# DRIMSeq: Dirichlet-multinomial framework for differential splicing and sQTL analyses in RNA-seq

Malgorzata Nowicka*, Mark Robinson

May 15, 2016

This vignette describes version 1.0.2 of the *DRIMSeq* package.

## Contents

## 1 Overview of Dirichlet-multinomial model

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 splicing or sQTL analysis where the multivariate features are transcripts or exons of a gene. Depending on the type of counts that are used in the analysis, you can test for differential splicing at the level of transcript or exon ratio changes. The implementation of Dirichlet-multinomial model in *DRIMSeq* package is customized for differential splicing and sQTL analyses, but the data objects used in *DRIMSeq* can contain different types of counts. Therefore, other types of multivariate differential analyses between groups 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 dispersion, i.e., the variability of feature ratios between samples (replicates) within conditions. Dispersion is needed in order to find the significant changes in feature ratios between conditions which should be sufficiently stronger than the changes/variability within conditions. Second, we use maximum likelihood to estimate the full model (estimated in every group/condition separately) and null model (estimated from all data) proportions and its corresponding likelihoods. Finally, we use the likelihood ratio statistics to test for the differences between feature proportions in different groups to identify the differentially spliced genes (differential splicing analysis) or the sQTLs (sQTL analysis).

## 2 Hints for DRIMSeq pipelines

In this vignette, we present how one could perform differential splicing analysis and sQTL analysis with the *DRIMSeq* package. We use small data sets so that the whole pipelines can be run within few minutes in *R*

*gosia.nowicka@uzh.ch

on a single core computer. In practice, the package is designed to take advantage of multicore computing for larger data sets.

Both pipelines consist of the initial steps where objects containing the data for the analysis are initiated and then filtered. Functions used for this purpose, such as `dmDSdata` or `dmSQTLdata` and `dmFilter`, have some parameters (like `counts`, `gene_id`, `min_samps_gene_expr`, etc.) for which no default values are predefined. These parameters must be specified by user in order to proceed with the pipeline.

Functions `dmDispersion`, `dmFit` and `dmTest`, which perform the actual statistical analyses described above, require that only one parameter `x` containing the data is specified by user. These functions have many other parameters available for tweaking, but they do have default values, which were chosen based on many real data analyses.

Some of the steps are quite time consuming, especially the dispersion estimation, where proportions of each gene are refitted for different dispersion 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, sQTL analyses are more computationally intensive than differential splicing analysis because one needs to do the analysis for every SNP in the surrounding region of a gene. It is indeed feasible to perform sQTL analysis for small chunks of genome, for example, per chromosome.

# 3 Differential splicing analysis work-flow

## 3.1 Example data

To demonstrate the usage of *DRIMSeq* in differential splicing 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 splicing 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.

## 3.2 Differential splicing analysis with DRIMSeq package

In order to do the analysis, we create a *dmDSdata* object, which contains feature 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 dispersion estimates, proportions estimates, likelihood ratio statistics, p-values, adjusted p-values. As new elements are added, the object also changes its name *dmDSdata* $\rightarrow$ *dmDSdispersion* $\rightarrow$ *dmDSfit* $\rightarrow$ *dmDStest*, but each container inherits slots and methods available for the previous one.

### 3.2.1 Loading pasilla data into R

The 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")

metadata <- read.table(file.path(data_dir, "metadata.txt"), header = TRUE,
  as.is = TRUE)
metadata
##     LibraryName LibraryLayout SampleName condition
## 1   Untreated-1        SINGLE  GSM461176       CTL
## 2   Untreated-3        PAIRED  GSM461177       CTL
## 3   Untreated-4        PAIRED  GSM461178       CTL
## 4 CG8144_RNAi-1        SINGLE  GSM461179        KD
## 5 CG8144_RNAi-3        PAIRED  GSM461180        KD
## 6 CG8144_RNAi-4        PAIRED  GSM461181        KD
## 7   Untreated-6        SINGLE  GSM461182       CTL
counts <- read.table(file.path(data_dir, "counts.txt"), header = TRUE,
  as.is = TRUE)
head(counts)
##    feature_id     gene_id   GSM461176   GSM461177   GSM461178 GSM461179   GSM461180
## 1 FBtr0300689 FBgn0031208 0.00000e+00 0.00000e+00 0.00000e+00  27.04866 0.00000e+00
## 2 FBtr0300690 FBgn0031208 5.01688e+00 4.00000e+00 1.00518e+00   0.00000 2.00000e+00
## 3 FBtr0330654 FBgn0031208 0.00000e+00 0.00000e+00 0.00000e+00   0.00000 0.00000e+00
## 4 FBtr0078166 FBgn0002121 1.19845e+02 1.65861e-06 9.54854e-02 685.96100 1.37563e-03
## 5 FBtr0078167 FBgn0002121 6.13896e-04 3.66545e-04 6.66691e-01 420.15300 1.44138e+00
## 6 FBtr0078168 FBgn0002121 9.76956e+01 2.73017e-06 5.50984e-06 279.84050 1.32354e-05
##     GSM461181   GSM461182
## 1 0.00000e+00 0.00000e+00
## 2 2.00464e+00 0.00000e+00
## 3 0.00000e+00 0.00000e+00
## 4 6.25907e+01 1.86834e+02
## 5 9.81068e-07 5.66896e-05
## 6 0.00000e+00 1.51869e+02
```

Load the *DRIMSeq* package.

```
library(DRIMSeq)
```

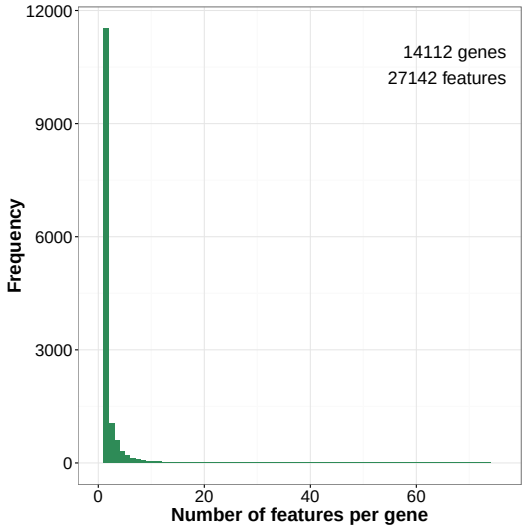
Create a `dmDSdata` object (saved as variable `d`), which contains counts and information about samples such as sample IDs and a variable `group` defining the experimental groups/conditions. Make sure that samples in `counts` have the same order as in `metadata`. When printing variable `d`, you can see its class, size and which accessor methods can be applied. For `dmDSdata` object, there are two methods that return data frames with counts and samples.

```
d <- dmDSdata(counts = counts[, metadata$SampleName], gene_id = counts$gene_id,
  feature_id = counts$feature_id, sample_id = metadata$SampleName,
```

```
  group = metadata$condition)
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 GSM461182 GSM461179 GSM461180
## 1 FBgn0031208 FBtr0300689   0.00000         0   0.00000         0  27.04866         0
## 2 FBgn0031208 FBtr0300690   5.01688         4   1.00518         0   0.00000         2
## 3 FBgn0031208 FBtr0330654   0.00000         0   0.00000         0   0.00000         0
##   GSM461181
## 1   0.00000
## 2   2.00464
## 3   0.00000
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 features per gene. For example, there are genes that have 74 isoforms.

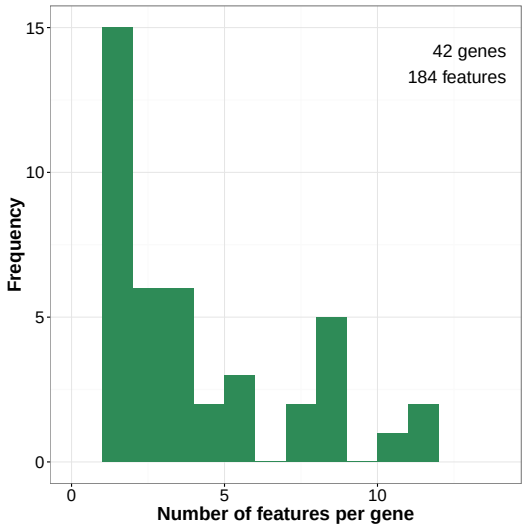
```
plotData(d)
```




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()
```

```
plotData(d)
```




After subsetting, `d` contains counts for 42 genes.

### 3.2.2 Filtering

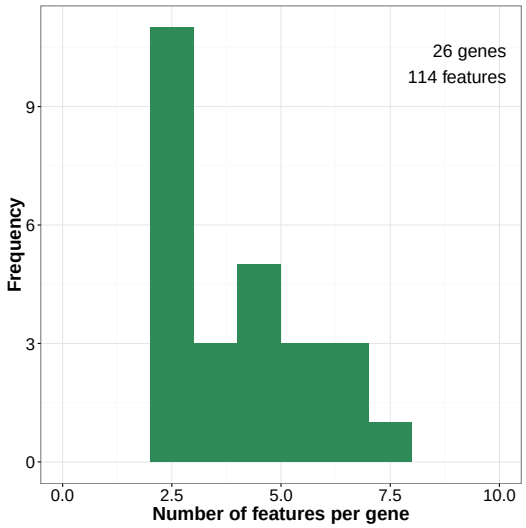
Genes may have many transcripts or exons that are lowly expressed or not expressed at all. You can remove them using the `dmFilter` function. Filtering of lowly expressed features can be done at two levels: minimal *expression* using `min_samps_feature_expr` and `min_feature_expr` parameters or minimal *proportion* with `min_samps_feature_prop` and `min_feature_prop`.

In the *pasilla* experiment we use a filtering based only on the feature 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 feature (here, a transcript) is expressed in one condition but not in another, which is a case of differential splicing. The level of feature expression is controlled by `min_feature_expr`. Our default value is set up to 10, which means that only the features 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 feature 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_samps_feature_prop = 0)
```

```
plotData(d)
```




### 3.2.3 Dispersion estimation

Ideally, we would like to get accurate dispersion estimates for every gene, which is problematic when analyzing small data sets because dispersion 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 dispersion and based on all the data, we could calculate a common dispersion, but we expect this to be too strong of an assumption. Moderated dispersion is a trade-off between gene-wise and common dispersion. 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 dispersion and gene-wise dispersions. In the default setting, all of them are computed and common dispersion is used as an initial value in the grid approach to estimate moderated gene-wise dispersions, which are shrunk toward the common dispersion.

This step of our pipeline is the most time consuming. Thus consider using `BPPARAM = BiocParallel::MulticoreParam()` with more than one worker.

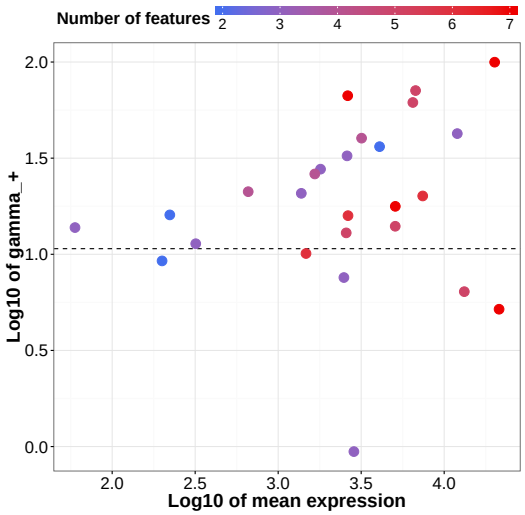
```
d <- dmDispersion(d, verbose = 1, BPPARAM = BiocParallel::SerialParam())
## * Calculating mean gene expression..
## Took 0 seconds.
## * Estimating common dispersion..
## Took 11.07 seconds.
## !  Using common_dispersion = 10.7 as disp_init !
## * Estimating genewise dispersion..
## Took 5.17 seconds.
```

```
d
## An object of class dmDSdispersion
## with 26 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
head(mean_expression(d), 3)
##       gene_id mean_expression
## 1 FBgn0000256        2622.286
## 2 FBgn0020309       13217.714
## 3 FBgn0259735       11992.903
common_dispersion(d)
## [1] 10.69633
head(genewise_dispersion(d), 3)
##       gene_id genewise_dispersion
## 1 FBgn0000256           66.824633
## 2 FBgn0020309            6.394084
## 3 FBgn0259735           42.450876
```

To inspect the behavior of dispersion estimates, you can plot them against the mean gene expression. Here, the effect of shrinking is not easily visible because our data set is very small.

If `out_dir = NULL` in any of the plotting functions from *DRIMSeq* package, a *ggplot* object is returned, and it can be further modified using *ggplot2* package. Here, we increase the size of points.

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




### 3.2.4 Proportion estimation

In this step, we estimate the full model proportions, meaning, that transcript or exon proportions are estimated for each condition separately. You can access this estimates and the corresponding statistics, such as log-

likelihoods, with `proportions` and `statistics` functions, respectively.

```
d <- dmFit(d, BPPARAM = BiocParallel::SerialParam())
d
## An object of class dmDSfit
## with 26 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
##   proportions(), statistics()
head(proportions(d), 3)
##       gene_id  feature_id         CTL          KD
## 1 FBgn0000256 FBtr0290077 0.355151653 0.077817640
## 2 FBgn0000256 FBtr0290078 0.044908202 0.268952808
## 3 FBgn0000256 FBtr0290082 0.007009802 0.002253173
head(statistics(d), 3)
##       gene_id   lik_CTL    lik_KD
## 1 FBgn0000256 -11836.78 -12928.59
## 2 FBgn0020309 -37232.88 -59448.10
## 3 FBgn0259735 -10730.32 -19654.76
```

### 3.2.5 Testing for differential splicing

Calling the `dmTest` function results in two calculations. First, null model proportions, i.e., feature ratios based on pooled (no grouping into conditions) counts, are estimated. Second, likelihood ratio statistics are used to test for the difference between feature proportions in different groups to identify the differentially spliced genes.

By default, `compared_groups` parameter in `dmTest` indicates all the levels of grouping variable, which means that we test for differences in splicing between any of the groups. In the *pasilla* example, there are only two conditions, and there is only one comparison that can be done. In the case where grouping variable has more that two levels, you could be interested in the pair-wise comparisons, which you can specify with `compared_groups` parameter.

Now, if you call `proportions` or `statistics` function, results of null estimation are added to the previous data frames.

```
d <- dmTest(d, verbose = 1, BPPARAM = BiocParallel::SerialParam())
## Running comparison between groups:  CTL, KD
## * Fitting null model..
## Took 0.12 seconds.
## * Calculating likelihood ratio statistics..
## Took 0.0019829273223877 seconds.
d
## An object of class dmDStest
## with 26 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
```

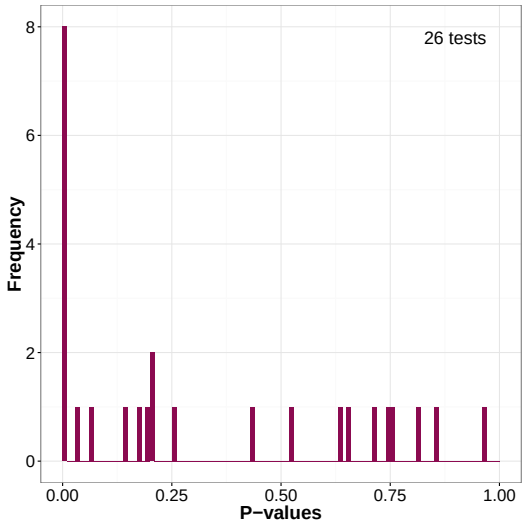
```
##   proportions(), statistics()
##   results()
head(proportions(d), 3)
##       gene_id  feature_id         CTL          KD        null
## 1 FBgn0000256 FBtr0290077 0.355151653 0.077817640 0.207148460
## 2 FBgn0000256 FBtr0290078 0.044908202 0.268952808 0.106036514
## 3 FBgn0000256 FBtr0290082 0.007009802 0.002253173 0.004951749
head(statistics(d), 3)
##       gene_id       CTL        KD  lik_null df
## 1 FBgn0000256 -11836.78 -12928.59 -24818.03  6
## 2 FBgn0020309 -37232.88 -59448.10 -96690.27  4
## 3 FBgn0259735 -10730.32 -19654.76 -30385.36  2
```

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 adjusted p-values for each gene.

```
head(results(d), 3)
##       gene_id          lr df       pvalue   adj_pvalue
## 1 FBgn0000256 105.3224556  6 1.940713e-20 5.045854e-19
## 2 FBgn0020309  18.5696496  4 9.546600e-04 3.545880e-03
## 3 FBgn0259735   0.5526324  2 7.585730e-01 8.575173e-01
```

You can plot a histogram of p-values.

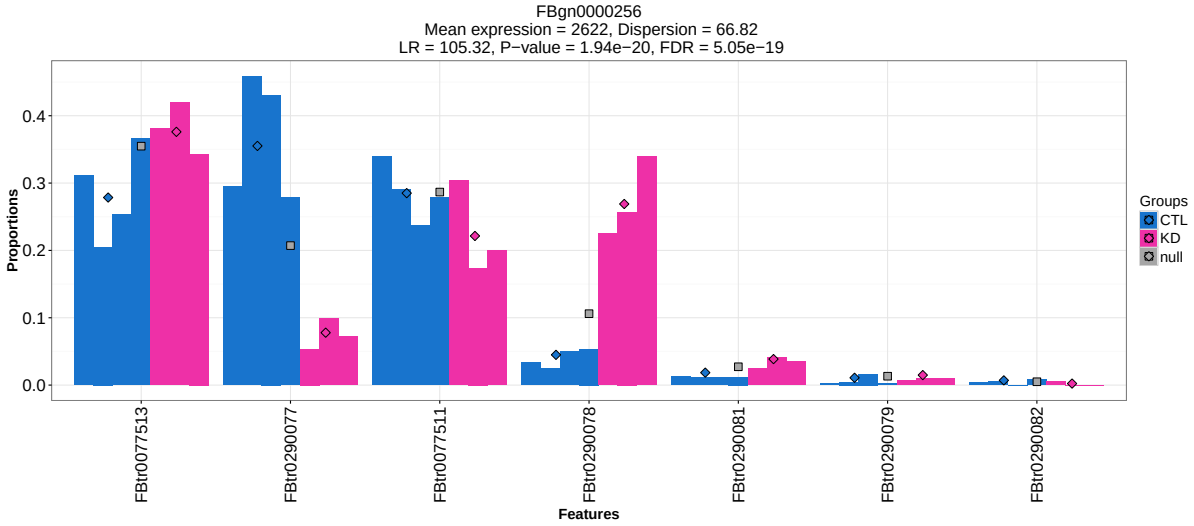
```
plotTest(d)
```



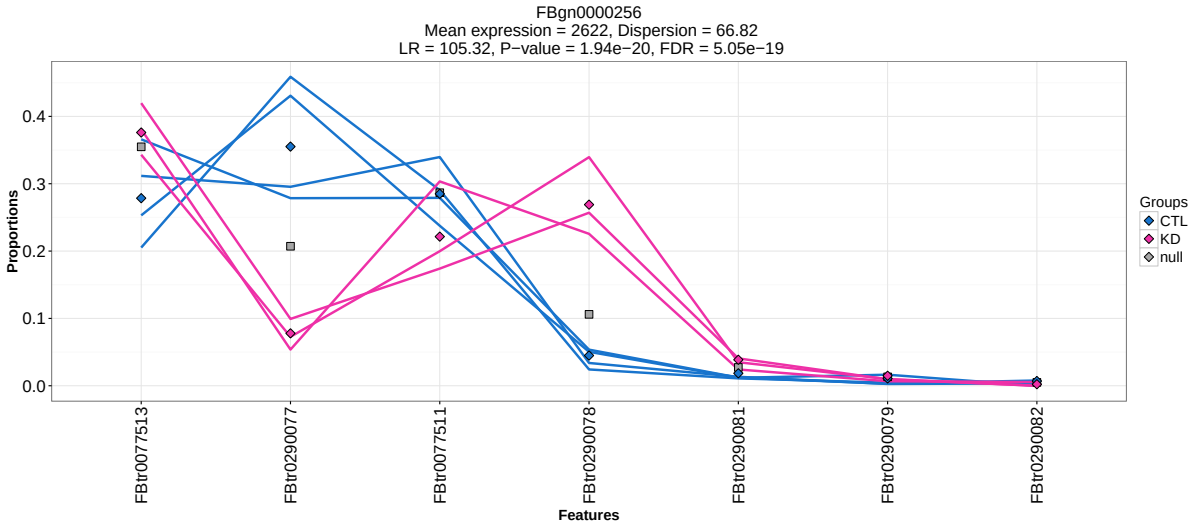

For genes of interest, you can make plots (bar plots, line plots, box plots, ribbon plots) of observed and estimated with Dirichlet-multinomial model feature ratios. Estimated proportions are marked with diamond shapes. Here, we plot the results for the top significant gene.

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

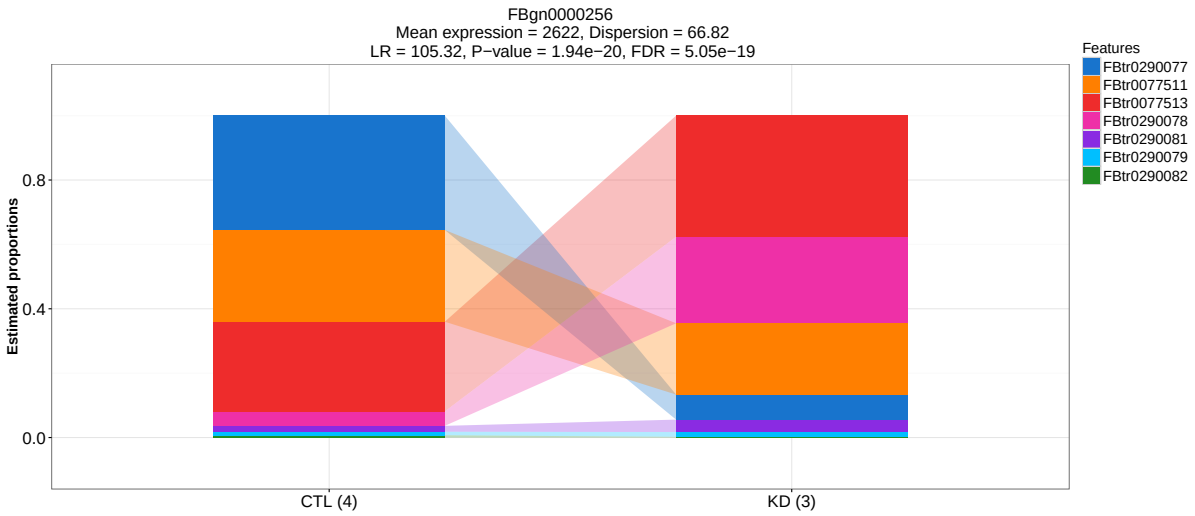
```
plotFit(d, gene_id = gene_id)
```




```
plotFit(d, gene_id = gene_id, plot_type = "lineplot")
```




```
plotFit(d, gene_id = gene_id, plot_type = "ribbonplot")
```




# 4 sQTL analysis work-flow

In sQTL analysis, we want to identify genetic variants (here, bi-allelic SNPs) that are associated with changes in splicing. Such SNPs are then called splicing quantitative trait locies (sQTLs).

Ideally, we would like to test associations of every SNP with every gene. However, such an approach would be very costly computationally and in terms of multiple testing correction. Under the assumption that SNPs that directly affect splicing 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.

## 4.1 Example data

To demonstrate the sQTL 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.

## 4.2 sQTL analysis with 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* work-flow for sQTL analysis is the same as for differential splicing.

First, we have to create a *dmSQTLdata* object, which contains feature counts and genotypes. Similarly as in differential splicing pipeline, results from every step are added to this object and at the end of the analysis, it contains dispersion estimates, proportions estimates, likelihood ratio statistics, p-values, adjusted p-values. As new elements are added, the object also changes its name *dmSQTLdata* → *dmSQTLdispersion* → *dmSQTLfit* → *dmSQTLtest*. For each object, slots and methods are inherited from the previous one.

### 4.2.1 Loading GEUVADIS data into R

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

```
library(GeuvadisTranscriptExpr)

counts <- GeuvadisTranscriptExpr::counts
genotypes <- GeuvadisTranscriptExpr::genotypes
gene_ranges <- GeuvadisTranscriptExpr::gene_ranges
snp_ranges <- GeuvadisTranscriptExpr::snp_ranges
```

Load the *DRIMSeq* package.

```
library(DRIMSeq)
```

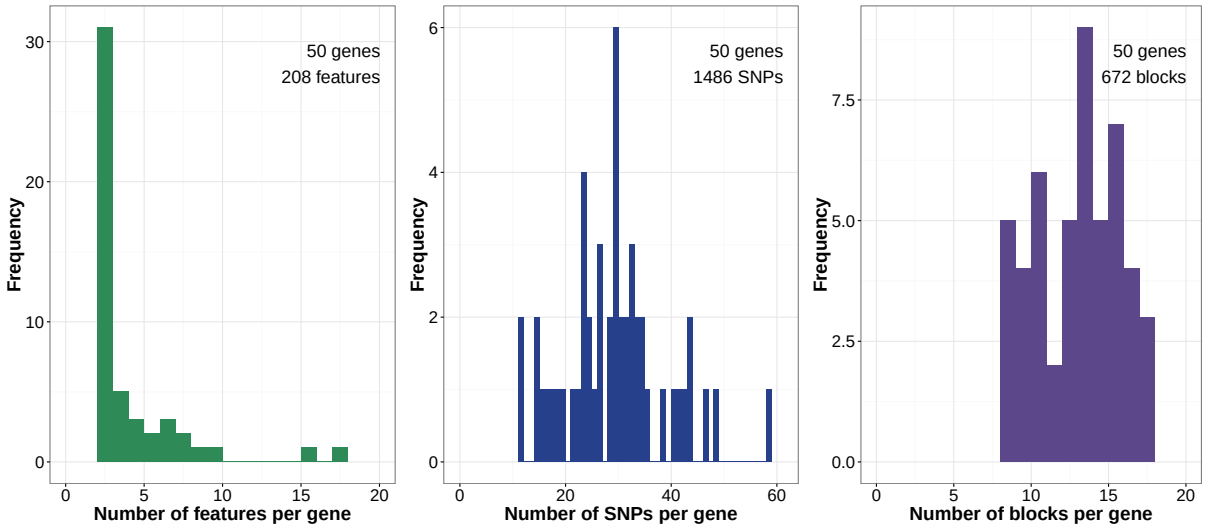
In the sQTL analysis, an initial data object `d` is of `dmSQTLdata` class and, additionally to feature counts, it contains genotypes of SNPs that are in some surrounding of genes. This surrounding is defined with the parameter `window`. In order to find out which SNPs should be tested with which genes we can use the `dmSQTLdataFromRanges` functions which requires as an input the location of genes and SNPs stored as *GRanges* objects. In the case you already know the match of SNPs and genes, you can use the `dmSQTLdata` function.

```
# Make sure that samples in counts and genotypes are in the same order
sample_id <- colnames(counts[, -(1:2)])

d <- dmSQTLdataFromRanges(counts = counts[, sample_id],
  gene_id = counts$Gene_Symbol, feature_id = counts$TargetID,
  gene_ranges = gene_ranges, genotypes = genotypes[, sample_id],
  snp_id = genotypes$snpId, snp_ranges = snp_ranges, sample_id = sample_id,
  window = 5e3, BPPARAM = BiocParallel::SerialParam())
d
## An object of class dmSQTLdata
## with 50 genes and 91 samples
```

In our sQTL 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.

Here, the data summary plot produces three histograms: the number of features per gene, the number of SNPs per gene and the number of blocks per gene. Total number of tests done in this analysis is equal to the total number of blocks. You can use the `multiplot` function from http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/ to plot this three figures next to each other.

```
multiplot(plotlist = plotData(d), cols = 3)
```

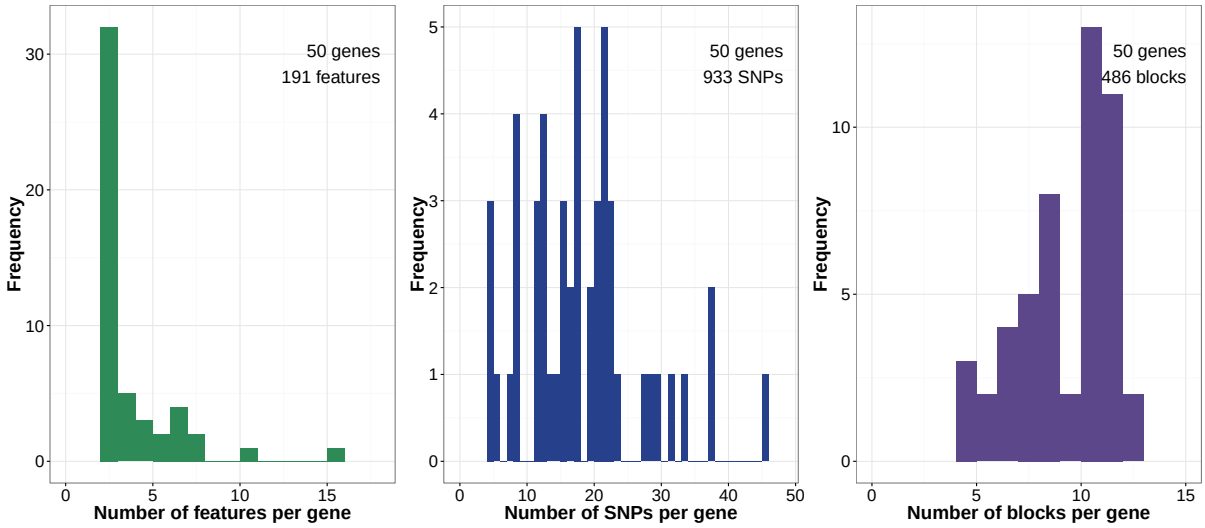



### 4.2.2 Filtering

The filtering step eliminates genes and features with low expression, as in the differential splicing analysis (see section **??**). 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 more or less 5% of 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 would exclude too many genes. 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,
  min_samps_feature_prop = 0, minor_allele_freq = 5,
  BPPARAM = BiocParallel::SerialParam())

multiplot(plotlist = plotData(d), cols = 3)
```




### 4.2.3 Dispersion estimation

As for differential splicing (see section **??**), `dmDispersion` may calculate three values: mean expression of genes, common dispersion and gene-wise dispersions. It has an additional parameter `speed`. If `speed = FALSE`, gene-wise dispersions are calculated per 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 dispersion calculated per gene and it will be assigned to all blocks matched to this gene.

In the default setting, `speed = TRUE` and common dispersion is used as an initial value in the grid approach to estimate gene-wise dispersions with NO moderation, since the sample size is quite big.

Again, this step of our pipeline is the most time consuming. Thus consider using `BPPARAM = BiocParallel::MulticorePa` with more than one worker when performing real data analysis.

```
d <- dmDispersion(d, verbose = 1, BPPARAM = BiocParallel::SerialParam())
```

```
## * Calculating mean gene expression..
```

```
## Took 0 seconds.
```

```
## * Estimating common dispersion..
```

```
## Took 12.19 seconds.
```

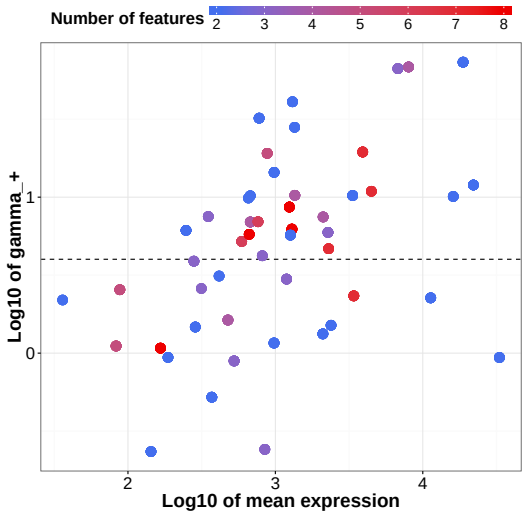
```
## !  Using common_dispersion = 4 as disp_init !
```

```
## * Estimating genewise dispersion..
```

```
## Took 7.85 seconds.
```

```
d
## An object of class dmSQTLdispersion
## with 50 genes and 91 samples
```

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




### 4.2.4 Proportion estimation

Full model proportions are estimated for each gene-block pair.

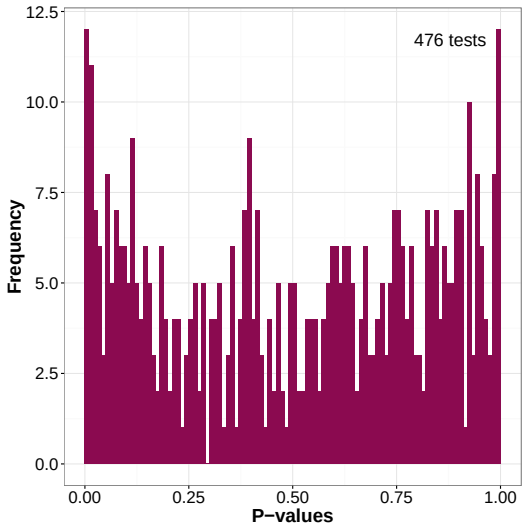
```
d <- dmFit(d, BPPARAM = BiocParallel::SerialParam())
d
## An object of class dmSQTLfit
## with 50 genes and 91 samples
```

### 4.2.5 Testing for sQTLs

As in section **??**, `dmTest` function fits null model proportions and performs the likelihood ratio test. 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.

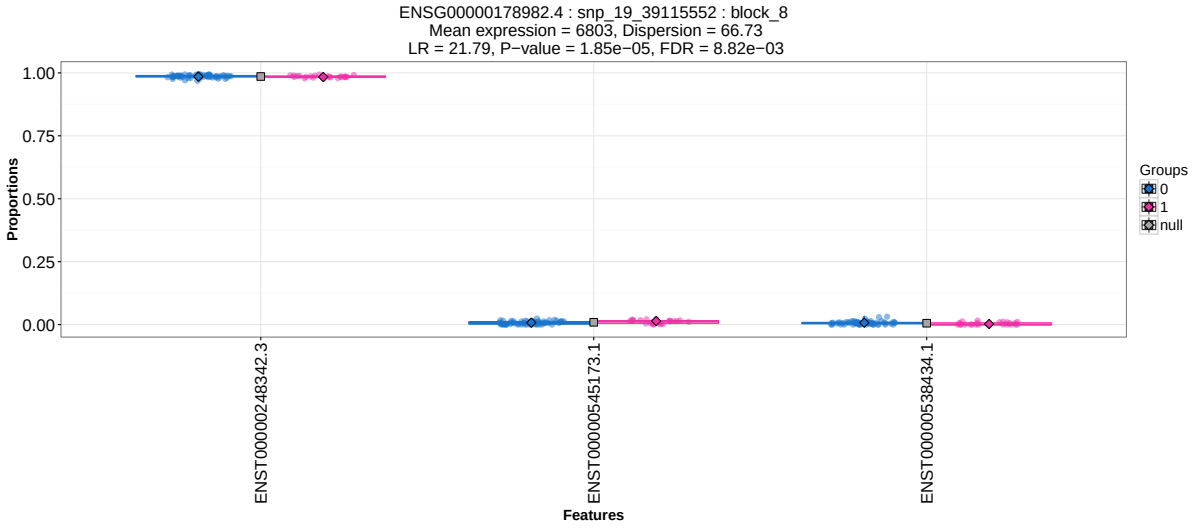
```
d <- dmTest(d, verbose = 1, BPPARAM = BiocParallel::SerialParam())
## * Fitting null model..
## Took 3.69 seconds.
## * Calculating likelihood ratio statistics..
## Took 0.0583069324493408 seconds.
d
## An object of class dmSQTLtest
## with 50 genes and 91 samples
## * data accessors: results()
head(results(d), 3)
##               gene_id block_id          snp_id       lr df    pvalue adj_pvalue
## 1   ENSG00000221983.2  block_3 snp_19_18678970 1.262112  2 0.5320296   0.999976
## 1.1 ENSG00000221983.2  block_3 snp_19_18685964 1.262112  2 0.5320296   0.999976
## 1.2 ENSG00000221983.2  block_3 snp_19_18687175 1.262112  2 0.5320296   0.999976
```

```
plotTest(d)
```

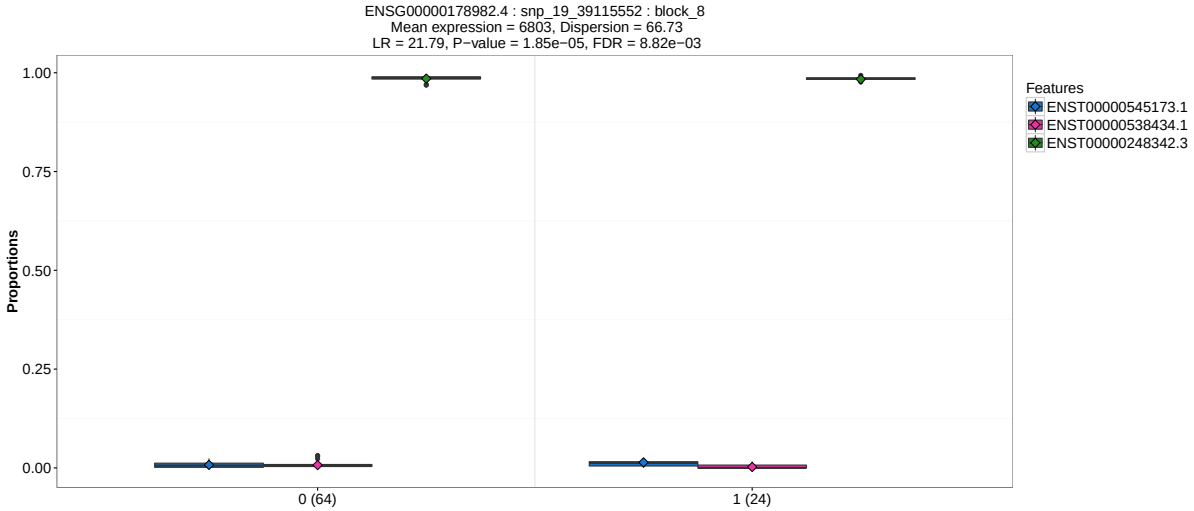



You can plot the observed and estimated Dirichlet-multinomial model feature ratios for the sQTLs of interest. When the sample size is large, we recommend using box plots as a `plot_type`. Here, we plot an sQTL with the lowest p-value.

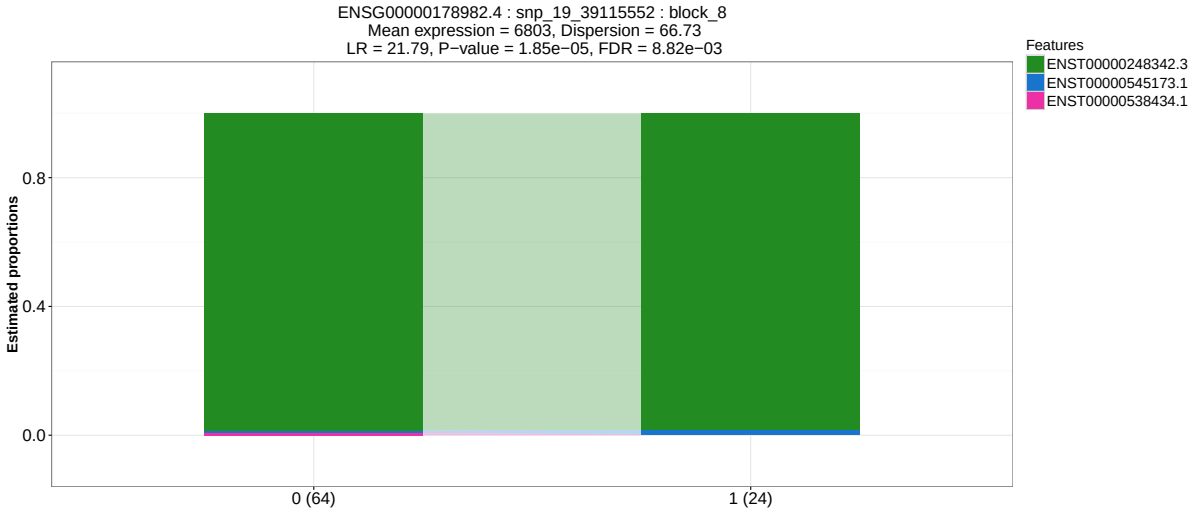
```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]

gene_id <- res$gene_id[1]
snp_id <- res$snp_id[1]

plotFit(d, gene_id, snp_id)
```




```
plotFit(d, gene_id, snp_id, plot_type = "boxplot2", order = FALSE)
```




```
plotFit(d, gene_id, snp_id, plot_type = "ribbonplot")
```



# APPENDIX

## A Session information

```
sessionInfo()
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] GeuvadisTranscriptExpr_1.0.0 ggplot2_2.1.0                DRIMSeq_1.0.2
## [4] PasillaTranscriptExpr_1.0.0  knitr_1.13
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.5          XVector_0.12.0       magrittr_1.5         edgeR_3.14.0
##  [5] GenomicRanges_1.24.0 BiocGenerics_0.18.0  zlibbioc_1.18.0      IRanges_2.6.0
##  [9] munsell_0.4.3        BiocParallel_1.6.2   colorspace_1.2-6     plyr_1.8.3
## [13] stringr_1.0.0        highr_0.6            GenomeInfoDb_1.8.2   tools_3.3.0
## [17] parallel_3.3.0       gtable_0.2.0         digest_0.6.9         reshape2_1.4.1
## [21] formatR_1.4          S4Vectors_0.10.0     evaluate_0.9         labeling_0.3
## [25] limma_3.28.4         stringi_1.0-1        scales_0.4.0         stats4_3.3.0
## [29] BiocStyle_2.0.2
```

## B References