Sequencing: RNA-seq data intro

Koen Van den Berge

8/6/2021

Contents

In this lecture we will start working with a real bulk RNA-seq dataset from Haglund *et al.* (2012). After importing the data, we will be working our way through four major challenges which, together, will form a full RNA-seq differential expression (DE) analysis pipeline where the result of our analysis will be a(n ordered) list of genes that we find to be differently expressed between our conditions of interest. The four main challenges we will look into are

- Choice of modeling assumptions (distribution).
- Normalization.
- Parameter estimation under a limited information setting.
- Statistical inference under high dimensionality (many genes).

1 Experimental design, data import and data exploration

1.1 Experimental design

Let's try to work out the experimental design using the following paragraph from the Methods section of the paper.

> Tissue for cell culturing was obtained from four chief cell parathyroid adenomas collected directly at surgery from female postmenopausal patients. Isolation of cells and culturing were performed essentially as previously described (22). Cells were plated and treated with 100 nM DPN (Tocris Bioscience, Minneapolis, MN) or 100 nM OHT (Sigma-Aldrich, St. Louis, MO) for 24 or 48 h, respectively. Untreated cells cultured in parallel were used as controls. Cells were harvested in RNeasy (QIAGEN AB, Hilden, Germany), and quality control was performed using Bioanalyzer (Agilent Technologies, Santa Clara, CA) and Nanodrop (Nanodrop Technology, Wilmington, DE) for all specimens. RNA samples were isolated from four different adenomas, and one sample (case 4, control 24 h) was omitted before transcriptome sequencing based on low RIN value. The entire sample set used for sequencing consisted of the treatment groups DPN 24 h (n = 4), DPN 48 h (n = 4), OHT 24 h (n = 4), OHT 48 h (n = 4), control 24 h (n = 3), and control 48 h (n = 4).

Figure 1: Figure: A paragraph from the Methods section.

1.2 Data import and exploration

We will be importing the dataset using the parathyroidSE data package from Bioconductor.

```
if (!requireNamespace("BiocManager", quietly = TRUE)){
  install.packages("BiocManager")
}
if(!"SummarizedExperiment" %in% installed.packages()[,1]){
  BiocManager::install("SummarizedExperiment")
}
# install package if not installed.
if(!"parathyroidSE" %in% installed.packages()[,1]) BiocManager::install("parathyroidSE")

suppressPackageStartupMessages({
  library(parathyroidSE)
  library(SummarizedExperiment)
})

# import data
data("parathyroidGenesSE", package="parathyroidSE")
# rename for convenience
se1 <- parathyroidGenesSE
rm(parathyroidGenesSE)

# three treatments
treatment1 <- colData(se1)$treatment
table(treatment1)
```

```
## treatment1
## Control     DPN     OHT
##       7      10      10
```

```
# two timepoints
time1 <- colData(se1)$time
table(time1)
```

```
## time1
## 24h 48h
##  13  14
```

```
# four donor patients
patient1 <- colData(se1)$patient
table(patient1)
```

```
## patient1
## 1 2 3 4
## 6 8 6 7
```

```
table(patient1, treatment1, time1)
```

```
## , , time1 = 24h
##
##         treatment1
## patient1 Control DPN OHT
##        1       1   1   1
##        2       1   2   2
##        3       1   1   1
##        4       0   1   1
##
## , , time1 = 48h
##
##         treatment1
## patient1 Control DPN OHT
##        1       1   1   1
##        2       1   1   1
##        3       1   1   1
##        4       1   2   2
```

- We observe that the number of samples that we are observing here is larger than what is described in the paper. As also described in the parathyroidSE vignette, some samples were spread over multiple sequencing runs (i.e., the same sample being sequenced repeatedly) and therefore constitute **technical replication**, rather than biological replication.
- We have previously seen that technical replicates can be considered to be distributed according to a Poisson distribution. One important property of Poisson random variables is that a sum of Poisson random variables still follow a Poisson distribution. Indeed, if $X \sim Poi(\mu_X)$ and $Y \sim Poi(\mu_Y)$, then $ X + Y = Z Poi(_X + _Y)$.

- For this reason, it is often suggested to sum technical replicates rather than, for example, averaging, which does not retain the Poisson property (try for yourself!). We'll therefore first sum the technical replicates.

```
dupExps <- as.character(colData(se1)$experiment[duplicated(colData(se1)$experiment)])
dupExps
```

```
## [1] "SRX140511" "SRX140513" "SRX140523" "SRX140525"
```

```
counts <- assays(se1)$counts
newCounts <- counts
cd <- colData(se1)
for(ss in 1:length(dupExps)){
  # check which samples are duplicates
  relevantId <- which(colData(se1)$experiment == dupExps[ss])
  # sum counts
  newCounts[,relevantId[1]] <- rowSums(counts[,relevantId])
  # keep which columns / rows to remove.
  if(ss == 1){
    toRemove <- relevantId[2]
  } else {
    toRemove <- c(toRemove, relevantId[2])
  }
}

# remove after summing counts (otherwise IDs get mixed up)
newCounts <- newCounts[,-toRemove]
newCD <- cd[-toRemove,]

# Create new SummarizedExperiment
se <- SummarizedExperiment(assays = list("counts" = newCounts),
                           colData = newCD,
                           metadata = metadata(se1))

treatment <- colData(se)$treatment
table(treatment)
```

```
## treatment
## Control     DPN     OHT
##       7       8       8
```

```
time <- colData(se)$time
table(time)
```

```
## time
## 24h 48h
##  11  12
```

```
patient <- colData(se)$patient
table(patient)
```

```
## patient
## 1 2 3 4
## 6 6 6 5
```

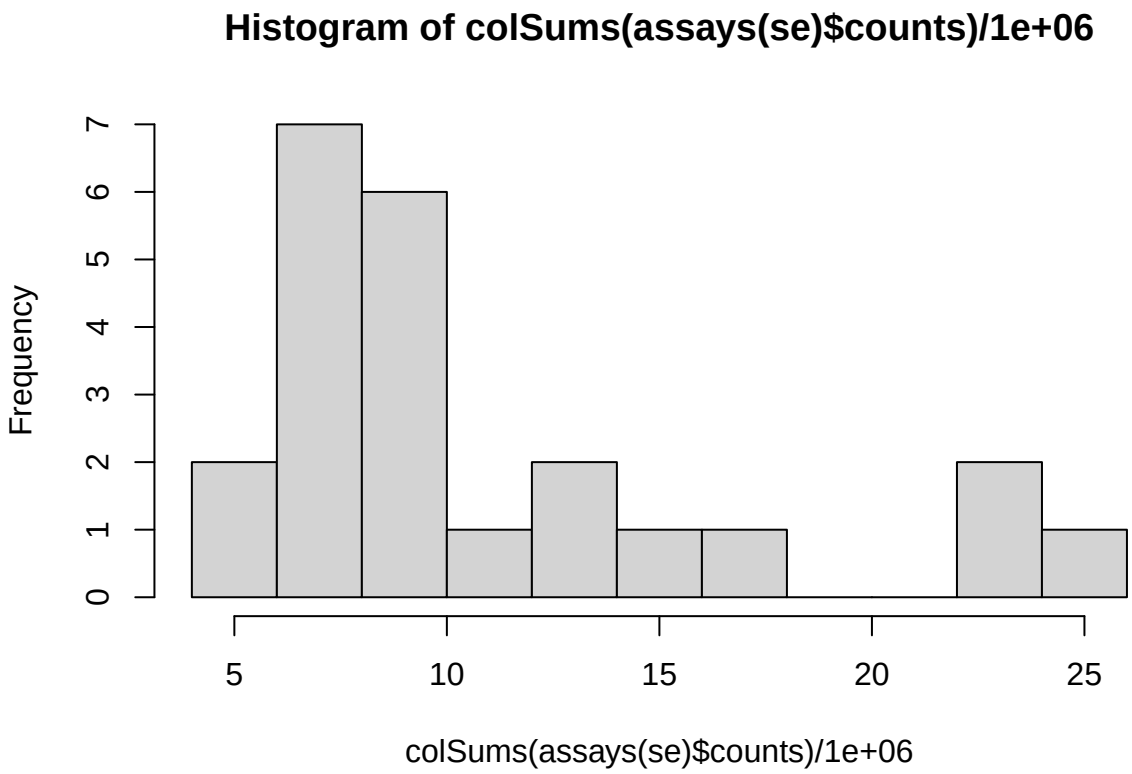
```
table(patient, treatment, time) # agrees with paper.
```

```
## , , time = 24h
##
##         treatment
## patient Control DPN OHT
##       1       1   1   1
##       2       1   1   1
##       3       1   1   1
##       4       0   1   1
##
## , , time = 48h
##
##         treatment
## patient Control DPN OHT
##       1       1   1   1
##       2       1   1   1
##       3       1   1   1
##       4       1   1   1
```

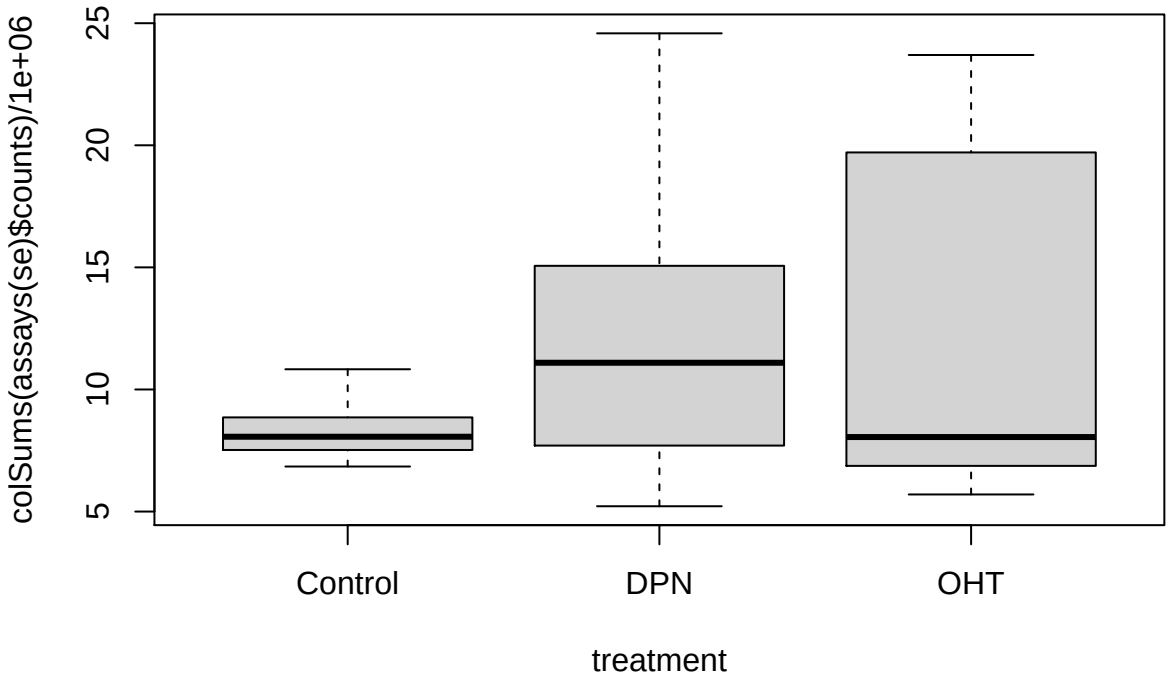
- After summing the technical replicates and appropriately updating the sample information, we again create a `SummarizedExperiment` object, which is essentially a data container that contains all relevant information about your experiment. Please see the vignette for more information on how to use this class.
- By directly matching columns (samples) and rows (genes) to their relevant metadata, the `SummarizedExperiment` class avoids mistakes by mis-matching columns and rows with each other (provided you haven't mismatched them when you creat the object).
- The `SummarizedExperiment` class is modular and extendable, and extensions exist for example for the analysis of single-cell RNA-seq data.
- Due to their convenient organization and widely supported usage within Bioconductor, we will typically work with such containers in the analysis of RNA-seq data.

1.3 Data exploration

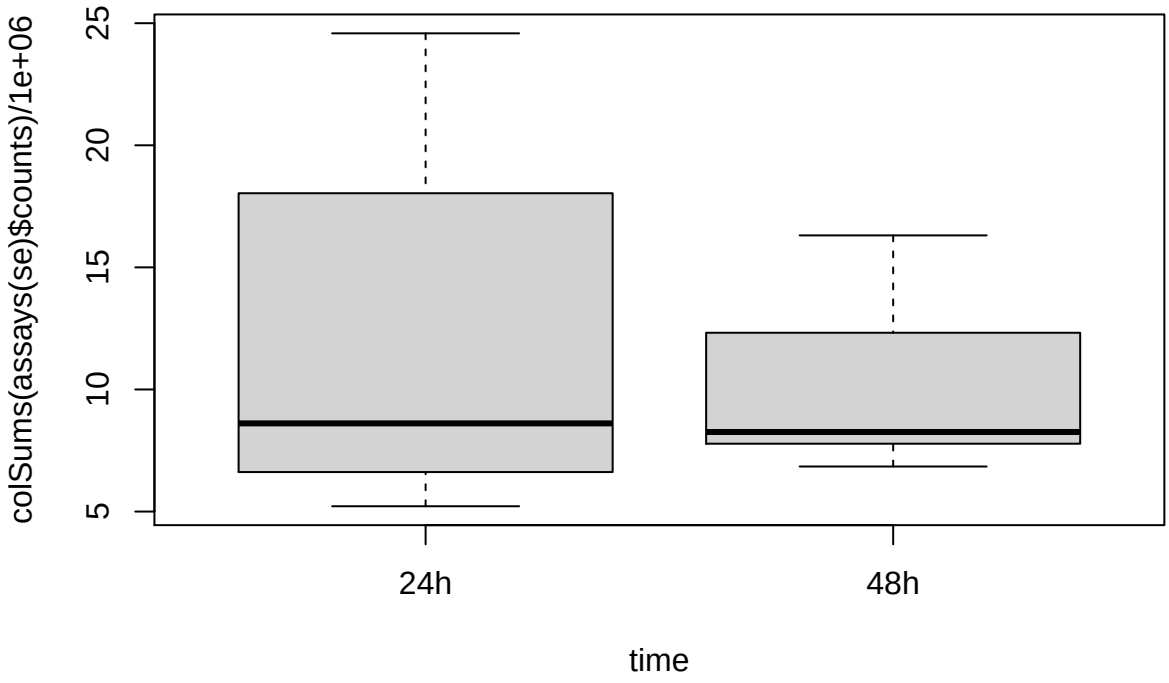
```
suppressPackageStartupMessages({
  library(limma)
  library(edgeR)
})

# library size distribution
hist(colSums(assays(se)$counts)/1e6, breaks=10)
```



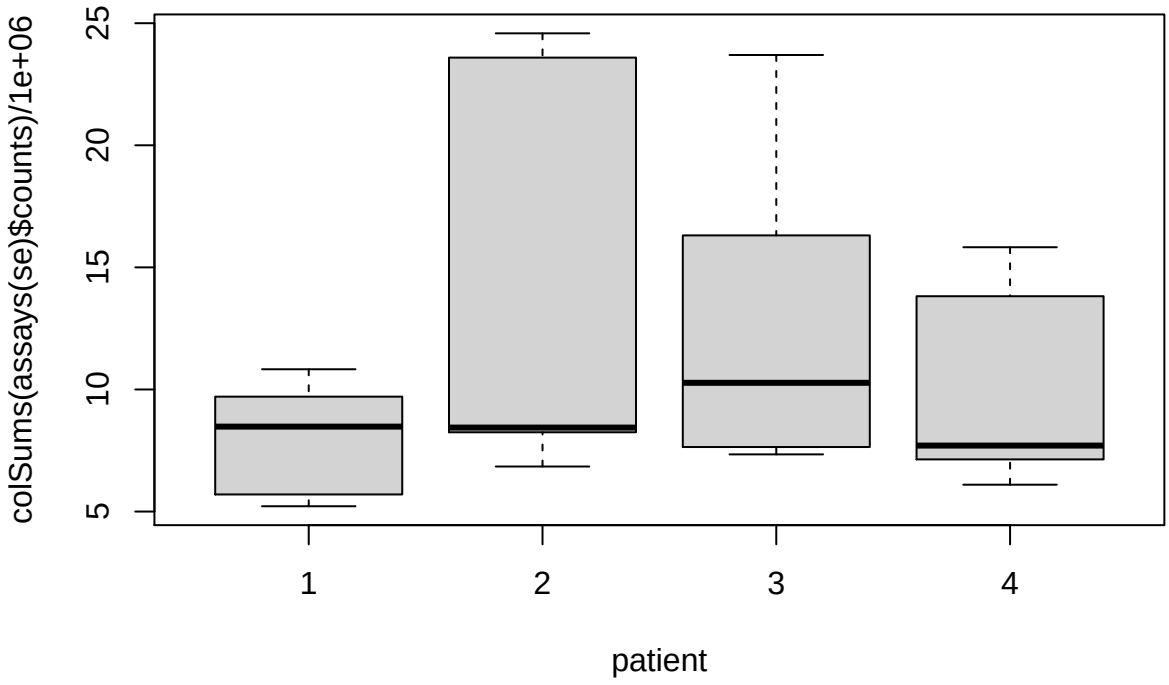

```
boxplot(colSums(assays(se)$counts)/1e6 ~ treatment)
```



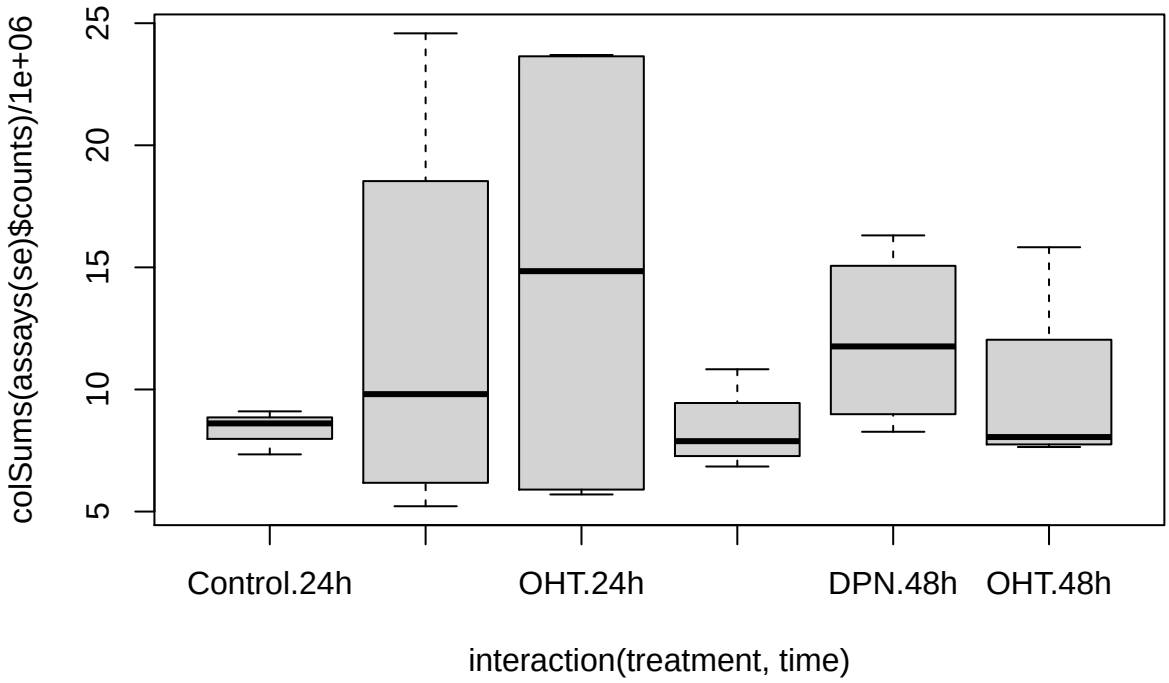

```
boxplot(colSums(assays(se)$counts)/1e6 ~ time)
```



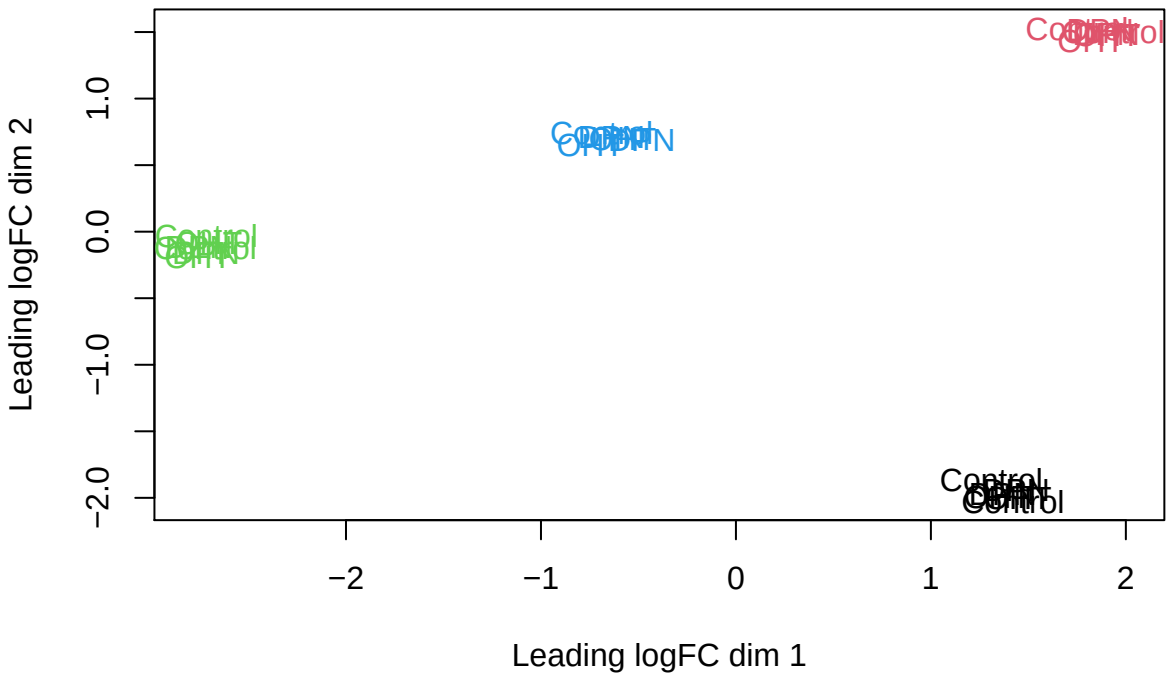

```
boxplot(colSums(assays(se)$counts)/1e6 ~ patient)
```




```
boxplot(colSums(assays(se)$counts)/1e6 ~ interaction(treatment, time))
```



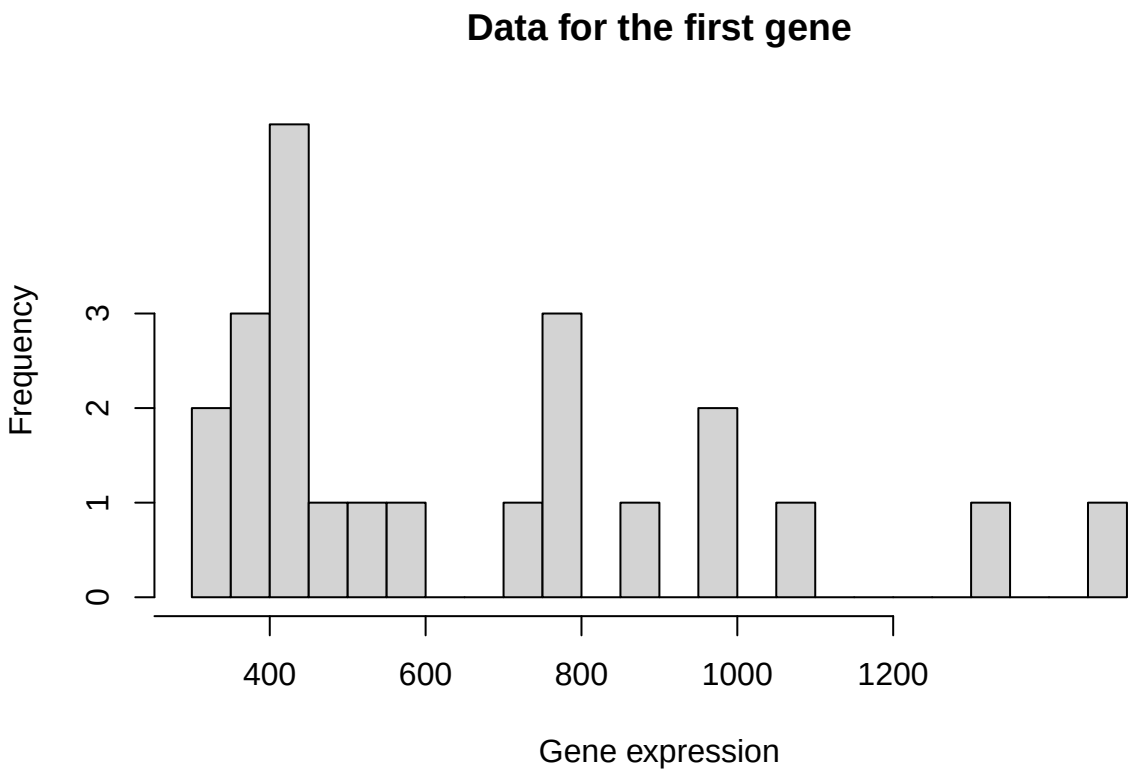

```
# MDS plot
plotMDS(se,
        labels = treatment,
        col=as.numeric(patient))
```



2 Challenge I: Choice of modeling assumptions

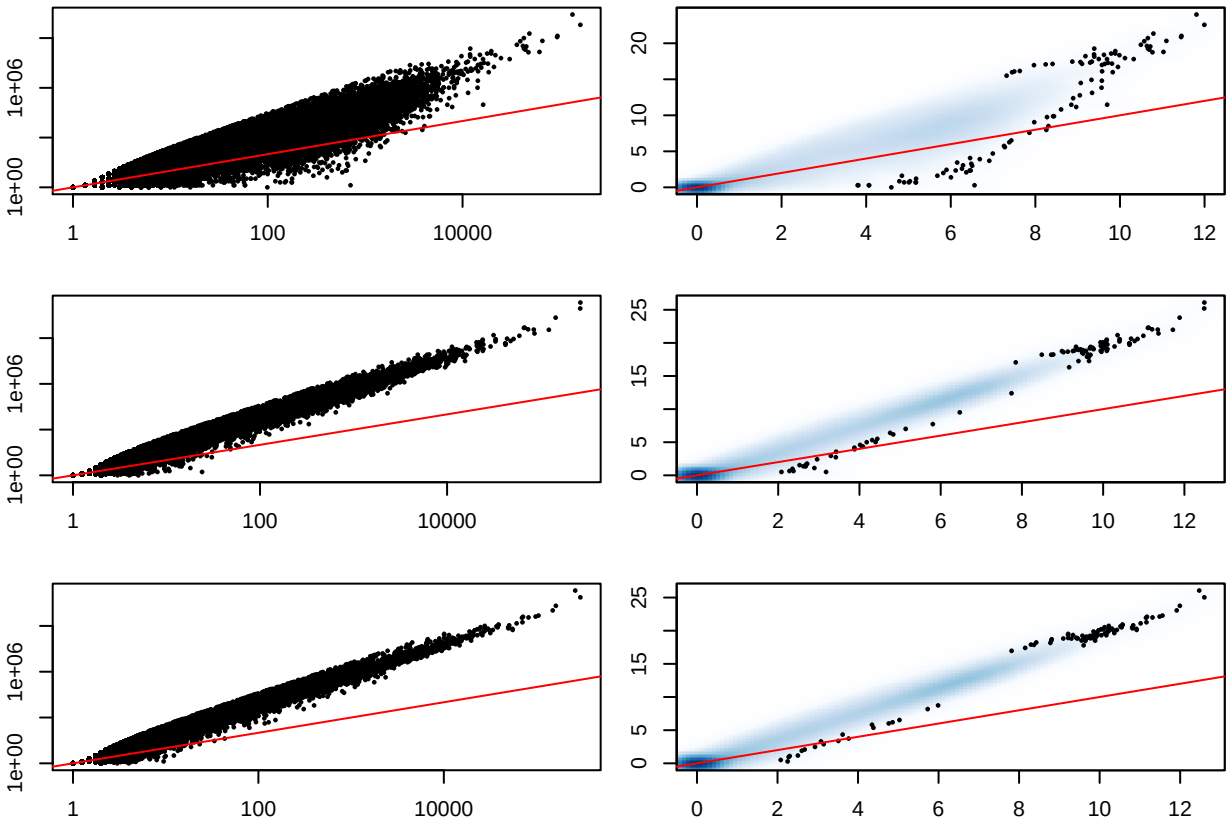
When working with a GLM, as part of the choices of modeling assumptions, we need to pick an appropriate distribution for the expression counts. Below we perform some exploratory analyses to investigate.

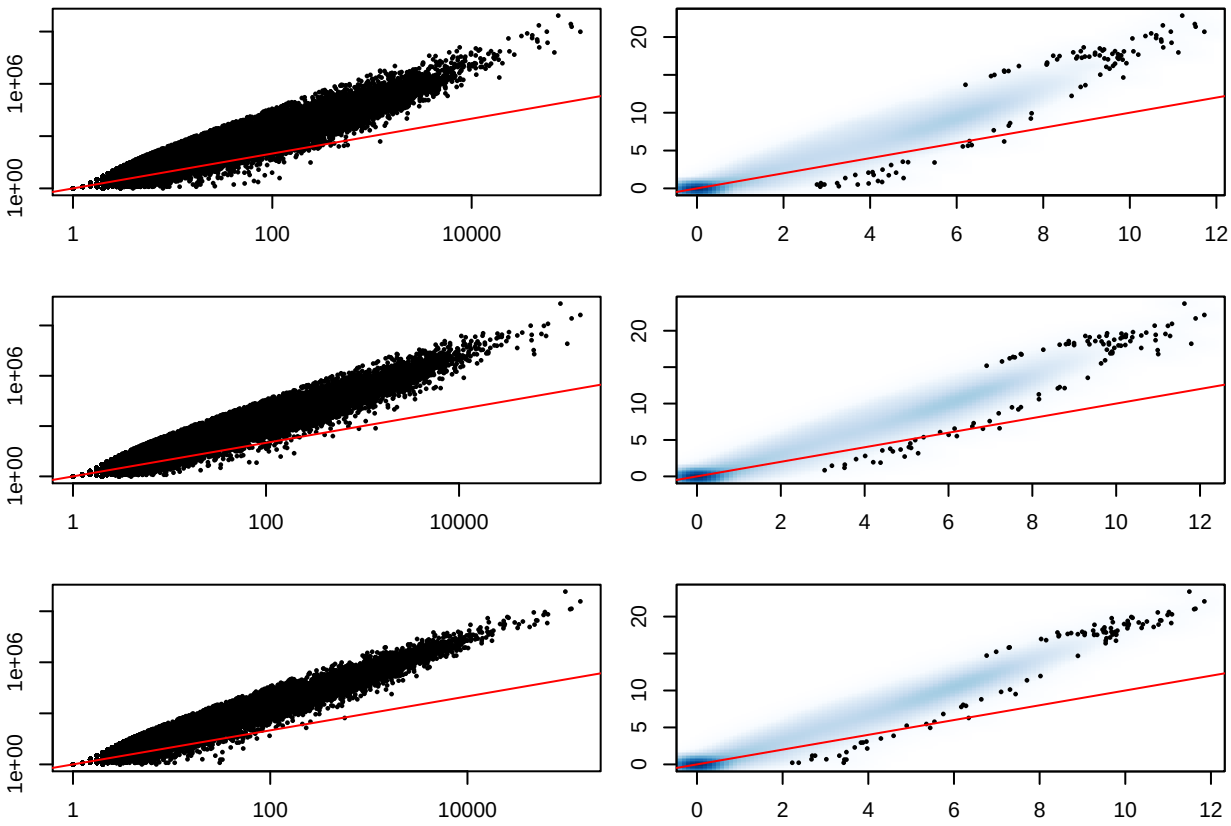
```
y <- assays(se)$counts[1,]
hist(y, breaks = 40,
     xlab = "Gene expression",
     xaxt = "n", yaxt = "n",
     main = "Data for the first gene")
axis(1, at = seq(200, 1200, by=200))
axis(2, at = 0:3)
```




```
# Mean-variance trend within each experimental condition
cont24ID <- which(treatment == "Control" & time == "24h")
DPN24ID <- which(treatment == "DPN" & time == "24h")
OHT24ID <- which(treatment == "OHT" & time == "24h")
cont48ID <- which(treatment == "Control" & time == "48h")
DPN48ID <- which(treatment == "DPN" & time == "48h")
OHT48ID <- which(treatment == "OHT" & time == "48h")
idList <- list(cont24ID, DPN24ID, OHT24ID,
               cont48ID, DPN48ID, OHT48ID)
names(idList) <- paste0(rep(levels(treatment),2), rep(levels(time), each=3))

par(mfrow=c(3,2), mar=c(2,2,2,1))
for(kk in 1:length(idList)){
  # extract counts for each condition
  curCounts <- assays(se)$counts[,idList[[kk]]]
  plot(x = rowMeans(curCounts)+1,
       y = rowVars(curCounts)+1,
       pch = 16, cex=1/2,
       xlab = "Mean", ylab="Variance",
       log="xy")
  abline(0,1, col="red")

  smoothScatter(x = log1p(rowMeans(curCounts)),
       y = log1p(rowVars(curCounts)),
       pch = 16, cex=1/2,
       xlab = "Mean", ylab="Variance")
```

```
  abline(0,1, col="red")
}
```





- Having data on thousands of genes provides the opportunity to empirically assess the mean-variance relationship.
- It is clear that the data is overdispersed with respect to the Poisson distribution. There also seems to be a quadratic trend of the variance as a function of the mean. This has motivated the **negative binomial distribution as the most popular choice to model (bulk) RNA-seq gene expression data**.

- The negative binomial distribution is also referred to as the Gamma-Poisson distribution as it can be formulated as such. Indeed, if

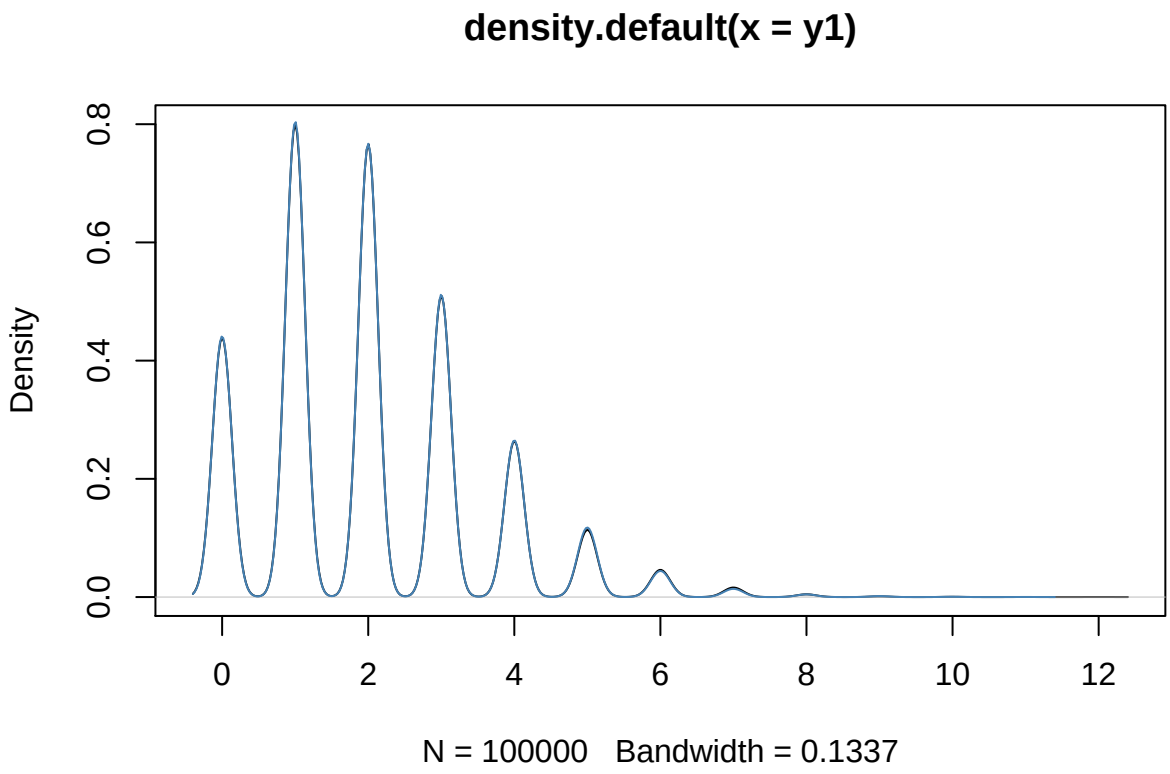
$$\lambda \sim \Gamma(\alpha, \beta) Y|\lambda \sim Poi(\lambda),$$

  then this is equivalent to

$$Y \sim NB(\mu = \alpha/\beta, \phi = 1/\alpha).$$

- This can be shown analytically, but is considered out of the scope of this course. Below, we show it empirically using simulation.
- This theoretical result has got some practical consequences. The Gamma-Poisson formulation makes it clear why we can sum technical replicates as the sum of Poisson random variables is again a Poisson random variable.
- The Poisson statement can thus be considered as capturing technical variation, while the Gamma statement can be considered to capture biological variation, i.e., variation in the expression mean across biological replicates.

```r
alpha <- 20
beta <- 10
lambda <- rgamma(n = 1e5, shape = alpha, rate = beta)
y1 <- rpois(n = 1e5, lambda = lambda)

# note phi = 1 / size
y2 <- rnbinom(n=1e5, mu=alpha / (beta), size=alpha)


plot(density(y1))
lines(density(y2), col="steelblue")
```



3 Challenge II: Normalization

Normalization is necessary to correct for several sources of technical variation:

- **Differences in sequencing depth** between samples. Some samples get sequenced deeper in the sense that they consist of more (mapped) reads and therefore can be considered to contain a higher amount of information, which we should be taking into account. In addition, if a sample is sequenced deeper, it is natural that the counts for each gene will be higher, jeopardizing a direct comparison of the expression counts.
- **Differences in RNA population composition** between samples. Suppose that two samples have been sequenced to the exact same depth. One sample is contaminated and has a very high concentration

of the contaminant cDNA being sequenced, but otherwise the two samples are identical. Since the contaminant will be taking up a significant proportion of the reads being sequenced, the counts will not be directly comparable between the samples. Hence, we may also want to correct for differences in the composition of the RNA population of the samples.

- **Other technical variation** such as sample-specific GC-content or transcript length effects may also be accounted for.

Let's take a look at how comparable different replicates are in the Control condition at 48h in our dataset. We will investigate this using MD-plots (mean-difference plots as introduced by Dudoit *et al.* (2002)), also sometimes referred to as MA-plots.

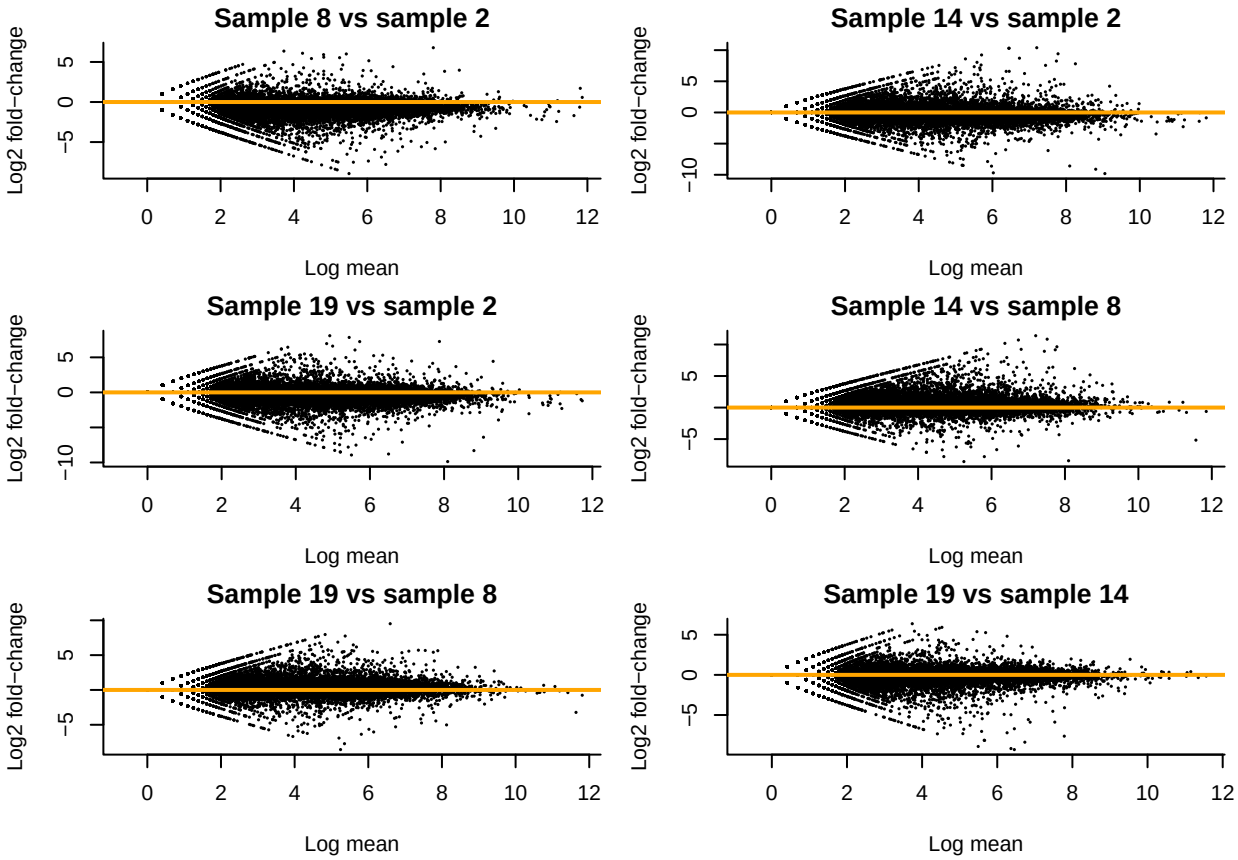
```
cont48ID # relevant samples
```

```
## [1]  2  8 14 19
```

```
colSums(assays(se)$counts[,cont48ID]) / 1e6
```

```
## [1] 10.827109  6.844144  8.064268  7.701432
```

```
combs <- combn(cont48ID, m=2) #pairwise combinations between samples

par(mfrow=c(3,2), mar=c(4,4,2,1))
for(cc in 1:ncol(combs)){
  curSamples <- combs[,cc]
  M <- rowMeans(assays(se)$counts[,curSamples])
  D <- assays(se)$counts[,curSamples[2]] / assays(se)$counts[,curSamples[1]]
  plot(x = log(M), y = log2(D),
       pch = 16, cex=1/3,
       main = paste0("Sample ", curSamples[2], " vs sample ", curSamples[1]),
       xlab = "Log mean", ylab = "Log2 fold-change",
       bty = 'l')
  abline(h = 0, col="orange", lwd=2)
}
```



- We see clear bias for some pairwise comparisons. For example, in the first plot comparing sample 8 versus sample 2, the log fold-changes are biased downwards. This means that, on average, a gene is lower expressed in sample 8 versus sample 2. Looking at the library sizes, we can indeed see that the library size for sample 2 is about 11×10^6 while it is only about 7×10^6 for sample 8! This is clear library size effect that we should take into account.

3.1 Count scaling versus GLM offsets

- We have previously discussed count scaling transformations such as CPM and TPM as normalization methods that directly scale counts. However, these transformations ignore inherent information such as the sequencing depth. A measured gene expression of zero in a sample where only 1000 reads are sequenced is very different from a sample where 10 million reads are sequenced.
- A more appropriate and natural way when working with GLMs is through the use of **offsets**. The general use of an offset is to account for the 'effort' performed in order to gather that observation of the response variable. Two examples:

    1. A biologist studying whale migration has one fixed spot where, in the migration season, she counts migrating whales day after day, over several years. For each day she records the number of spotted whales. Of course, the time spent whale-watching may differ from day to day and it is natural that you are more likely to spot more whales if you spend more time looking for them. The time spent spotting whales can then be used as an offset.
    2. In our case, a sample being sequenced deeper contains more information, i.e., more 'effort' has been performed, as compared to a sample being sequenced relatively shallow. We have more confidence of a count from a deeply sequenced sample than from a shallowly sequenced sample. We can therefore use the sequencing depth $N_i = \sum_g Y_{gi}$ as off set in the model.

- Adding an offset to the model is different from adding a new variable to the model. For each new variable we add, we will estimate its average effect β on the response variable. When adding an offset, however, we are implicitly assuming that $\beta = 1$.
- Offsets are typically added on the scale of the linear predictor. Suppose we have a gene g and sample i specific offset O_{ij}, then we can define a negative binomial GLM including the offset as

$$\left\{ \begin{array}{ccc} Y_{gi} & \sim & NB(\mu_{gi}, \phi_g) \\ \log \mu_{gi} & = & \eta_{gi} \\ \eta_{gi} & = & \mathbf{X}_i^T \beta_g + \log(O_{gi}). \end{array} \right.$$

3.2 How to normalize?

Many approaches are available for normalizing RNA-seq data. Most methods basically calculate an offset that is added to the GLM used to model gene expression. One notable method, full-quantile normalization, does not calculate an offset.

3.2.1 TMM method (default of `edgeR`)

The trimmed mean of M-values (TMM) method introduced by Robinson & Oshlack (2010) is a global-scaling normalization procedure. As the name suggests, it is based on a trimmed mean of fold-changes (M-values) as the scaling factor. A trimmed mean is an average after removing a set of "extreme" values. Specifically, TMM calculates a normalization factor $F_i^{(r)}$ across genes g for each sample i as compared to a reference sample r,

$$\log_2(F_i^{(r)}) = \frac{\sum_{g \in G^*} w_{gi}^r M_{gi}^r}{\sum_{g \in G^*} w_{gi}^r},$$

where M_{gi}^r represents the $\log_2$-fold-change of the gene expression fraction as compared to a reference sample r, i.e.,

$$M_{gi}^r = \log_2 \left(\frac{Y_{gi}/N_i}{Y_{gr}/N_r} \right),$$

w_{gi}^r represents a weight calculated as

$$w_{gi}^r = \frac{N_i - Y_{gi}}{N_i Y_{gi}} + \frac{N_r - Y_{gr}}{N_r Y_{gr}},$$

and G^* represents the set of peaks after trimming those with the most extreme values.

The procedure only takes peaks into account where both $Y_{gi} > 0$ and $Y_{gr} > 0$. By default, TMM trims peaks with the 30% most extreme M-values and 5% most extreme average gene expression, and chooses as reference r the sample whose upper-quartile is closest to the across-sample average upper-quartile. The normalized counts are then given by $\tilde{Y}_{gi} = Y_{gi}/N_i^s$, where

$$N_i^s = \frac{N_i F_i^{(r)}}{\sum_i N_i F_i^{(r)}/n}.$$

TMM normalization may be performed from the `calcNormFactors` function implemented in `edgeR`:

```
dge <- edgeR::calcNormFactors(se)
dge$samples #normalization factors added to colData
```

```
##          group lib.size norm.factors       run experiment patient treatment
## Sample1      1  9102683    0.9782830 SRR479052  SRX140503       1   Control
## Sample2      1 10827109    0.9728700 SRR479053  SRX140504       1   Control
## Sample3      1  5217761    0.9898593 SRR479054  SRX140505       1       DPN
## Sample4      1  9706035    0.9930169 SRR479055  SRX140506       1       DPN
## Sample5      1  5700022    0.9850867 SRR479056  SRX140507       1       OHT
## Sample6      1  7854568    0.9897270 SRR479057  SRX140508       1       OHT
## Sample7      1  8610014    0.9266581 SRR479058  SRX140509       2   Control
## Sample8      1  6844144    0.9544240 SRR479059  SRX140510       2   Control
## Sample9      1 24584280    0.9188545 SRR479060  SRX140511       2       DPN
## Sample10     1  8267977    0.9398000 SRR479062  SRX140512       2       DPN
## Sample11     1 23590411    0.9096695 SRR479063  SRX140513       2       OHT
## Sample12     1  8247122    0.9369050 SRR479065  SRX140514       2       OHT
## Sample13     1  7341000    1.0668032 SRR479066  SRX140515       3   Control
## Sample14     1  8064268    1.0552688 SRR479067  SRX140516       3   Control
## Sample15     1 12481958    1.0461698 SRR479068  SRX140517       3       DPN
## Sample16     1 16310090    1.0260056 SRR479069  SRX140518       3       DPN
## Sample17     1 23697329    1.0268459 SRR479070  SRX140519       3       OHT
## Sample18     1  7642648    1.0409451 SRR479071  SRX140520       3       OHT
## Sample19     1  7701432    1.0559132 SRR479072  SRX140521       4   Control
## Sample20     1  7135899    1.0675040 SRR479073  SRX140522       4       DPN
## Sample21     1 13818393    1.0327004 SRR479074  SRX140523       4       DPN
## Sample22     1  6099942    1.0890994 SRR479076  SRX140524       4       OHT
## Sample23     1 15825211    1.0286470 SRR479077  SRX140525       4       OHT
##          time submission     study    sample
## Sample1   24h  SRA051611 SRP012167 SRS308865
## Sample2   48h  SRA051611 SRP012167 SRS308866
## Sample3   24h  SRA051611 SRP012167 SRS308867
## Sample4   48h  SRA051611 SRP012167 SRS308868
## Sample5   24h  SRA051611 SRP012167 SRS308869
## Sample6   48h  SRA051611 SRP012167 SRS308870
## Sample7   24h  SRA051611 SRP012167 SRS308871
## Sample8   48h  SRA051611 SRP012167 SRS308872
## Sample9   24h  SRA051611 SRP012167 SRS308873
## Sample10  48h  SRA051611 SRP012167 SRS308874
## Sample11  24h  SRA051611 SRP012167 SRS308875
## Sample12  48h  SRA051611 SRP012167 SRS308876
## Sample13  24h  SRA051611 SRP012167 SRS308877
## Sample14  48h  SRA051611 SRP012167 SRS308878
## Sample15  24h  SRA051611 SRP012167 SRS308879
## Sample16  48h  SRA051611 SRP012167 SRS308880
## Sample17  24h  SRA051611 SRP012167 SRS308881
## Sample18  48h  SRA051611 SRP012167 SRS308882
## Sample19  48h  SRA051611 SRP012167 SRS308883
## Sample20  24h  SRA051611 SRP012167 SRS308884
## Sample21  48h  SRA051611 SRP012167 SRS308885
## Sample22  24h  SRA051611 SRP012167 SRS308886
## Sample23  48h  SRA051611 SRP012167 SRS308887
```

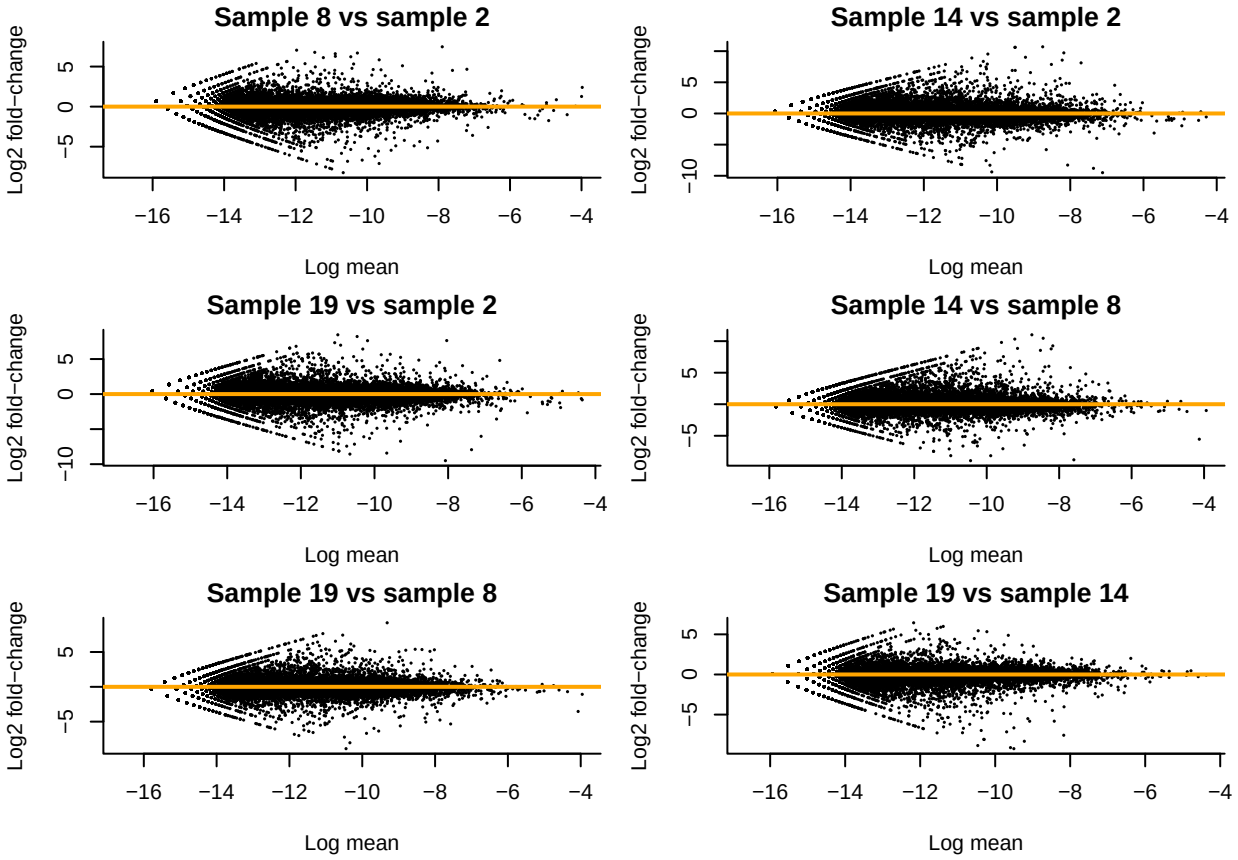
Let's check how our MD-plots look like after normalization. Note that, we can rewrite the GLM as

$$\log\left(\frac{\mu_{gi}}{O_{gi}}\right) = \mathbf{X}_i^T \beta_g$$

and so $\frac{\mu_{gi}}{O_{gi}}$ can be considered as an 'offset-corrected count'.

We see that all MD-plots are now nicely centered around a log-fold-change of zero!

```
## normalize
effLibSize <- dge$samples$lib.size * dge$samples$norm.factors
normCountTMM <- sweep(assays(se)$counts, 2, FUN="/", effLibSize)


par(mfrow=c(3,2), mar=c(4,4,2,1))
for(cc in 1:ncol(combs)){
  curSamples <- combs[,cc]
  M <- rowMeans(normCountTMM[,curSamples])
  D <- normCountTMM[,curSamples[2]] / normCountTMM[,curSamples[1]]
  plot(x = log(M), y = log2(D),
       pch = 16, cex=1/3,
       main = paste0("Sample ", curSamples[2], " vs sample ", curSamples[1]),
       xlab = "Log mean", ylab = "Log2 fold-change",
       bty = 'l')
  abline(h = 0, col="orange", lwd=2)
}
```



3.3 Median-of-ratios method (default of DESeq2)

The median-of-ratios method is used in DESeq2 as described in Love *et al.* (2014). It assumes that the expected value $\mu_{gi} = E(Y_{gi})$ is proportional to the true expression of the gene, q_{gi}, scaled by a normalization

factor s_i for each sample,

$$\mu_{gi} = s_i q_{gi}.$$

The normalization factor s_i is then estimated using the median-of-ratios method compared to a synthetic reference sample r defined based on geometric means of counts across samples

$$s_i = \text{median}_{\{g:Y^*_{gr} \neq 0\}} \frac{Y_{gi}}{Y^*_{gr}},$$

with

$$Y^*_{gr} = \left(\prod_{i=1}^{n} Y_{gi} \right)^{1/n}.$$

From this, we calculate the normalized count as $\tilde{Y}_{gi} = Y_{gi}/s_i$.

Median-of-ratios normalization is implemented in the `DESeq2` package:

```
dds <- DESeq2::DESeqDataSetFromMatrix(countData = assays(se)$counts,
                                      colData = colData(se),
                                      design = ~ 1) #just add intercept to showcase normalization
```

```
## converting counts to integer mode
```

```
dds <- DESeq2::estimateSizeFactors(dds)
sizeFactors(dds)
```

```
##  [1] 0.9187624 1.0644717 0.5232069 0.9826698 0.5676212 0.7807662 0.8284089
##  [8] 0.6600848 2.3955069 0.7906917 2.2784591 0.7860386 0.7807842 0.8445577
## [15] 1.3345483 1.7011973 2.5001652 0.7889061 0.8138720 0.7632306 1.4518320
## [22] 0.6553677 1.6696064
```

3.4 Conditional quantile normalization (`cqn`)

The `cqn` method, introduced by Hansen *et al.* (2012) and implemented in the `cqn` `R` package, starts by assuming a Poisson model for the gene expression counts Y_{gi}. Median regression is used to model, for each sample, the log-transformed accessibility count as a smooth function of GC-content as well as gene length, focusing on genes with high average count (above 50 by default). Next, subset quantile normalization (Wu *et al.* (2010)) is performed on the residuals of that model (i.e., on the counts adjusted for GC-content) for between-sample normalization. The method could intuitively be thought of as full-quantile normalization after removing a smoothed sample-specific GC-content effect. Normalized counts are calculated as recommended in the `cqn` vignette, i.e.,

$$\tilde{Y}_{gi} = \left(\frac{(Y_{gi} + 1)10^6}{\sum_g Y_{gi}} \right) 2^{O_{gi}},$$

with O_{gi} the normalization offset estimated by `cqn`, which is on the $\log_2$ scale. Note that `cqn` normalization works with a gene- and sample-specific offset, unlike TMM and median-of-ratios normalization which only work with a sample-specific offset.

3.5 Full quantile (FQ) normalization

In full-quantile normalization, originally introduced in the context of microarrays by Bolstad *et al.* (2003), the samples are forced to each have a distribution identical to the distribution of the median/average of the quantiles across samples. In practice, we implement full-quantile normalization using the following procedure

1. Given a data matrix $\mathbf{Y}_{G\times n}$ for G genes (rows) and n samples (columns),
2. sort each column to get $\mathbf{Y}^S$,
3. replace all elements of each row by the median (or average) for that row,
4. obtain the normalized counts $\tilde{\mathbf{Y}}$ by re-arranging (i.e., unsorting) each column.

4 Challenge III: Parameter estimation (under limited information setting)

Empirical Bayes, but also the factors we should incorporate into the mean model: show screenshot of paper that doesn't include interaction effect. Discuss with students.

5 Challenge IV: Statistical inference across many genes

6 The importance of reproducible analysis

https://www.youtube.com/watch?v=8QJfNS7XXwA