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

Malgorzata Nowicka*, Mark Robinson

December 11, 2015

This vignette describes version 0.3.3 of the *DRIMSeq* package.

Contents

1 Overview of Dirichlet-multinomial model **2**

2 Hints for DRIMSeq pipelines **2**

3 Differential splicing analysis work-flow **2**
3.1 Example data . . . 3
3.2 Differential splicing analysis with DRIMSeq package . . . 3
3.2.1 Loading pasilla data into R . . . 3
3.2.2 Filtering . . . 6
3.2.3 Dispersion estimation . . . 7
3.2.4 Proportions estimation . . . 8
3.2.5 Testing for differential splicing . . . 9

4 sQTL analysis work-flow **12**
4.1 Example data . . . 12
4.2 GEUVADIS data preprocessing . . . 12
4.3 sQTL analysis with DRIMSeq package . . . 17
4.3.1 Loading GEUVADIS data into R . . . 17
4.3.2 Filtering . . . 18
4.3.3 Dispersion estimation . . . 19
4.3.4 Proportions estimation . . . 20
4.3.5 Testing for sQTLs . . . 20

APPENDIX **24**

A Session information **24**

B References **24**

*gosia.nowicka@uzh.ch

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 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 *DRIMSeq* can contain different types of counts, and thus, other types of multivariate differential analysis between groups can be performed.

The statistical details of *DRIMSeq* are presented in our paper *fixme: citation to DRIMSeq paper*. 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 features ratios between conditions which should be sufficiently stronger than the changes/variability within conditions. Second, we use the 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 statistic 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 *DRIMSeq* package. We use small subsets of data so that you can run the whole pipelines within few minutes in *R* on a single core computer.

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 analysis 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 analysis.

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 implemented in many functions a parallelization approach from *BiocParallel*. Thus, if possible, we recommend to increase the number of workers in `BPPARAM`.

3 Differential splicing analysis work-flow

3.1 Example data

To demonstrate the usage of *DRIMSeq* in differential splicing analysis, we will use a *pasilla* data set produced by Brooks et al. [1]. The aim of their study was to identify exons that are regulated by *pasilla* protein, the Drosophila melanogaster ortholog of mammalian NOVA1 and NOVA2 which are one of the 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 the *HTSeq* counts available in *pasilla* package, where you can find all the steps needed, for processing the GEO data, to get a table with exonic bin counts.

3.2 Differential splicing analysis with DRIMSeq package

In oder to do the analysis, we have to 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. So that, 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 $dmDSdata -> dmDSdispersion -> dmDSfit -> dmDStest$, but every following one inherits slots and methods available for the previous one.

3.2.1 Loading pasilla data into R

The counts obtained from *HTSeq* are saved in separate text files for each sample. We want to put them together into one data frame.

```
library(pasilla)

data_dir  <- system.file("extdata", package="pasilla")
count_files <- list.files(data_dir, pattern="fb.txt$", full.names=TRUE)
count_files
## [1] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/treated1fb.txt"
## [2] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/treated2fb.txt"
## [3] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/treated3fb.txt"
## [4] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/untreated1fb.txt"
## [5] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/untreated2fb.txt"
## [6] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/untreated3fb.txt"
## [7] "/home/gosia/R/libraries/BioCdevel/pasilla/extdata/untreated4fb.txt"
## Read the HTSeq files into R
htseq_list <- lapply(1:length(count_files), function(i){
  htseq <- read.table(count_files[i], header = FALSE, as.is = TRUE)
  colnames(htseq) <- c("group_id", gsub("fb.txt", "", strsplit(count_files[i],
    "extdata/")[[1]][2]))
  return(htseq)
})

## Merge them into one data frame
```

```
htseq_counts <- Reduce(function(...) merge(..., by = "group_id", all=TRUE,
  sort = FALSE), htseq_list)

## Remove the summary elements
tail(htseq_counts)
##              group_id treated1 treated2 treated3 untreated1 untreated2 untreated3
## 70462 FBgn0261575:001        2        0        0          0          1          0
## 70463 FBgn0261575:002       10        1        3          5          7          1
## 70464      _ambiguous     1317        0        0       1096       1144          0
## 70465          _empty 11173758 16686814 16057634    7756745   12150698   13990064
## 70466       _lowaqual 67756093        0        0   22017200   42760359          0
## 70467     _notaligned        0        0        0          0          0          0
##       untreated4
## 70462          0
## 70463          0
## 70464          0
## 70465   14248758
## 70466          0
## 70467          0
htseq_counts <- htseq_counts[!grepl(pattern = "_", htseq_counts$group_id), ]
```

The exon bin IDs produced by *HTSeq* consist of the gene IDs and the bin number separated by colon. To create the *dmDSdata* object, we need a vector of gene IDs and a vector of feature IDs, here bin IDs.

```
group_split <- limma::strsplit2(htseq_counts[, 1], ":")
```

Finally, load the *DRIMSeq* package.

```
suppressPackageStartupMessages(library(DRIMSeq))
```

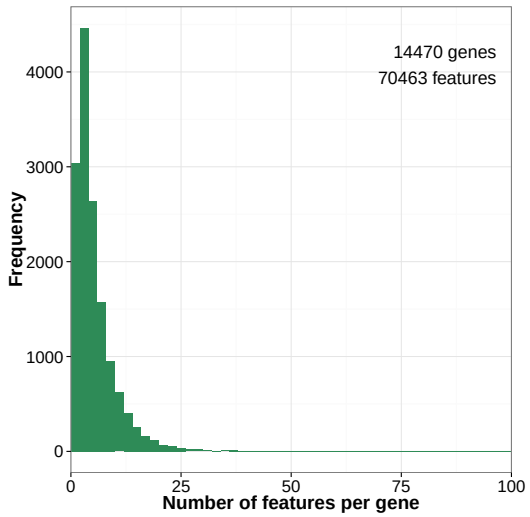
Create a `dmDSdata` object (saved as variable `d`), which contains counts and information about samples such as sample IDs and a variable `group` defining the experimental groups/conditions. When printing variable `d`, you can see its class, size and which accessor methods can be applied. For `dmDSdata`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

```
d <- dmDSdata(counts = htseq_counts[, -1], gene_id = group_split[, 1],
  feature_id = group_split[, 2], sample_id = colnames(htseq_counts)[-1],
  group = gsub("[1-4]", "", colnames(htseq_counts)[-1]))
d
## An object of class dmDSdata
## with 14470 genes and 7 samples
## * data accessors: counts(), samples()
head(counts(d), 3)
##       gene_id feature_id treated1 treated2 treated3 untreated1 untreated2 untreated3
## 1 FBgn0000003        001        0        0        1          0          0          0
## 2 FBgn0000008        001        0        0        0          0          0          0
## 3 FBgn0000008        002        0        0        0          0          0          1
##   untreated4
## 1          0
```

```
## 2          0
## 3          0
head(samples(d), 3)
##   sample_id   group
## 1  treated1 treated
## 2  treated2 treated
## 3  treated3 treated
```

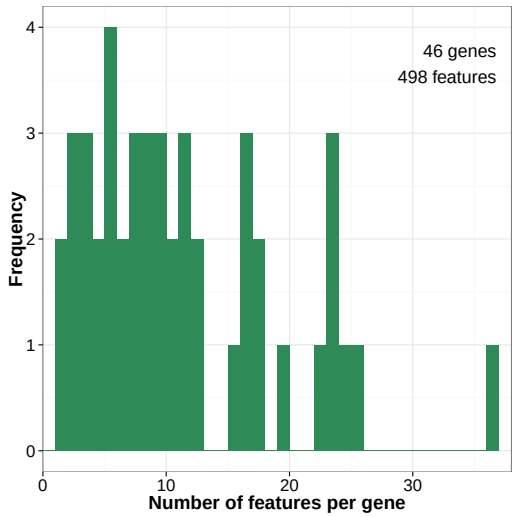
You can also make a data summary plot, which is a histogram of the number of features per gene. There are genes, that have nearly 100 of exonic bins.

```
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.

```
genes_subset = readLines(file.path(data_dir, "geneIDsinsubset.txt"))
d <- d[names(d) %in% genes_subset, ]
d
## An object of class dmDSdata
## with 46 genes and 7 samples
## * data accessors: counts(), samples()
plotData(d)
```



After subsetting, `d` contains counts for 46 genes.

3.2.2 Filtering

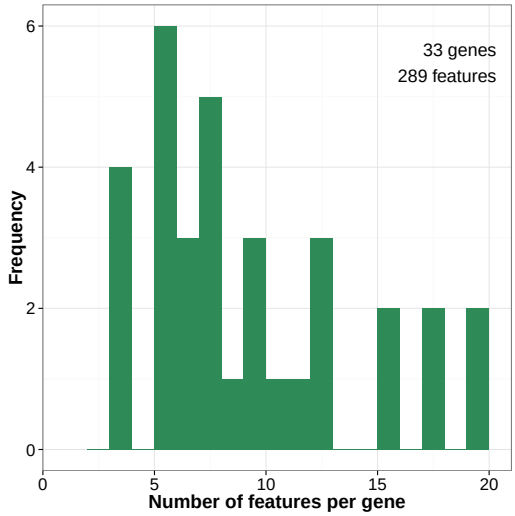
Filtering of genes and features with low expression improves the performance of Dirichlet-multinomial model. Genes may have many features that are expressed very lowly (so that even if there was any differential expression, it would not be biologically significant) or not expressed at all. Removing such features from the analysis has two effects. First, it reduces the number of parameters (proportions) to estimate. Second, improves the estimation of dispersion *fixme: Do simulation*. However, too stringent filtering may remove genes and features where the differential changes happen, and, in consequence, lead to loss of power.

Thus, finding an optimal filter is a challenge. In differential splicing analysis, we recommend to adjust the filtering parameters according to the number of replicates per condition. At the feature level, we suggest using `min_samps_feature_prop` equal to the minimal number of replicates in any of the conditions. For example, in *pasilla* assay with 3 knock-down and 4 control samples, we would set this parameters to 3. Like this, we allow a situation where a feature (here, an exonic bin) is expressed in one condition but may not be expressed at all in another one. The level of feature expression is controlled by `min_feature_prop`. Our default value is set up to 0.01, which means that only the features with ratio of at least 1% in 3 samples are kept.

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, a minimal filtering for the gene expression removes the genes with mostly zero counts and reduces the number of tests in multiple test correction. The default value that we propose is `min_samps_gene_expr = 3` and `min_gene_expr = 1`, which means that only genes that are expressed at the level of 1 cpm in at least 3 samples are kept for the downstream analysis.

```
# Check what is the minimal number of replicates per condition
table(samples(d)$group)
##
##   treated untreated
##         3         4
d <- dmFilter(d, min_samps_gene_expr = 6, min_samps_feature_expr = 3,
```

```
  min_samps_feature_prop = 3)
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 sample size 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, which is less inaccurate. However, such assumption is quite unrealistic. Moderated dispersion is a trade-off between gene-wise and common dispersion. The moderated estimates are a weighted combination of common and individual dispersion and we recommend such 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.

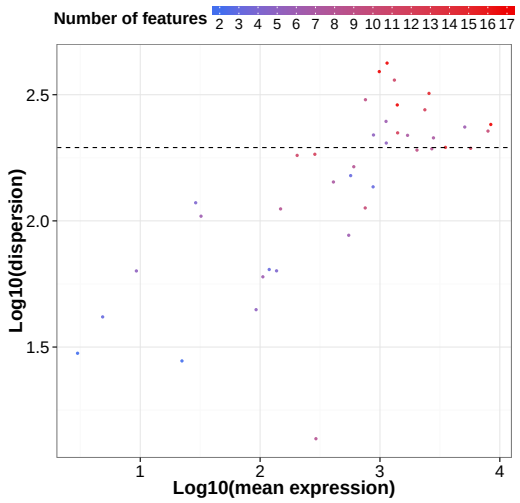
This step of our pipeline is the most time consuming, thus consider increasing the number of workers in `BPPARAM`.

```
d <- dmDispersion(d, BPPARAM = BiocParallel::MulticoreParam(workers = 2))
## * Calculating mean gene expression..
## Took  0.913  seconds.
## * Estimating common dispersion..
## Took  56.918  seconds.

## !  Using common_dispersion = 195.27 as disp_init !

## * Estimating genewise dispersion..
## Took  5.846  seconds.
d
## An object of class dmDSdispersion
## with 33 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        1318.286
## 2 FBgn0000578        3515.000
## 3 FBgn0002921        7962.286
common_dispersion(d)
## [1] 195.2712
head(genewise_dispersion(d), 3)
##       gene_id genewise_dispersion
## 1 FBgn0000256            302.8640
## 2 FBgn0000578            182.4581
## 3 FBgn0002921            228.6716
```

To inspect the behaviour of dispersion estimates, you can plot them against the mean gene expression. Here, the effect of shrinking is not so well visible because our data set is very small. To see how it works on real data sets, go to our paper *fixme: citation to DRIMSeq paper*.

```
plotDispersion(d)
```



3.2.4 Proportions 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::MulticoreParam(workers = 1))
## * Fitting full model..
## Took  14.711  seconds.
d
## An object of class dmDSfit
```

```
## with 40 genes and 7 samples
## * data accessors: counts(), samples()
##   mean_expression(), common_dispersion(), genewise_dispersion()
##   proportions(), statistics()
head(proportions(d), 3)
##       gene_id feature_id    treated  untreated
## 1 FBgn0000256        001 0.04754941 0.04804901
## 2 FBgn0000256        002 0.09192056 0.07878007
## 3 FBgn0000256        003 0.26135466 0.26411140
head(statistics(d), 3)
##       gene_id lik_treated lik_untreated
## 1 FBgn0000256   -6866.356     -12197.61
## 2 FBgn0000578  -22743.567     -37572.26
## 3 FBgn0002921  -42450.168     -48057.55
```

3.2.5 Testing for differential splicing

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

By default, a parameter `compared_groups` in `dmTest` equals to `1:nlevels(samples(x)$group)`, which means that we test for differences in splicing between any of the groups specified in `samples(d)$group`. In *pasilla* example, there are only two conditions, and there is only one comparison that can be done. In the case where the 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, BPPARAM = BiocParallel::MulticoreParam(workers = 1))
```

```
## Running comparison  between groups:  treated, untreated
```

```
## * Fitting null model..
## Took  15.393  seconds.
## * Calculating likelihood ratio statistics..
## Took  0.002892733  seconds.
d
## An object of class dmDStest
## with 40 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    treated  untreated       null
## 1 FBgn0000256        001 0.04754941 0.04804901 0.04813549
## 2 FBgn0000256        002 0.09192056 0.07878007 0.08457247
```

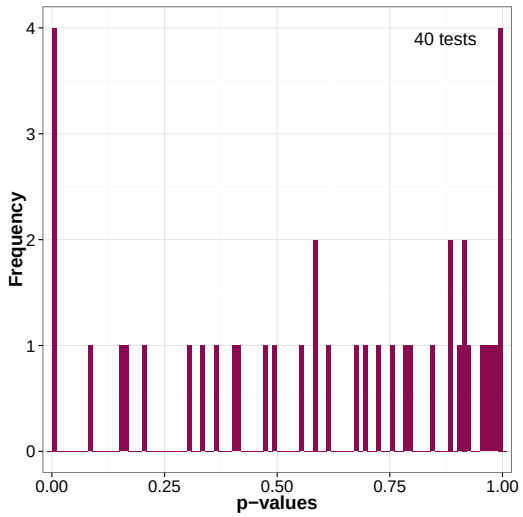
```
## 3 FBgn0000256        003 0.26135466 0.26411140 0.26459246
head(statistics(d), 3)
##       gene_id lik_treated lik_untreated  lik_null df
## 1 FBgn0000256   -6866.356     -12197.61 -19079.64 10
## 2 FBgn0000578  -22743.567     -37572.26 -60318.54 14
## 3 FBgn0002921  -42450.168     -48057.55 -90509.34  9
```

To obtain the results of likelihood ratio test, 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 31.349649 10 0.0005135684 0.006847578
## 2 FBgn0000578  5.425912 14 0.9789583045 0.999025040
## 3 FBgn0002921  3.239646  9 0.9540310716 0.999025040
```

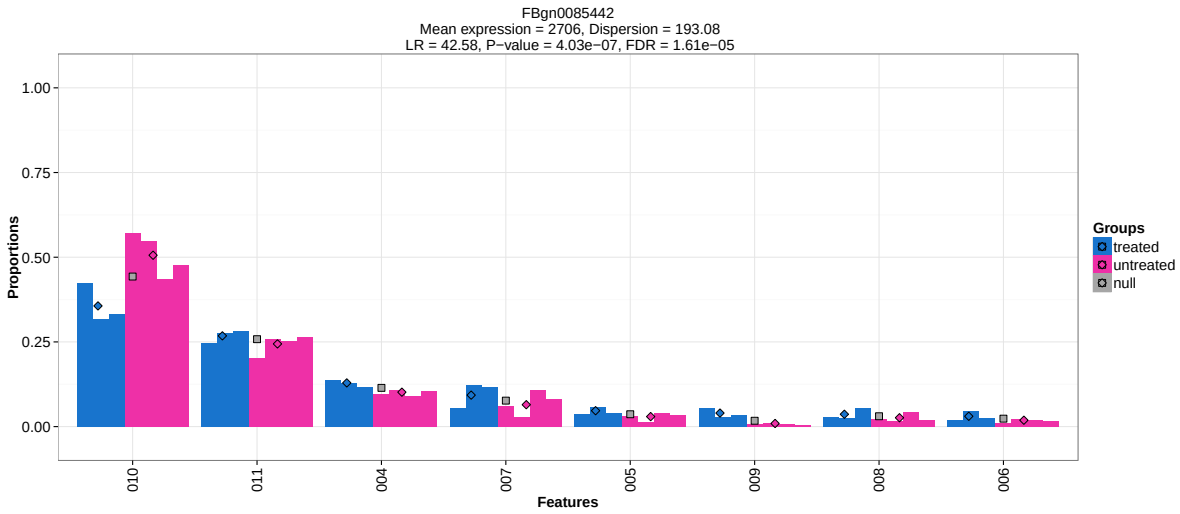
You can plot the histogram of p-values.

```
plotTest(d)
```

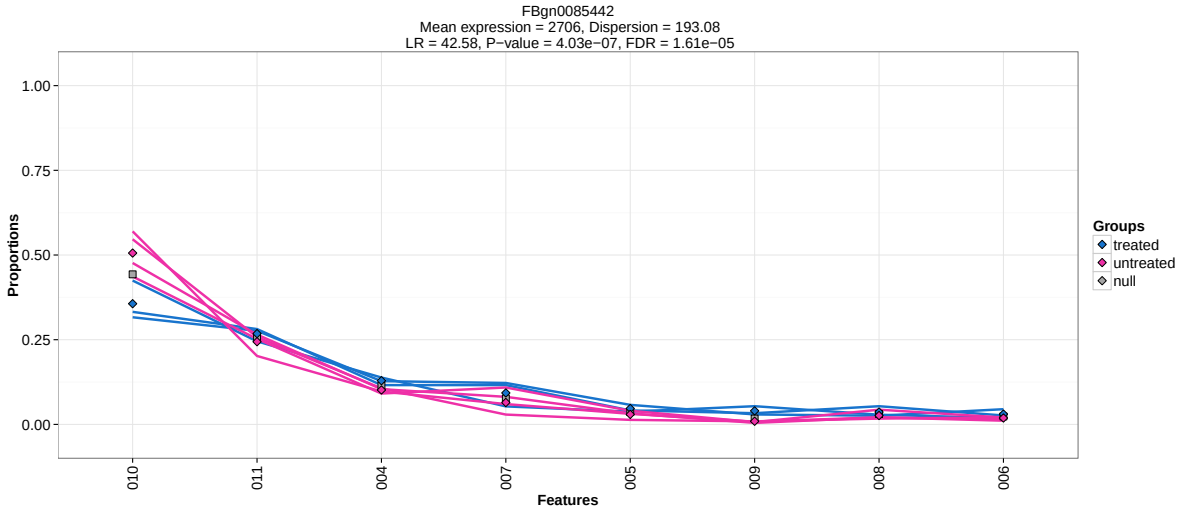


For genes of interest, you can make plots (bar plots, line plots, box plots, ribbon plots) of observed and estimated with Dirichlet-multinomial model feature ratios. Estimated proportions are marked with diamond shapes. Here, we plot the results for the top significant gene.

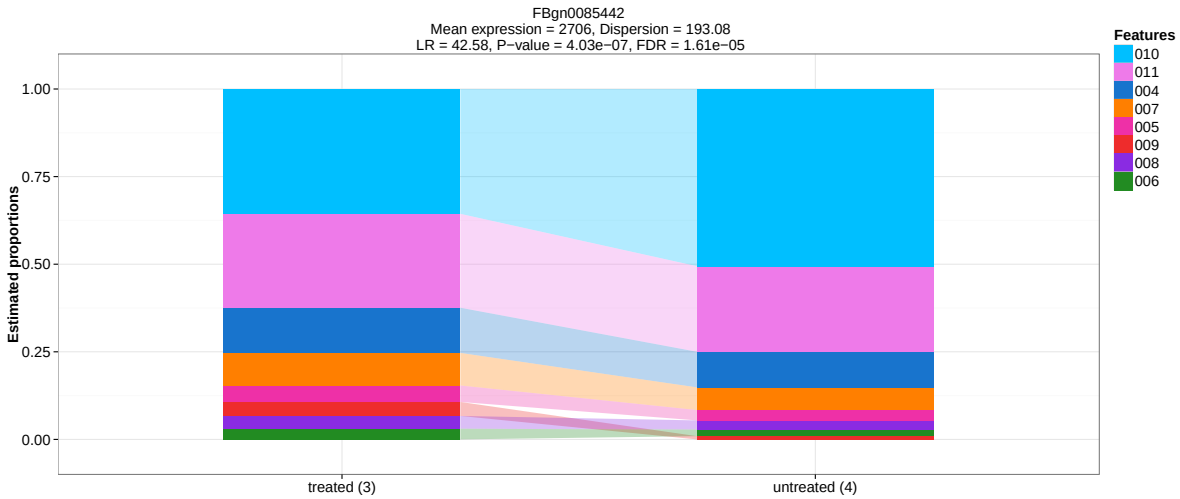
```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]
gene_id <- res$gene_id[1]

plotFit(d, gene_id = gene_id)
## Plot gene 1: FBgn0085442
```




```
plotFit(d, gene_id = gene_id, plot_type = "lineplot")
## Plot gene 1: FBgn0085442
```




```
plotFit(d, gene_id = gene_id, plot_type = "ribbonplot")
## Plot gene 1: FBgn0085442
```



4 sQTL analysis work-flow

4.1 Example data

For the sQTL analysis, we use data from the GEUVADIS project [2], 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 quantifications (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 available on http://www.gencodegenes.org/releases/12.html.

In order to make the vignette runnable, we do the *DRIMSeq* 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. In the next section, we present how this subsets were obtained.

4.2 GEUVADIS data preprocessing

This section shows how you can prepare the GEUVADIS data for the analysis with *DRIMSeq*. It preprocesses counts and genotypes for all the populations. At the end, it generates subsets used in examples and *DRIMSeq* pipeline presented further in this vignette. The *R* code in this section does not run when the vignette is generated, mainly because it takes some time, but it is executable.

In the 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 the associations of every SNP with every gene. However, such 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 SNPs that are located within the gene body and within some range upstream and downstream the gene.

In order to find out which SNPs should be tested with which genes, we need to know the location of genes. We can get it from the GFT file with Gencode v12 gene annotation available on http://www.gencodegenes.org/releases/12.html. Let's save this GFT file in `geuvadis_annotation/gencode.v12.annotation.gtf`. We can import it into *R* and extract locations of protein coding genes with the following code.

```
### main path where the data is saved
data_dir <- "."

library(GenomicRanges)
library(rtracklayer)
library(limma)

### prepare annotation
gtf0 <- import(paste0(data_dir, "geuvadis_annotation/gencode.v12.annotation.gtf"))

# keep protein coding genes
keep_index <- mcols(gtf0)$gene_type == "protein_coding" & mcols(gtf0)$type == "gene"
gtf <- gtf0[keep_index]
# remove 'chr' from the chromosome names
seqlevels(gtf) <- gsub(pattern = "chr", replacement = "", x = seqlevels(gtf))
# create BED file with gene location
genes_bed <- data.frame(chr = seqnames(gtf), start =  start(gtf), end = end(gtf),
   geneId = mcols(gtf)$gene_id)

dir.create(paste0(data_dir, "annotation"))
write.table(genes_bed, paste0(data_dir, "annotation/genes.bed"), quote = FALSE,
   sep = "\t", row.names = FALSE, col.names = FALSE)
```

From http://www.ebi.ac.uk/arrayexpress/files/E-GEUV-1/analysis_results/, we can download a file with information about samples `E-GEUV-1.sdrf.txt` and a file with transcript quantifications `GD660.TrQuantCount.txt`. Let's save them in `geuvadis_analysis_results/` folder.

We use `samples` to keep the names of the samples and the population they come from.

```
### prepare samples
samples <- read.table(paste0(data_dir, "geuvadis_analysis_results/E-GEUV-1.sdrf.txt"),
    header = T, sep="\t", as.is=TRUE)

samples <- samples[c("Assay.Name", "Characteristics.population.")]
samples <- unique(samples)

dim(samples)
table(samples$Characteristics.population.)

colnames(samples) <- c("sample_id", "population")
samples$sample_id_short <- strsplit2(samples$sample_id, "\\.")[,1]
```

File `GD660.TrQuantCount.txt` contains quantification for all the populations. We split this table by population and keep the samples specified in data frame `samples`.

```
### prepare counts
expr_all <- read.table(paste0(data_dir, "geuvadis_analysis_results/GD660.TrQuantCount.txt"),
   header = T, sep="\t", as.is = TRUE)

dim(expr_all)

expr_all <- expr_all[, c("TargetID", "Gene_Symbol", samples$sample_id)]
# use short names
colnames(expr_all) <- c("trId", "geneId", samples$sample_id_short)

dir.create(paste0(data_dir, "expression"))

for(i in unique(samples$population)){

  expr <- expr_all[, c("trId", "geneId", samples$sample_id_short[samples$population == i])]
  write.table(expr, paste0(data_dir, "expression/TrQuantCount_",i,".tsv"),
     quote = FALSE, sep = "\t", row.names = FALSE, col.names = TRUE)

}
```

The VCF files with genotypes are available on http://www.ebi.ac.uk/arrayexpress/files/E-GEUV-1/genotypes/ in the compressed format. Let's save them in `geuvadis_genotypes/` folder. We can load them into *R* using the *VariantAnnotation* package. To do so, we have to first "big zip" them with `bgzip` and create indexes with `indexTabix`.

```
library(Rsamtools)
library(VariantAnnotation)
library(tools)

files <- list.files(path = paste0(data_dir, "geuvadis_genotypes"),
   pattern = "genotypes.vcf.gz", full.names = TRUE, include.dirs = FALSE)

## bigzip and index the vcf files
for(i in 1:length(files)){
  zipped <- bgzip(files[i])
  idx <- indexTabix(zipped, format = "vcf")
}
```

We are interested in bi-allelic SNPs that lay within a 5000 bases surrounding of genes. Additionally, we want to convert the genotype information into 0 for ref/ref, 1 for ref/not ref, 2 for not ref/not ref, -1 or NA for missing values. Newly encoded genotypes are saved as text files, one file for each chromosome and population.

```
### prepare genotypes
# extended gene ranges
window <- 5000
gene_ranges <- resize(gtf, GenomicRanges::width(gtf) + 2 * window, fix = "center")

population <- unique(samples$population)
chr <- gsub("chr", "", strsplit2(files, split = "\\.")[, 2])
```

```
dir.create(paste0(data_dir, "genotypes"))

for(j in 1:length(population)){
  for(i in 1:length(files)){

    cat(population[j], chr[i], fill = TRUE)

    zipped <- paste0(file_path_sans_ext(files[i]), ".bgz")
    idx <- paste0(file_path_sans_ext(files[i]), ".bgz.tbi")
    tab <- TabixFile(zipped, idx)

    ## Explore the file header with scanVcfHeader
    hdr <- scanVcfHeader(tab)
    print(all(samples$sample_id_short %in% samples(hdr)))

    ## Read VCF file - SNPs for one population and within extended gene ranges
    gene_ranges_tmp <- gene_ranges[seqnames(gene_ranges) == chr[i]]
    param <- ScanVcfParam(which = gene_ranges_tmp,
       samples = samples$sample_id_short[samples$population == population[j]])
    vcf <- readVcf(tab, "hg19", param)

    ## Keep only the bi-allelic SNPs
    # width of ref seq
    rw <- width(ref(vcf))
    # width of first alt seq
    aw <- unlist(lapply(alt(vcf), function(x) {width(x[1])}))
    # number of alternate genotypes
    nalt <- elementLengths(alt(vcf))
    # select only bi-allelic SNPs (monomorphic OK, so aw can be 0 or 1)
    snp <- rw == 1 & aw <= 1 & nalt == 1
    # subset vcf
    vcfbi <- vcf[snp,]

    rowdata <- rowData(vcfbi)

    ## Convert genotype into a number of alleles different from reference
    geno <- geno(vcfbi)$GT
    geno01 <- geno
    geno01[,] <- -1
    geno01[geno %in% c("0/0", "0|0")] <- 0 # REF/REF
    geno01[geno %in% c("0/1", "0|1", "1/0", "1|0")] <- 1 # REF/ALT
    geno01[geno %in% c("1/1", "1|1")] <- 2 # ALT/ALT
    # geno01 should be integer, not character
    mode(geno01) <- "integer"

    genotypes <- unique(data.frame(chr = seqnames(rowdata),
       start = start(rowdata), end = end(rowdata), snpId = rownames(geno01),
```

```
       geno01, stringsAsFactors = FALSE))

    ## sorting SNPs by position
    genotypes <- genotypes[order(genotypes[,2]), ]

    write.table(genotypes,
      paste0(data_dir, "genotypes/genotypes_", j, "_chr", chr[i], ".tsv"),
      quote = FALSE, sep = "\t", row.names = FALSE, col.names = TRUE)

  }
}
```

Now, we show how the example data sets were generated. The lists of sample genes and SNPs are predefined in the files in `extdata/` of this package.

```
data_ex_dir <- system.file("extdata", package = "DRIMSeq")

genes_subset = readLines(file.path(data_ex_dir, "/gene_id_subset.txt"))
snps_subset = readLines(file.path(data_ex_dir, "/snp_id_subset.txt"))
```

```
### Subset gene ranges
gene_bed <- read.table(paste0(data_dir, "annotation/genes.bed"), header = FALSE,
    as.is = TRUE)
gene_bed <- gene_bed[gene_bed[, 4] %in% genes_subset, ]

write.table(gene_bed, paste0("genes_subset.bed"), quote = FALSE,
    sep = "\t", row.names = FALSE, col.names = FALSE)

### Subset counts
counts <- read.table(paste0(data_dir, "expression/TrQuantCount_CEU.tsv"),
    header = TRUE, as.is = TRUE)
counts <- counts[counts$geneId %in% genes_subset, ]

write.table(counts, paste0("TrQuantCount_CEU_subset.tsv"),
    quote = FALSE, sep = "\t", row.names = FALSE, col.names = TRUE)

### Subset genotypes
genotypes <- read.table(paste0(data_dir, "genotypes/genotypes_CEU_chr19.tsv"),
    header = TRUE, sep = "\t", as.is = TRUE)
genotypes <- genotypes[genotypes$snpId %in% snps_subset, ]

write.table(genotypes, paste0("genotypes_CEU_subset.tsv"),
   quote = FALSE, sep = "\t", row.names = FALSE, col.names = TRUE)
```

4.3 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 every following object, slots and methods are inherited from the previous one.

4.3.1 Loading GEUVADIS data into R

Load the *DRIMSeq* package.

```
suppressPackageStartupMessages(library(DRIMSeq))
```

Let's load the subsets of GEUVADIS data that are prepared for the demonstration of the *DRIMSeq* work-flow. Feature counts, genotypes and gene annotation can be found in the `extdata/` folder attached with this package. In order to find which SNPs are in the surrounding of genes we have to prepare *GRanges* objects with gene and SNP locations. All this is done with the following code.

```
suppressPackageStartupMessages(library(GenomicRanges))
library(rtracklayer)

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

# gene_ranges with names!
gene_ranges <- import(paste0(data_dir, "/genes_subset.bed"))
names(gene_ranges) <- mcols(gene_ranges)$name

counts <- read.table(paste0(data_dir, "/TrQuantCount_CEU_subset.tsv"),
   header = TRUE, sep = "\t", as.is = TRUE)

genotypes <- read.table(paste0(data_dir, "/genotypes_CEU_subset.tsv"),
   header = TRUE, sep = "\t", as.is = TRUE)

# snp_ranges with names!
snp_ranges <- GRanges(Rle(genotypes$chr), IRanges(genotypes$start,
   genotypes$end))
names(snp_ranges) <- genotypes$snpId

## Check if samples in count and genotypes are in the same order
all(colnames(counts[, -(1:2)]) == colnames(genotypes[, -(1:4)]))
## [1] TRUE
sample_id <- colnames(counts[, -(1:2)])
```

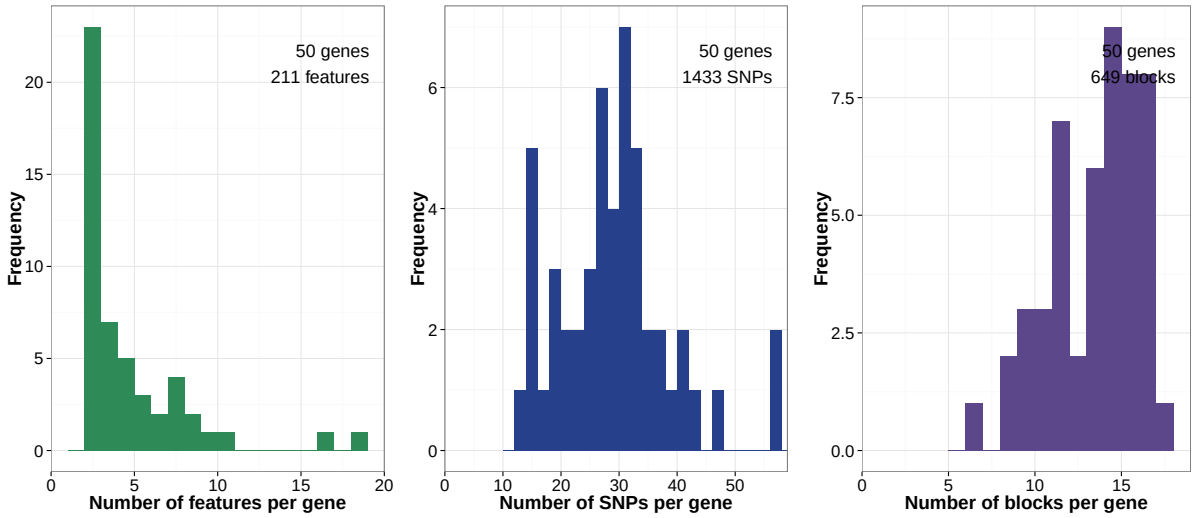
In the sQTL analysis, an initial data object `d` is of `dmSQTLdata` class and, additionally to feature counts, it contains genotypes.

```
d <- dmSQTLdataFromRanges(counts = counts[, -(1:2)], gene_id = counts$geneId,
   feature_id = counts$trId, gene_ranges = gene_ranges,
   genotypes = genotypes[, -(1:4)], snp_id = genotypes$snpId,
   snp_ranges = snp_ranges, sample_id = sample_id, window = 5e3,
   BPPARAM = BiocParallel::MulticoreParam(workers = 1))
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. We identify SNPs with identical genotypes across the samples and assign them to blocks. Estimation and testing is 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)
```



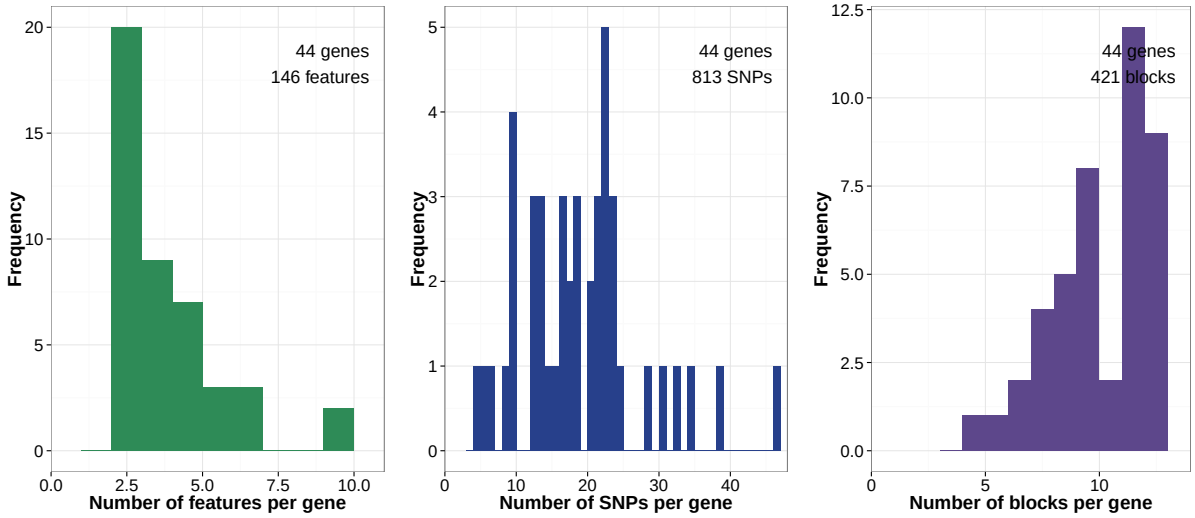
4.3.2 Filtering

The filtering step eliminates genes and features with low expression, as in differential splicing analysis 3.2.2. Additionally, it filters out the SNPs/blocks which 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. `min_samps_feature_prop` defines the minimal number of samples where feature has to be expressed at the level of minimum `min_feature_prop`. It should be equal to the size of the minimal group that can be created based on genotypes which is equal to `minor_allele_freq`. Like this, we allow a situation where a feature is expressed in one group, but may not be expressed at all in another one, which is a case of

differential splicing. Because `min_samps_feature_prop` is quite low in comparison to the total sample size, we can be more conservative about the `min_feature_prop` which is set to 0.1.

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 are present in the counts data, or genes are 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 = 5, minor_allele_freq = 5,
    BPPARAM = BiocParallel::MulticoreParam(workers = 1))

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



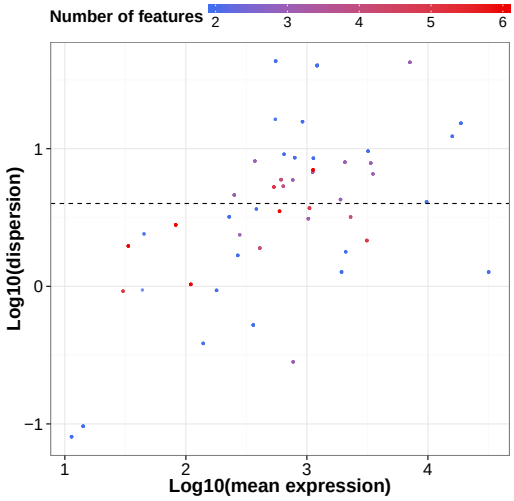
4.3.3 Dispersion estimation

As for differential splicing 3.2.3, `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. Such 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 the blocks matched with 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.

```
d <- dmDispersion(d, BPPARAM = BiocParallel::MulticoreParam(workers = 2))
## * Calculating mean gene expression..
## Took  1.399  seconds.
## * Estimating common dispersion..
## Took  69.01  seconds.

## !  Using common_dispersion = 4 as disp_init !

## * Estimating genewise dispersion..
## Took  5.209  seconds.
```

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



4.3.4 Proportions estimation

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

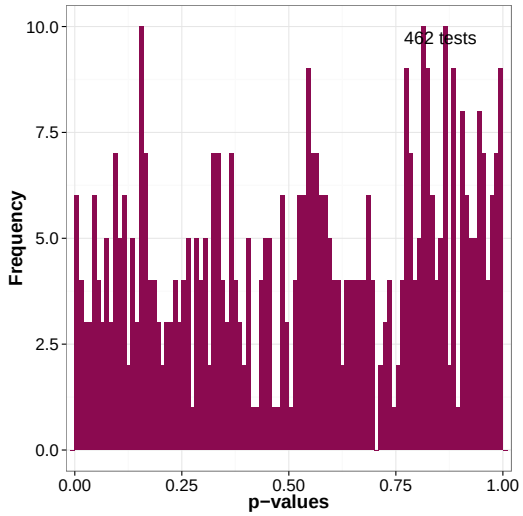
```
d <- dmFit(d, BPPARAM = BiocParallel::MulticoreParam(workers = 1))
## * Fitting full model..
## Took  35.669  seconds.
d
## An object of class dmSQTLfit
## with 50 genes and 91 samples
```

4.3.5 Testing for sQTLs

As in 3.2.5, `dmTest` function fits null model proportions and does the likelihood ratio test. To obtain the results, 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-block/SNP pair.

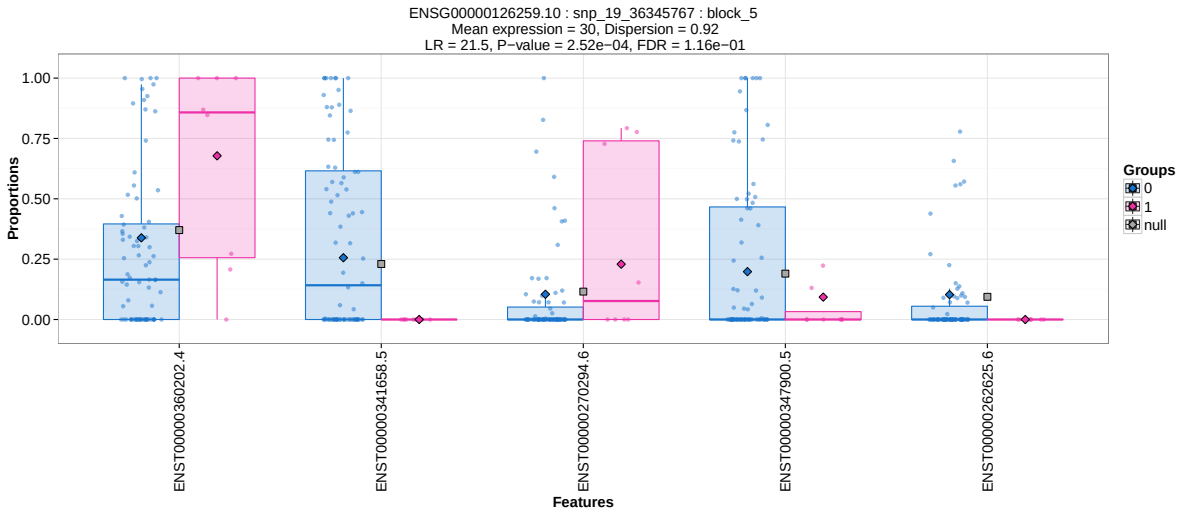
```
d <- dmTest(d, BPPARAM = BiocParallel::MulticoreParam(workers = 1))
## * Fitting null model..
## Took  34.134  seconds.
## * Calculating likelihood ratio statistics..
## Took  28.91702  seconds.
d
## An object of class dmSQTLtest
## with 50 genes and 91 samples
## * data accessors: results()
```

```
head(results(d), 3)
##             gene_id block_id          snp_id        lr df    pvalue adj_pvalue
## 1 ENSG00000095066.6  block_1 snp_19_12868839 0.1613363  1 0.6879292  0.9996592
## 2 ENSG00000095066.6  block_5 snp_19_12875197 0.3959769  1 0.5291743  0.9996592
## 3 ENSG00000095066.6  block_6 snp_19_12876278 0.8339018  1 0.3611467  0.9996592
plotTest(d)
```

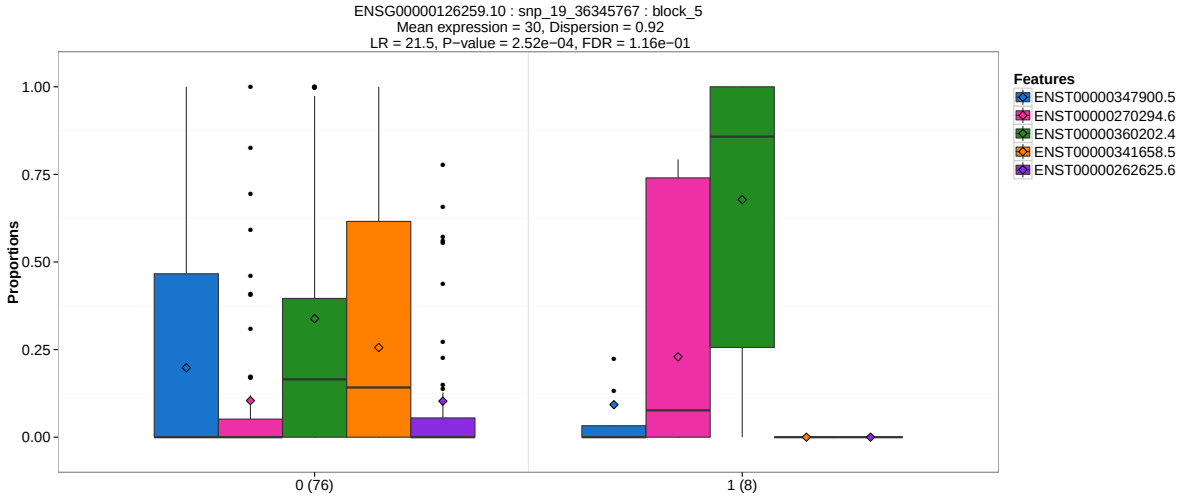


You can plot the observed and estimated with Dirichlet-multinomial model feature ratios for the sQTLs of interest. When the sample size is big, we recommend using box plots as a `plot_type`. Here, we plot an sQTL with the lowest p-value.

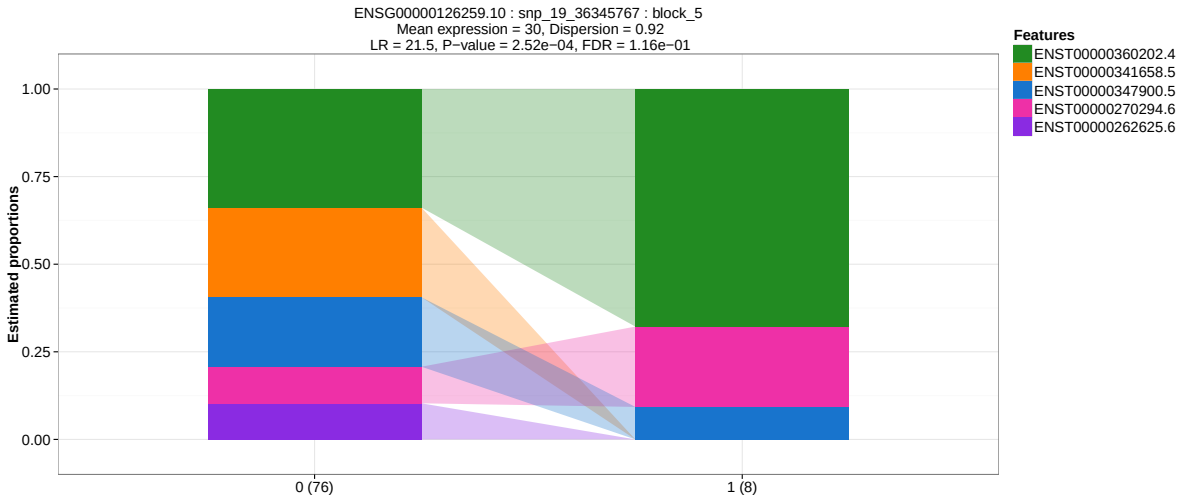
```
res <- results(d)
res <- res[order(res$pvalue, decreasing = FALSE), ]

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

plotFit(d, gene_id, snp_id)
## Plot pair 1: ENSG00000126259.10:snp_19_36345767
```




```
plotFit(d, gene_id, snp_id, plot_type = "boxplot2", order = FALSE)
## Plot pair 1: ENSG00000126259.10:snp_19_36345767
```




```
plotFit(d, gene_id, snp_id, plot_type = "ribbonplot")
## Plot pair 1: ENSG00000126259.10:snp_19_36345767
```


ENSG00000126259.10 : snp_19_36345767 : block_5
Mean expression = 30, Dispersion = 0.92
LR = 21.5, P−value = 2.52e−04, FDR = 1.16e−01
Estimated proportions
1.00
0.75
0.50
0.25
0.00
0 (76)
1 (8)
Features
ENST00000360202.4
ENST00000341658.5
ENST00000347900.5
ENST00000270294.6
ENST00000262625.6

APPENDIX

A Session information

```
sessionInfo()
## R version 3.2.2 (2015-08-14)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.3 LTS
##
## locale:
##  [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C               LC_TIME=en_CA.UTF-8
##  [4] LC_COLLATE=en_CA.UTF-8     LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] grid      stats4    parallel  stats     graphics  grDevices utils     datasets
##  [9] methods   base
##
## other attached packages:
##  [1] rtracklayer_1.30.0   GenomicRanges_1.22.0 GenomeInfoDb_1.6.0   IRanges_2.4.0
##  [5] S4Vectors_0.8.0      DRIMSeq_0.3.1        BiocGenerics_0.16.0  pasilla_0.10.0
##  [9] knitr_1.11           colorout_1.1-1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.1                formatR_1.2.1              futile.logger_1.4.1
##  [4] highr_0.5.1                plyr_1.8.3                 XVector_0.10.0
##  [7] futile.options_1.0.0       bitops_1.0-6               tools_3.2.2
## [10] zlibbioc_1.16.0            digest_0.6.8               evaluate_0.8
## [13] gtable_0.1.2               proto_0.3-10               stringr_1.0.0
## [16] Biostrings_2.38.0          Biobase_2.30.0             XML_3.98-1.3
## [19] BiocParallel_1.4.0         limma_3.26.0               ggplot2_1.0.1
## [22] reshape2_1.4.1             lambda.r_1.1.7             edgeR_3.12.0
## [25] magrittr_1.5               GenomicAlignments_1.6.0    scales_0.3.0
## [28] Rsamtools_1.22.0           codetools_0.2-14           MASS_7.3-44
## [31] SummarizedExperiment_1.0.0 BiocStyle_1.8.0            colorspace_1.2-6
## [34] labeling_0.3               stringi_0.5-5              RCurl_1.95-4.7
## [37] munsell_0.4.2
```

B References

References

[1] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley, "Conservation of an RNA regulatory map between Drosophila and mammals.," *Genome research*, vol. 21, no. 2, pp. 193–202, 2011.

[2] T. Lappalainen, M. Sammeth, M. R. Friedländer, P. A. C. 't Hoen, J. Monlong, M. A. Rivas, M. Gonzàlez-Porta, N. Kurbatova, T. Griebel, P. G. Ferreira, M. Barann, T. Wieland, L. Greger, M. van Iterson, J. Almlöf, P. Ribeca, I. Pulyakhina, D. Esser, T. Giger, A. Tikhonov, M. Sultan, G. Bertier, D. G. MacArthur, M. Lek, E. Lizano, H. P. J. Buermans, I. Padioleau, T. Schwarzmayr, O. Karlberg, H. Ongen, H. Kilpinen, S. Beltran, M. Gut, K. Kahlem, V. Amstislavskiy, O. Stegle, M. Pirinen, S. B. Montgomery, P. Donnelly, M. I. McCarthy, P. Flicek, T. M. Strom, H. Lehrach, S. Schreiber, R. Sudbrak, A. Carracedo, S. E. Antonarakis, R. Häsler, A.-C. Syvänen, G.-J. van Ommen, A. Brazma, T. Meitinger, P. Rosenstiel, R. Guigó, I. G. Gut, X. Estivill, and E. T. Dermitzakis, "Transcriptome and genome sequencing uncovers functional variation in humans.," *Nature*, vol. 501, no. 7468, pp. 506–11, 2013.