Sequencing: Selected technical topics

Koen Van den Berge

Last edited on 14 October, 2021

Contents

1 Independent filtering

Independent filtering, where genes/transcripts/proteins are filtered out prior to statistical analysis, is a common practice in 'omics experiments. Typically, lowly expressed features are filtered out, and one can argue that these features' expression is too low to be deemed biologically relevant. In addition, low-count features are also associated with a low statistical power for differential expression (remember relative uncertainty of counts, and `edgeR`'s BCV plot), and will increase the number of tests performed, and therefore lead to a more severe multiple testing correction.

Independent filtering has been formalized by Bourgon *et al.* (2010), and the concept can be summarized as follows.

- For each feature we calculate two statistics, S_F and S_T, respectively used for two stages: filtering and testing (e.g., differential expression).
- In order for a feature to be deemed significant, both of its statistics must be greater than some cut-off.
- We want to control the type I error rate of the second stage (testing). But note that **the second stage is conditional on the first stage**, as we only test features passing the filter, and basically ignore the fact that filtering was performed. Indeed, one criticism is that computing and correcting the p-values as if filtering had not been performed may lead to overoptimistic adjusted p-values.
- Bourgon *et al.* (2010) show that filtering is only appropriate (i.e., does not inflate type I error rate) if the conditional null distribution of test statistics for features passing the filter is the same as the unconditional null distribution. Therefore, **filtering is appropriate if the statistic used for filtering is independent of the statistic used for testing under the null hypothesis**.

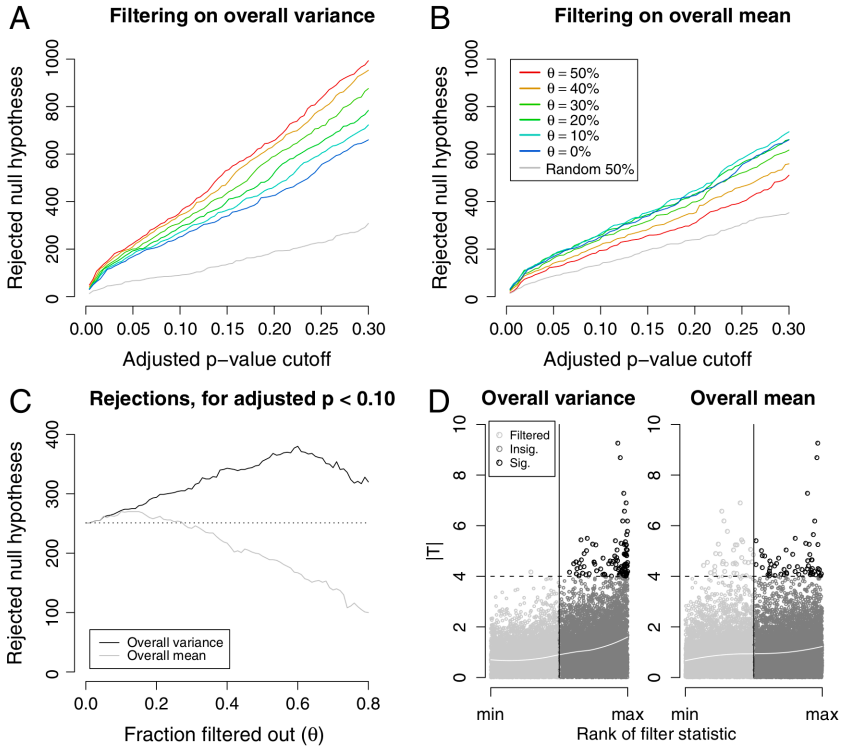


Fig. 1. Power assessment of filtering applied to the ALL data (12,625 genes). *R*, the number of genes called differentially expressed between the two cytogenetic groups, was computed for different stage-one filters, filtering stringencies, and FDR-adjusted *p*-value cutoffs. In all cases, a standard *t*-statistic (*T*) was used in stage two, and adjustment for multiple testing was by the method of ref. 24. Similar results were obtained with other adjustment procedures. Filter cutoffs were selected so that a fraction θ of genes were removed. A random filter, which arbitrarily selected and removed one half of the genes, was also considered. (*A*) Filtering on overall variance (S^2). At all FDR cutoffs, increasingly stringent filtering increased total discoveries, even though fewer genes were tested. This effect was not, however, due to the reduction in the number of hypotheses alone: filtering half of the genes at random reduced total discoveries by approximately one half, as expected. (*B*) Filtering on overall mean ($\bar{Y}$), on the other hand, produced a small increase in rejections at low stringency, but then substantially reduced rejections, and thus power, at higher stringencies. (*C*) Effect of increasing filtering stringency for fixed adjusted *p*-value cutoff $\alpha = 0.1$. At higher stringencies, both filters eventually reduced rejections. For the ALL data, this effect occurred much more quickly for the overall mean filter. With the overall variance filter, the number of rejections increased by up to 50%. (*D*) Filtering on overall mean ($\theta = 0.5$ is shown) removed many significant $|T_i|$ (e.g., $|T_i| > 4$), while filtering on overall variance retained them.

Figure 1: Figure 1 from Bourgon *et al.* (2010).

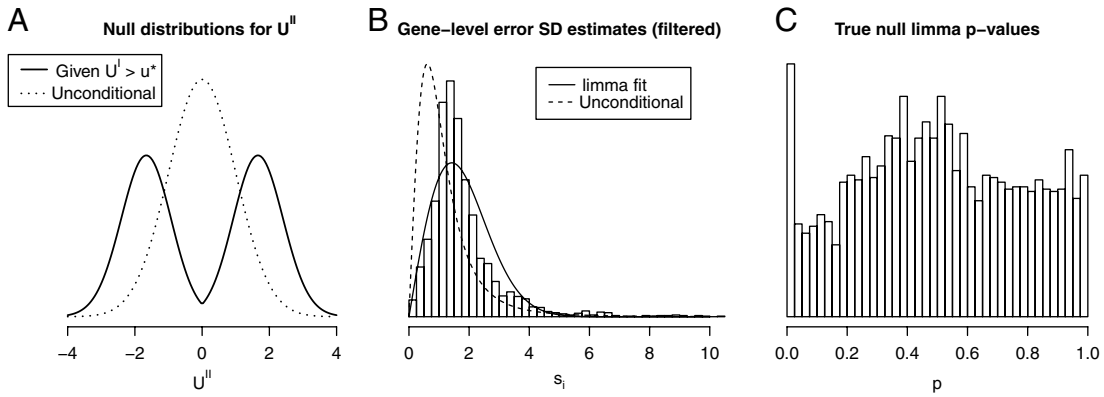


Fig. 2. (*A*) The null distribution of the test statistic is affected by filtering on the maximum of within-class averages. In this example, all genes have a known common variance, the filter statistic is the maximum of within-class means, and the test statistic is a *z*-score. The unconditional distribution of the test statistic for nondifferentially expressed genes is a standard normal. Its conditional null distribution, given that the filter statistic (U^I) exceeds a certain threshold (u^*), however, has much heavier tails. Using the unconditional null distribution to compute *p*-values after filtering would therefore be inappropriate. See *SI Text* for full details. (*B* and *C*) Overall variance filtering and the *limma* moderated *t*-statistic. Data for 5,000 nondifferentially expressed genes were generated according to the *limma* Bayesian model ($n_1 = n_2 = 2$, $d_0 = 3$, $s_0^2 = 1$). (*B*) Filtering on overall variance ($\theta = 0.5$) preferentially eliminated genes with small s_i, causing gene-level standard deviation estimates for genes passing the filter (histogram) to be shifted relative to the unconditional distribution used to generate the data (*dashed curve*). The *limma* inverse χ^2 model was unable to provide a good fit (*solid curve*) to the s_i passing the filter. (*C*) The fitting problems lead to a posterior degrees-of-freedom estimate of ∞. As a consequence, *p*-values were computed using an inappropriate null distribution, producing too many true-null *p*-values close to zero, i.e., loss of type I error rate control. An analogous analysis comparing biological replicates from the ALL study—so that real array data were used but no gene was expected to exhibit significant differential expression—yielded qualitatively similar results.

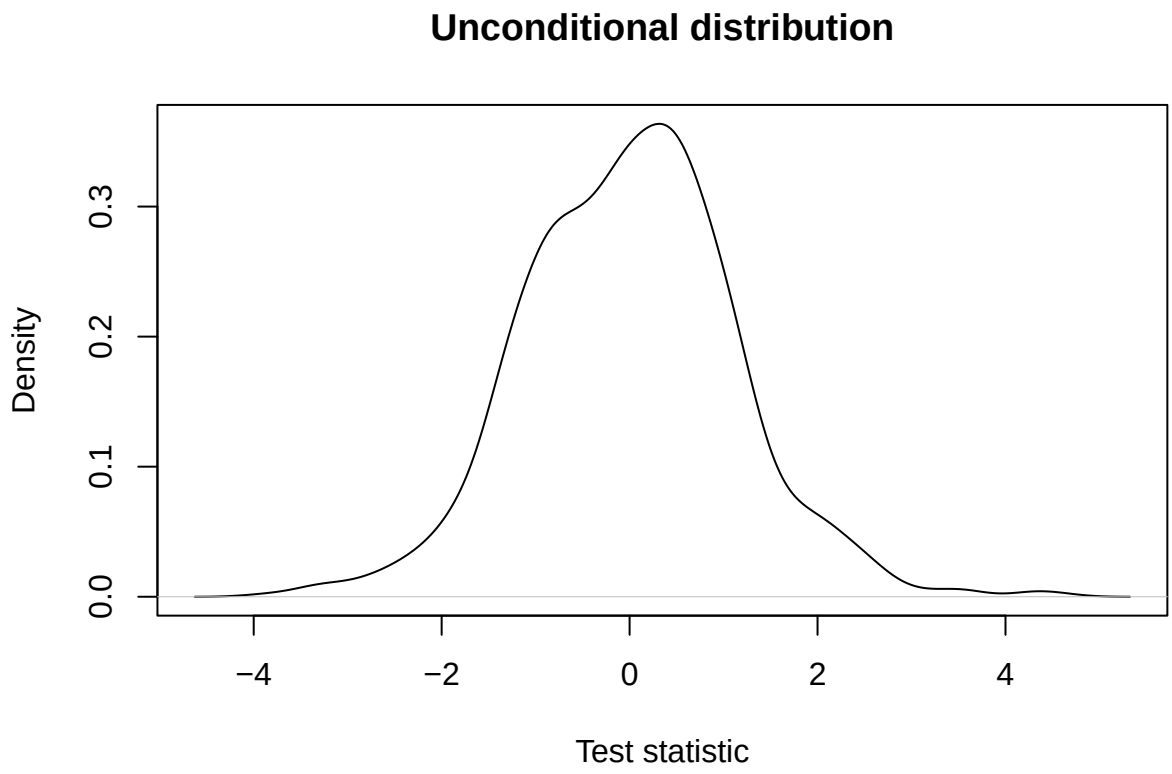
Figure 2: Figure 2 from Bourgon *et al.* (2010).

Let's try a couple of examples to get some intuition using simulated data.

```
suppressPackageStartupMessages(library(DESeq2))
set.seed(24)
dds <- DESeq2::makeExampleDESeqDataSet()
simCounts <- counts(dds)
group <- dds$condition
```

1.1 A dependent test statistic

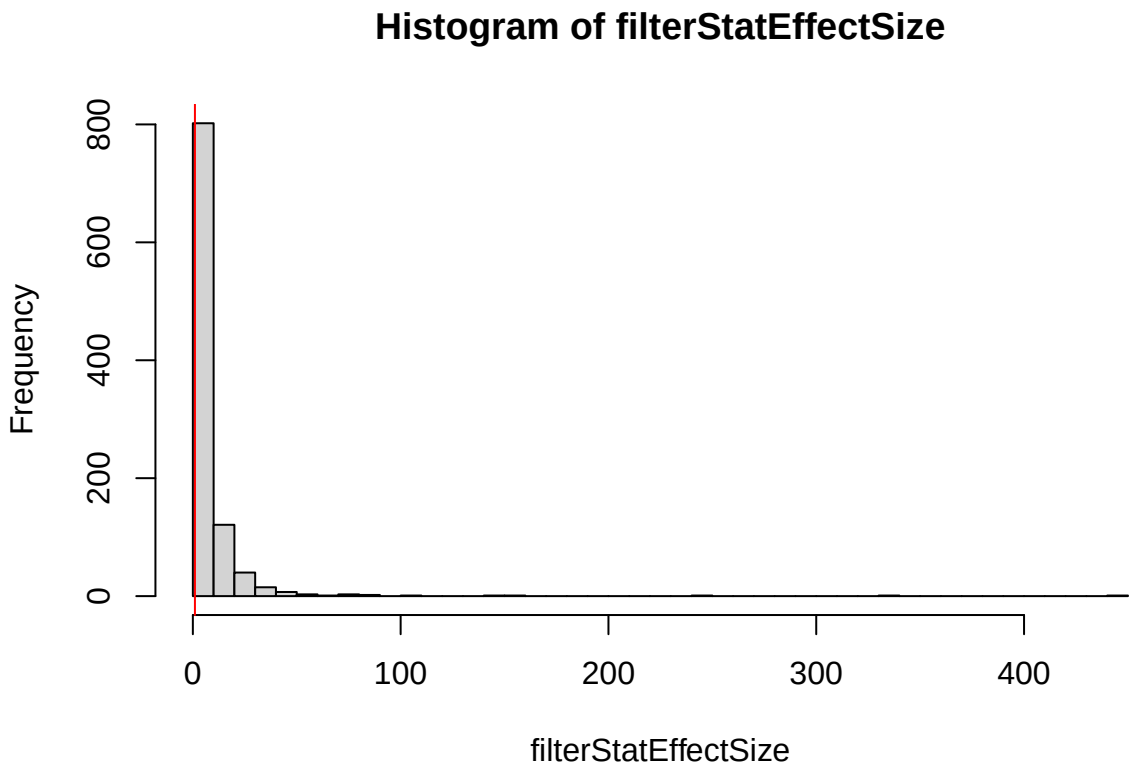
```
filterStatEffectSize <- abs(rowMeans(simCounts[,group == "A"]) - rowMeans(simCounts[,group == "B"]))
testStat <- genefilter::rowttests(simCounts, group)

## unconditional distribution
plot(density(testStat$statistic, na.rm=TRUE),
     xlab = "Test statistic",
     main = "Unconditional distribution")
```



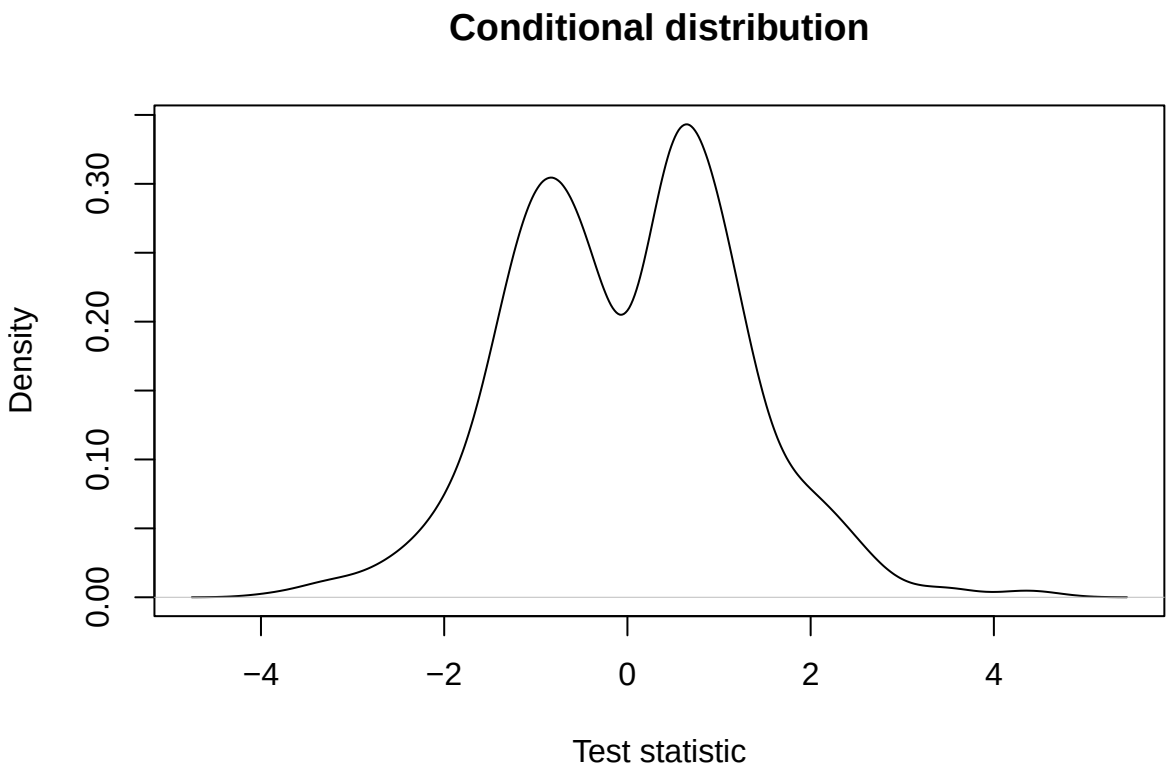

```
## conditional distribution: very different!
mean(filterStatEffectSize > 1)
```

```
## [1] 0.792
```

```
hist(filterStatEffectSize, breaks=40)
abline(v=1, col="red")
```




```
keepEffectSize <- filterStatEffectSize > 1
plot(density(testStat$statistic[keepEffectSize], na.rm=TRUE),
     xlab = "Test statistic",
     main = "Conditional distribution")
```

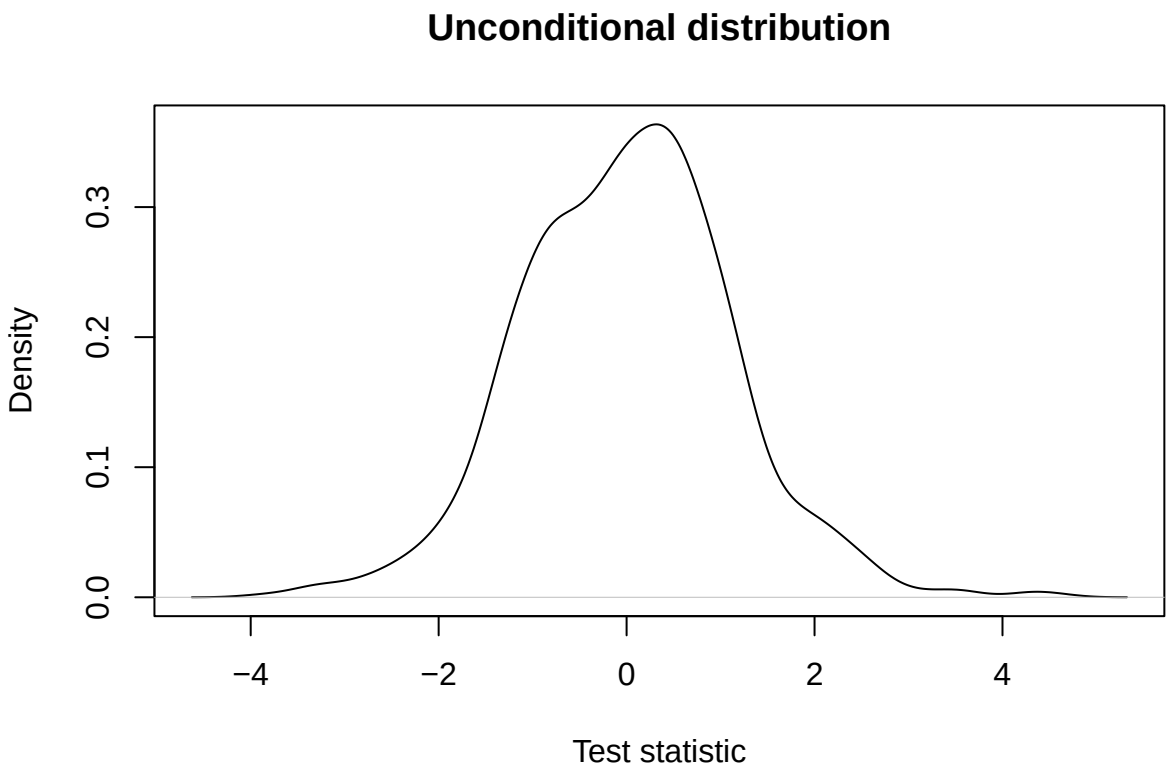


1.2 An independent test statistic

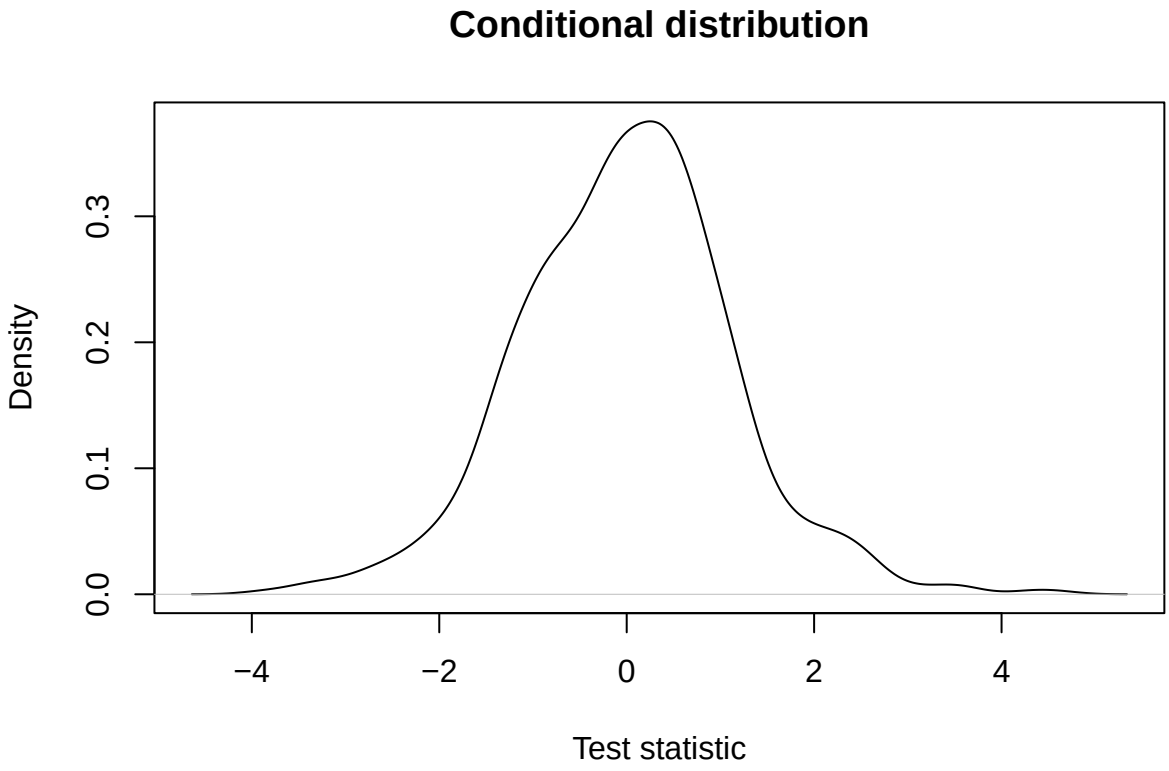
```
filterStatGlobalMean <- rowMeans(simCounts)
mean(filterStatGlobalMean > 5) # we remove a similar fraction
```

```
## [1] 0.771
```

```
keepGlobalMean <- filterStatGlobalMean > 5

## unconditional distribution
plot(density(testStat$statistic, na.rm=TRUE),
     xlab = "Test statistic",
     main = "Unconditional distribution")
```




```
## conditional distribution: the same.
plot(density(testStat$statistic[keepGlobalMean], na.rm=TRUE),
     xlab = "Test statistic",
     main = "Conditional distribution")
```



2 Aliasing

Suppose we are working with the following experimental design on colon cancer. Studying the effect of a drug on gene expression, researchers gather RNA-seq data from four colon cancer patients and four healthy individuals. For each individual, they obtain RNA-seq data from a blood sample before as well as two weeks after taking a daily dose of the drug. The research question relates to differential expression after vs. before taking the drug, in particular whether this is different for the diseased versus healthy group (i.e., the interaction between time (before/after taking the drug) and disease status (healthy/colon cancer)).

In terms of the model matrix, we could imagine a design such as `~ patient + disease*time`, where

- `disease` is a binary indicator referring to colon cancer versus control sample.
- `time` defines if the sample is taken before or after taking the drug.
- `patient` defines the individual donor the sample comes from.

The research question could then amount to testing the `disease * time` interaction.

Let's try this, by simulating random data for one gene.

```
set.seed(2)
patient <- factor(rep(letters[1:8], each=2)) # 2 samples per patient
disease <- factor(c(rep("healthy",8), rep("cancer",8)), levels=c("healthy", "cancer")) # first four are
time <- factor(rep(c("before", "after"), 8), levels=c("before", "after")) # one before and one after sa

table(patient, disease, time)
```

```
## , , time = before
##
##         disease
## patient healthy cancer
##       a       1      0
##       b       1      0
##       c       1      0
##       d       1      0
##       e       0      1
##       f       0      1
##       g       0      1
##       h       0      1
##
## , , time = after
##
##         disease
## patient healthy cancer
##       a       1      0
##       b       1      0
##       c       1      0
##       d       1      0
##       e       0      1
##       f       0      1
##       g       0      1
##       h       0      1
```

```
## simulate data for one gene
n <- 16
y <- rpois(n = n, lambda = 50)

## fit a Poisson model
m <- glm(y ~ patient + disease*time,
         family = "poisson")
summary(m)
```

```
##
## Call:
## glm(formula = y ~ patient + disease * time, family = "poisson")
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.52772  -0.43544   0.00013   0.44162   1.34650
##
## Coefficients: (1 not defined because of singularities)
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)              3.76900    0.11916  31.631   <2e-16 ***
## patientb                 0.06744    0.14999   0.450   0.6530
## patientc                 0.06744    0.14999   0.450   0.6530
## patientd                 0.27304    0.14310   1.908   0.0564 .
## patiente                 0.16449    0.16224   1.014   0.3107
## patientf                 0.02565    0.16644   0.154   0.8775
## patientg                -0.01784    0.16785  -0.106   0.9154
## patienth                 0.05706    0.16544   0.345   0.7302
## diseasecancer                 NA         NA      NA       NA
```

```
## timeafter               -0.01567    0.10220  -0.153   0.8782
## diseasecancer:timeafter  0.12374    0.14407   0.859   0.3904
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 16.1200  on 15  degrees of freedom
## Residual deviance:  8.8417  on  6  degrees of freedom
## AIC: 120.16
##
## Number of Fisher Scoring iterations: 4
```

We find that one of the coefficients is `NA`! This is obviously not because we're dealing with `NA` values in the data as we've just simulated the response variable. What's going on?

One of the parameters, in this case the parameter distinguishing cancer from healthy patients **cannot be estimated as it is a linear combination of other parameters**. In our case, estimating the diseased effect would use information that is already used to estimate the patient-level intercepts. In other words, **once you know the patient, you immediately also know the disease status**, so estimating the diseased vs healthy effect on top of the patient effect provides no additional information if we have already estimated the patient-level effects. This concept is called aliasing, and is a common technical issue in 'omics experiments with complex experimental designs.

While to understand the origin of the aliasing it is crucial to understand the relationship between the variables in the experimental design, we can also investigate it in detail using the `alias` function, to give us an idea.

```
alias(m)
```

```
## Model :
## y ~ patient + disease * time
##
## Complete :
##               (Intercept) patientb patientc patientd patiente patientf patientg
## diseasecancer 0           0        0        0        1        1        1
##               patienth timeafter diseasecancer:timeafter
## diseasecancer 1        0         0
```

We see that the effect `diseasecancer` is a linear combination of the patient-specific effects of the cancer patients. This makes sense!

For clarity, let's reproduce this using our design matrix.

```
X <- model.matrix(~ patient + disease*time) # this is the design used in glm()

## these are indeed identical.
X[,"diseasecancer"]
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
```

```
X[,"patiente"] + X[,"patientf"] + X[,"patientg"] + X[,"patienth"]
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
```

Since one of our parameters is a linear combination of other parameters, it cannot be estimated simultaneously with the other parameters. In this case, we can actually drop the `disease` main effect from the model, since we know that it is already included in the `patient` effect.

We will have to carefully construct our design matrix in order to account for all important sources of variation while still allowing us to answer the research question of interest. The aliasing exploration above has made it clear we may drop the `disease` main effect, so let's start by constructing this design matrix.

```
X <- model.matrix(~ patient + time + disease:time)

m2 <- glm(y ~ -1 + X,
          family = "poisson")
summary(m2)
```

```
##
## Call:
## glm(formula = y ~ -1 + X, family = "poisson")
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.52772  -0.43544   0.00013   0.44162   1.34650
##
## Coefficients: (1 not defined because of singularities)
##                             Estimate Std. Error z value Pr(>|z|)
## X(Intercept)                 3.76900    0.11916  31.631   <2e-16 ***
## Xpatientb                    0.06744    0.14999   0.450   0.6530
## Xpatientc                    0.06744    0.14999   0.450   0.6530
## Xpatientd                    0.27304    0.14310   1.908   0.0564 .
## Xpatiente                    0.28823    0.16077   1.793   0.0730 .
## Xpatientf                    0.14939    0.16500   0.905   0.3653
## Xpatientg                    0.10590    0.16643   0.636   0.5246
## Xpatienth                    0.18081    0.16400   1.102   0.2703
## Xtimeafter                  -0.01567    0.10220  -0.153   0.8782
## Xtimebefore:diseasecancer -0.12374    0.14407  -0.859   0.3904
## Xtimeafter:diseasecancer        NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 4489.2752  on 16  degrees of freedom
## Residual deviance:    8.8417  on  6  degrees of freedom
```

```
## AIC: 120.16
##
## Number of Fisher Scoring iterations: 4
```

```
alias(m2)
```

```
## Model :
## y ~ -1 + X
##
## Complete :
##                            X(Intercept) Xpatientb Xpatientc Xpatientd Xpatiente
## Xtimeafter:diseasecancer  0            0         0         0         1
##                            Xpatientf Xpatientg Xpatienth Xtimeafter
## Xtimeafter:diseasecancer  1         1         1         0
##                            Xtimebefore:diseasecancer
## Xtimeafter:diseasecancer -1
```

We are still confronted with aliasing as the model matrix contains an interaction effect `timebefore:diseasecancer` as well as `timeafter:diseasecancer`, while only the latter is relevant. Indeed, we know that we can derive the `timebefore:diseasecancer` effect by averaging the patient effects of the cancer patients.

```
X <- X[,!colnames(X) %in% "timebefore:diseasecancer"]


## fit a Poisson model
m2 <- glm(y ~ -1 + X,
          family = "poisson")
summary(m2)
```

```
##
## Call:
## glm(formula = y ~ -1 + X, family = "poisson")
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.52772  -0.43544   0.00013   0.44162   1.34650
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## X(Intercept)               3.76900    0.11916  31.631   <2e-16 ***
## Xpatientb                  0.06744    0.14999   0.450   0.6530
## Xpatientc                  0.06744    0.14999   0.450   0.6530
## Xpatientd                  0.27304    0.14310   1.908   0.0564 .
## Xpatiente                  0.16449    0.16224   1.014   0.3107
## Xpatientf                  0.02565    0.16644   0.154   0.8775
## Xpatientg                 -0.01784    0.16785  -0.106   0.9154
## Xpatienth                  0.05706    0.16544   0.345   0.7302
## Xtimeafter                -0.01567    0.10220  -0.153   0.8782
## Xtimeafter:diseasecancer  0.12374    0.14407   0.859   0.3904
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 4489.2752  on 16  degrees of freedom
## Residual deviance:    8.8417  on  6  degrees of freedom
## AIC: 120.16
##
## Number of Fisher Scoring iterations: 4
```

We see that all coefficients can now be estimated. The `timeafter` effect may be interpreted as the time effect for healthy patients, while the `timeafter:diseasecancer` effect may be interpreted as the difference in the time effect for cancer patients, with the time effect for healthy patients, i.e., it is the relevant interaction effect we are interested in.

Question. Taking this further, suppose that we can safely assume that there is no interaction effect between disease status and time. How would you now test for differential expression between healthy and cancer patients at the first timepoint? Specify the experimental design and contrast used.

Answer.

Assuming no interaction, we can specify the design as follows:

```
XMain <- model.matrix(~ patient + time)
head(XMain)
```

```
##   (Intercept) patientb patientc patientd patiente patientf patientg patienth
## 1           1        0        0        0        0        0        0        0
## 2           1        0        0        0        0        0        0        0
## 3           1        1        0        0        0        0        0        0
## 4           1        1        0        0        0        0        0        0
## 5           1        0        1        0        0        0        0        0
## 6           1        0        1        0        0        0        0        0
##   timeafter
## 1         0
## 2         1
## 3         0
## 4         1
## 5         0
## 6         1
```

In order to set up the contrast testing for healthy versus diseased patients at the first timepoint, we need to take the average of the appropriate patient-level intercepts. The average expression for healthy patients is

$$\log \mu_{healthy} = \frac{1}{4} \left\{ \beta_0 + (\beta_0 + \beta_1) + (\beta_0 + \beta_2) + (\beta_0 + \beta_3) \right\}.$$

Similar, for the diseased patients it equals

$$\log \mu_{diseased} = \frac{1}{4} \left\{ (\beta_0 + \beta_4) + (\beta_0 + \beta_5) + (\beta_0 + \beta_6) + (\beta_0 + \beta_7) \right\}.$$

And thus the relevant contrast

$$\log \frac{\mu_{diseased}}{\mu_{healthy}} = \frac{1}{4}(\beta_4 + \beta_5 + \beta_6 + \beta_7) - \frac{1}{4}(\beta_1 + \beta_2 + \beta_3).$$

3 limma-voom as an alternative approach to modeling counts

`limma` is a powerful linear model based framework for modeling microarray gene expression data and inferring differential expression results, and has been introduced in the proteomics module of this course. With the inception of RNA-seq, the `limma` developers got creative and extended their framework to also model count data, hence creating limma-voom.

3.1 The limma framework for the analysis of microarrays

In the proteomics module, we have previously introduced the powerful linear model based framework `limma`, and how it uses an empirical Bayes strategy to borrow information across proteins to derive a posterior variance estimate. In its default implementation, `limma` cannot be used to model count data, as it can not account for their mean-variance relationship. The developers, however, came up with a creative approach to use the `limma` framework to model coutn data.

3.2 limma-voom: extending limma for RNA-seq data

- Count models such as `edgeR` and `DESeq2` automatically account for the mean-variance relationship of the data, given that the observed mean-variance relationship is close to the one assumed by the distribution. However, they are also more complex, both computationally as well as statistically and conceptually.
- limma-voom (Law *et al.* (2014)) is a method that unlocks Gaussian linear models to analyze count data in the context of RNA-seq, by first estimating the mean-variance relationship of the dataset at hand, and subsequently incorporating it in the analysis through observation-level weights in a linear regression model.

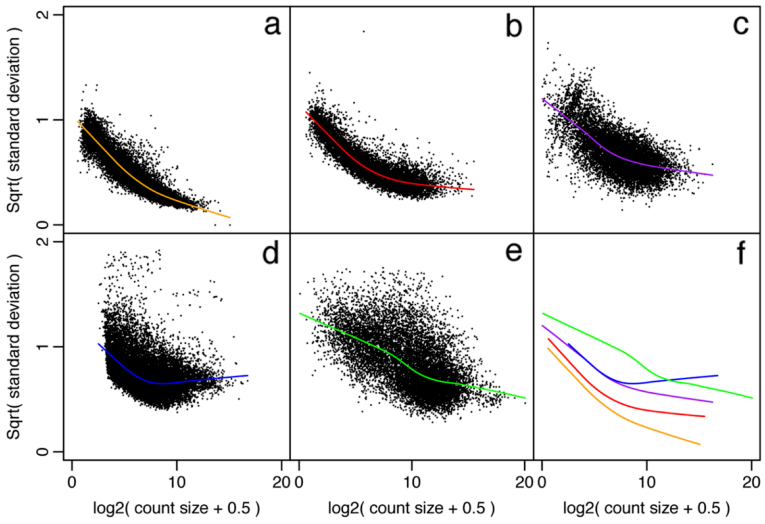


Figure 1 Mean-variance relationships. Gene-wise means and variances of RNA-seq data are represented by black points with a LOWESS trend. Plots are ordered by increasing levels of biological variation in datasets. **(a)** voom trend for HBRR and UHRR genes for Samples A, B, C and D of the SEQC project; technical variation only. **(b)** C57BL/6J and DBA mouse experiment; low-level biological variation. **(c)** Simulation study in the presence of 100 upregulating genes and 100 downregulating genes; moderate-level biological variation. **(d)** Nigerian lymphoblastoid cell lines; high-level biological variation. **(e)** *Drosophila melanogaster* embryonic developmental stages; very high biological variation due to systematic differences between samples. **(f)** LOWESS voom trends for datasets (a)–(e). HBRR, Ambion's Human Brain Reference RNA; LOWESS, locally weighted regression; UHRR, Stratagene's Universal Human Reference RNA.

Figure 3: Figure 1 from Law *et al.* (2014).

- The mean-variance relationship is dataset-specific and needs to be estimated separately for each dataset.
- The mean-variance trend is estimated nonparametrically across all genes, using a global mean and variance for each gene. Using this trend, observation-level variances are estimated for each individual observation.
- These observation-level variances are then used as inverse weights in the linear modeling framework, to account for heteroscedasticity.

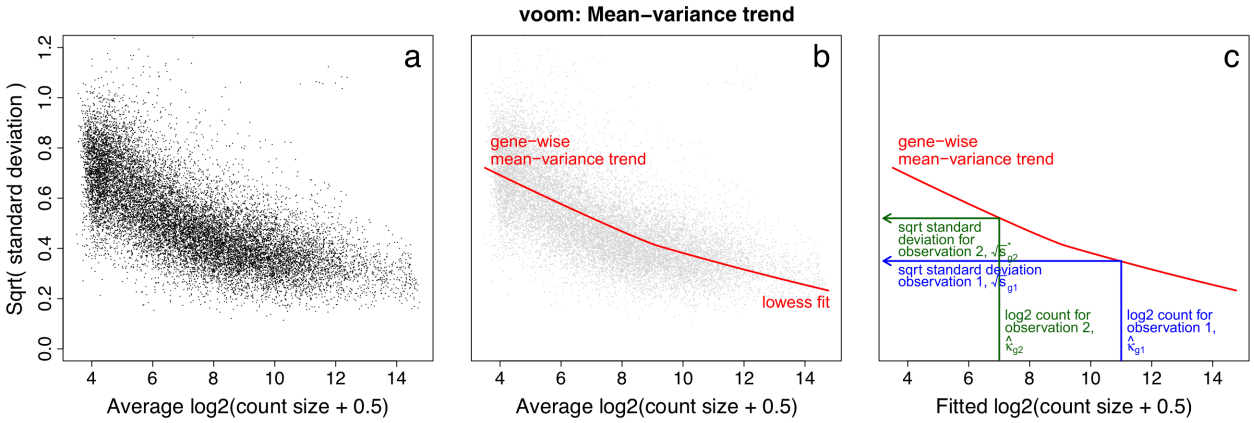


Figure 2 voom mean-variance modeling. (a) Gene-wise square-root residual standard deviations are plotted against average log-count. **(b)** A functional relation between gene-wise means and variances is given by a robust LOWESS fit to the points. **(c)** The mean-variance trend enables each observation to map to a square-root standard deviation value using its fitted value for log-count. LOWESS, locally weighted regression.

Figure 4: Figure 1 from Law *et al.* (2014).