Normalizing read counts

Read counts to DGE, Part I

Friederike Dündar, ABC, WCM

02/25/2020

Contents

featureCounts results **1**

DESeq2 object setup **2**
- countData . . . 3
- colData . . . 3
- Generate the DESeqDataSet . . . 4

Normalizing for sequencing depth and RNA composition differences **6**
- Assumptions of DESeq's size factor method:** . . . 6
- Calculating and applying the size factor . . . 6

Understanding more properties of read count data **8**
- Reducing the dependence of the variance on the mean . . . 9

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).

```
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.

```
readcounts <- read.table("featCounts_Gierlinski_genes.txt", header=TRUE)
str(readcounts)
```

```
## 'data.frame':    6692 obs. of  16 variables:
##  $ Geneid                                            : chr  "YAL012W" "YAL069W" "YAL068W-A" "YAL068C" ...
##  $ Chr                                               : chr  "chrI" "chrI" "chrI" "chrI" ...
##  $ Start                                             : chr  "130799" "335" "538" "1807" ...
##  $ End                                               : chr  "131983" "649" "792" "2169" ...
##  $ Strand                                            : chr  "+" "+" "+" "-" ...
##  $ Length                                            : int  1185 315 255 363 228 1782 309 387 381 381 ...
##  $ ...alignment.SNF2_1_Aligned.sortedByCoord.out.bam: int  7351 0 0 2 0 103 2 5 13 46 ...
##  $ ...alignment.SNF2_2_Aligned.sortedByCoord.out.bam: int  7180 0 0 2 0 51 0 9 8 58 ...
##  $ ...alignment.SNF2_3_Aligned.sortedByCoord.out.bam: int  7648 0 0 2 0 44 0 6 10 45 ...
##  $ ...alignment.SNF2_4_Aligned.sortedByCoord.out.bam: int  8119 0 0 1 0 90 0 3 9 61 ...
##  $ ...alignment.SNF2_5_Aligned.sortedByCoord.out.bam: int  5944 0 0 0 0 53 0 1 6 40 ...
##  $ ...alignment.WT_1_Aligned.sortedByCoord.out.bam  : int  4312 0 0 0 0 12 0 10 9 33 ...
##  $ ...alignment.WT_2_Aligned.sortedByCoord.out.bam  : int  3767 0 0 0 0 23 0 5 12 41 ...
##  $ ...alignment.WT_3_Aligned.sortedByCoord.out.bam  : int  3040 0 0 0 0 21 0 2 4 31 ...
##  $ ...alignment.WT_4_Aligned.sortedByCoord.out.bam  : int  5604 0 0 2 0 30 0 4 4 45 ...
##  $ ...alignment.WT_5_Aligned.sortedByCoord.out.bam  : int  4167 0 0 2 0 29 0 3 8 25 ...
```

The sample names are a bit annoying since they contain the entire BAM file name. Let's turn these into more useful identifiers.

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

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

```r
### 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!

```r
str(readcounts)
```

```
## 'data.frame':    6692 obs. of  16 variables:
##  $ Geneid: chr  "YAL012W" "YAL069W" "YAL068W-A" "YAL068C" ...
##  $ Chr   : chr  "chrI" "chrI" "chrI" "chrI" ...
##  $ Start : chr  "130799" "335" "538" "1807" ...
##  $ End   : chr  "131983" "649" "792" "2169" ...
##  $ Strand: chr  "+" "+" "+" "-" ...
##  $ Length: int  1185 315 255 363 228 1782 309 387 381 381 ...
##  $ 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 ...
```

DESeq2 object setup

We will use the `DESeq2` package to normalize the samples for differences in their sequencing depth and to explore them. First, you will therefore need to make sure that you have the package installed.

```r
## not available via install.packages(), but through bioconductor
BiocManager::install("DESeq2") # only needs to be done once!
```

```r
library(DESeq2)
```

We will have to generate a `DESeqDataSet`, which is a specific R object 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`!*

© *Copyright 2020 – Weill Cornell Medicine*

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

In principle, our `readcounts` is pretty much already in the format that we'll need (columns = Samples, rows = genes), but we're missing `row.names` and 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)]

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`, the meta data containing information about the samples (= columns)

colData

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

```
# let's use the info from our readcounts object
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()
```

© Copyright 2020 – Weill Cornell Medicine

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))
```

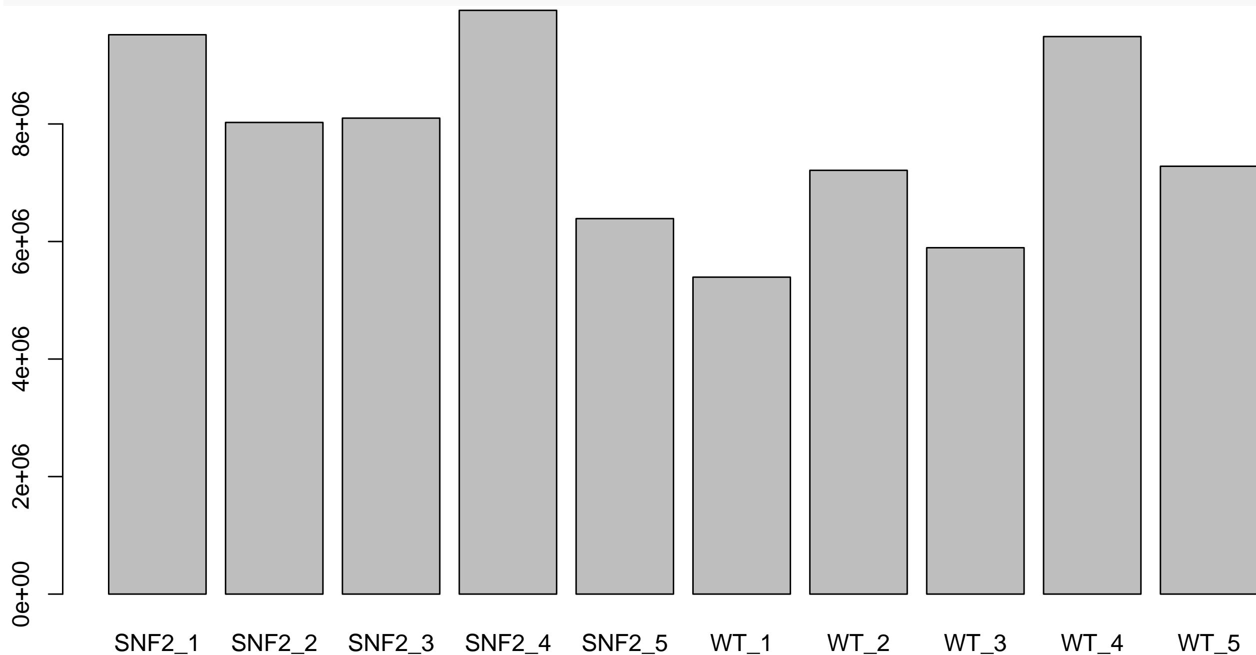
```
##           SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
## YAL012W     7351   7180   7648   8119   5944 4312 3767 3040 5604 4167
## YAL069W        0      0      0      0      0    0    0    0    0    0
## YAL068W.A      0      0      0      0      0    0    0    0    0    0
## YAL068C        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 sequenced 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
```



© Copyright 2020 – Weill Cornell Medicine

Remove genes with no reads.

```
dim(DESeq.ds)
```

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

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

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

As you can see, there are now fewer features stored in the `DESeq.ds` (first entry of the `dim()` result). The filtering was also translated to the count matrix that we store in that object (and all other matrices stored in the `assay` slot).

```
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.

The underlying code is similar to this:

```
## define a function to calculate the geometric mean
gm_mean <- function(x, na.rm=TRUE){ exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x)) }

## calculate the geometric mean for each gene using that function
## note the use of apply(), which we instruct to apply the gm_mean()
## function per row (this is what the second parameter, 1, indicates)
pseudo_refs <- counts(DESeq.ds) %>% apply(., 1, gm_mean)

## divide each value by its corresponding pseudo-reference value
pseudo_ref_ratios <- counts(DESeq.ds) %>% apply(., 2, function(cts){ cts/pseudo_refs})

## if you want to see what that means at the single-gene level,
## compare the result of this:
counts(DESeq.ds)[1,]/pseudo_refs[1]
## with
pseudo_ref_ratios[1,]

## determine the median value per sample to get the size factor
apply(pseudo_ref_ratios , 2, median)
```

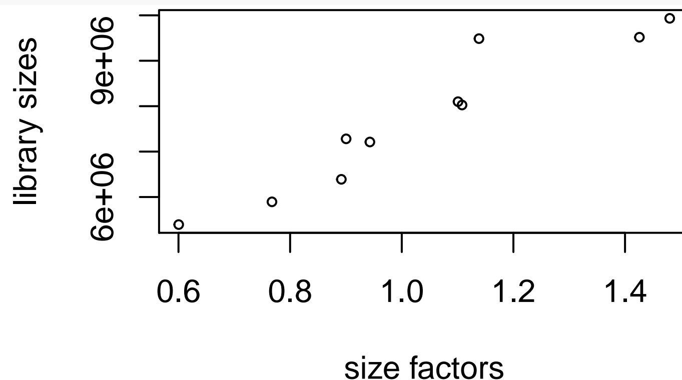
If you want to see the source code for how exactly `DESeq2` calculates the size factors, you can use the following command: `getMethod("estimateSizeFactors", "DESeqDataSet")`.

Assumptions of `DESeq`'s size factor method:**

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

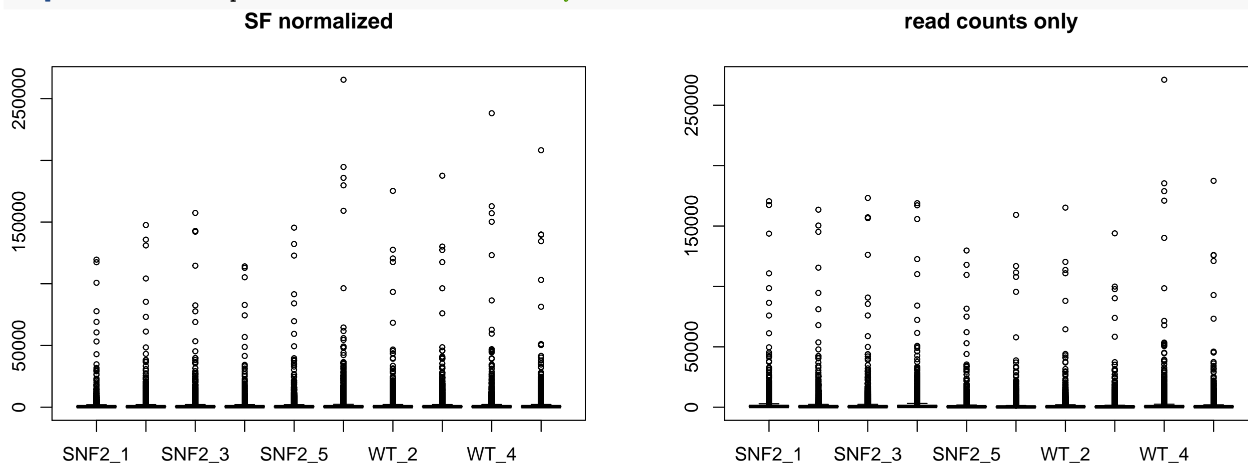
Calculating and applying the size factor

```
DESeq.ds <- estimateSizeFactors(DESeq.ds) # calculate SFs, add them to object
plot( sizeFactors(DESeq.ds), colSums(counts(DESeq.ds)), # assess them
      ylab = "library sizes", xlab = "size factors", cex = .6 )
```



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.

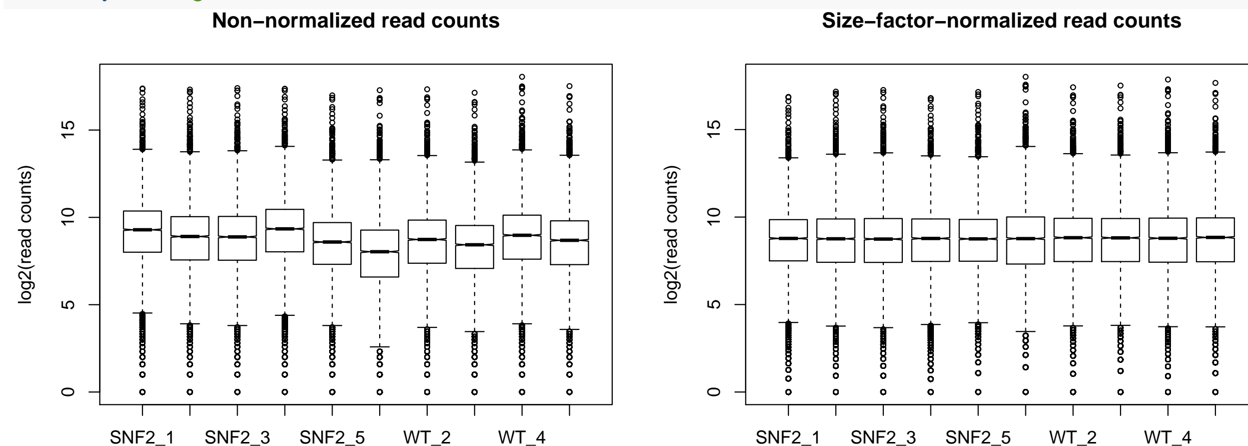
```
## setting up the plotting layout
par(mfrow=c(1,2))

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

## adding the boxplots
boxplot(counts.sf_normalized, main = "SF normalized", cex = .6)
boxplot(counts(DESeq.ds), main = "read counts only", cex = .6)
```



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)", cex = .6)
## 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)", cex = .6)
```



Understanding more properties of read count data

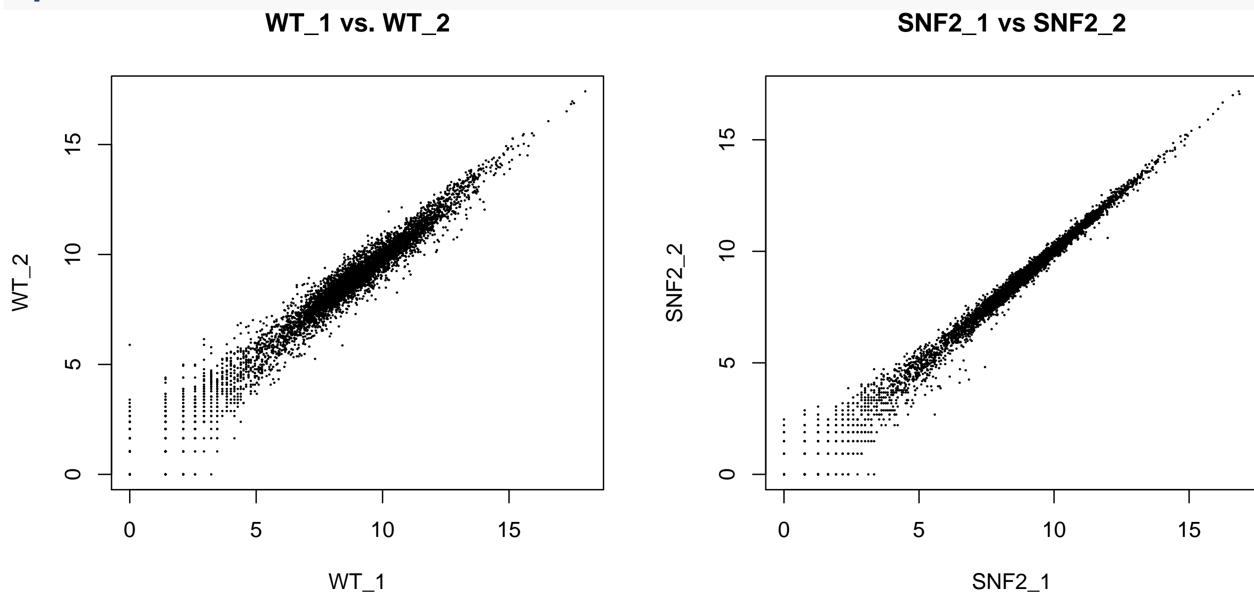
Characteristics we've seen so far:

- 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")
```

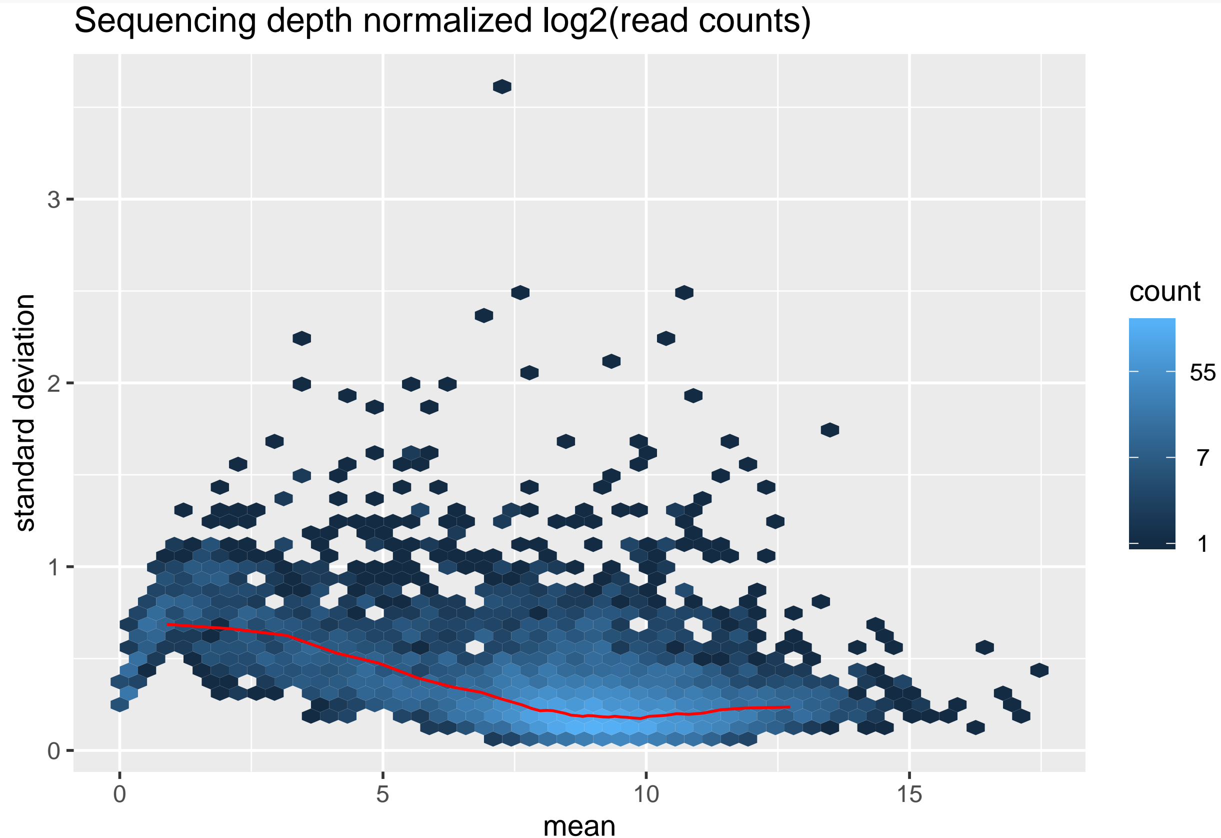


Every dot = one gene.
The fanning out of the points in the lower left corner (points below $2^5 = 32$) indicates that read counts correlate less well between replicates when they are low.
This observation indicates that the standard deviation of the expression levels may depend on the mean: the lower 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.

```
## generate the base meanSdPlot using sequencing depth normalized log2(read counts)
log.norm.counts <- log2(counts(DESeq.ds, normalized=TRUE) + 1)
## set up ploting frames
par(mfrow=c(1,1))
## generate the plot
msd_plot <- vsn::meanSdPlot(log.norm.counts,
                            ranks=FALSE, # show the data on the original scale
```

© Copyright 2020 – Weill Cornell Medicine

```
                        plot = FALSE)
## since vsn::meanSdPlot generates a ggplot2 object, this can be
## manipulated in the usual ways
msd_plot$gg +
  ggtitle("Sequencing depth normalized log2(read counts)") +
  ylab("standard deviation")
```



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.

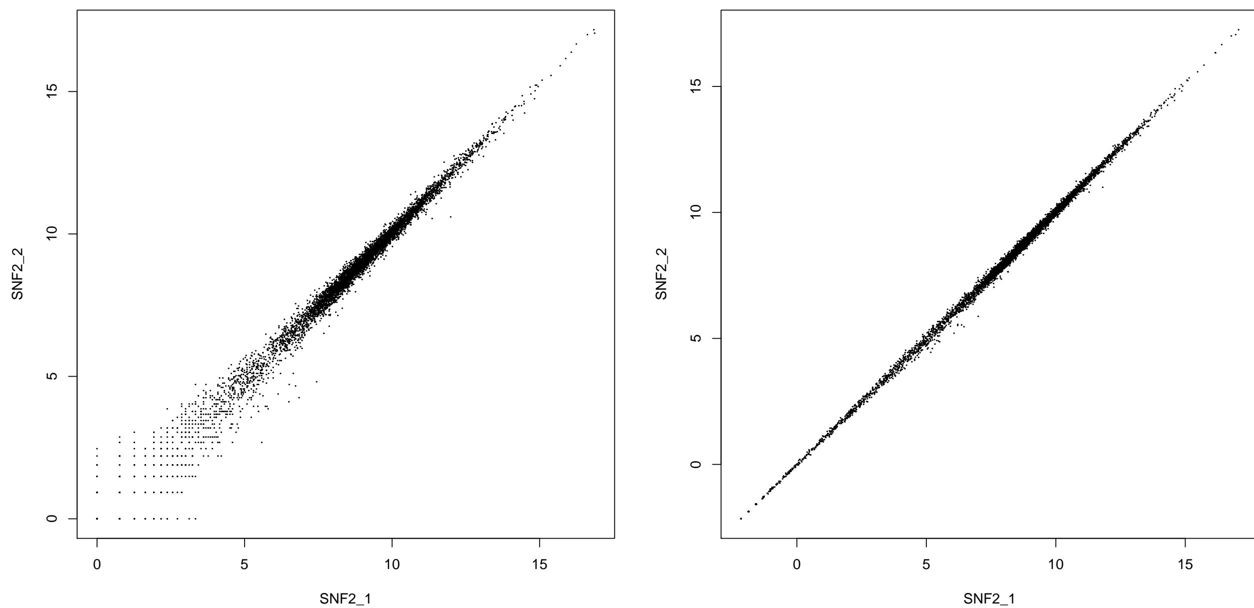
Reducing the dependence of the variance on the mean

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:

```r
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])) )
```



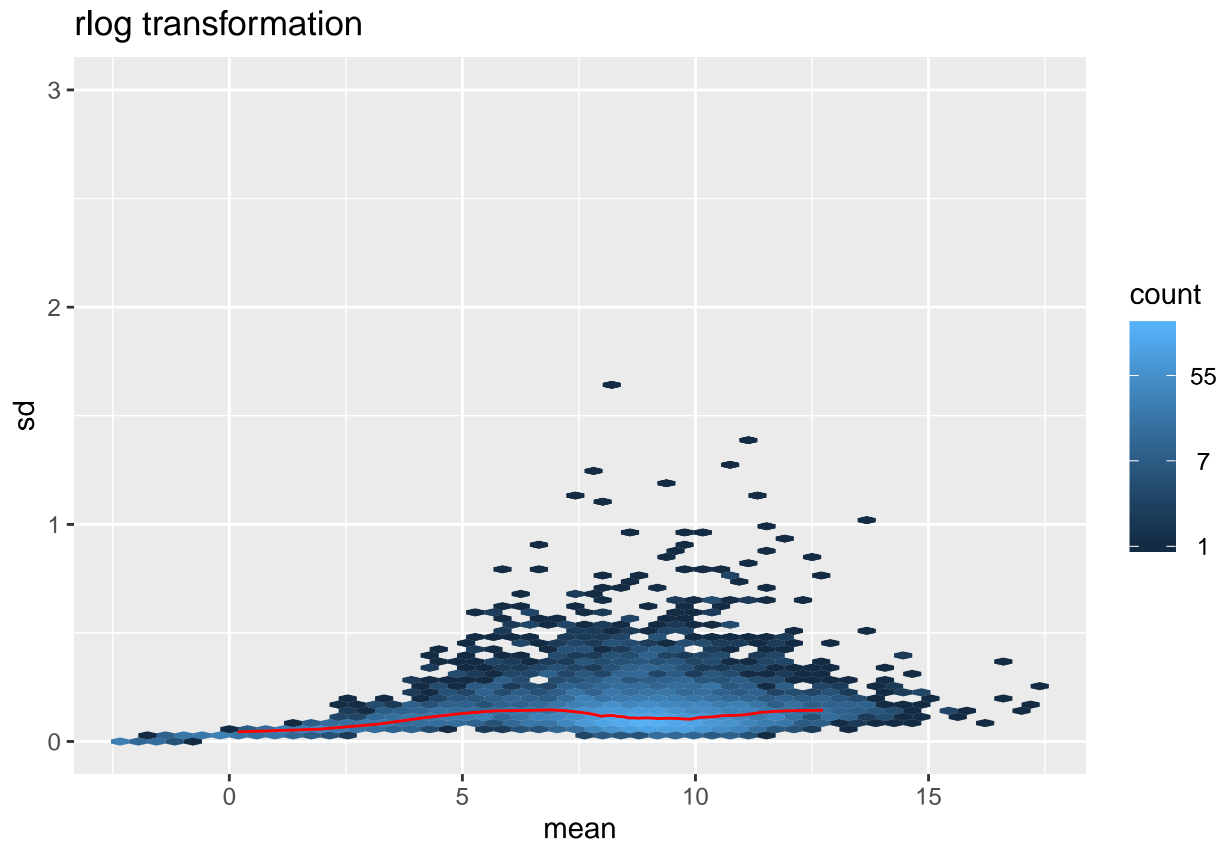

```r
rlog.norm.counts <- assay(DESeq.rlog)
```

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?

```r
## rlog-transformed read counts
msd_plot <- vsn::meanSdPlot( rlog.norm.counts, ranks=FALSE, plot = FALSE)
msd_plot$gg + ggtitle("rlog transformation") + coord_cartesian(ylim = c(0,3))
```

© Copyright 2020 – Weill Cornell Medicine



It's not perfect, but it looks much better than before.
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).

Before we exit the session, let's make sure our objects are stored on disk to be loaded into future sessions:

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