Differential expression analysis of RNA–Seq data using *DESeq2*

Bernd Klaus[1]

European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany
[1]`bernd.klaus@embl.de`

November 3, 2014

Contents

-

1 Required packages and other preparations

```
library(geneplotter)
library(ggplot2)
library(plyr)
library(LSD)
library(DESeq2)
library(gplots)
library(RColorBrewer)
library(stringr)
library(topGO)
library(genefilter)
library(biomaRt)
library(dplyr)
library(EDASeq)
library(fdrtool)
#library(xlsx)

data.dir <- file.path("/g/huber/users/klaus/Data/NCBI")
```

2 Introduction

In this document we introduce a complete sample workflow for a typical RNA-Seq data analysis. Data import, preprocessing, differential expression and enrichment analysis are discussed. We re–analyze RNA–Seq data obtained by comparing the expression profiles of WT mice to mice harboring a deletion that is associated with a MYC enhancer which is required for proper lip/palate formation. The data was published along with the following paper:

- Uslu et. al. - Long-range enhancers regulating Myc expression are required for normal facial morphogenesis, 2014

All necessary steps are detailed, including the download of the raw data and the alignment as well as the creation of a count table.

Note that in the workflows below, we do not aim at giving a comprehensive treatment of of possible options, but rather try to introduce sensible default approaches for the necessary steps. It is based on a protocol published in Nature Methods, which contains more detailed information on various steps. Additionally, the "Beginners guide to DESeq2" is well worth reading and contains a lot of additional background information.

- Anders et. al. - Count-based differential expression analysis of RNA sequencing data using R and Bioconductor, 2013
- Love et. al. – A Beginner's guide to the "DESeq2" package

3 RNA–Seq data preprocessing

An RNA–Seq experiment data analysis starts with FASTQ–files obtained as the output of the sequencing runs. Commonly, there is one FASTQ–file per sample for single–end reads and two FASTQ files for paired–end data.

To download the published data from the gene expression omnibus repository, we need the SRA toolkit, which allows to download the FASTQ files from short read archive. Binaries are available for ubuntu and centOS. The accession numbers (and links to the landing pages) are given in the article, they are GSM1279692 – GSM1279701

The command line script to download the files is given below. The variable *FQ* has to point to the *fastq–dump* binary, which is part of the SRA toolkit and allows to turn the SRA format into FASTQ.

```
cat(paste(readLines("getRawData.sh"), "\n"))

   #! /bin/sh
```

```
    cd /g/huber/users/klaus/Data/NCBI
    FQ=/g/huber/users/klaus/Own-Software/sratoolkit.2.3.5-2-centos_linux64/bin/fastq-dump

    $FQ -v --gzip SRR1042885
    $FQ -v --gzip SRR1042886
    $FQ -v --gzip SRR1042887
    $FQ -v --gzip SRR1042888
    $FQ -v --gzip SRR1042889
    $FQ -v --gzip SRR1042890
    $FQ -v --gzip SRR1042891
    $FQ -v --gzip SRR1042892
```

3.1 Creation of a sample metadata table

Now we can create a sample metadata table, containing all the sample information. We first get a list of the zipped FASTQ–files

```
fastqDir= file.path("/g/huber/users/klaus/Data/NCBI/fastQCQuality")
fastq <- list.files(fastqDir, pattern = "*.fastq.gz")
```

Then we subset the strings, the running number in the beginning serves as sample numbers.

```
sampleNO <- str_sub(fastq, 1,10)
```

The first four samples are labelled as WT, the other 4 as deletion mutants.

warning: there is an annotation error here: sample 2 and 7 belong to the opposite group. We will fix this later in the workflow

```
condition = c(rep("WT",4), rep("Del_8_17_homo",4))
```

The library name combines the sample number with the condition. The function `paste` can be used to concatenate strings.

```
libraryName = paste(condition,"-",sampleNO, sep = "")
```

We now create a new data frame with the sample metadata information

```
metadata <- data.frame(sampleNO = sampleNO,
                              condition = condition,
                                        fastq = fastq,
                                   libraryName = libraryName)

metadata

       sampleNO     condition              fastq              libraryName
   1 SRR1042885            WT SRR1042885.fastq.gz            WT-SRR1042885
   2 SRR1042886            WT SRR1042886.fastq.gz            WT-SRR1042886
   3 SRR1042887            WT SRR1042887.fastq.gz            WT-SRR1042887
   4 SRR1042888            WT SRR1042888.fastq.gz            WT-SRR1042888
   5 SRR1042889 Del_8_17_homo SRR1042889.fastq.gz Del_8_17_homo-SRR1042889
   6 SRR1042890 Del_8_17_homo SRR1042890.fastq.gz Del_8_17_homo-SRR1042890
   7 SRR1042891 Del_8_17_homo SRR1042891.fastq.gz Del_8_17_homo-SRR1042891
   8 SRR1042892 Del_8_17_homo SRR1042892.fastq.gz Del_8_17_homo-SRR1042892
```

Form this sample metadata, all subsequent commands will be constructed. This way, we can be sure that the commands are correct, and we avoid sample swaps etc.

3.2 Quality control commands

After the FASTQ files have been obtained. One should perform initial checks on sequence quality. This can be conveniently done using the java–based program *fastqc*, which creates a comprehensive html–report and is very easy to use: One just specifies the the pathway to the FASTQ files and then program creates the report. We create the command in the variable `fastQC.cmd` and then use the function `sink` to write out the command to a file.

```
fastQC.binary = "/g/huber/users/klaus/Own-Software/FastQC/fastqc"
fastQC.cmd = paste(fastQC.binary, with(metadata, paste0( fastqDir, fastq, collapse = " ")))
cat('#!/bin/sh \n\n')
sink(file = "fastQC-report.sh", type="output")
cat(fastQC.cmd)
sink()
```

When inspecting the html reports, one can for example check for persistence of low–quality scores, over–representation of adapter sequences and other potential problems. From these inspections, users may choose to remove low-quality samples, trim ends of reads or adapt alignment parameters.

3.3 Alignment of reads

After initial checks on sequence quality, reads are mapped to a reference genome with a splice-aware aligner. Here we use *TopHat* , which is a splicing–aware addition to the short–read aligner *Bowtie*.

In order to create the alignment commands we need a reference genome sequence and and an indexed version of it, which is created by the alignment program.

Furthermore, providing genomic feature annotation (e.g. exons, genes) in a GTF files helps the aligner to perform the mapping of spliced reads. Later, we will also need the GTF file to count reads into feature bins.

For simplicity and to avoid problems with mismatching chromosome identifiers and inconsistent genomic coordinate systems, it is recommended to use the prebuilt indices packaged with GTF files from iGenomes whenever possible together with *TopHat*.

```
bowidx = "/g/huber/users/klaus/Data/Sandro/Mus_musculus/Ensembl/NCBIM37/Sequence/Bowtie2Index/genome"
gtf = "/g/huber/users/klaus/Data/Sandro/Mus_musculus/Ensembl/NCBIM37/Annotation/Genes/genes.gtf"
output.dir = "/g/huber/users/klaus/Data/NCBI/MassimoAligned"
```

The following script creates the *TopHat* commands necessary for the alignments.

```
tophat.cmd = with(metadata, paste("tophat -G ", gtf ," -p 5 -o ", output.dir ,
libraryName , " " ,bowidx, " ", fastqDir,fastq, "\n\n",  sep = "") )

sink(file = "tophat-commands.sh", type="output")
cat('#!/bin/sh \n\n')
cat(tophat.cmd)
sink()
```

In the call to *TopHat*, the option -G points *TopHat* to a GTF file of annotation to facilitate mapping reads across exon-exon junctions (some of which can be found de novo), —o specifies the output directory, —p specifies the number of threads to use (this may affect run times and can vary depending on the resources available).

The first argument is the name of the index (built in advance), and the second argument is a list of all FASTQ files containing reads of the specific sample. Note that the FASTQ files are concatenated with commas, without spaces. For experiments with paired–end reads, pairs of FASTQ files are given as separate arguments and the order in both arguments must match.

Note that other parameters can be specified here as needed; see the appropriate documentation for the version you are using. For example *TopHat* has special options to only keep only the aligned reads with proper orientation if you have strand specific data.

3.4 Sorting and indexing of the alignment files

TopHat returns the alignment as BAM files. This format, and equivalently SAM, (an uncompressed text version of BAM), are the de facto standard file formats for alignments. The software *samtools* can be used to handle the BAM/SAM format.

In order to count the reads overlapping genomic features and to view the alignments in a genome browser like IGV, one needs to sort the aligned reads and create an index for random access to the BAM files.

Note that we here use commands suitable for *samtools* up to 0.1.9. The 1.0 version has a slightly different way of specifying input and output files for the commands.

```
sink(file = "sam-commands.sh")
cat('#!/bin/sh \n\n')
cat(paste("cd", output.dir, "\n\n"))
ob = file.path(output.dir, metadata$libraryName, "accepted_hits.bam")
for(i in seq_len(nrow(metadata))) {
  lib = metadata$libraryName[i]

  # sort by position and index files for IGV and HTSeq count
  # for single end reads no sorting by name is required!
  cat(paste0("samtools sort ",ob[i]," ",lib,"_s"),"\n")
  cat(paste0("samtools index ",lib,"_s.bam"),"\n\n")
}
sink()
```

3.5 Counting features with HTSeq

We now use the count script from the HTSeq python library to count the aligned reads to features. Since HTSeq-count requires SAM input, we first convert our BAM to SAM files.

Note that it is very important to set the correct strand option (—s) in HTSeq-count. Otherwise, a lot of overlaps will not be counted. Here, we set it to "no" since we do not have a strand specific protocol.

```
HTSeqcommands <- character(nrow(metadata)*2)


for(i in seq_len(nrow(metadata))) {
  lib = metadata$libraryName[i]


  # create sam files from bam files
  HTSeqcommands [2*(i-1) +1 ]    = paste0("samtools view " ,  lib, "_s.bam", " > ",  lib, "_s.sam"   ,"\n")
  # count features for DESeq
  HTSeqcommands [2*(i-1) +2 ]  = paste0("htseq-count-0.6.1p1 -a 10" , " -s \'no\'", " ",  lib, "_s.sam ", 
}


sink(file = "countDESeq-no.sh", type = "output")
cat('#!/bin/sh \n\n')
cat("cd ", output.dir, "\n\n")
cat(HTSeqcommands)
sink()
```

HTSeq-count returns the counts per gene for every sample in a '.txt' file.

3.6 Creating a count table for DESeq2

We first add the names of HTSeq-count count–file names to the metadata table we have.

```
### add names of HTSeq count file names to the data

metadata = mutate(metadata,
                  countFile = paste0(metadata$libraryName, "_s_no_DESeq.txt"))

metadata <- lapply(metadata, as.character)
```

We can now create a *DESeq2* summarized experiment object from the count files. In order to fit the requirements of *DESeq2*, we reorganize the our metadata beforehand.

```
sampleTable <- data.frame(sampleName = metadata$libraryName,
                          fileName = metadata$countFile,
                          condition = metadata$condition,
                          sampleNO = metadata$sampleNO,
                          fastq = metadata$fastq
)

DESeq2Table <- DESeqDataSetFromHTSeqCount(sampleTable = sampleTable,
                                          directory = output.dir,
                                          design = ~ condition)
```

The *DESeq2* objects contains metadata on the genomic features (genes in our case) accessible via `rowData`. This feature–metadata is stores as *GRanges* objects. The class *GRanges* can be used to store very general sequence related data, and has its own metadata slot, accessible via `mcols`. Here, additional information are added after the dispersion estimation will have been performed.

```
rowData(DESeq2Table)

   GRangesList object of length 37991:
   $ENSMUSG00000000001
   GRanges object with 0 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>

   $ENSMUSG00000000003
   GRanges object with 0 ranges and 0 metadata columns:
        seqnames ranges strand

   $ENSMUSG00000000028
   GRanges object with 0 ranges and 0 metadata columns:
        seqnames ranges strand

   ...
   <37988 more elements>
   -------
   seqinfo: no sequences

mcols(rowData(DESeq2Table))

   DataFrame with 37991 rows and 0 columns
```

3.7 Add additional annotation information using biomaRt

The bioconductor package *biomaRt* allows to retrieve additional information from the ENSEMBL and other databases using the Bio Mart platform (http://biomart.org/). For an ENSEMBL specific direct web access see also http://www.ensembl.org/biomart/martview.

We first set up the mart and then get the annotation via the function getBM. Here we set up a mart using an archived version since the iGenomes files contain only an older build of the mouse genome (ensembl67–NCBIM37, corresponding to mm9), while the paper uses ensembl76–GRCm38, corresponding to mm10.

```
## get NCBIM37 (mm9)
ensembl67 <- useMart(host='may2012.archive.ensembl.org', biomart='ENSEMBL_MART_ENSEMBL',
                     dataset = "mmusculus_gene_ensembl")
## current annotation (GRCm38, mm10)
ensembl76 <- useMart("ensembl", dataset="mmusculus_gene_ensembl")
```

getBM takes a list of attributes, which basically correspond to an identifier for a certain annotation and a filter which specifies the database keys/identifiers to be used in the query. Here we simply use the gene ID as a key.

You can find out about available attributes and filters with the functions listAttributes and listFilters.

Here we just obtain gene symbols and descriptions for our genes.

```
bm <- getBM(
  attributes=
    c("ensembl_gene_id", "external_gene_id", "description"),
  filter="ensembl_gene_id",
  values= rownames(DESeq2Table),
  mart=ensembl67 )

## arrange rows of bm in ascending order according to the gene ids
bm <- arrange(bm, ensembl_gene_id)

head(bm)

        ensembl_gene_id external_gene_id
   1 ENSMUSG00000000001            Gnai3
   2 ENSMUSG00000000003             Pbsn
   3 ENSMUSG00000000028            Cdc45
   4 ENSMUSG00000000031              H19
   5 ENSMUSG00000000037            Scml2
   6 ENSMUSG00000000049             Apoh
                                                                                        description
   1 guanine nucleotide binding protein (G protein), alpha inhibiting 3 [Source:MGI Symbol;Acc:MGI:95773]
   2                                                       probasin [Source:MGI Symbol;Acc:MGI:1860484]
   3                  cell division cycle 45 homolog (S. cerevisiae) [Source:MGI Symbol;Acc:MGI:1338073]
   4                                              H19 fetal liver mRNA [Source:MGI Symbol;Acc:MGI:95891]
   5                          sex comb on midleg-like 2 (Drosophila) [Source:MGI Symbol;Acc:MGI:1340042]
   6                                                  apolipoprotein H [Source:MGI Symbol;Acc:MGI:88058]
```

We now add this as row (or feature) metadata to the DESeq2 table using a join command. The gene names of the DESeq2 table can be conveniently extracted using the function rownames.

warning: We have to make sure that rows AND that the data types match of the respective columns match, otherwise the join will not work.

Note that instead of a join operation, we also could have used the function cbind to bind the columns to each other.

However, left_join makes sure that the rows joined together do actually match, i.e. that the genes they correspond to are the same. Additionally, one can have multiple hits per query–gene ID (e.g. for biotype). Specifically `left_join` means that all the rows in the left table (our DESeq2 row names) are to be kept and joined to matching entries of the obtained annotation data. In the case of multiple matches, all combination of the matches are returned. (We do not have multiple matches here).

```
DESeq2Features <- data.frame(ensembl_gene_id = rownames(DESeq2Table))
DESeq2Features$ensembl_gene_id <- as.character(DESeq2Features$ensembl_gene_id)

### join them together
rowData <- dplyr::left_join(DESeq2Features, bm, by = "ensembl_gene_id")
rowData <- as(rowData, "DataFrame")

### add the annotation to the DESeq2 table
mcols(rowData(DESeq2Table)) <- c(mcols(rowData(DESeq2Table)),rowData)

#save(DESeq2Table, file = "geneCounts.RData")
```

For convenience, we can load the count data from a pre–saved file.

```
load(url("http://www-huber.embl.de/users/klaus/geneCounts.RData"))
DESeq2Table
```

4 Quality control and Normalization of the count data

Having created a count table and fed it into a DESeq2 object, the next step in the workflow is the quality control of the data. Let's check how many genes we capture by counting the number of genes that have non–zero counts in all samples.

```
GeneCounts <- counts(DESeq2Table)
idx.nz <- apply(GeneCounts, 1, function(x) { all(x > 0)})
sum(idx.nz)

   [1] 16149

### random sample from the count matrix
nz.counts <- subset(GeneCounts, idx.nz)
sam <- sample(dim(nz.counts)[1], 5)
nz.counts[sam, ]

                      WT-SRR1042885 WT-SRR1042886 WT-SRR1042887 WT-SRR1042888 Del_8_17_homo-SRR1042889
   ENSMUSG00000048949             6             6             4            11                        4
   ENSMUSG00000032115          1763          1468          3129          3073                     2134
   ENSMUSG00000021706           457           393           770           760                      599
   ENSMUSG00000026565           579           550          1162          1211                      883
   ENSMUSG00000058396            29            20            35            45                       39
                      Del_8_17_homo-SRR1042890 Del_8_17_homo-SRR1042891 Del_8_17_homo-SRR1042892
   ENSMUSG00000048949                        2                        4                       14
   ENSMUSG00000032115                     1012                     1765                     5608
   ENSMUSG00000021706                      264                      460                     1536
   ENSMUSG00000026565                      354                      635                     2209
   ENSMUSG00000058396                       12                       21                       77
```

In a typical RNA–Seq experiment, there will be at least several thousand genes that are expressed in all samples. If the number of non–zero genes is very low, there is usually something wrong with at least some of the samples (e.g. the sample preparation was not done properly, or there was some problem with the sequencing).

warning: We now correct the annotation error: SRRSRR1042891 and SRR1042886 have the opposite genotype.

```
con <- as.character(colData(DESeq2Table)$condition)
con[2] <- "Del_8_17_homo"
con[7] <- "WT"
```

4.1 Normalization

As different libraries will be sequenced to different depths, offsets are built in the statistical model of *DESeq2* to ensure that parameters are comparable.

The term normalization is often used for that, but it should be noted that the raw read counts are not actually altered. *DESeq2* defines a virtual reference sample by taking the median of each gene's values across samples and then computes size factors as the median of ratios of each sample to the reference sample.

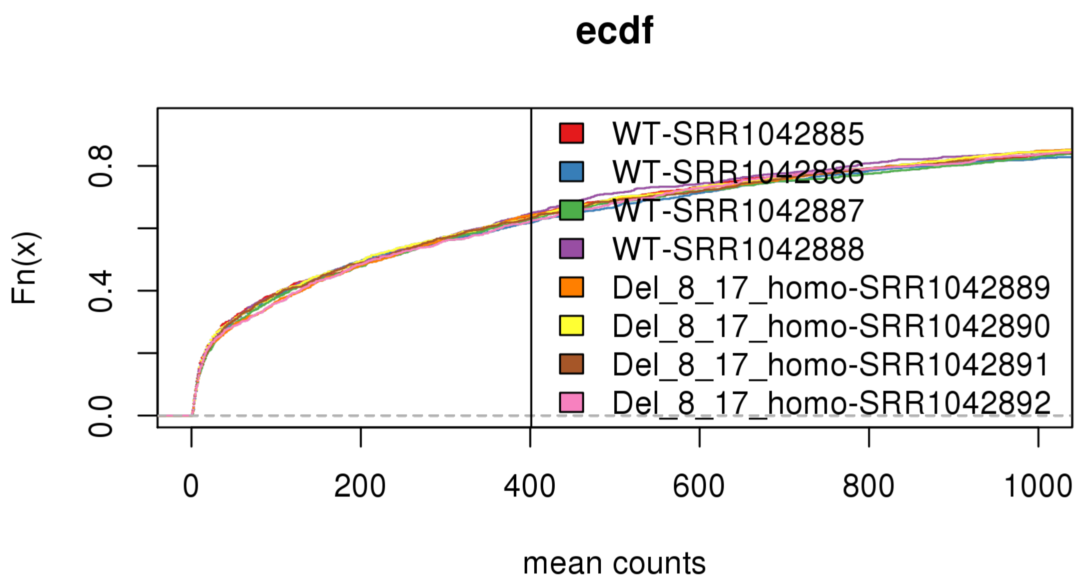
Generally, the ratios of the size factors should roughly match the ratios of the library sizes. Dividing each column of the count table by the corresponding size factor yields normalized count values, which can be scaled to give a counts per million interpretation.

Thus, if all size factors are roughly equal to one, the libraries have been sequenced equally deeply.

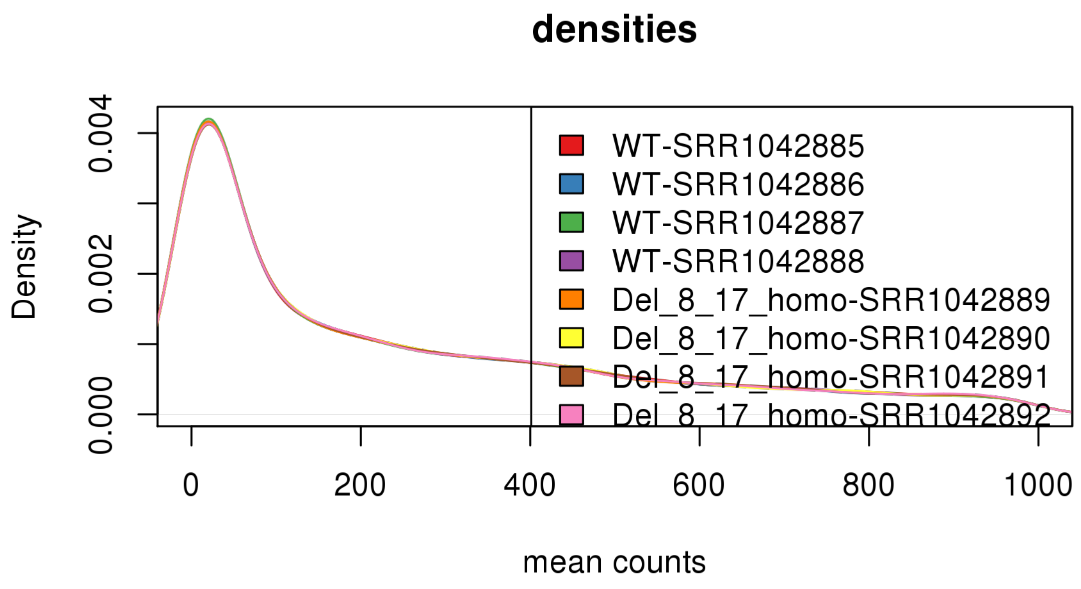
Count data sets typically show a (left-facing) trombone shape in average vs mean–difference plots (MA–plots), reflecting the higher variability of log ratios at lower counts. In addition, points will typically be centered around a log ratio of 0 if the normalization factors are calculated appropriately, although this is just a general guide.

In the code below we produce density estimates of the sample counts and pairwise mean–difference plots after the computation of the size factors. The MA–plots are saved to a file in order to avoid cluttering the document.

```
### make sure to get fold change WT-deletion
colData(DESeq2Table)$condition <- factor(con, levels = c("WT", "Del_8_17_homo"))

#### estimate size factors
DESeq2Table <- estimateSizeFactors(DESeq2Table)
sizeFactors(DESeq2Table)

              WT-SRR1042885           WT-SRR1042886           WT-SRR1042887           WT-SRR1042888
                      0.836                   0.632                   1.398                   1.371
   Del_8_17_homo-SRR1042889 Del_8_17_homo-SRR1042890 Del_8_17_homo-SRR1042891 Del_8_17_homo-SRR1042892
                      1.009                   0.447                   0.860                   2.609

# plot densities of counts for the different samples
### to assess their distributions

 multiecdf( counts(DESeq2Table, normalized = T)[idx.nz ,],
    xlab="mean counts", xlim=c(0, 1000))
```



```
  multidensity( counts(DESeq2Table, normalized = T)[idx.nz ,],
   xlab="mean counts", xlim=c(0, 1000))
```



```
### looks good!

## check pairwise MA plots

pdf("pairwiseMAs.pdf")
  MA.idx = t(combn(1:8, 2))
  for( i in  1:15){
         MDPlot(counts(DESeq2Table, normalized = T)[idx.nz ,],
                c(MA.idx[i,1],MA.idx[i,2]),
        main = paste( colnames(DESeq2Table)[MA.idx[i,1]], " vs ",
         colnames(DESeq2Table)[MA.idx[i,2]] ), ylim = c(-3,3) )
        }
dev.off()

   pdf
     2

## looks good, no systematic shift visible!
```

4.2 PCA and sample heatmaps

A useful first step in an RNA-Seq analysis is often to assess overall similarity between samples: Which samples are similar to each other, which are different? Does this fit to the expectation from the experiment's design? We use the *R* function `dist` to calculate the Euclidean distance between samples. To avoid that the distance measure is dominated by a few highly variable genes, and have a roughly equal contribution from all genes, we use it on the regularized log–transformed data.

The aim of the regularized log–transform is to stabilize the variance of the data and to make its distribution roughly symmetric since many common statistical methods for exploratory analysis of multidimensional data, especially methods for clustering and ordination (e.g., principal-component analysis and the like), work best for (at least approximately) homoskedastic data; this means that the variance of an observable quantity (i.e., here, the expression strength of a gene) does not depend on the mean.

In RNA-Seq data, however, variance grows with the mean. For example, if one performs PCA directly on a matrix of normalized read counts, the result typically depends only on the few most strongly expressed genes because they show the largest absolute differences between samples.

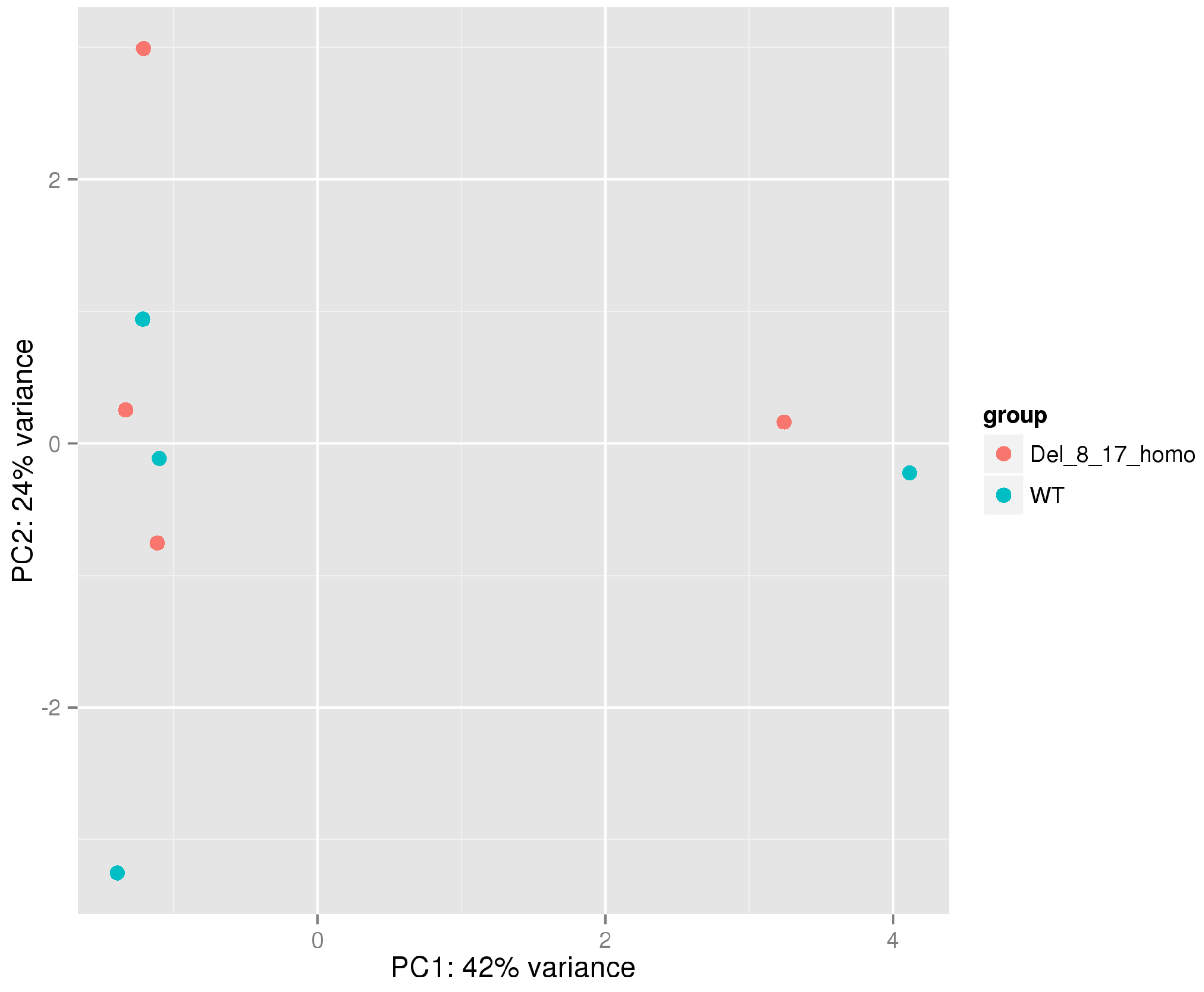
A simple and often used strategy to avoid this is to take the logarithm of the normalized count values plus a small pseudocount; however, now the genes with low counts tend to dominate the results because, due to the strong Poisson noise inherent to small count values, they show the strongest relative differences between samples. Note that this effect can be diminished by adding a relatively high number of pseudocounts, e.g. 32, since this will also substantially reduce the variance of the fold changes.

As a solution, *DESeq2* offers the regularized–logarithm transformation, or rlog for short. For genes with high counts, the rlog transformation differs not much from an ordinary log2 transformation.

For genes with lower counts, however, the values are shrunken towards the genes' averages across all samples. Using an empirical Bayesian prior in the form of a ridge penality, this is done such that the rlog-transformed data are approximately homoskedastic. Note that the rlog transformation is provided for applications other than differential testing. For differential testing it is always recommended to apply the DESeq function to raw counts.

Note the use of the function `t` to transpose the data matrix. We need this because dist calculates distances between data rows and our samples constitute the columns. We visualize the distances in a heatmap, using the function `heatmap.2` from the *gplots* package. The heatmap is saved as a '.pdf' file.

```
### produce rlog-transformed data

rld <- rlogTransformation(DESeq2Table, blind=TRUE)
## create a distance matrix between the samples

pdf("HeatmapPlots.pdf")
distsRL <- dist(t(assay(rld)))
mat <- as.matrix(distsRL)

hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
```



PCA plot

```
heatmap.2(mat, trace="none", col = rev(hmcol), margin=c(13, 13))
dev.off()

   pdf
     2
```

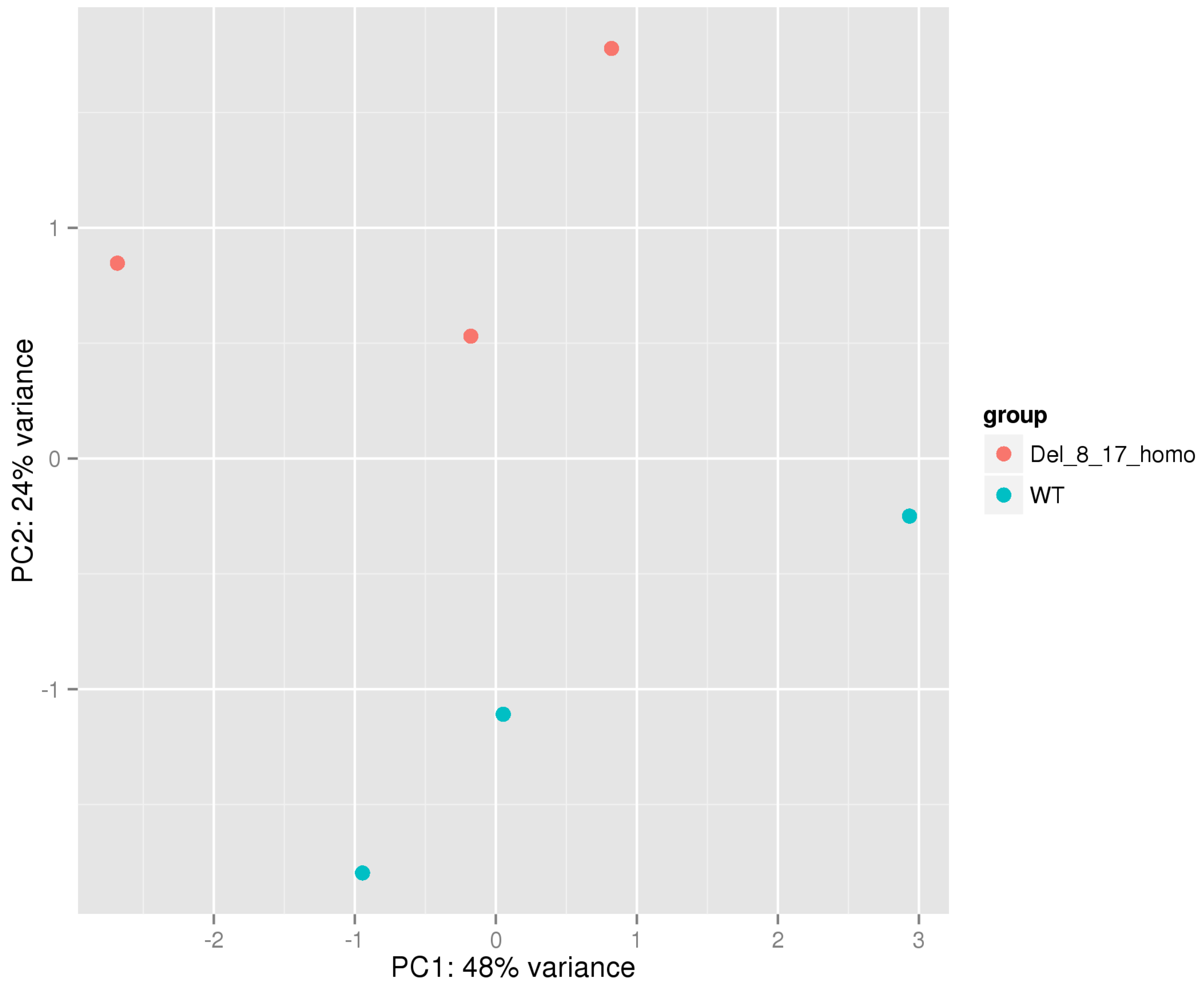
Another way to visualize sample–to–sample distances is a principal–components analysis (PCA). In this ordination method, the data points (i.e., here, the samples) are projected onto the 2D plane such that they spread out optimally.

Here, we use the function `plotPCA` which comes with *DESeq2*. The term specified as `intgroup` are the column names from our sample data; they tell the function to use them to choose colors.

```
DESeq2::plotPCA(rld, intgroup=c("condition"))
```

```
#dev.off()
```

We can clearly identify to outliers in the PCA plot, one in each experimental groups. These two outliers basically provide the separation according to the first principal component. Thus, they outliers will most likely increase the overall variability and thus diminish statistical power later when testing for differential expression. Therefore we remove them.



PCA plot without outliers

```
pdf("HeatmapPlots_no_outliers.pdf")
outliers <- c( "WT-SRR1042888", "Del_8_17_homo-SRR1042890")
DESeq2Table.sub <- DESeq2Table[, !(colnames(DESeq2Table) %in% outliers)]
rld <- rlogTransformation(DESeq2Table.sub, blind=TRUE)
distsRL <- dist(t(assay(rld)))
mat <- as.matrix(distsRL)

hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
heatmap.2(mat, trace="none", col = rev(hmcol), margin=c(13, 13))
dev.off()

   pdf
     2
```

```
DESeq2::plotPCA(rld, intgroup=c("condition"))
```

```
#dev.off()
```

```
## remove outliers
outliers <- c( "WT-SRR1042888", "Del_8_17_homo-SRR1042890")
DESeq2Table <- DESeq2Table[, !(colnames(DESeq2Table) %in% outliers)]
```

5 Differential expression analysis

5.1 Dispersion estimation

It is known that a standard Poisson model can only account for the technical noise in RNA–Seq data. In the Poisson model the variance is equal to the mean, while in real RNA–Seq data the variance is greater than the mean, a phenomenon often encountered in count data in general and referred to as "overdispersion".

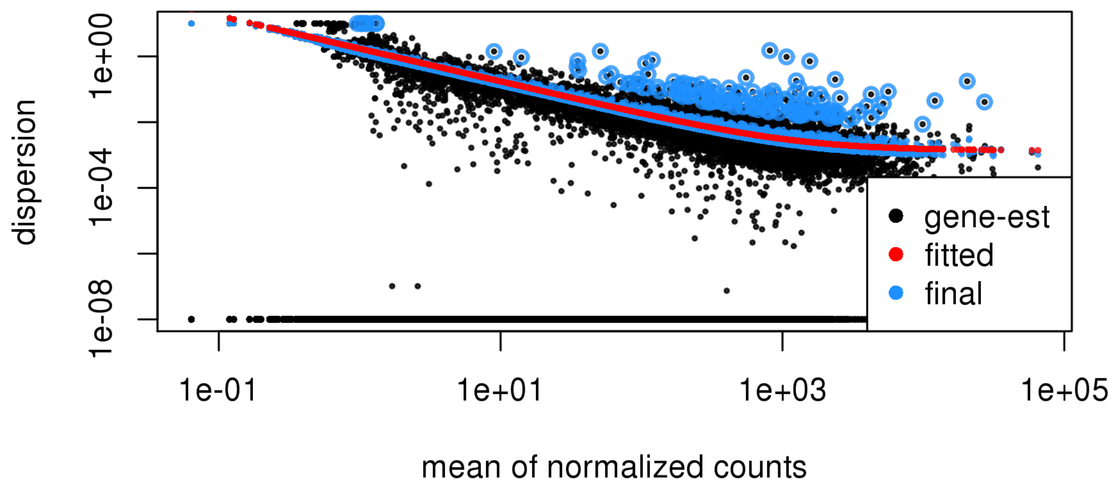
A popular way to model this is to use a Negative–Binomial-distribution (NB), which includes an additional parameter dispersion parameter α such that $E(NB) = \mu$ and

$$\text{Var}[\text{NB}(\mu, \alpha)] = \mu + \alpha \cdot \mu^2$$

Hence, the variance is greater than the mean. *DESeq2* also uses the NB model. Hence, the first step in the analysis of differential expression, is to obtain an estimate of the dispersion parameter for each gene. The typical shape of the dispersion fit is an exponentially decaying curve. The asymptotic dispersion for highly expressed genes can be seen as a measurement of biological variability in the sense of a squared coefficient of variation: a dispersion value of 0.01 means that the gene's expression tends to differ by typically $\sqrt{0.01} = 10\%$ between samples of the same treatment group.

For weak genes, the Poisson noise is an additional source of noise, which is added to the dispersion. The function `plotDispEsts` visualizes *DESeq2*'s dispersion estimates:

```
DESeq2Table <- estimateDispersions(DESeq2Table)

  gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates

plotDispEsts(DESeq2Table)
```



The black points are the dispersion estimates for each gene as obtained by considering the information from each gene separately. Unless one has many samples, these values fluctuate strongly around their true values.

Therefore, the red trend line is fitted, which shows the dispersions' dependence on the mean, and then shrink each gene's estimate towards the red line to obtain the final estimates (blue points) that are then used in the hypothesis test.

The blue circles above the main "cloud" of points are genes which have high gene–wise dispersion estimates which are labelled as dispersion outliers. These estimates are therefore not shrunk toward the fitted trend line.

The warnings just indicate that the dispersion estimation failed for some genes.

5.2 Statistical testing of Differential expression

We can perform the statistical testing for differential expression and extract its results. Calling `nbinomWaldTest` performs the test for differential expression, while the call to the `results` function extracts the results of the test and returns adjusted p–values according to the Benjamini–Hochberg rule to control the FDR. The test–performed is Wald test, which is a test for coefficients in a regression model. It is based on a z–score, i.e. a $N(0,1)$ distributed (under H_0) test statistic.

```
DESeq2Table <-  nbinomWaldTest(DESeq2Table)
DESeq2Res <- results(DESeq2Table, pAdjustMethod = "BH")

### number of siginificant DE-genes
table(DESeq2Res$padj < 0.1)


  FALSE  TRUE
  14526   135
```

We identify 135 differentially expressed genes.

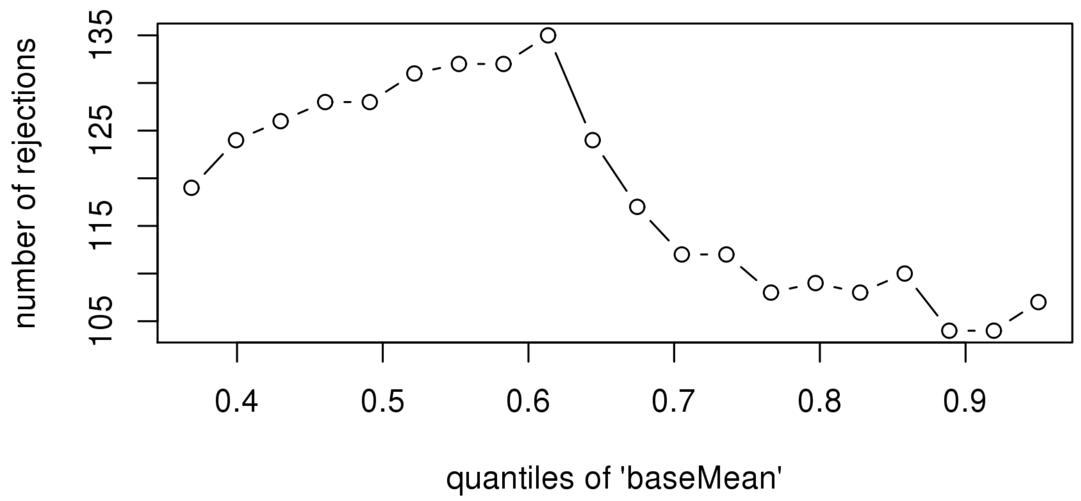
5.2.1 Independent filtering

In addition to this, *DESeq2* performs automated independent filtering: For weakly expressed genes, we have no chance of seeing differential expression, because the low read counts suffer from so high Poisson noise that any biological effect is drowned in the uncertainties from the read counting.

At first sight, there may seem to be little benefit in filtering out these genes. After all, the test found them to be non–significant anyway. However, these genes have an influence on the multiple testing procedure, whose performance commonly improves if such genes are removed. By removing the weakly–expressed genes from the input to the BH–FDR procedure, we can find more genes to be significant among those which we keep, and so improve the power of our test. This approach is known as independent filtering.

The *DESeq2* software automatically performs independent filtering which maximizes the number of genes which will have a BH–adjusted p–value less than a critical value (by default, alpha is set to 0.1). This automatic independent filtering is performed by, and can be controlled by, the results function. We can observe how the number of rejections changes for various cutoffs based on mean normalized count. The following optimal threshold and table of possible values is stored as an attribute of the results object.

```
attr(DESeq2Res,"filterThreshold")

   61.3%
    6.35

plot(attr(DESeq2Res,"filterNumRej"),type="b", xlab="quantiles of 'baseMean'",
     ylab="number of rejections")
```



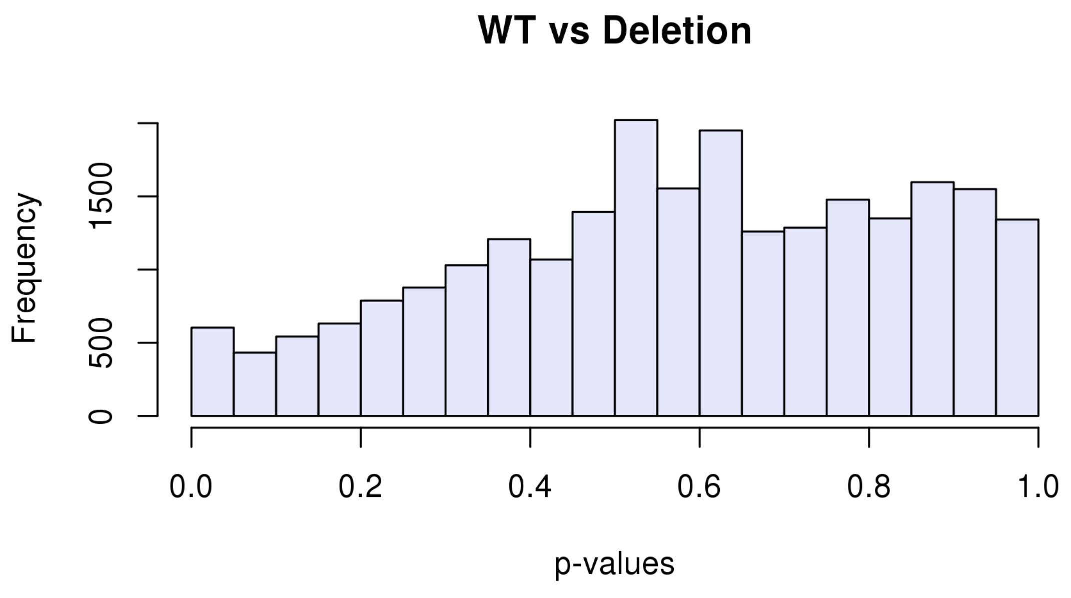
The term *independent* highlights an important caveat. Such filtering is permissible only if the filter criterion is independent of the actual test statistic under the null–hypothesis.

Otherwise, the filtering would invalidate the test and consequently the assumptions of the BH procedure. This is why we filtered on the average over all samples: this filter is blind to the assignment of samples to the treatment and control group and hence independent under the null hypothesis of equal expression.

5.2.2 Inspection and correction of p–values

The null–p–values follow a uniform distribution on the unit interval [0,1] if they are computed using a continuous null distribution. Significant p–values thus become visible as an enrichment of p–values near zero in the histogram.

Thus, p–value histogram of "correctly" computed p–values will have a rectangular shape with a peak at 0.



p-values, wrong null distribution

A histogram of p–values should always be plotted in order to check whether they have been computed correctly. We also do this here:

```r
hist(DESeq2Res$pvalue, col = "lavender",
      main = "WT vs Deletion", xlab = "p-values")
```

We can see that this is clearly not the case for the p–values returned by *DESeq2* in this case.

Very often, if the assumed variance of the null distribution is too high, we see hill–shaped p–value histogram. If the variance is too low, we get a U–shaped histogram, with peaks at both ends.

Here we have a hill–shape, indicating an overestimation of the variance in the null distribution. Thus, the $N(0,1)$ null distribution of the Wald test is not appropriate here.

The dispersion estimation is not condition specific and estimates only a single dispersion estimate per gene. This is sensible, since the number of replicates is usually low.

However, if we have e.g. batches or "outlying" samples that are consistently a bit different from others within a group, the dispersion within the experimental group can be different and a single dispersion parameter not be appropriate.

For an example of the estimation of multiple dispersions, see the analysis performed in: Reyes et. al. ???- Drift and conservation of differential exon usage across tissues in primate species, 2013

Fortunately, there is software available to estimate the variance of the null–model from the test statistics. This is commonly referred to as "empirical null modelling".

Here we use the *fdrtool* for this using the Wald statistic as input. This packages returns the estimated null variance, as well as estimates of various other FDR–related quantities and the p–values computed using the estimated null model parameters. An alternative and widely used–package for this task is *locfdr*.

```r
### remove filtered out genes by independent filtering,
### they have NA adj. pvals
DESeq2Res <- DESeq2Res[ !is.na(DESeq2Res$padj), ]
```

```
### remove genes with NA pvals (outliers)
DESeq2Res <- DESeq2Res[ !is.na(DESeq2Res$pvalue), ]

### remove adjsuted pvalues, since we add the fdrtool results later on
### (based on the correct p-values)
DESeq2Res <- DESeq2Res[, -which(names(DESeq2Res) == "padj")]


### use z-scores as input to FDRtool to re-estimate the p-value

FDR.DESeq2Res <- fdrtool(DESeq2Res$stat, statistic= "normal", plot = T)

   Step 1... determine cutoff point
   Step 2... estimate parameters of null distribution and eta0
   Step 3... compute p-values and estimate empirical PDF/CDF
   Step 4... compute q-values and local fdr
   Step 5... prepare for plotting
```

```
### null model variance

FDR.DESeq2Res$param[1, "sd"]

      sd
   0.817

### add values to the results data frame, also ad new BH- adjusted p-values
DESeq2Res[,"padj"]  <- p.adjust(FDR.DESeq2Res$pval, method = "BH")
```

fdrtool estimates the variance of the null model as 0.817, which is less than the theoretical sd of 1, as expected from the p–value histogram. We can plot the histogram of the “correct” p–values.

```
hist(FDR.DESeq2Res$pval, col = "royalblue4",
     main = "WT vs Deletion, correct null model", xlab = "CORRECTED p-values")
```

As we can see, the null model is now correct and the histogram has the expected shape.

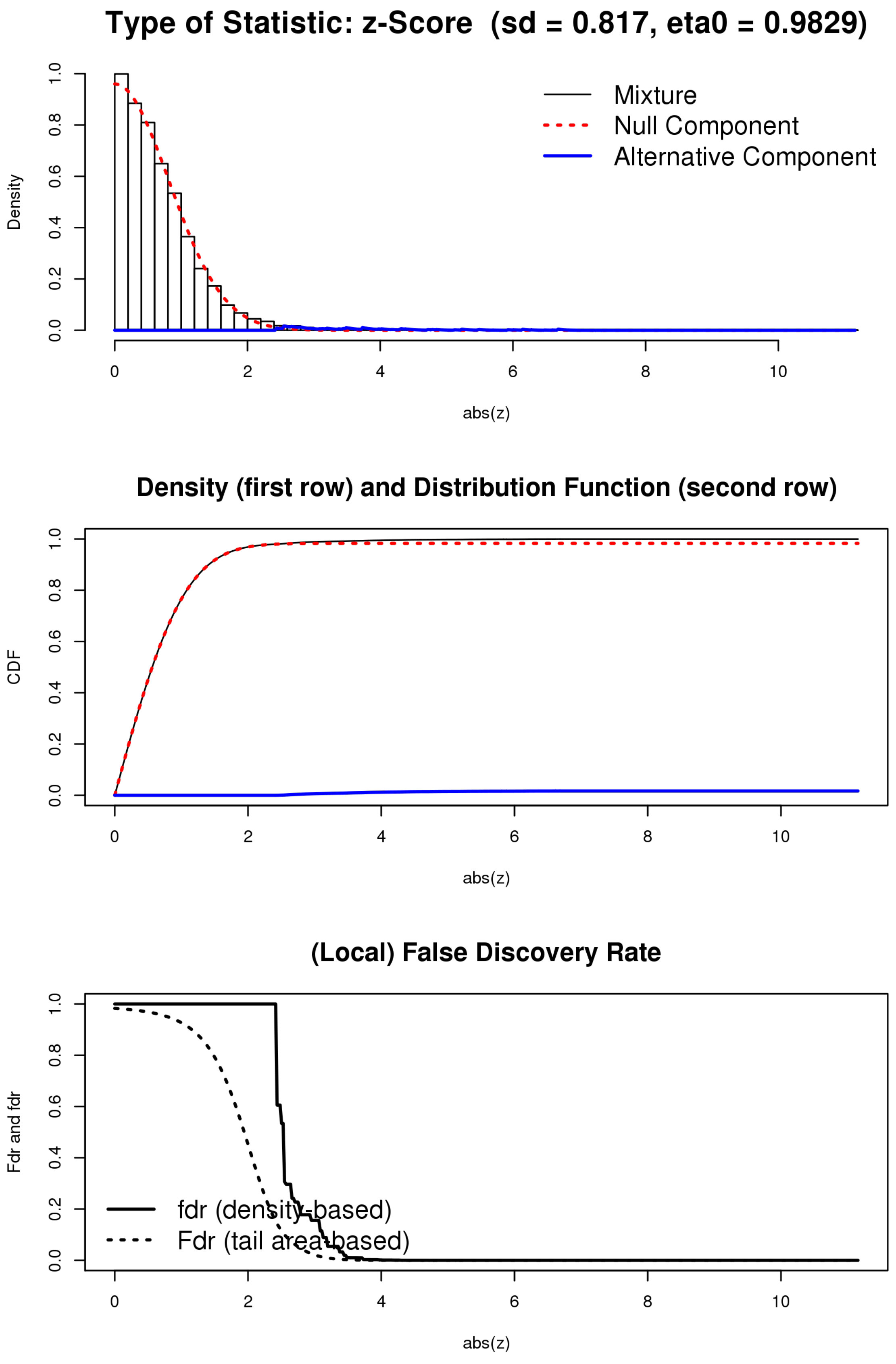
5.2.3 Extracting differentially expressed genes

We can now extract the number of differentially expressed genes and produce an MA plot of the two–group comparison as shown in figure 4a of the original paper. Similar to its usage in the initial quality control, a MA plot provides a useful overview for an experiment with a two-group comparison:

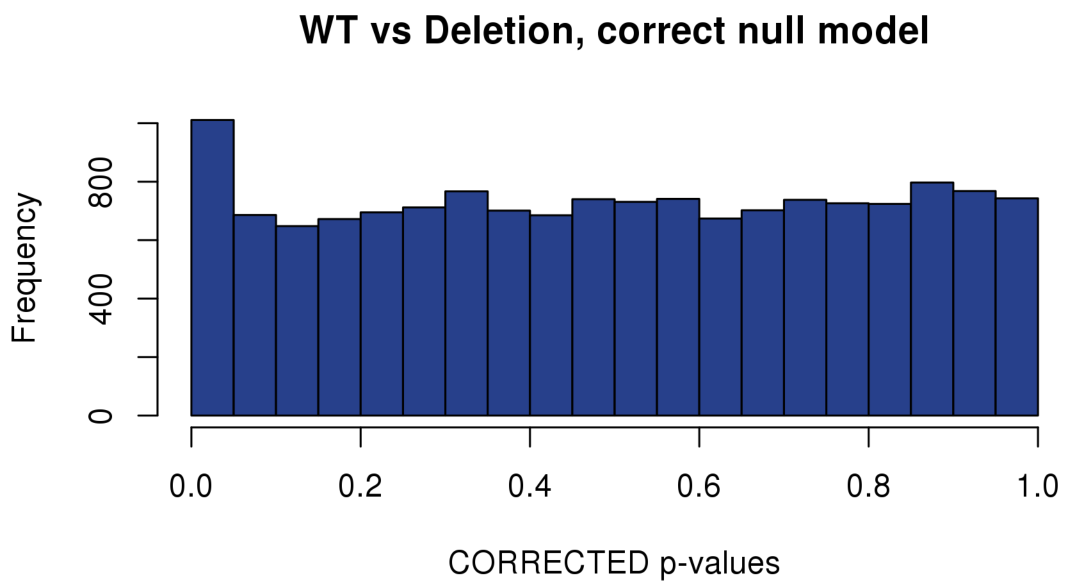
The plot represents each gene with a dot. The x–axis is the average expression over all samples, the y axis the log2 fold change between treatment and control. Genes with an FDR value below a threshold (here 0.1) are shown in red. This plot demonstrates that only genes with a large average normalized count contain more information to yield a significant call.

Also note *DESeq2*'s shrinkage estimation of log fold changes (LFCs): When count values are too low to allow an accurate estimate of the LFC, the value is “shrunken” towards zero to avoid that these values, which otherwise would frequently be unrealistically large, dominate the top–ranked log fold changes.

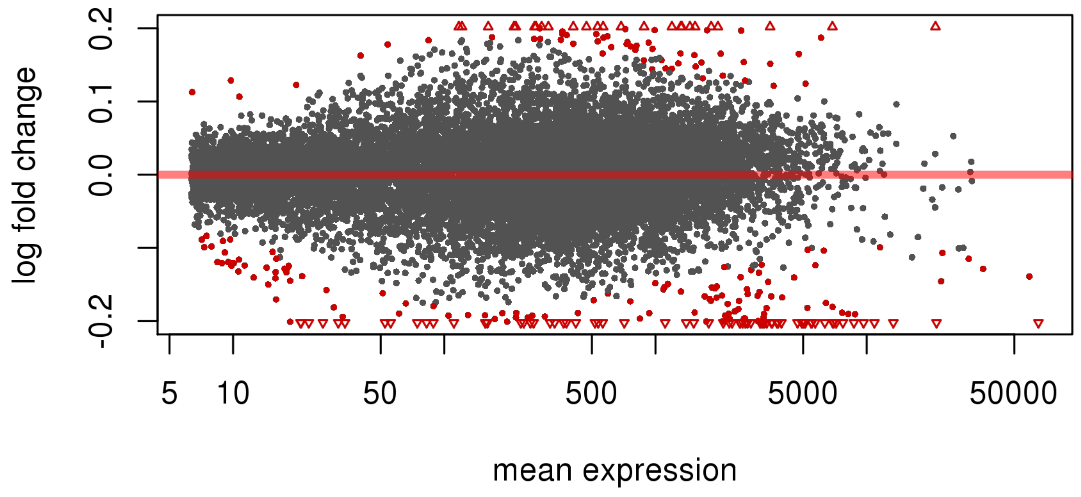
```
table(DESeq2Res[,"padj"] < 0.1)


   FALSE  TRUE
   14400   261

plotMA(DESeq2Res)
```



fdrtool output, including estimating null model standard deviation



p-values, correct null distribution



MA plot for DESeq2 analysis

We now identify 261 differentially expressed genes at an FDR of 0.1.

5.2.4 Check overlap with the paper results

We can now check the overlap with results of the paper. In the original publication 100 genes with an FDR value of less than 0.05 were identified. The excel table "ng.2971–S3.xls" containing their FC and p–values is available as supplementary table 4 of the original publication.

```
paperRes <- read.csv("ng.2971-S3.csv", skip = 1, header =T)
names(paperRes)

    [1] "Gene_ID"        "baseMean"       "baseMean.WT"    "baseMean.DEL"   "fold.change"
    [6] "log2FoldChange" "pval"           "padj"           "GROUP"          "Gene.name"
   [11] "chr"            "start.mm10."    "end.mm10."      "strand"         "Gene.name.1"
   [16] "KEGG"           "GO"

sigGenes <- rownames(subset(DESeq2Res, padj < 0.1))

## extract annotation of significant genes
anno <- mcols(rowData(DESeq2Table))[,1:3]
anSig <- subset(anno, ensembl_gene_id %in% sigGenes)

### check overlap
overlap <- subset(paperRes, Gene.name  %in% anSig$external_gene_id )
dim(overlap)

   [1] 208  17

### how many genes that we identify with FDR < 0.1 also had
### FDR < 0.1 in the original paper?
dim(subset(overlap, padj < 0.1))[1]

   [1] 87

### total number of genes with FDR < 0.1 in the original paper
dim(subset(paperRes, padj < 0.1))[1]

   [1] 116

#write.csv( subset(overlap, padj < 0.1), file = "overlapSig.csv")
#save(anSig, DESeq2Res, overlap, file = "ResDEAnalysis.RData")
```

Out of the 261 differentially expressed genes that we identify at an FDR of 0.1, 87 have also been identified in the original paper (out of 116 in total in the original paper with an FDR of 0.1).

The MA–plot also looks very similar to the paper plot. Thus, we can reproduce the paper results with a slightly different software setting and can gain power by correcting the null model. The "blood" and "ribosomal proteins" parts of the MA–plot marked in the paper figure are also clearly visible.

6 Gene ontology enrichment analysis

We can now try characterize the identified differentially expressed genes a bit better by performing an GO enrichment analysis. Essentially the gene ontology (`http://www.geneontology.org/`) is hierarchically organized collection of functional gene sets. For a nice introduction to the GO see

- du Plessis et. al. – The what, where, how and why of gene ontology primer for bioinformaticians- Bioinformatics, 2011

6.1 Matching the background set

The function `genefinder` from the *genefilter* package will be used to find background genes that are similar in expression to the differentially expressed genes. The function tries to identify 10 genes for each DE–gene that match its expression strength.

We then check whether the background has roughly the same distribution of average expression strength as the foreground by plotting the densities.

We do this in order not to select a biased background since the gene set testing is performed by a simple Fisher test on a 2x2 table, which uses only the status of a gene, i.e. whether it is differentially expressed or not and not its fold–change or absolute expression.

Note that the chance of a gene being identified as DE will most probably depend its gene expression for RNA–Seq data (potentially also its gene length etc.). Thus it is important to find a matching background. Our the testing approach here is very similar to web tools like DAVID, however, we explicitly model the background here.

```
## get average expressions
overallBaseMean <- as.matrix(DESeq2Res[, "baseMean", drop = F])

backG <- genefinder(overallBaseMean, anSig$ensembl_gene_id, 10, method = "manhattan")
## get identified similar genes
backG <- rownames(overallBaseMean)[as.vector(sapply(backG, function(x)x$indices))]
## remove DE genes from background
backG <- setdiff(backG,  anSig$ensembl_gene_id)
## number of genes in the background
length(backG)

   [1] 2049

  multidensity( list(
       all= log2(DESeq2Res[,"baseMean"]) ,
       fore=log2(DESeq2Res[anSig$ensembl_gene_id, "baseMean"]),
       back=log2(DESeq2Res[backG, "baseMean"])),
     xlab="log2 mean counts", main = "Matching for enrichment analysis")
```

We can see that the matching returned a sensible result, we can now perform the actual testing. For this purpose we use the *topGO* package which implements a nice interface to Fisher testing and also has additional algorithms taking the GO structure into account, by e.g. only reporting the most specific gene set in the hierarchy. For more details, see the paper:

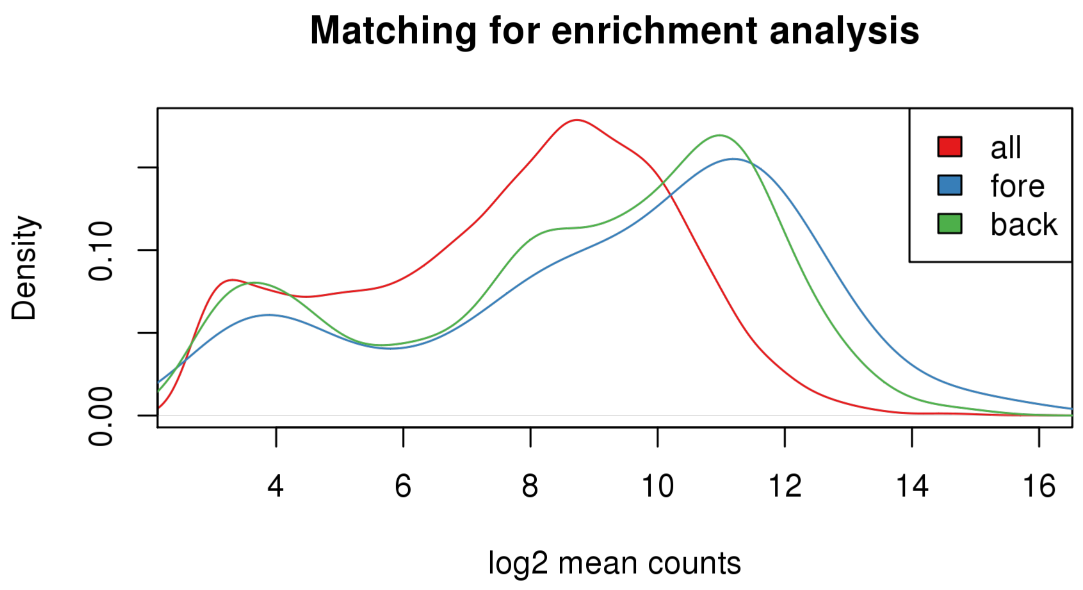
- Alexa et. al. – Improved scoring of functional groups from gene expression data by decorrelating GO graph structure - Bioinformatics, 2006

The GO has three top ontologies, cellular component (CC), biological processes (BP), and molecular function (MF). Here we test all off them subsequently.

6.2 Running topGO

We first create a factor `alg` which indicates for every gene in our universe (union of background and DE-genes), whether it is differentially expressed or not. It has the ENSEMBL ID's of the genes in our universe as names and contains 1 if the gene is DE and 0 otherwise.

```
onts = c( "MF", "BP", "CC" )

geneIDs = rownames(overallBaseMean)
inUniverse = geneIDs %in% c(anSig$ensembl_gene_id,  backG)
inSelection =  geneIDs %in% anSig$ensembl_gene_id
```



matching foreground and background

```
alg <- factor( as.integer( inSelection[inUniverse] ) )
names(alg) <- geneIDs[inUniverse]
```

We first initialize the *topGO* data set for each top ontology, using the GO annotations contained in the annotation data base for mouse *org.Mm.eg.db*. The `nodeSize` parameter specifies a minimum size of a GO category we want to use: i.e. here categories with less than 5 genes are not included in the testing. Since we have ENSEMBL IDs as our key, we have to specify it since *topGO* uses ENTREZ identifiers by default.

Now the tests can be run. *topGO* offers a wide range of options, for details see the paper or the package vignette.

Here we run two common tests: an ordinary Fisher test for every GO category, and the "elim" algorithm, which tries to incorporate the hierarchical structure of the GO and to "decorrelate" it.

The "elim" algorithm starts processing the nodes/GO category from the highest (bottommost) level and then iteratively moves to nodes from a lower level. If a node is scored as significant, all of its genes are marked as removed in all ancestor nodes. This way, the "elim" algorithm aims at finding the most specific node for every gene.

The tests use a 0.01 p–value cutoff by default. We order by the classic algorithm to make the analysis comparable to the one performed in the original paper. The enrichment results of the original publication are reported in supplementary table 6 and 7.

```
tab = as.list(onts)
names(tab) = onts
### test all three top level ontologies
  for(i in 1:3){

  ## prepare data
        tgd <- new( "topGOdata", ontology=onts[i], allGenes = alg, nodeSize=5,
                 annot=annFUN.org, mapping="org.Mm.eg.db", ID = "ensembl" )

  ## run tests
        resultTopGO.elim <- runTest(tgd, algorithm = "elim", statistic = "Fisher" )
```

```
      resultTopGO.classic <- runTest(tgd, algorithm = "classic", statistic = "Fisher" )

  ## look at results
      tab[[i]] <- GenTable( tgd, Fisher.elim = resultTopGO.elim,
             Fisher.classic = resultTopGO.classic,
             orderBy = "Fisher.classic" , topNodes = 200)

      }
```

We can now look at the results, we look at the top 200 GO categories according to the "Fisher classic" algorithm. The function GenTable produces a table of significant GO categories. Finally, we bind all resulting tables together and write them to an '.csv'–file that can be view with spreadsheet programs.

```
topGOResults <- rbind.fill(tab)
write.csv(topGOResults, file = "topGOResults.csv")
```

Gene set enrichment analysis has been and is a field of very extensive research in bioinformatics. For additional approaches see the *topGO* vignette and the references therein and also GeneSetEnrichment workflow in Bioconductor:

- http://bioconductor.org/packages/release/BiocViews.html#___GeneSetEnrichment