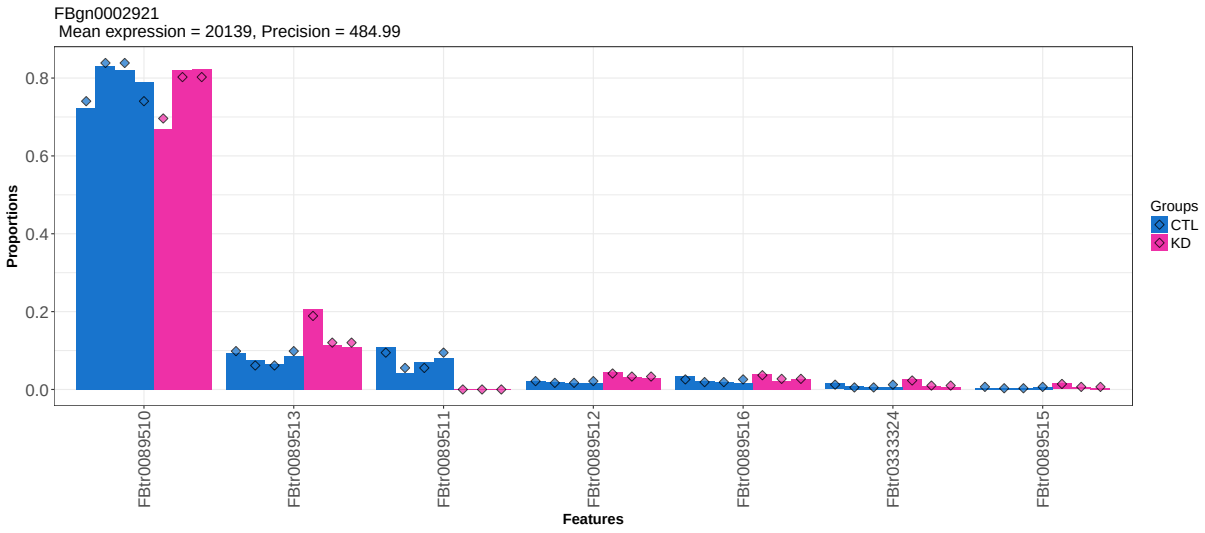



```
## Fit proportions
d2 <- dmFit(d2, design = design_full2, verbose = 1)
## * Fitting the DM model..
##    Using the regression approach.
## Took 0.3812 seconds.
## * Fitting the BB model..
##    Using the regression approach.
## Took 0.0852 seconds.
## Test for DTU
d2 <- dmTest(d2, coef = "groupKD", verbose = 1)
## * Fitting the DM model..
##    Using the one way approach.
## Took 0.2029 seconds.
## * Calculating likelihood ratio statistics..
## Took 7e-04 seconds.
## * Fitting the BB model..
##    Using the one way approach.
## Took 0.0597 seconds.
## * Calculating likelihood ratio statistics..
## Took 5e-04 seconds.
design(d2)
##   (Intercept) library_layoutSINGLE
## 1           1                     1
## 2           1                     0
## 3           1                     0
## 4           1                     1
## 5           1                     0
## 6           1                     0
## 7           1                     1
head(results(d2), 3)
##       gene_id         lr df       pvalue   adj_pvalue
## 1 FBgn0000256 297.219823  6 3.224375e-61 2.579500e-60
```

```
## 2 FBgn0020309  18.721572  4 8.913637e-04 2.037403e-03
## 3 FBgn0259735   0.924745  2 6.297877e-01 7.197574e-01
## Plot p-value distribution
plotPValues(d2)
```




```
## Plot the top significant gene
res2 <- results(d2)
res2 <- res2[order(res2$pvalue, decreasing = FALSE), ]
top_gene_id2 <- res2$gene_id[1]
plotProportions(d2, gene_id = top_gene_id2, group_variable = "group")
```



6 tuQTL analysis workflow

In the transcript usage QTL analysis, we want to identify genetic variants (here, bi-allelic SNPs) that are associated with changes in transcript usage. Such SNPs are then called transcript usage quantitative trait locies (tuQTLs).

Ideally, we would like to test associations of every SNP with every gene. However, such an ap-

proach would be very costly computationally and in terms of multiple testing correction. Under the assumption that SNPs that directly affect transcript usage are likely to be placed in the close surrounding of genes, we test only the SNPs that are located within the gene body and within some range upstream and downstream of the gene.

6.1 Example data

To demonstrate the tuQTL analysis with the *DRIMSeq* package, we use data from the GEUVADIS project [**?**], where 462 RNA-Seq samples from lymphoblastoid cell lines were obtained. The genome sequencing data of the same individuals is provided by the 1000 Genomes Project. The samples in this project come from five populations: CEPH (CEU), Finns (FIN), British (GBR), Toscani (TSI) and Yoruba (YRI). We use transcript quantification (expected counts from FluxCapacitor) and genotypes available on the GEUVADIS project website http://www.ebi.ac.uk/Tools/geuvadis-das/, and the Gencode v12 gene annotation is available at http://www.gencodegenes.org/releases/12.html.
In order to make this vignette runnable, we perform the analysis on subsets of bi-allelic SNPs and transcript expected counts for CEPH population (91 individuals) that correspond to 50 randomly selected genes from chromosome 19. The full dataset can be accessed from *GeuvadisTranscriptExpr* package along with the description of preprocessing steps.

6.2 tuQTL analysis with the DRIMSeq package

Assuming you have gene annotation, feature counts and bi-allelic genotypes that are expressed in terms of the number of alleles different from the reference, the *DRIMSeq* workflow for tuQTL analysis is analogous to the one for differential transcript usage.
First, we have to create a *dmSQTLdata* object, which contains feature counts, sample information and genotypes. Similarly as in the differential transcript usage pipeline, results from every step are added to this object and at the end of the analysis, it contains precision estimates, proportions estimates, likelihood ratio statistics, p-values, adjusted p-values. As new elements are added, the object also changes its name *dmSQTLdata* $\rightarrow$ *dmSQTLprecision* $\rightarrow$ *dmSQTLfit* $\rightarrow$ *dmSQTLtest*. For each object, slots and methods are inherited from the previous one.

6.2.1 Loading GEUVADIS data into R

We use the subsets of data defined in the *GeuvadisTranscriptExpr* package.

```
library(GeuvadisTranscriptExpr)

geuv_counts <- GeuvadisTranscriptExpr::counts
geuv_genotypes <- GeuvadisTranscriptExpr::genotypes
geuv_gene_ranges <- GeuvadisTranscriptExpr::gene_ranges
geuv_snp_ranges <- GeuvadisTranscriptExpr::snp_ranges
```

Load the *DRIMSeq* package.

```
library(DRIMSeq)
```

In the tuQTL analysis, an initial data object `d` is of `dmSQTLdata` class and, additionally to feature counts and sample information, it contains genotypes of SNPs that are in some surrounding of genes. This surrounding is defined with the parameter `window`. In order to find out which SNPs should be tested with which genes, the `dmSQTLdata` functions requires as an input the location of genes

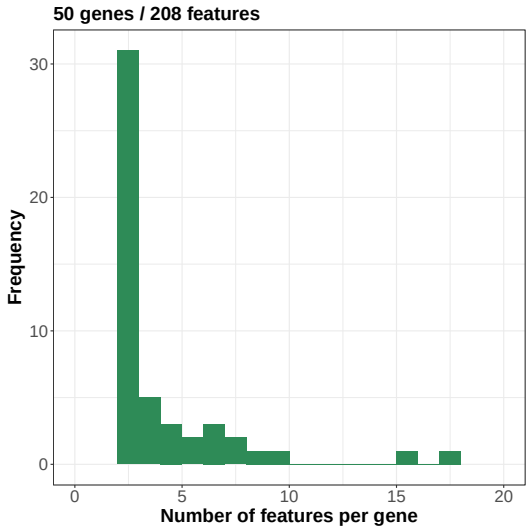
(`gene_ranges`) and SNPs (`snp_ranges`) stored as *GRanges* objects. Variables with transcript IDs and gene IDs in the `counts` data frame must have names `feature_id` and `gene_id`, respectively. In the `genotypes` data frame, the variable with SNP IDs must have name `snp_id`.

```
colnames(geuv_counts)[c(1,2)] <- c("feature_id", "gene_id")
colnames(geuv_genotypes)[4] <- "snp_id"
geuv_samples <- data.frame(sample_id = colnames(geuv_counts)[-c(1,2)])

d <- dmSQTLdata(counts = geuv_counts, gene_ranges = geuv_gene_ranges,
  genotypes = geuv_genotypes, snp_ranges = geuv_snp_ranges,
  samples = geuv_samples, window = 5e3)
d
## An object of class dmSQTLdata
## with 50 genes and 91 samples
## * data accessors: counts(), samples()
```

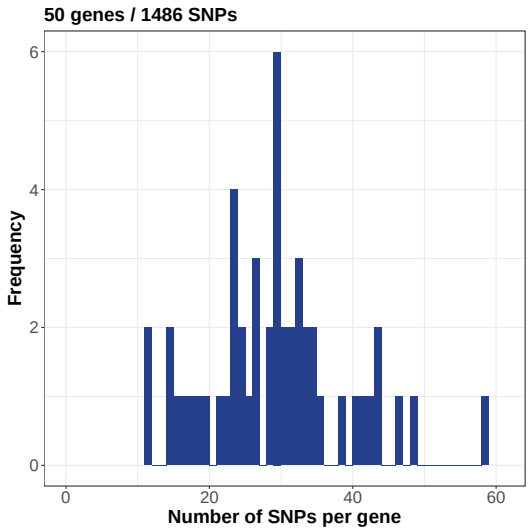
In our tuQTL analysis, we do not repeat tests for the SNPs that define the same grouping of samples (genotype). We identify SNPs with identical genotypes across the samples and assign them to blocks. Estimation and testing are done at the block level, but the returned results are extended to a SNP level by repeating the block statistics for each SNP that belongs to a given block.

The data summary plot `plotData` produces three histograms: the number of features per gene, the number of SNPs per gene and the number of blocks per gene.

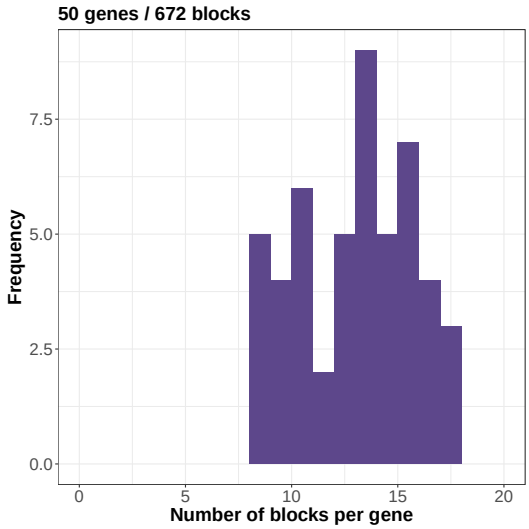
```
plotData(d, plot_type = "features")
```




```
plotData(d, plot_type = "snps")
```




```
plotData(d, plot_type = "blocks")
```



6.2.2 Filtering

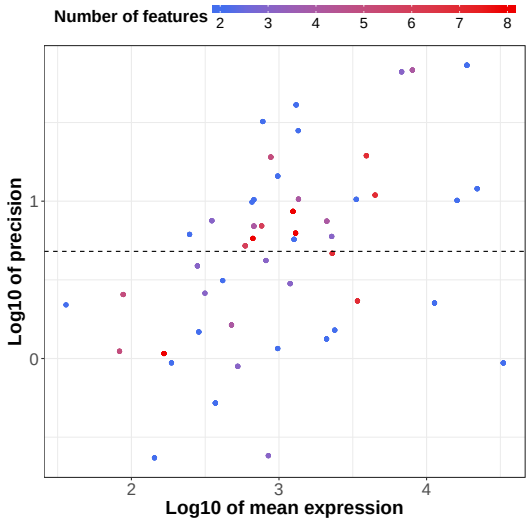
The filtering step eliminates genes and features with low expression, as in the differential transcript usage analysis (see section 5.2.2). Additionally, it filters out the SNPs/blocks that do not define at least two genotypes where each of them is present in at least `minor_allele_freq` individuals. Usually, `minor_allele_freq` is equal to roughly 5% of the total sample size.
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. However, for some genes, missing values may be present in the counts data, or genes may be lowly expressed in some samples. Setting up `min_samps_gene_expr` to 91 may exclude too many genes from the analysis. We can be slightly less stringent by taking, for example, `min_samps_gene_expr = 70`.

```
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)
```

6.2.3 Precision estimation

In the DTU analysis (section 5.2.3), the full model used in precision estimation has to be defined by the user. Here, full models are defined by genotypes. For a given SNP, genotype can have numeric values of 0, 1, and 2. When `one_way = TRUE`, 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 genotype values 0, 1, and 2.
For the tuQTL analysis, it has an additional parameter called `speed`. If `speed = FALSE`, gene-wise precisions are calculated for each gene-block. This calculation may take a long time, since there can be hundreds of SNPs/blocks per gene. If `speed` is set to `TRUE`, there will be only one precision calculated per gene (assuming a null model, i.e., model with intercept only), and it will be assigned to all the blocks matched to this gene. In the default setting, `speed = TRUE` and common precision is used as an initial value in the grid approach to estimate gene-wise precisions with NO moderation, since the sample size is quite large.
Again, this step of the pipeline is one of the most time consuming. Thus consider using `BPPARAM = BiocParallel::MulticoreParam()` with more than one worker when performing real data analysis.

```
## To make the analysis reproducible
set.seed(123)
## Calculate precision
d <- dmPrecision(d)
## !  Using a subset of 0.1 genes to estimate common precision !
## !  Using common_precision = 4.7973 as prec_init !
plotPrecision(d)
```



6.2.4 Proportion estimation

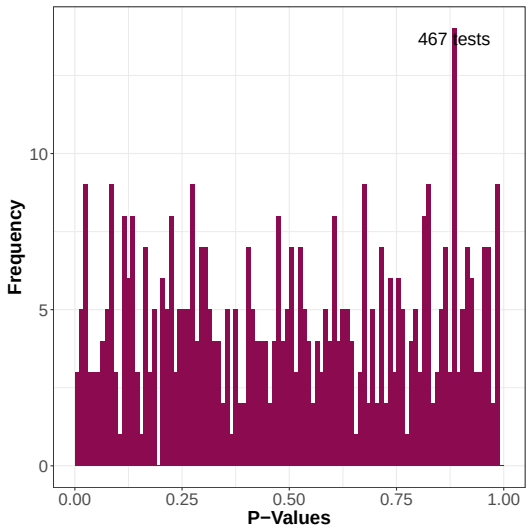
Dirichlet-multinomial full model proportions/coefficients and likelihoods are estimated for each gene-block pair. Currently, no transcript-level analysis are implemented in the tuQTL workflow.

```
d <- dmFit(d)
```

6.2.5 Testing for tuQTLs

`dmTest` function estimates gene-level null model proportions/coefficients and likelihoods and performs the likelihood ratio test. The null models equal to models with intercept only.
In contrast to the DTU analysis, there are some additional challenges that have to handled in the tuQTL analysis. They include a large number of tests per gene with highly variable allele frequencies (models) and linkage disequilibrium. As in other sQTL studies, we apply a permutation approach to empirically assess the null distribution of associations and use it for the adjustment of nominal p-values.
There are two permutation schemes available. When `permutation_mode` equals to `"all_genes"`, the null p-value distribution is calculated from all the genes. 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, which is also the default one.
The function `results` returns a data frame with likelihood ratio statistics, degrees of freedom, p-values and Benjamini and Hochberg adjusted p-values for each gene-block/SNP pair.

```
d <- dmTest(d)
plotPValues(d)
```

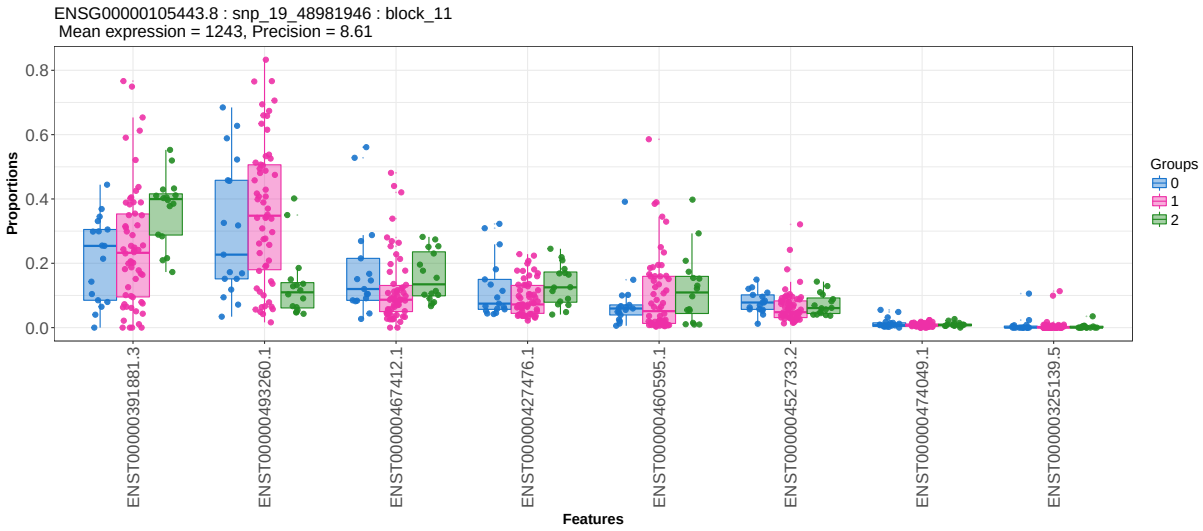



```
head(results(d))
##             gene_id block_id          snp_id       lr df    pvalue adj_pvalue
## 1 ENSG00000221983.2  block_3 snp_19_18678970 1.257939  2 0.4077350  0.9880683
## 2 ENSG00000221983.2  block_3 snp_19_18685964 1.257939  2 0.4077350  0.9880683
## 3 ENSG00000221983.2  block_3 snp_19_18687175 1.257939  2 0.4077350  0.9880683
## 4 ENSG00000221983.2  block_3 snp_19_18689904 1.257939  2 0.4077350  0.9880683
## 5 ENSG00000221983.2  block_4 snp_19_18679155 1.072372  2 0.4488788  0.9880683
## 6 ENSG00000221983.2  block_4 snp_19_18679379 1.072372  2 0.4488788  0.9880683
```

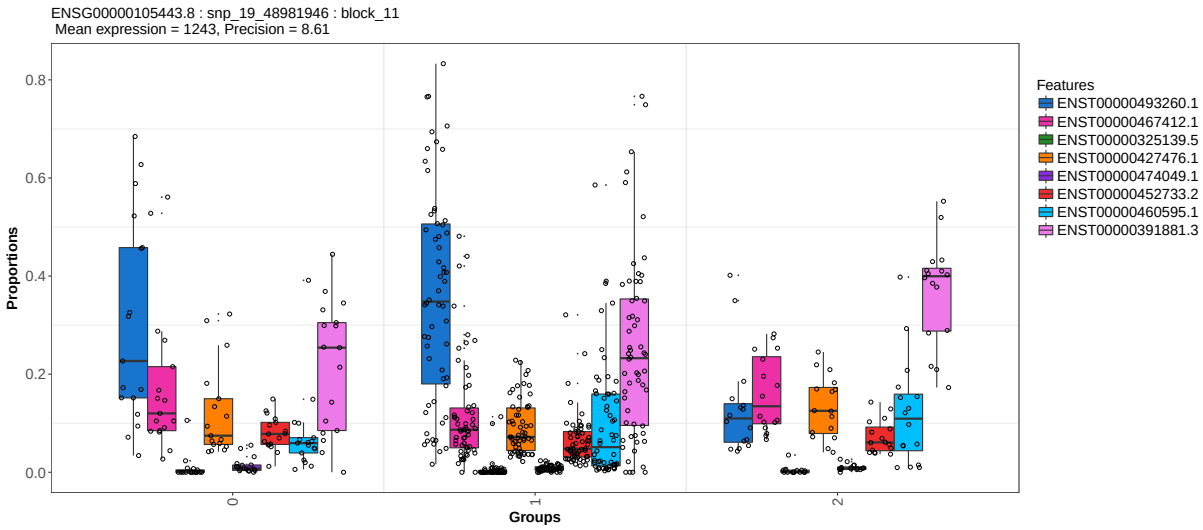
You can plot the observed transcript ratios for the tuQTLs of interest. Plotting the fitted values is not possible as we do not return this estimates due to their size. When the sample size is large, we recommend using box plots as a `plot_type`. We plot a tuQTL with the lowest p-value.

```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]
```

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

plotProportions(d, gene_id = top_gene_id, snp_id = top_snp_id)
```




```
plotProportions(d, gene_id = top_gene_id, snp_id = top_snp_id,
  plot_type = "boxplot2", order = FALSE)
```



7 Session information

```
sessionInfo()
## R Under development (unstable) (2016-12-23 r71840)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.5 LTS
##
## locale:
```

```
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] GeuvadisTranscriptExpr_1.3.0 ggplot2_2.2.0               DRIMSeq_1.3.3
## [4] PasillaTranscriptExpr_1.3.0  knitr_1.15.1                colorout_1.1-2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.8           compiler_3.4.0        GenomeInfoDb_1.11.6
##  [4] highr_0.6             plyr_1.8.4            XVector_0.15.0
##  [7] tools_3.4.0           zlibbioc_1.21.0       digest_0.6.10
## [10] evaluate_0.10         tibble_1.2            gtable_0.2.0
## [13] lattice_0.20-34       yaml_2.1.14           parallel_3.4.0
## [16] stringr_1.1.0         S4Vectors_0.13.5      IRanges_2.9.14
## [19] stats4_3.4.0          locfit_1.5-9.1        rprojroot_1.1
## [22] grid_3.4.0            BiocParallel_1.9.3    rmarkdown_1.3
## [25] limma_3.31.7          reshape2_1.4.2        magrittr_1.5
## [28] edgeR_3.17.5          MASS_7.3-45           backports_1.0.4
## [31] scales_0.4.1          htmltools_0.3.5       BiocGenerics_0.21.1
## [34] GenomicRanges_1.27.17 assertthat_0.1        BiocStyle_2.3.28
## [37] colorspace_1.3-2      labeling_0.3          stringi_1.1.2
## [40] lazyeval_0.2.0        munsell_0.4.3
```