Read counts to DGE, Part I

Friederike Duendar, ABC, WCM

02/26/2019

Contents

`featureCounts` results

We aligned five samples for the WT and SNF2 condition, respectively.

- *How can you check which command was used to generate those `BAM` files?*

Let's read the result file into R (you'll have to download it to your laptop first).

Loading additional libraries:

```
library(ggplot2) # for making plots
library(magrittr) # for "pipe"-like coding in R
```

First, make sure you set the path to your working directory which should contain the count table.

```
folder <- "~/Documents/Teaching/ANGSD/RNA/" # download count table!
folder <- "./"
setwd(folder)
## reading in featureCounts output
readcounts <- read.table(paste0(folder, "featCounts_Gierlinski_genes.txt"),
                         header=TRUE)
```

`DESeq2` setup

We will use the `DESeq2` package to normalize the samples for differences in their sequencing depth and to explore them.

```
## not available via install.packages(), but through bioconductor
source("http://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```

```
library(DESeq2)
```

We will have to generate a `DESeqDataSet`, which is a specific R class that combines `data.frames` and one or more `matrices` into one object. The `data.frames` typically contain metadata about the samples and genes (e.g. gene IDs, sample conditions), while the matrices contain the expression values.

- *Find out via `?DESeqDataSetFromMatrix` how to generate a `DESeqDataSet`!*

We need two tables: `countData` and `colData`.

- `colData`: `data.frame` with all the variables you know about your samples, e.g., experimental condition, the type, and date of sequencing and so on. Its `row.names` should correspond to the unique sample names.
- `countData`: should contain a `matrix` of the actual values associated with the genes and samples. Conveniently, this is almost exactly the format of the `featureCounts` output.

`countData`

```
head(readcounts)
```

```
##      Geneid  Chr  Start    End Strand Length
## 1   YAL012W chrI 130799 131983      +   1185
## 2   YAL069W chrI    335    649      +    315
## 3 YAL068W-A chrI    538    792      +    255
## 4   YAL068C chrI   1807   2169      -    363
## 5 YAL067W-A chrI   2480   2707      +    228
## 6   YAL067C chrI   7235   9016      -   1782
##   ...alignment.SNF2_1_Aligned.sortedByCoord.out.bam
## 1                                              7351
## 2                                                 0
## 3                                                 0
## 4                                                 2
## 5                                                 0
## 6                                               103
##   ...alignment.SNF2_2_Aligned.sortedByCoord.out.bam
## 1                                              7180
## 2                                                 0
## 3                                                 0
## 4                                                 2
## 5                                                 0
## 6                                                51
##   ...alignment.SNF2_3_Aligned.sortedByCoord.out.bam
## 1                                              7648
## 2                                                 0
## 3                                                 0
## 4                                                 2
## 5                                                 0
## 6                                                44
##   ...alignment.SNF2_4_Aligned.sortedByCoord.out.bam
## 1                                              8119
## 2                                                 0
## 3                                                 0
## 4                                                 1
## 5                                                 0
## 6                                                90
##   ...alignment.SNF2_5_Aligned.sortedByCoord.out.bam
## 1                                              5944
```

```
## 2                                                    0
## 3                                                    0
## 4                                                    0
## 5                                                    0
## 6                                                   53
##   ...alignment.WT_1_Aligned.sortedByCoord.out.bam
## 1                                                 4312
## 2                                                    0
## 3                                                    0
## 4                                                    0
## 5                                                    0
## 6                                                   12
##   ...alignment.WT_2_Aligned.sortedByCoord.out.bam
## 1                                                 3767
## 2                                                    0
## 3                                                    0
## 4                                                    0
## 5                                                    0
## 6                                                   23
##   ...alignment.WT_3_Aligned.sortedByCoord.out.bam
## 1                                                 3040
## 2                                                    0
## 3                                                    0
## 4                                                    0
## 5                                                    0
## 6                                                   21
##   ...alignment.WT_4_Aligned.sortedByCoord.out.bam
## 1                                                 5604
## 2                                                    0
## 3                                                    0
## 4                                                    2
## 5                                                    0
## 6                                                   30
##   ...alignment.WT_5_Aligned.sortedByCoord.out.bam
## 1                                                 4167
## 2                                                    0
## 3                                                    0
## 4                                                    2
## 5                                                    0
## 6                                                   29
```

In principle, this is the format that we'll need (columns = Samples, rows = genes), but particularly the sample names are a bit unwielding and we're completely missing `row.names`. In addition, the first couple of columns contain meta data information that need to be separated from the counts (e.g. gene IDs, gene lengths etc.).

```
## gene IDs should be stored as row.names
row.names(readcounts) <- make.names(readcounts$Geneid)

## exclude the columns without read counts (columns 1 to 6 contain additional
## info such as genomic coordinates)
readcounts <- readcounts[ , -c(1:6)]

## give meaningful sample names - there are many ways to achieve this
orig_names <- names(readcounts) # keep a back-up copy of the original names
```

```
names(readcounts) <- c("SNF2_1", "SNF2_2", "SNF2_3", "SNF2_4", "SNF2_5",
                       "WT_1", "WT_2", "WT_3", "WT_4", "WT_5" ) # most error-prone way!

### alternatives:
names(readcounts) <- c( paste("SNF2", c(1:5), sep = "_"),
                       paste("WT", c(1:5), sep = "_") ) # less potential for typos
### even safer
names(readcounts) <- gsub(".*(WT|SNF2)(_[0-9]+).*", "\\1\\2", orig_names)
```

Always check your data set after you manipulated it!

```
str(readcounts)
```

```
## 'data.frame':    6692 obs. of  10 variables:
##  $ SNF2_1: int  7351 0 0 2 0 103 2 5 13 46 ...
##  $ SNF2_2: int  7180 0 0 2 0 51 0 9 8 58 ...
##  $ SNF2_3: int  7648 0 0 2 0 44 0 6 10 45 ...
##  $ SNF2_4: int  8119 0 0 1 0 90 0 3 9 61 ...
##  $ SNF2_5: int  5944 0 0 0 0 53 0 1 6 40 ...
##  $ WT_1  : int  4312 0 0 0 0 12 0 10 9 33 ...
##  $ WT_2  : int  3767 0 0 0 0 23 0 5 12 41 ...
##  $ WT_3  : int  3040 0 0 0 0 21 0 2 4 31 ...
##  $ WT_4  : int  5604 0 0 2 0 30 0 4 4 45 ...
##  $ WT_5  : int  4167 0 0 2 0 29 0 3 8 25 ...
```

```
head(readcounts)
```

```
##           SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
## YAL012W     7351   7180   7648   8119   5944 4312 3767 3040 5604 4167
## YAL069W        0      0      0      0      0    0    0    0    0    0
## YAL068W.A      0      0      0      0      0    0    0    0    0    0
## YAL068C        2      2      2      1      0    0    0    0    2    2
## YAL067W.A      0      0      0      0      0    0    0    0    0    0
## YAL067C      103     51     44     90     53   12   23   21   30   29
```

This would be the data that we will store in the `counts` (or `assay`) slot of the `DESeq2` object. Now, we turn to the `colData`.

`colData`

According to `?colData`, this should be a `data.frame`, where the *rows* directly match the *columns* of the count data.

Here's how this could be generated in `R` matching the `readcounts` `data.frame` we already have:

```
sample_info <- DataFrame(condition = gsub("_[0-9]+", "", names(readcounts)),
                         row.names = names(readcounts) )
sample_info
```

```
## DataFrame with 10 rows and 1 column
##          condition
##        <character>
## SNF2_1        SNF2
## SNF2_2        SNF2
## SNF2_3        SNF2
## SNF2_4        SNF2
```

```
## SNF2_5        SNF2
## WT_1            WT
## WT_2            WT
## WT_3            WT
## WT_4            WT
## WT_5            WT
```

```
str(sample_info)
```

```
## Formal class 'DataFrame' [package "S4Vectors"] with 6 slots
##   ..@ rownames       : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
##   ..@ nrows          : int 10
##   ..@ listData       :List of 1
##   .. ..$ condition: chr [1:10] "SNF2" "SNF2" "SNF2" "SNF2" ...
##   ..@ elementType    : chr "ANY"
##   ..@ elementMetadata: NULL
##   ..@ metadata       : list()
```

Generate the DESeqDataSet

```
DESeq.ds <- DESeqDataSetFromMatrix(countData = readcounts,
                                   colData = sample_info,
                                   design = ~ condition)
DESeq.ds
```

```
## class: DESeqDataSet
## dim: 6692 10
## metadata(1): version
## assays(1): counts
## rownames(6692): YAL012W YAL069W ... YMR325W YMR326C
## rowData names(0):
## colnames(10): SNF2_1 SNF2_2 ... WT_4 WT_5
## colData names(1): condition
```

```
head(counts(DESeq.ds))
```

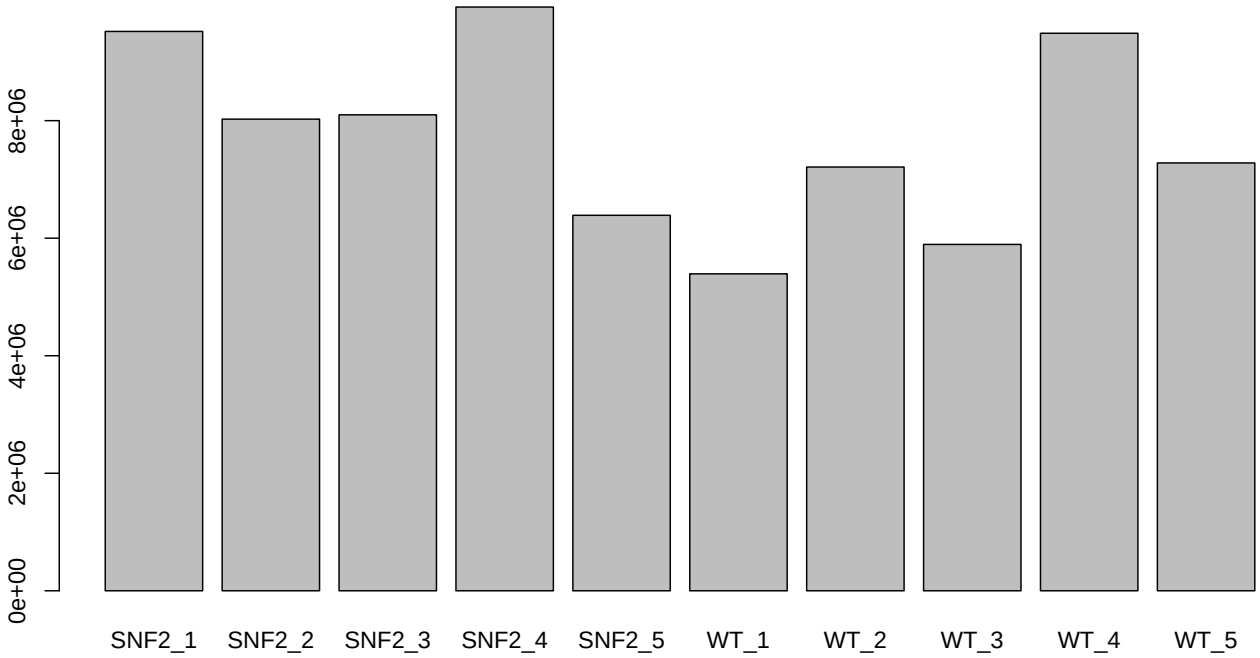
```
##           SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
## YAL012W     7351   7180   7648   8119   5944 4312 3767 3040 5604 4167
## YAL069W        0      0      0      0      0    0    0    0    0    0
## YAL068W.A      0      0      0      0      0    0    0    0    0    0
## YAL068C        2      2      2      1      0    0    0    0    2    2
## YAL067W.A      0      0      0      0      0    0    0    0    0    0
## YAL067C      103     51     44     90     53   12   23   21   30   29
```

How many reads were counted for each sample (= library sizes)?

```
colSums(counts(DESeq.ds))
```

```
##  SNF2_1  SNF2_2  SNF2_3  SNF2_4  SNF2_5    WT_1    WT_2    WT_3    WT_4
## 9518261 8025575 8099295 9933479 6389328 5393487 7211200 5894001 9487091
##    WT_5
## 7280514
```

```
colSums(counts(DESeq.ds)) %>% barplot
```



Remove genes with no reads.

```
keep_genes <- rowSums(counts(DESeq.ds)) > 0
dim(DESeq.ds)
```

```
## [1] 6692   10
```

```
DESeq.ds <- DESeq.ds[ keep_genes, ]
dim(DESeq.ds)
```

```
## [1] 6394   10
```

```
counts(DESeq.ds) %>% str
```

```
##  int [1:6394, 1:10] 7351 2 103 2 5 13 46 17 20 249 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:6394] "YAL012W" "YAL068C" "YAL067C" "YAL066W" ...
##   ..$ : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
```

```
assay(DESeq.ds) %>% str
```

```
##  int [1:6394, 1:10] 7351 2 103 2 5 13 46 17 20 249 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:6394] "YAL012W" "YAL068C" "YAL067C" "YAL066W" ...
##   ..$ : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
```

Normalizing for sequencing depth and RNA composition differences

Now that we have the data, we can start using `DESeq2`'s functions, e.g. `estimateSizeFactors()` for sequencing depth normalization.

The size factor is calculated as follows:

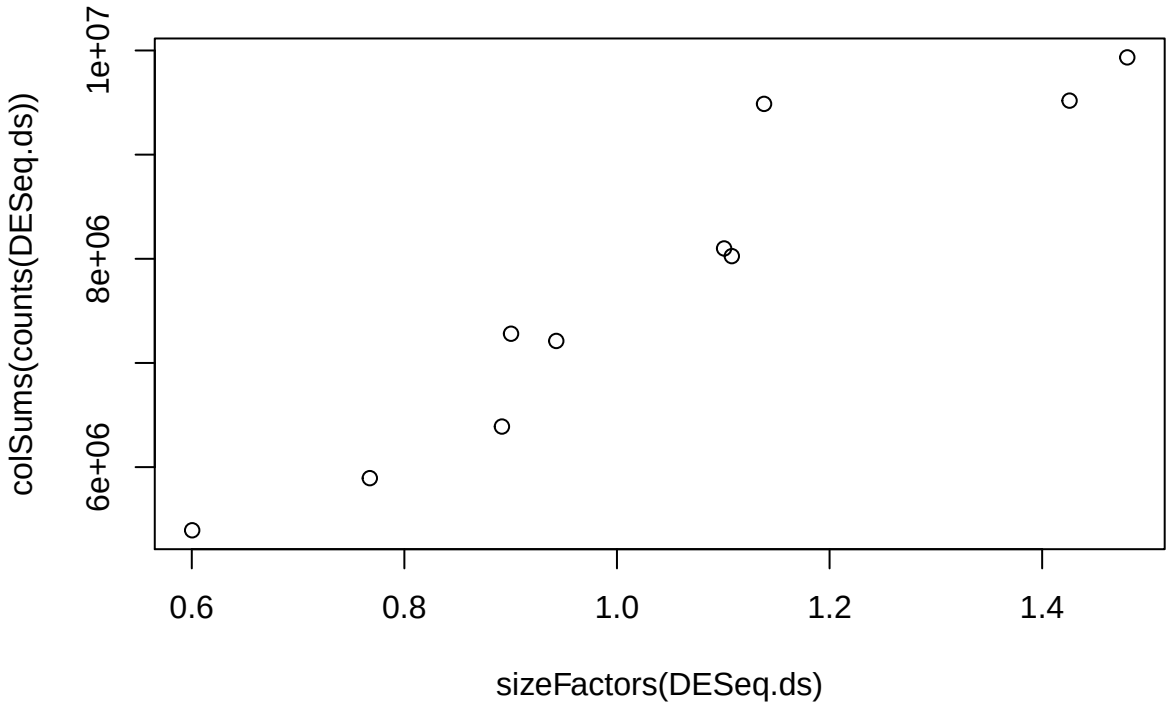
1. For every gene, the geometric mean of counts is calculated across all samples (= "pseudo baseline expression").
2. For every gene, the ratio of its counts within a specific sample to the pseudo-baseline is calculated (e.g., `Sample A/pseudo baseline`, `Sample B/pseudo baseline`).
3. For every *sample* (columns!), the median of the ratios from step 2 is calculated. This is the size factor.

- There is the assumption that some genes are not changing across conditions!
- Size factors should be around 1.
- Normalized counts are calculated via $counts_{geneX,sampleA}/sizefactor_{sampleA}$

```
DESeq.ds <- estimateSizeFactors(DESeq.ds)
sizeFactors(DESeq.ds)
```

```
##    SNF2_1    SNF2_2    SNF2_3    SNF2_4    SNF2_5      WT_1      WT_2
## 1.4257612 1.1080380 1.1007930 1.4800919 0.8917712 0.6003659 0.9428913
##      WT_3      WT_4      WT_5
## 0.7674773 1.1383612 0.9003437
```

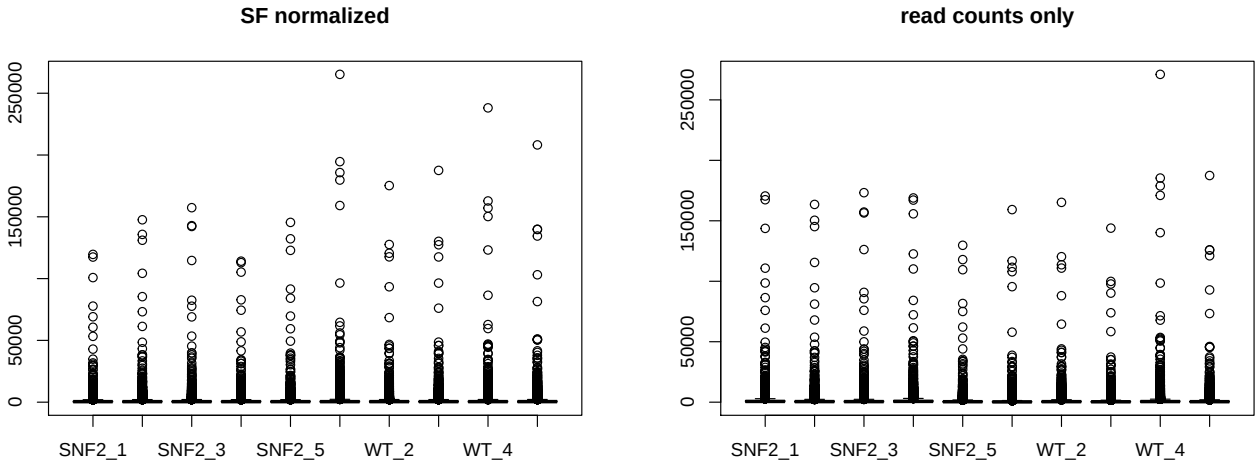
```
plot(sizeFactors(DESeq.ds), colSums(counts(DESeq.ds)))
```



The read counts normalized for sequencing depth can be accessed via `counts(..., normalized = TRUE)`.

Let's check whether the normalization helped adjust global differences between the samples.

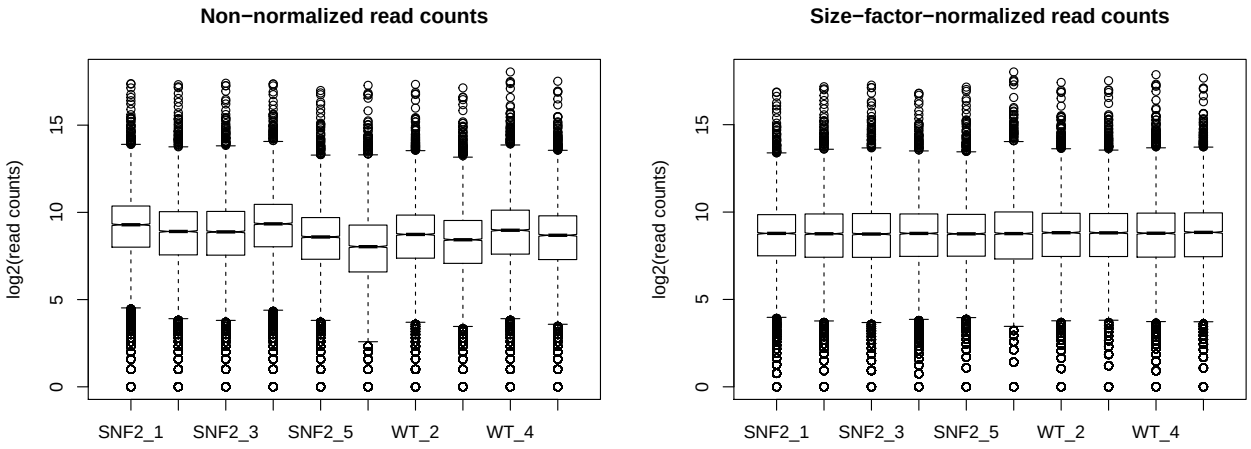
```
## setting up the plotting layout
par(mfrow=c(1,2))

## extracting normalized counts
counts.sf_normalized <- counts(DESeq.ds, normalized=TRUE)

## adding the boxplots
boxplot(counts.sf_normalized, main = "SF normalized")
boxplot(counts(DESeq.ds), main = "read counts only")
```



We can't really see anything because the range of the read counts is so large that it covers several orders of magnitude. For those cases, it is usually helpful to *transform* the normalized read counts to bring them onto more similar scales.

- *To see the influence of the sequencing depth normalization, make two box plots of log2(read counts):*
  - *one for non-normalized counts*
  - *the other one for normalized counts*

```
par(mfrow=c(1,2)) # to plot the two box plots next to each other

## bp of non-normalized
boxplot(log2(counts(DESeq.ds)+1), notch=TRUE,
        main = "Non-normalized read counts",
        ylab="log2(read counts)")
## bp of size-factor normalized values
boxplot(log2(counts(DESeq.ds, normalize= TRUE) +1), notch=TRUE,
        main = "Size-factor-normalized read counts",
        ylab="log2(read counts)")
```



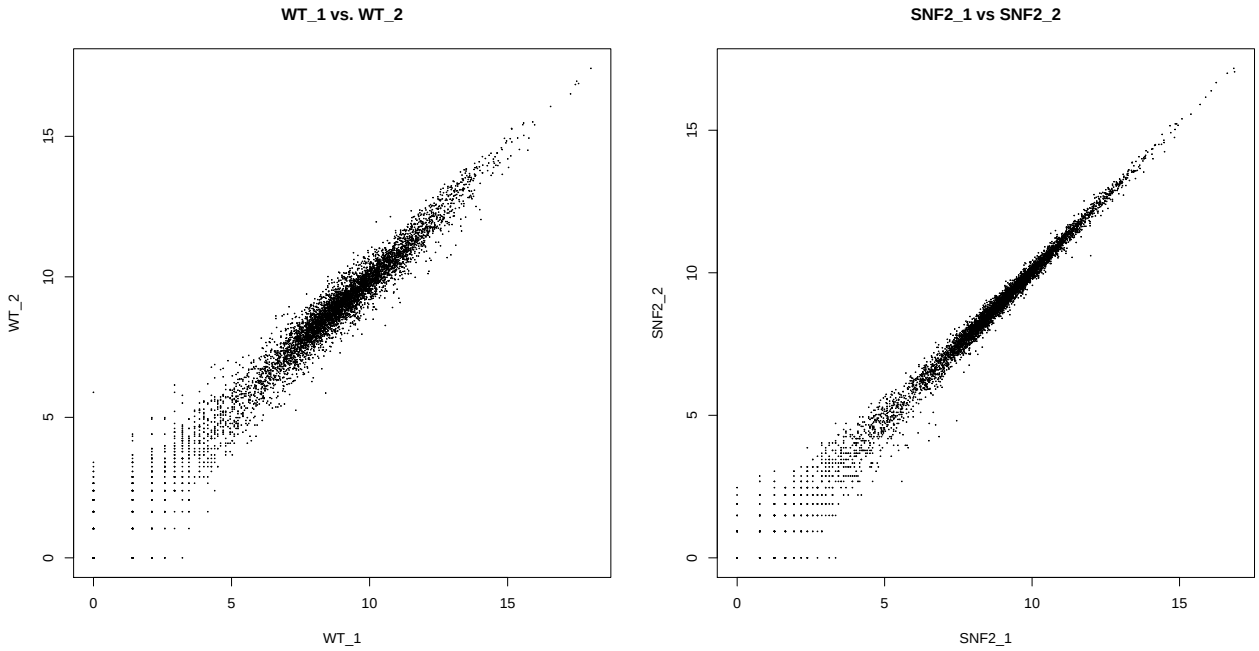
Understanding more properties of read count data

Characteristics we've seen so far:

- zeros can mean two things: no expression or no detection
- fairly large dynamic range
- *Make a scatterplot of log normalized counts against each other to see how well the actual values correlate which each other per sample and gene. Focus on two samples.*

```
## non-normalized read counts plus pseudocount
log.counts <- log2(counts(DESeq.ds, normalized = FALSE) + 1)

## instead of creating a new object, we could assign the values to a distinct matrix
## within the DESeq.ds object
assay(DESeq.ds, "log.counts") <- log2(counts(DESeq.ds, normalized = FALSE) + 1)

## normalized read counts
log.norm.counts <- log2(counts(DESeq.ds, normalized=TRUE) + 1)
assay(DESeq.ds, "log.norm.counts") <- log.norm.counts
```

```
par(mfrow=c(1,2))

DESeq.ds[, c("WT_1","WT_2")] %>%
  assay(.,  "log.norm.counts") %>%
  plot(., cex=.1, main = "WT_1 vs. WT_2")

DESeq.ds[, c("SNF2_1","SNF2_2")] %>%
  assay(.,  "log.norm.counts") %>%
  plot(., cex=.1, main = "SNF2_1 vs SNF2_2")
```



Every dot = one gene.

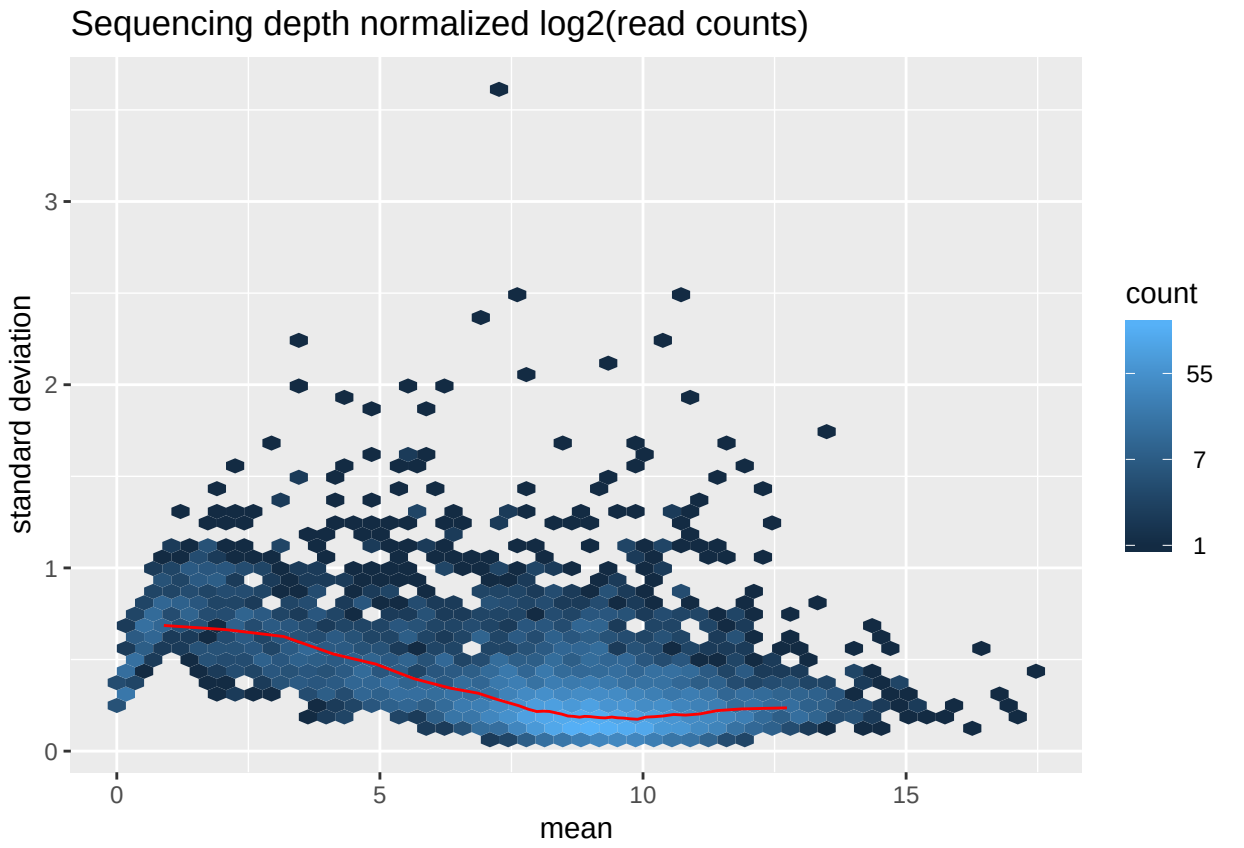
The fanning out of the points in the lower left corner (points below $2^5 = 32$) indicates that read counts correlate less well between replicates when they are low.

This observation indicates that the standard deviation of the expression levels may depend on the mean: the lower the mean read counts per gene, the higher the standard deviation.

This can be assessed visually; the package `vsn` offers a simple function for this.

```
par(mfrow=c(1,1))

## generate the base meanSdPlot using sequencing depth normalized log2(read counts)
log.norm.counts <- log2(counts(DESeq.ds, normalized=TRUE) + 1)
msd_plot <- vsn::meanSdPlot(log.norm.counts,
                        ranks=FALSE, # show the data on the original scale
                        plot = FALSE)
msd_plot$gg +
  ggtitle("Sequencing depth normalized log2(read counts)") +
  ylab("standard deviation")
```

```
## Warning: package 'hexbin' was built under R version 3.4.3
```



From the help for `meanSdPlot`: *The red dots depict the running median estimator (window-width 10 percent). If there is no variance-mean dependence, then the line formed by the red dots should be approximately horizontal.*

The plot here shows that there is some variance-mean dependence for genes with low read counts. This means that the data shows signs of *heteroskedasticity*.

Many tools expect data to be *homoskedastic*, i.e., all variables should have similar variances.

Reducing the dependence of the variance on the mean

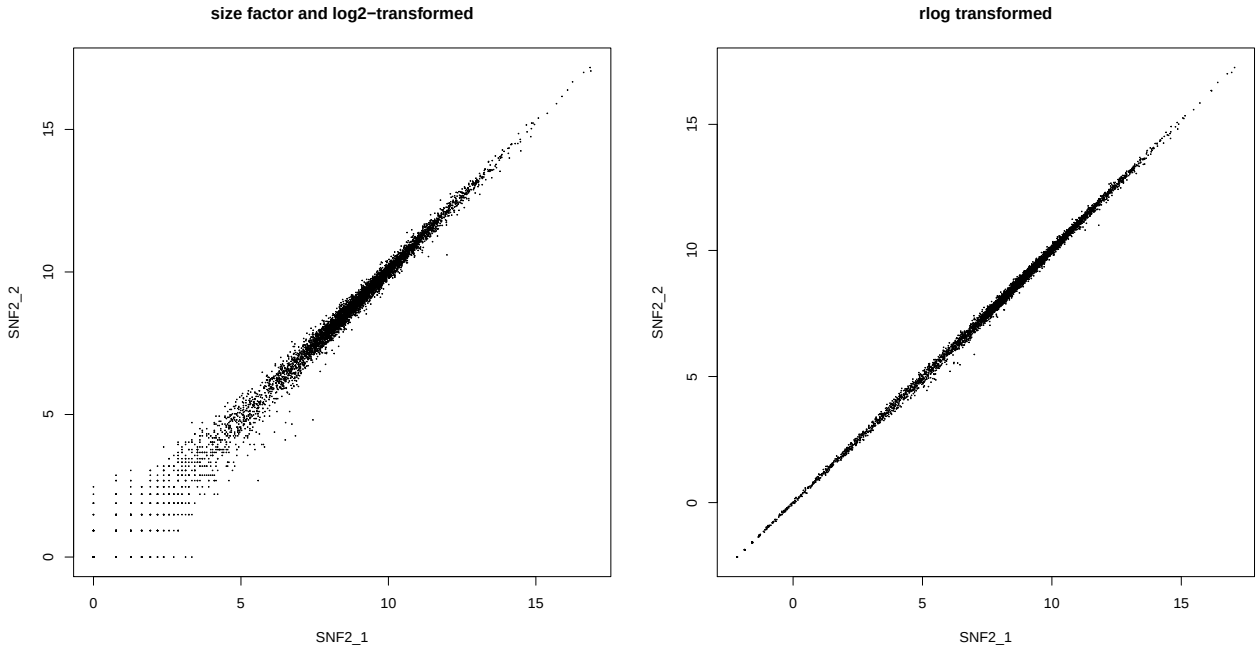
DESeq offers two ways to shrink the log-transformed counts for genes with very low counts: `rlog` and `varianceStabilizingTransformation` (`vst`).

We'll use `rlog` here as it is an optimized method for RNA-seq read counts: it transforms the read counts to the log2 scale while simultaneously minimizing the difference between samples for rows with small counts and taking differences between library sizes of the samples into account. `vst` tends to depend a bit more on the size factors, but generally, both methods should return similar results.

```
## this actually generates a different type of object!
DESeq.rlog <- rlog(DESeq.ds, blind = TRUE)
## set blind = FALSE if the conditions are expected to introduce
## strong differences in a large proportion of the genes
```

Let's visually check the results of the `rlog` transformation:

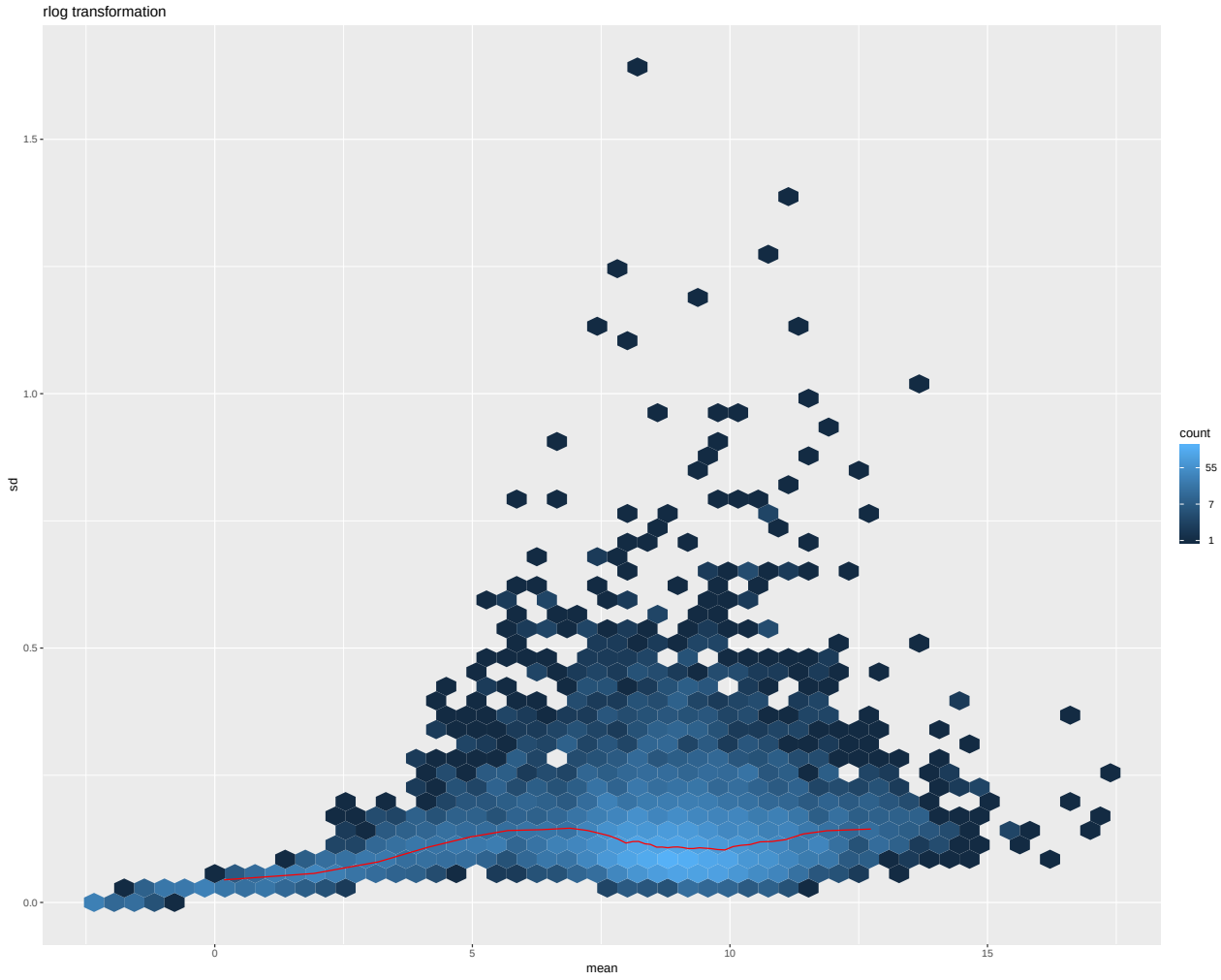
```
par(mfrow=c(1,2))

plot(log.norm.counts[,1:2], cex=.1,
     main = "size factor and log2-transformed")

## the rlog-transformed counts are stored in the accessor "assay"
plot(assay(DESeq.rlog)[,1],
     assay(DESeq.rlog)[,2],
     cex=.1, main = "rlog transformed",
     xlab = colnames(assay(DESeq.rlog[,1])),
     ylab = colnames(assay(DESeq.rlog[,2])) )
```




```
rlog.norm.counts <- assay(DESeq.rlog)
```

As you can see in the left plot the variance - that is higher for small read counts - is tightened significantly using `rlog`. What does the mean-sd-plot show?

```
## rlog-transformed read counts
msd_plot <- vsn::meanSdPlot( rlog.norm.counts, ranks=FALSE, plot = FALSE)
msd_plot$gg + ggtitle("rlog transformation")
```



It's not perfect, but it looks much better than before (compare the y-axis ranges of this plot and the previous one!).

Now, we have **expression values** that have been adjusted for:

- differences in sequencing depth
- differences in RNA composition
- heteroskedasticity
- large dynamic range

These values can now be used for **exploratory analyses** – for DE analyses, we will eventually supply the **raw counts**, though (because the DE tests will require their own modeling of the gene counts).

```
save.image(file = "RNAseqGierlinski.RData")
```