Sequencing: Selected technical topics

Lieven Clement & Koen Van den Berge

Last edited on 28 November, 2022

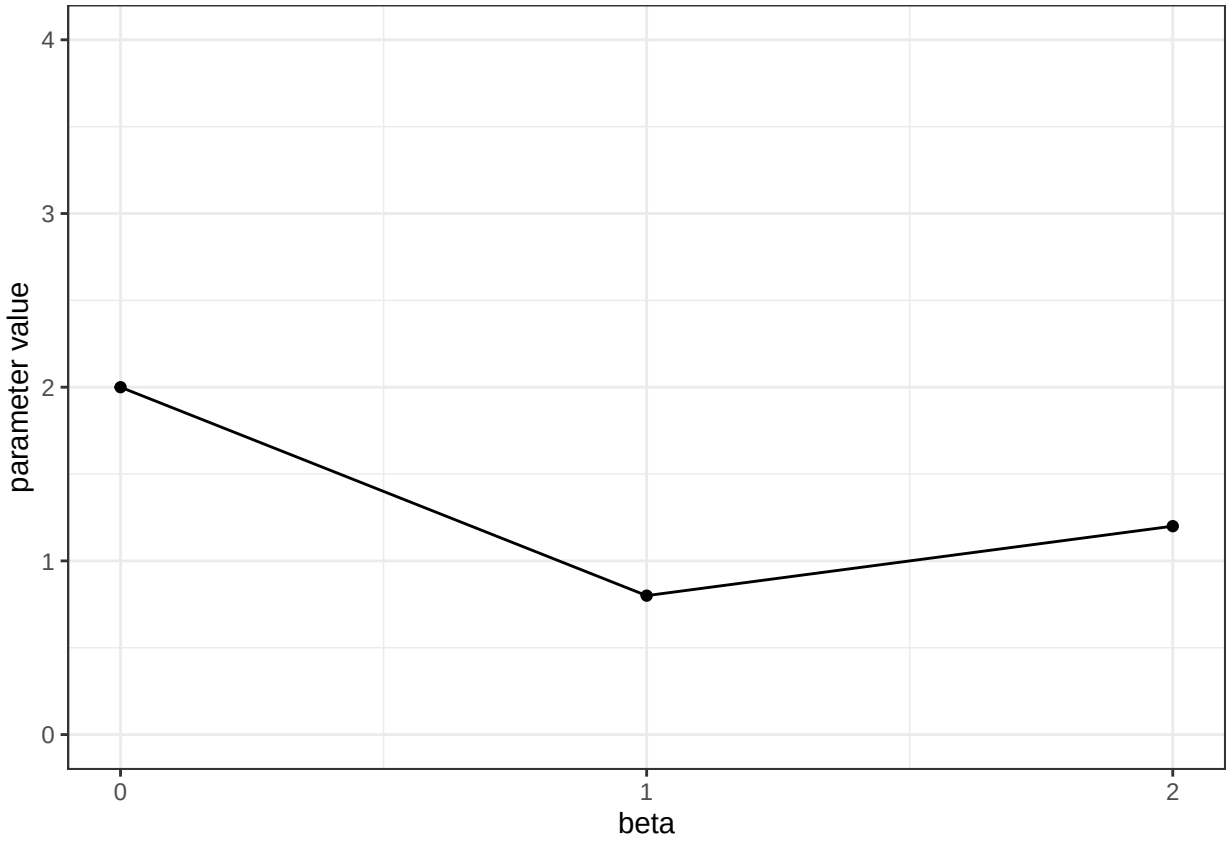
Contents

1 Parameter Estimation and Inference in Generalized linear models

1.1 Simulate Poisson Data

- We simulate data for 100 observations.
- Covariates x are simulated from normal distribution
- The β are chosen at $\beta_0 = 2$, $\beta_1 = 0.8$, $\beta_2 = 1.2$

```
set.seed(300)
xhlp<-cbind(1,rnorm(100),rnorm(100))
betasTrue<-c(2,0.8,1.2)
etaTrue<-xhlp%*%betasTrue
y<-rpois(100,exp(etaTrue))
data.frame(coef =0:2,betasTrue=betasTrue) %>%
  ggplot(aes(x=coef,y=betasTrue)) +
  geom_point() +
  geom_line() +
  ylab("parameter value") +
  xlab("beta") +
  ylim(0,4) +
  theme_bw() +
  scale_x_continuous(breaks=c(0,1,2))
```



1.2 Exponential Family

$$f(y_i|\theta_i, \phi) = \exp\left\{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\right\}$$

with

- θ_i: canonical parameters
- ϕ: dispersion parameter
- $a(.)$, $b(.)$, $c(.)$: specific functions that depend on the distribution,

e.g. for normal distribution

- $\phi = \sigma^2$,
- $\theta = \mu$,
- $a(\phi) = \phi = \sigma^2$,
- $b(\theta_i) = \theta_i^2/2$,
- $c(y_i, \phi) = -\frac{1}{2}[y^2/\phi + \log(2\pi\phi)]$

1.2.1 Poisson Distribution

$$y_i \sim \frac{\mu_i^{y_i} e^{-\mu_i}}{y_i!}$$

In format of exponential family:

$$y_i \sim \exp\{y_i \log(\mu_i) - \mu_i - \log(y_i!)\}$$

1.3 Components of Generalized Linear Model

$$\begin{cases} y_i|x_i & \sim & f(y_i|\theta_i, \phi) \\ \mathrm{E}[y_i|\mathbf{x}_i] & = & \mu_i \\ g(\mu_i) & = & \eta(\mathbf{x}_i) \\ \eta(\mathbf{x}_i) & = & \mathbf{x}_i^T\beta \end{cases},$$

with $g(.)$ the link function, e.g.

- $g(.) = .$: identity link for Normal distribution
- $g(.) = \log(.)$: canonical link for Poisson distribution
- $g(.) = \mathrm{logit}(.) = \log\left[\frac{(.)}{(1-.)}\right]$: canonical link for Bernouilli distribution.

1.3.1 Poisson GLM

$$\begin{cases} y_i & \sim & Poisson(\mu_i) \\ E[y_i] & = & \mu_i \\ \log(\mu_i) & = & \eta_i \\ \eta_i & = & \mathbf{x}_i\beta \end{cases}$$

1.4 Likelihood

We start from a sample, and consider it as fixed and known.

- In particular we do NOT consider the sample observations as random variables.
- Therefore we write the observed sample as $y_i, ..., y_n$
- The theory is based on the likelihood function, which can be interpreted as a measure for the probability that the given sample is observed as a realisation of a sequence of random variables $Y_1, ... Y_n$
- The random variables Y_i are characterized by a distribution or density function which has typically unknown parameters, e.g. a Poisson distribution $f(Y_i) \sim \text{Poisson}(\theta_i)$.
- When the subjects are mutually independent the joint likelihood to observe $y_1, ..., y_n$ equals

$$\prod_{i=1}^{n} f(y_i, \theta_i, \phi)$$

- The densities are actually also a function of the parameters θ_i, ϕ. To stress this, we indicated that in the density formulation.
- The likelihood function is a function of all parameters

$$L(\theta, \phi|y) = \prod_{i=1}^{n} f(y_i, \theta_i, \phi)$$

- The log-likelihood function is often used, which is defined as

$$l(\theta, \phi|y) = \log L(\theta, \phi|y) = \sum_{i=1}^{n} \log f(y_i, \theta_i, \phi)$$

1.4.1 Poisson Example

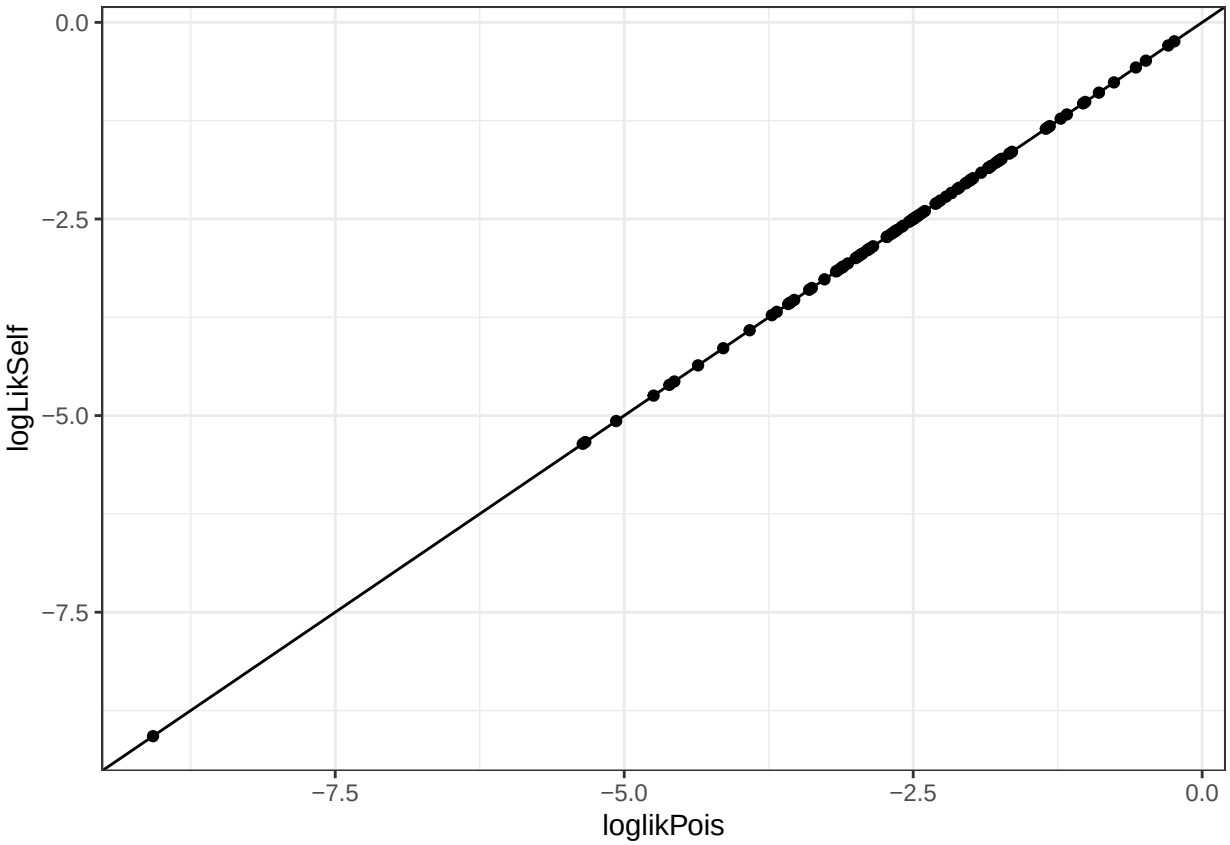
The log-likelihood for our simulated dataset given the real model parameters is:

For one observation:

$$l(\mu_i|y_i) = y_i \log \mu_i - \mu_i - \log y_i!$$

- Verify in R.

```
muTrue <- exp(etaTrue)
loglikPois <- dpois(y,muTrue,log = TRUE)
logLikSelf <- y*log(muTrue) - muTrue - lfactorial(y)
qplot(loglikPois,logLikSelf) + theme_bw() + geom_abline(intercept = 0, slope=1)
```



The log-likelihood can also be written in terms of the canonical model parameters θ

$$l(\mu_i|y_i) = y_i\theta_i - e^{\theta_i} - \log y_i!$$

- Note that $\theta_i = \eta_i$. The canonical parameter for the poisson equals the linear predictor!

$$\theta_i = \eta_i = \mathbf{x}_i^t\beta$$

Log-likelihood for all observations, given that they are independent:

$$l(\mu|\mathbf{y}) = \sum_{i=1}^{n} \{y_i\theta_i - e^{\theta_i} - \log y_i!\}$$

- Calculate in R:

```
dpois(y,lambda = muTrue, log = TRUE) %>%
  sum()
```

```
## [1] -260.485
```

1.4.2 Properties of the Log-Likelihood

$$l(\theta_i, \phi|y_i) = \left\{ \frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi) \right\}$$

- $E[y_i] = \mu_i = b'(\theta_i)$
- $\text{var}[y_i] = b''(\theta_i)a(\phi)$

Note that,

- Variance $\text{var}[y_i]$ depends on mean!
- Often there is no dispersion parameter e.g. Bernouilli: $\text{var}[y_i] = \mu_i(1-\mu_i)$,

1.4.3 Poisson Example

- Canonical model parameter $\theta_i = \log \mu_i$.
- $b(\theta_i) = \exp(\theta_i)$
- $c(y_i, \phi) = -\log(y_i!)$
- $\phi = 1$
- $a(\phi) = 1$
- $\mu_i = \frac{\partial b(\theta_i)}{\partial \theta_i} = \frac{\partial \exp(\theta_i)}{\partial \theta_i} = \exp(\theta_i)$
- $\text{Var}[y_i] = a(\phi)\frac{\partial^2 b(\theta_i)}{(\partial \theta_i)^2} = \frac{\partial^2 \exp(\theta_i)}{\partial \theta_i^2} = \exp(\theta_i)$.
- Mean is equal to variance for Poisson!

1.5 Parameter Estimation: Maximum Likelihood

Choose the parameters β so that the likelihood to observe the sample under the statistical model is maximized.

It is easier and equivalent to maximize the log-likihood

$$\text{argmax}_\beta l(\mu|\mathbf{y})$$

$$\frac{\partial l(\mu|\mathbf{y})}{\partial \beta} = 0$$

$\frac{\partial l(\mu|\mathbf{y})}{\partial \beta}$ is also referred to as the score function.

1.5.1 Score function

$$S_i(\theta_i) = \frac{\partial l(\theta_i, \phi|y_i)}{\partial \theta_i}$$

$$S_i(\theta_i) = \frac{\partial l(\theta_i, \phi|y_i)}{\partial \theta_i} = \frac{y_i - \mu_i}{a(\phi)}$$

when canonical link function is used:

- $\mu_i = b'(\theta_i)$

Regression (chain rule and $i = 1, \ldots, n$ i.i.d observations)

$$\begin{aligned} S(\beta) &= \frac{\partial \sum_{i=1}^{n} \left\{ \frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi) \right\}}{\partial \beta} \\ &= \sum_{i=1}^{n} \frac{\partial \left\{ \frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi) \right\}}{\partial \theta} \frac{\partial \theta}{\partial \mu} \frac{\partial \mu}{\partial \eta} \frac{\partial \eta}{\partial \beta} \\ &= \sum_{i=1}^{n} \frac{y_i - \mu_i}{a(\phi)} \frac{1}{b''(\theta)} \frac{\partial \mu}{\partial \eta} \mathbf{x}^t \\ &= \mathbf{X}^T \mathbf{A} (\mathbf{y} - \mu) \end{aligned}$$

- A is a diagonal matrix: $a_{ii} = \left(\text{var}[y_i] \frac{\partial \eta_i}{\partial \mu_i} \right)^{-1}$, $y = [y_1, \ldots, y_n]^T$, $\mu = [\mu_1, \ldots, \mu_n]^T$

1.5.1.1 Poisson Example

For poisson data: $a(\phi)b''(\theta) = \mu$ and $\frac{\partial \mu}{\partial \eta} = \mu$. So $\mathbf{A} = \mathbf{I}$ and

$$S(\beta) = \mathbf{X}^T \{\mathbf{Y} - \mu\}$$

1.5.2 Solve Score Equations

Find parameter estimator $\hat{\beta}$ so that

$$S(\beta) = \mathbf{0}$$
$$\mathbf{X}^T \mathbf{A} (\mathbf{y} - \mu) = \mathbf{0}$$

Problem! Non linear in β due to

- link function: $\mu = h^{-1}(\eta)$
- $a_{ii} = \left(\text{var}[y_i] \frac{\partial \eta_i}{\partial \mu_i} \right)^{-1}$

$\rightarrow$ Find roots of score equation by using Newton-Raphson method.

1.5.3 Newton-Raphson

Newton Raphson algorithm to find the root of the score function.

1. Choose initial parameter estimate $\beta^k = \beta^0$
2. Calculate score $S(\beta)|_{\beta=\beta^k}$
3. Calculate derivative of the function for which you want to calculate the roots
4. Walk along first derivative until line (plane) of the derivative crosses zero
5. Update the betas β^{k+1}
6. Iterate from step 2 - 5 until convergence.

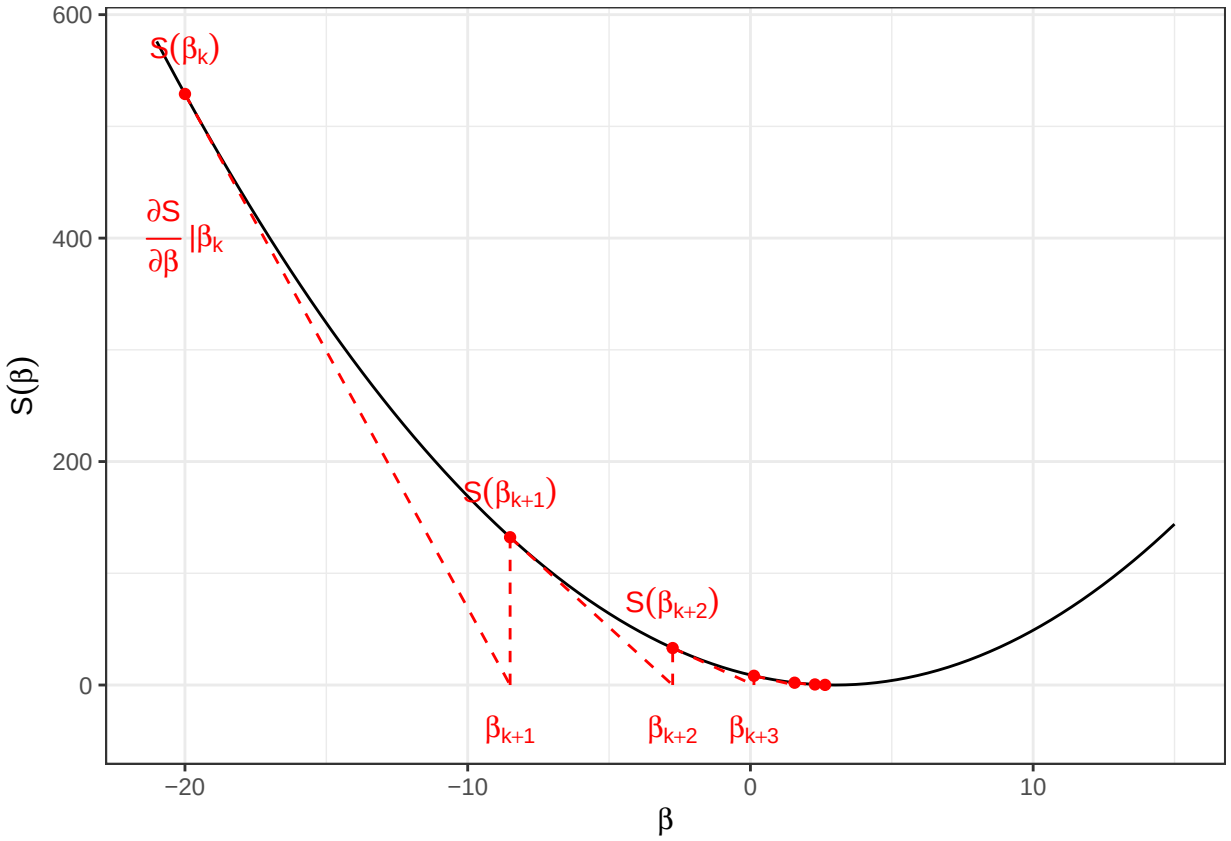


Figure 1: Newton-Raphson algorithm: The algorithm starts at an initial guess bk for the root of the score function S. It walks along the tangent line of the score function in S(bk) to move into the direction where the score function becomes zero (and where the log-likelihood function is maximal). The new estimate of the root is taken at bk+1 where the tangent line becomes zero. It then calculates the score for the updated parameter estimator and the whole process is repeated until the root is found.

1.5.3.1 Derivative of Score

We have to implement an interative algorithm for optimisation. To make things tractable we will act as if $\mathbf{A}$ is known and fix it using the current values of β^k. Note, that for Poisson regression $\mathbf{A} = \mathbf{I}$.

$$\begin{aligned}
\frac{\partial S(\beta)}{\partial \beta} &= \frac{\mathbf{X}^T\mathbf{A}\left\{\mathbf{Y}-\mu\right\}}{\partial \beta} \\
&= -\mathbf{X}^T\mathbf{A}\begin{bmatrix} \frac{\partial \mu_1}{\partial \eta_1} & 0 & ... & 0 \\ 0 & \frac{\partial \mu_2}{\partial \eta_2} & ... & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & \frac{\partial \mu_n}{\partial \eta_n} \end{bmatrix}\frac{\partial \eta}{\partial \beta} \\
&= -\mathbf{X}^T\mathbf{W}\mathbf{X}
\end{aligned}$$

1.5.3.2 Define equation of Tangent Line (Plane)

- We know two points of the tangent plane $(\beta^k, S(\beta^k))$ and $(\beta^{k+1}, 0)$
- We know the direction of the plane $S'(\beta) = \frac{\partial S(\beta)}{\partial \beta}$
- Equation of Plane:

$$S(\beta) = \alpha_0 + S'|_{\beta^k}\beta$$

- Get β_{k+1}

$$\begin{aligned}
\mathbf{0} &= \alpha_0 + S'|_{\beta^k}\beta^{k+1} \\
\beta^{k+1} &= -\left(S'|_{\beta^k}\right)^{-1}\alpha_0
\end{aligned}$$

- Get α_0

$$\begin{aligned}
S(\beta^k) &= \alpha_0 + S'|_{\beta^k}\beta^k \\
\alpha_0 &= -S'|_{\beta^k}\beta^k + S(\beta^k)
\end{aligned}$$

- Get β_{k+1}

$$\begin{aligned}
\beta^{k+1} &= \beta^k - \left(S'|_{\beta^k}\right)^{-1}S(\beta^k) \\
\beta^{k+1} &= \beta^k + \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}S(\beta^k)
\end{aligned}$$

With $J(\beta) = I(\beta) = \mathbf{X}^T\mathbf{W}\mathbf{X}$ the Fisher information matrix.

1.5.4 Fisher Scoring

Because we use the canonical model parameters the observed Fisher information matrix equals the expected Fisher information matrix $J(\beta) = I(\beta)$. Indeed, the observed Fisher information matrix is not depending on the observations, but only on the design and the variance of the data (via the weights).

Hence, Newton-Raphson is equivalent to Fisher scoring when the canonical link function is used.

Note, that the Fisher matrix, minus second derivative (or hessian) of the likelihood to the model parameters, is also the inverse of the variance covariance matrix of the model parameters. It is thus related to the precision.

1.5.5 Iteratively Reweighted Least Squares (IRLS).

We can rewrite Newton Raphson or Fisher scoring as IRLS.

$$\begin{aligned}
\beta^{k+1} &= \beta^k + \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1} S(\beta^k) \\
\beta^{k+1} &= \beta^k + \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{A}\left(\mathbf{Y}-\mu\right) \\
\beta^{k+1} &= \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{W}\mathbf{X}\beta^k + \left(X^T\mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{W}\frac{\partial\eta}{\partial\mu}\left(\mathbf{Y}-\mu\right) \\
\beta^{k+1} &= \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{W}\left[\mathbf{X}\beta^k + \frac{\partial\eta}{\partial\mu}\left(\mathbf{Y}-\mu\right)\right] \\
\beta^{k+1} &= \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{W}\mathbf{z}
\end{aligned}$$

with $\mathbf{z} = \left[\mathbf{X}\beta^k + \frac{\partial\eta}{\partial\mu}\left(\mathbf{Y}-\mu\right)\right]$

So we can fit the model by performing iterative regressions of the pseudo data $\mathbf{z}$ on $\mathbf{X}$. In each iteration we will update $\mathbf{z}$, the weights $\mathbf{W}$ and the model parameters.

For Poisson data

- $\frac{\partial\eta}{\partial\mu} = \frac{\partial\log\mu}{\partial\mu} = \frac{1}{\mu} = \exp(-\eta)$
- $\mathbf{W} = \mathbf{A}\frac{\partial\mu}{\partial\eta}$ is a diagonal matrix with $[\frac{\partial\mu_i}{\partial\eta_i}]_{ii} = [\mu_i]_{ii} = [\exp(\eta_i)]_{ii}$ on its diagonal elements.

1.5.6 Variance-Covariance Matrix of Mean Model Parameters?

In the IRWLS algorithm, the data is weighted according to the variance of $\mathbf{Y}$. We correct for the fact that the data are heteroscedastic.

Count data have a mean variance relation (e.g. in Poisson case $\mathrm{E}[Y] = \mathrm{var}[Y] = \mu$). The IRWLS also corrects for the scale parameter ϕ in $\mathbf{W}$. (Note that the scale parameter for Poisson is $\phi = 1$).

So IRWLS the variance-covariance matrix for the model parameter equals

$$_{\hat{\beta}} = \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}.$$

Note, that the Fisher Information Matrix equals the inverse of the variance-covariance matrix of the experiment. The larger the Fisher Information Matrix the more information we have on the experiment to estimate the model parameters. FIM ↑, precision ↑, SE ↓

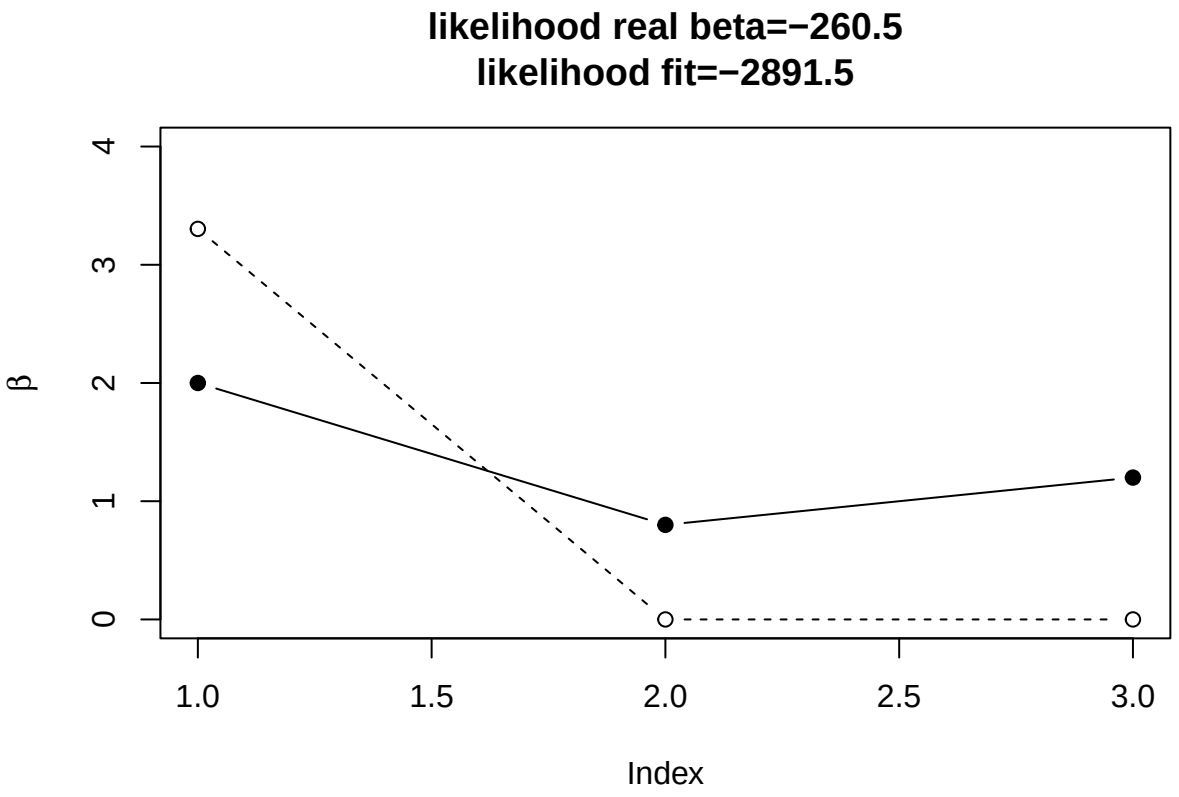
1.6 Poisson Example

1.6.1 Initial Estimate

This is a very poor initial estimate used to illustrate the algorithm. Otherwise convergence for this simple example is way too quick

```
iteration=0
betas<-c(log(mean(y)),0,0)
plot(betasTrue,
     ylab=expression(beta),
```

```
     ylim=c(0,4),
     pch=19,
     type="b",
     main=paste0("likelihood real beta=",
                 round(sum(dpois(y,exp(etaTrue),log=TRUE)),1),"\nlikelihood fit=", round(sum(dpois(y,exp
)
lines(betas,type="b",lty=2 )
```



1.6.2 Iteratively Reweighted Least Squares

1.6.2.1 Pseudo Data

$$z_i = \eta_i + \frac{\partial \eta_i}{\partial \mu_i}(y_i - \mu_i)$$

$$z_i = \eta_i + e^{-\eta_i} y_i - 1$$

1.6.2.2 Weight Matrix?

$$[w_{ii}] = var_{y_i}^{-1} \left(\frac{\partial \mu}{\partial \eta}\right)^2$$

$$[w_{ii}] = e^{\eta_i}$$

1.6.2.3 Run Update Step Multiple Times

First 3 times (colors are black 0, red iteration 1, green iteration 2, blue iteration 3)

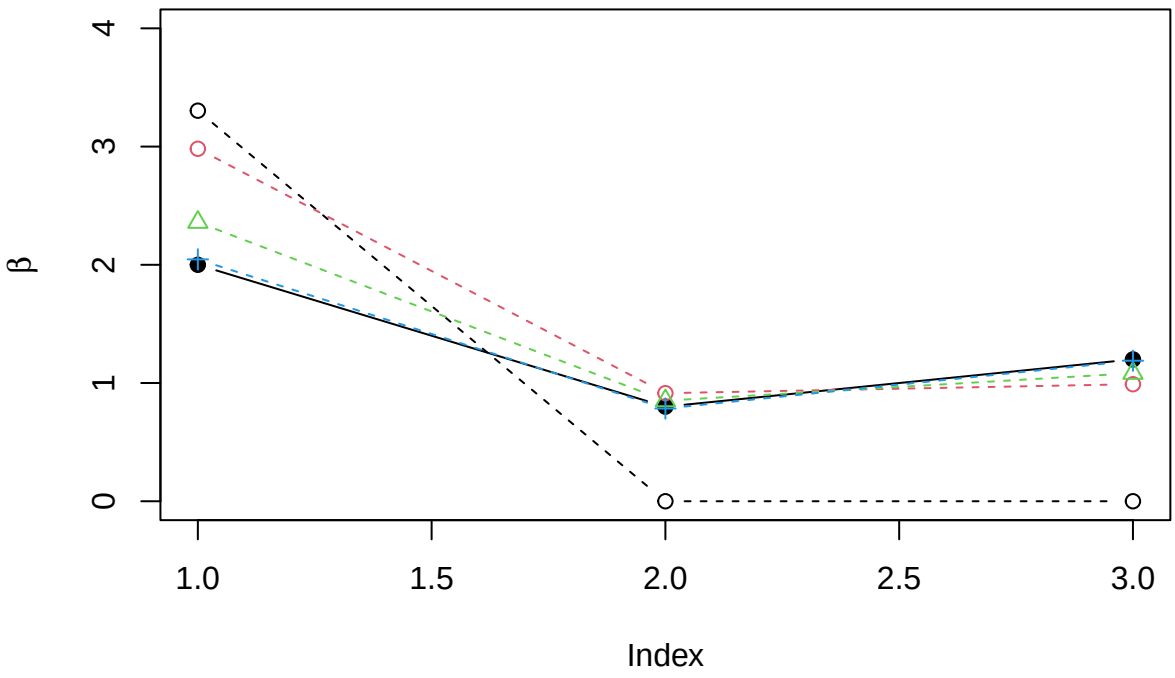
```
plot(betasTrue,ylab=expression(beta),ylim=c(0,4),pch=19,type="b")
lines(betas,type="b",lty=2)

cat("\nlikelihood TRUE=", round(sum(dpois(y,exp(xhlp%*%betasTrue),log=TRUE)),1))
```

```
##
## likelihood TRUE= -260.5
```

```
cat("\nlikelihood initial fit=", round(sum(dpois(y,exp(xhlp%*%betas),log=TRUE)),1))
```

```
##
## likelihood initial fit= -2891.5
```

```
#Calculate current eta
eta<-xhlp%*%betas

iteration=0
for (i in 1:3)
{
#start IRLS UPDATE STEP
iteration=iteration+1
#calculate pseudo data based on current betas
z=eta+exp(-eta)*(y-exp(eta))
#calculate new weights: diagonal elements
w<-c(exp(eta))

#update betas
lmUpdate<-lm(z~-1+xhlp,weight=w)
#eta<-xhlp%*%betas
eta<-lmUpdate$fitted
betas<-lmUpdate$coef
lines(betas,type="b",col=iteration+1,pch=iteration,lty=2)
cat("\nlikelihood current fit=", round(sum(dpois(y,exp(xhlp%*%betas),log=TRUE)),1))
}
```

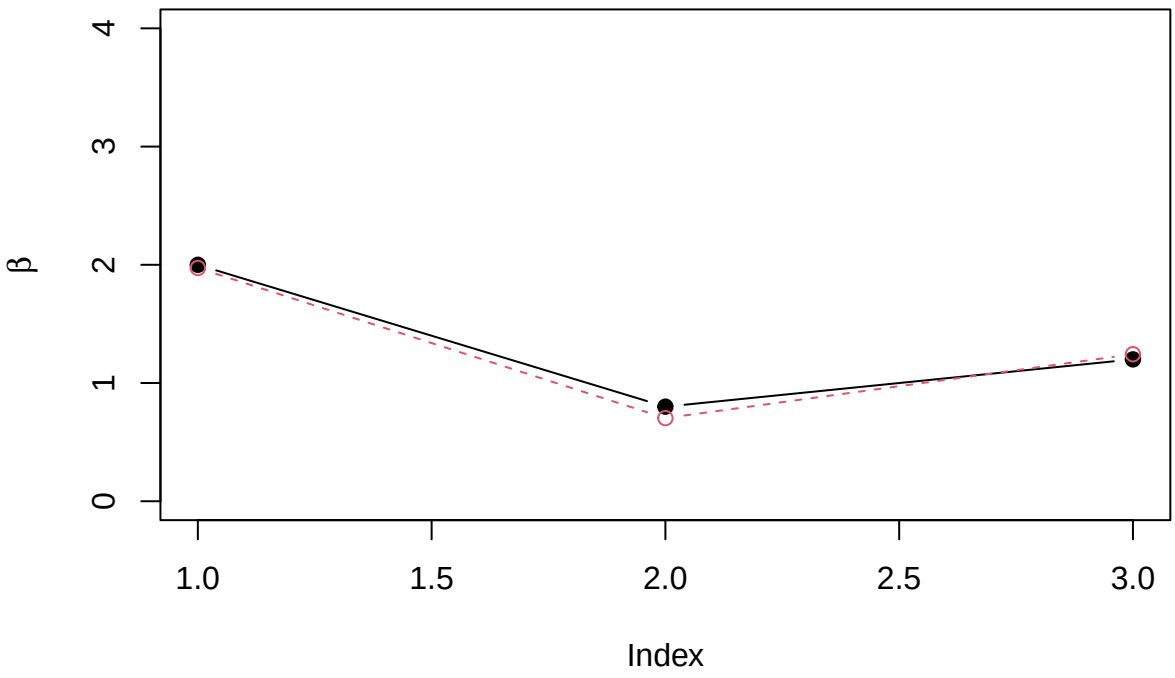



```
##
## likelihood current fit= -1883.9
## likelihood current fit= -431.8
## likelihood current fit= -261.7
```

1.7 Comparison with GLM Function

1.7.1 Smarter Initialisation

```
z<-log(y+.5)
betas<-lm(z~-1+xhlp)$coef
plot(betasTrue,ylab=expression(beta),ylim=c(0,4),pch=19,type="b")
lines(betas,col=2,type="b",lty=2)
```




```
#calculate current eta
eta<-xhlp%*%betas

cat("\nlikelihood TRUE=", round(sum(dpois(y,exp(xhlp%*%betasTrue),log=TRUE)),1))
```

```
##
## likelihood TRUE= -260.5
```

```
cat("\nlikelihood initial fit=", round(sum(dpois(y,exp(xhlp%*%betas),log=TRUE)),1))
```

```
##
## likelihood initial fit= -263.8
```

1.7.2 Evaluation Stopping Criterion

- Residual deviance: Is 2 log of LR between best possible fit and current fit

$$LR = \frac{L_{\text{best}}}{L_{\text{current}}}$$

$$D = 2(\log L_{\text{best}} - \log L_{\text{current}})$$

$$D = 2(l_{\text{best}} - l_{\text{current}})$$

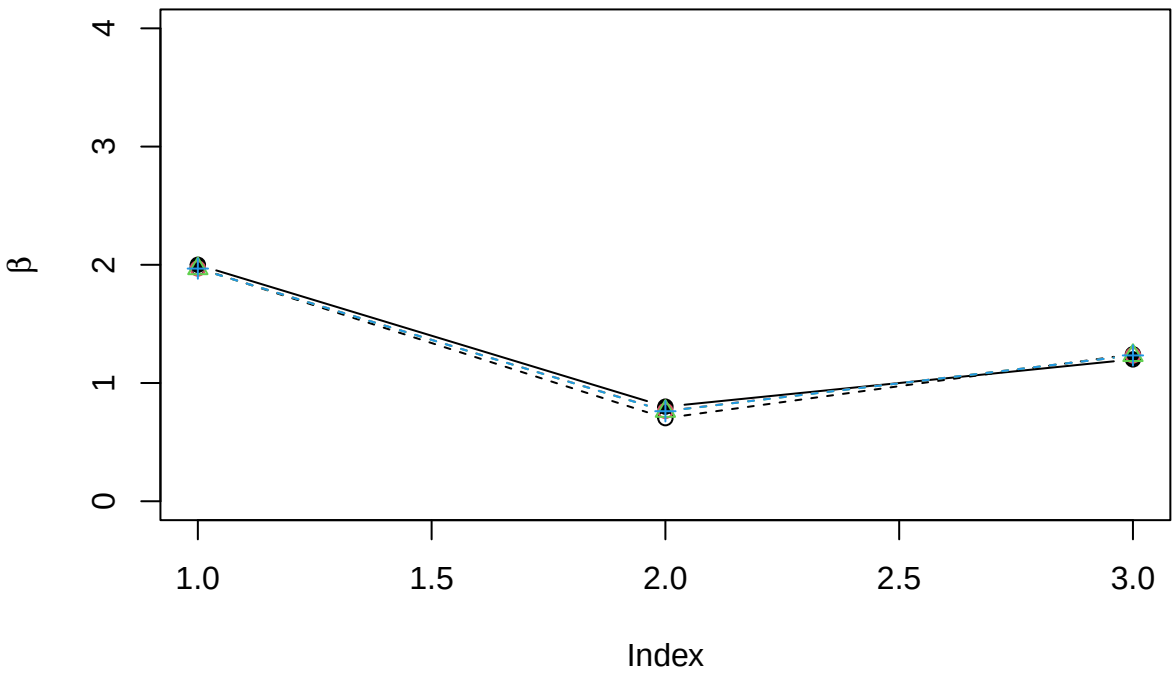
- Best fit: $\mu = y$

- Optimal poisson:
$$l_{\text{best}} = \sum [y_i \log(y_i) - y_i - \log(y_i!)]$$
- Current fit
$$l_{\text{current}} = \sum [y_i \eta_i - e^{\eta_i} - log(y_i!)]$$
- Deviance D:
$$D = 2 \sum [y_i log(y_i) - y_i \eta_i - (y_i - e^{\eta_i})]$$
- Problem to calculate it if y=0 but by apply l'Hopital's rule we know
$$\lim_{y_i \to 0} y_i \log(y_i) = 0$$

```
ylogy<-function(y)
{
return(ifelse(y==0,rep(0,length(y)),y*log(y)))
}

deviance<-2*sum(ylogy(y)-y*eta-(y-exp(eta)))

devianceOld<-1e30
```

1.7.3 Run Update Step until Convergence

```
plot(betasTrue,ylab=expression(beta),ylim=c(0,4),pch=19,type="b")
lines(betas,type="b",lty=2)

tol<-1e-6
iteration=0
while(((devianceOld-deviance)/devianceOld)>tol)
{
#start IRLS UPDATE STEP
iteration=iteration+1
#calculate pseudo data based on current betas
z=eta+exp(-eta)*(y-exp(eta))
#calculate new weights: diagonal elements
w<-c(exp(eta))

#update betas
lmUpdate<-lm(z~-1+xhlp,weight=w)
#eta<-xhlp%*%betas
eta<-lmUpdate$fitted
betas<-lmUpdate$coef
lines(betas,type="b",col=iteration+1,pch=iteration,lty=2)

#criterion for convergence
devianceOld<-deviance
deviance<-2*sum(ylogy(y)-y*eta-(y-exp(eta)))
cat("iteration",iteration,"Deviance Old",devianceOld,"Deviance", deviance,"\n")
}
```




```
## iteration 1 Deviance Old 129.1127 Deviance 114.3748
## iteration 2 Deviance Old 114.3748 Deviance 114.3374
## iteration 3 Deviance Old 114.3374 Deviance 114.3374
```

1.7.4 Variance β?

$$\Sigma_\beta = (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}$$

```
varBeta=solve(t(xhlp)%*%diag(w)%*%xhlp)
```

1.7.5 Comparison with GLM fit

Use -1 because intercept is already in xhlp

```
glmfit=glm(y~-1+xhlp,family=poisson)
comp=data.frame(glmfit=c(glmfit$deviance,glmfit$coef,summary(glmfit)$coef[,2]),ourFit=c(deviance,betas,s
row.names(comp)=c("deviance",paste("beta",1:3,sep=""),paste("se",1:3,sep=""))
comp
```

	glmfit	ourFit
deviance	114.3373950	114.3373950
beta1	1.9680569	1.9680569

	glmfit	ourFit
beta2	0.7613664	0.7613664
beta3	1.2333003	1.2333003
se1	0.0381438	0.0381438
se2	0.0189121	0.0189120
se3	0.0255667	0.0255666

1.8 Hypothesis testing: Large sample theory

1.8.1 Wald test

- Follows immediately from the information matrix for generalized linear models

$$I(\beta) = \mathbf{X}^T\mathbf{W}\mathbf{X}$$

so large sample distribution of the maximum likelihood estimator $\hat{\beta}$ is multivariate normal

$$\hat{\beta} \sim MVN\left[\beta, \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\right]$$

We can perform a Wald Test for a single model parameter

$$W = \frac{\hat{\beta}_m}{\hat{\text{se}}_{\hat{\beta}_m}} \approx N(0,1)|H_0$$

to test for

$$H_0 : \beta_p = 0 \leftrightarrow H_1 : \beta_p \neq 0$$

Again, we can also assess contrasts! Indeed, linear combinations of model parameter estimators also follow a normal distribution.

$$\mathbf{L}^T\hat{\beta} \sim N\left[\mathbf{L}^T\beta, \mathbf{L}^T\hat{\Sigma}_{\hat{\beta}}\mathbf{L}\right]$$

With $\mathbf{L}$ a vector for a single contrast.

$$W = \frac{\mathbf{L}^T\hat{\beta}}{\hat{\text{se}}_{\mathbf{L}^T\hat{\beta}}} \approx N(0,1)|H_0$$

testing for

$$H_0 : \mathbf{L}^T\beta = 0 \leftrightarrow H_1 : \mathbf{L}^T\beta \neq 0$$

We can also test for multiple contrasts simultaneously, e.g. by assuming that multiple model parameters are zero. Suppose that $\mathbf{L}$ is the contrast matrix that corresponds testing for c model parameters, simultaneously. Then

$$\mathbf{L}^T\hat{\beta} \sim MVN\left[\mathbf{L}^T\beta, \mathbf{L}^T\hat{\Sigma}_{\hat{\beta}}\mathbf{L}\right]$$

and

$$W = \mathbf{L}^T\hat{\beta}\left(\mathbf{L}^T\hat{\Sigma}_{\hat{\beta}}\mathbf{L}\right)^{-1}\hat{\beta}\mathbf{L} \sim \chi^2_c|H_0$$

to test for

$$H_0 : \mathbf{L}^T\beta = \mathbf{0} \leftrightarrow H_1 : \mathbf{L}^T\beta = \mathbf{0} \neq 0$$

In general, when we test for $c \geq 1$ contrasts, then the test statistic $W \sim \chi^2_r | H_0$, with r the rank of the contrast matrix.

1.8.2 Likelihood ratio test

The likelihood ratio test (LRT) measures the discrepancy in log-likelihood between our current model (sometimes also referred to as full model) and a reduced model (sometimes also referred to as null or alternative model).

The reduced model must be nested in (and therefore of lower dimension as compared to) the full model.

While adding more covariates will always explain more variability in our response variable, the LRT tests whether this is actually significant.

For example, in the example of gene differential expression between healthy versus tumoral tissue, the full model could be a GLM where the mean is modeled according to an intercept and a tissue indicator variable (healthy / tumor), while the alternative model could be a GLM with just an intercept. Indeed, if the gene is similarly expressed between healthy and tumor tissue, the log-likelihood of the alternative model will decrease only a little as compared to the full model.

As the name suggests, the likelihood ratio test assesses whether the ratio of the log-likelihoods provides sufficient evidence for a worse fit of the alternative versus full model

$$\lambda = 2\left[l(\hat{\beta}_{\text{full}}) - 2l(\hat{\beta}_0)\right]$$

`Asymptotically, under the null hypothesis it can be shown that`

$$L \sim \chi^2_c | H_0,$$

with c the number of parameters dropped in the alternative model versus the full model.

Let $\mathbf{C}$ denote the $c \times p$ contrast matrix denoting the contrast for the parameters being dropped, the null and alternative hypothesis are as in the Wald test setting:

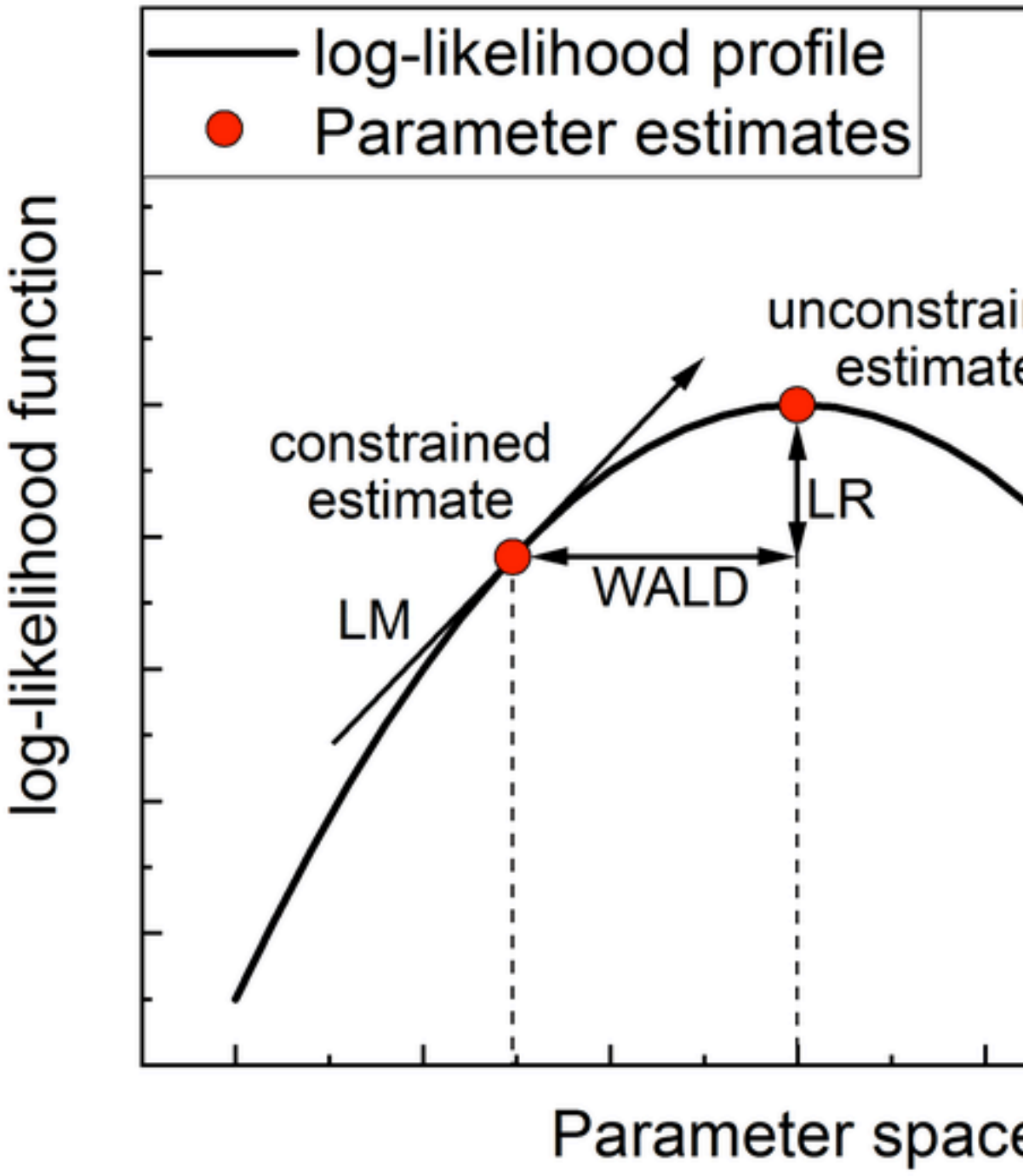
$$H_0 : \mathbf{C}\beta = 0$$

$$H_1 : \mathbf{C}\beta \neq 0$$

- It is important to keep in mind that standard statistical inference theory in GLMs works **asymptotically in terms of the sample size**.

Thus we need many data points in order for the theory to hold in practice. In order for the p-values to be correct, our parametric (distributional) assumptions as well as the independence assumption, must also hold.

- In bulk RNA-seq, we are often working with a limited number of samples and so we typically do not expect asymptotic theory to hold yet. In single-cell RNA-seq, we often perform several preprocessing steps before calculating p-values for each gene and so we may be 'using the data multiple times'. Rather than attaching strong probabilistic interpretations to the p-values, we therefore advice to view the p-values simply as useful numerical summaries for ranking the genes for further inspection in genomics applications.


log-likelihood profile
Parameter estimates
log-likelihood function
unconstrai
estimate
constrained
estimate
LR
WALD
LM
Parameter space

2 EdgeR

McCarthy and Smyth, 2012

MATERIALS AND METHODS

Biological coefficient of variation

RNA-Seq profiles are formed from n RNA samples. Let π_{gi} be the fraction of all cDNA fragments in the i-th sample that originate from gene g. Let G denote the total number of genes, so $\sum_{g=1}^{G} \pi_{gi} = 1$ for each sample. Let $\sqrt{\phi_g}$ denote the coefficient of variation (CV) (standard deviation divided by mean) of π_{gi} between the replicates i. We denote the total number of mapped reads in library i by N_i and the number that map to the g-th gene by y_{gi}. Then

$$E(y_{gi}) = \mu_{gi} = N_i \pi_{gi}.$$

Assuming that the count y_{gi} follows a Poisson distribution for repeated sequencing runs of the same RNA sample, a well known formula for the variance of a mixture distribution implies:

$$\mathrm{var}(y_{gi}) = E_\pi[\mathrm{var}(y|\pi)] + \mathrm{var}_\pi[E(y|\pi)] = \mu_{gi} + \phi_g \mu_{gi}^2.$$

Dividing both sides by μ_{gi}^2 gives

$$\mathrm{CV}^2(y_{gi}) = 1/\mu_{gi} + \phi_g.$$

The first term $1/\mu_{gi}$ is the squared CV for the Poisson distribution and the second is the squared CV of the unobserved expression values. The total CV^2 therefore is the technical CV^2 with which π_{gi} is measured plus the biological CV^2 of the true π_{gi}. In this article, we call ϕ_g the dispersion and $\sqrt{\phi_g}$ the biological CV although, strictly speaking, it captures all sources of the inter-library variation between replicates, including perhaps contributions from technical causes such as library preparation as well as true biological variation between samples.

GLMs

GLMs are an extension of classical linear models to non-normally distributed response data (42,43). GLMs specify probability distributions according to their mean–variance relationship, for example the quadratic mean–variance relationship specified above for read counts. Assuming that an estimate is available for ϕ_g, so the variance can be evaluated for any value of μ_{gi}, GLM theory can be used to fit a log-linear model

$$\log \mu_{gi} = \mathbf{x}_i^T \beta_g + \log N_i$$

for each gene (32,41). Here $\mathbf{x}_i$ is a vector of covariates that specifies the treatment conditions applied to RNA sample i, and β_g is a vector of regression coefficients by which the covariate effects are mediated for gene g. The quadratic variance function specifies the negative binomial GLM distributional family. The use of the negative binomial distribution is equivalent to treating the π_{gi} as gamma distributed.

Fitting the GLMs

The derivative of the log-likelihood with respect to the coefficients β_g is $X^T \mathbf{z}_g$, where X is the design matrix with columns $\mathbf{x}_i$ and $z_{gi} = (y_{gi} - \mu_{gi})/(1 + \phi_g \mu_{gi})$. The Fisher information matrix for the coefficients can be written as $\mathcal{I}_g = X^T W_g X$, where W_g is the diagonal matrix of working weights from standard GLM theory (43). The Fisher scoring iteration to find the maximum likelihood estimate of β_g is therefore $\beta_g^{\mathrm{new}} = \beta_g^{\mathrm{old}} + \delta$ with $\delta = (X^T W_g X)^{-1} X^T \mathbf{z}_g$. This iteration usually produces an increase in the likelihood function, but the likelihood can also decrease representing divergence from the required solution. On the other hand, there always exists a stepsize modifier α with $0 < \alpha < 1$ such that $\beta_g^{\mathrm{new}} = \beta_g^{\mathrm{old}} + \alpha\delta$ produces an increase in the likelihood. Choosing α so that this is so at each iteration is known as a line search strategy (44,45).

Fisher's scoring iteration can be viewed as an approximate Newton-Raphson algorithm, with the Fisher information matrix approximating the second derivative matrix. The line search strategy may be used with any approximation to the second derivative matrix that is positive definite. Our implemention uses a computationally convenient approximation. Without loss of generality, the linear model can be parametrized so that $X^T X = I$. If this is done, and if the μ_{gi} also happen to be constant over i for a given gene g, then the information matrix simpifies considerably to $\mu_g/(1 + \phi_g \mu_g)$ times the identity matrix I. Taking this as the approximation to the information matrix, the Fisher scoring step with line search modification becomes simply $\delta = \alpha X^T \mathbf{z}_g$, where the multiplier $\mu_g/(1 + \phi_g \mu_g)$ has been absorbed into the stepsize factor α. In this formulation, α is no longer constrained to be less than one. In our implementation, each gene has its own stepsize α that is increased or decreased as the iteration proceeds.

Cox–Reid adjusted profile likelihood

The adjusted profile likelihood (APL) for ϕ_g is the penalized log-likelihood

$$\mathrm{APL}_g(\phi_g) = \ell(\phi_g; \mathbf{y}_g, \hat{\beta}_g) - \frac{1}{2} \log\det \mathcal{I}_g.$$

where $\mathbf{y}_g$ is the vector of counts for gene g, $\hat{\beta}_g$ is the estimated coefficient vector, $\ell()$ is the log-likelihood function and $\mathcal{I}_g$ is the Fisher information matrix. The Cholesky decomposition (46) provides a numerically stable and efficient algorithm for computing the determinant of the information matrix. Specifically, logdet $\mathcal{I}_g$ is the sum of the logarithms of the diagonal elements of the Cholesky factor R, where $\mathcal{I}_g = R^T R$ and R is upper triangular. The matrix R can be obtained as a by product of the QR-decomposition used in standard linear model fitting. In our implementation, the Cholesky calculations are carried out in a vectorized fashion, computed for all genes in parallel.

Simulations

Artificial data sets were generated with negative binomial distributed counts for a fixed total number of 10 000 genes. The expected count size varied between genes according to a gamma distribution with shape parameter 0.5, an *ad hoc* choice that happened to mimic the size distribution of the carcinoma data. The average dispersion was set to 0.16 (BCV = 0.4). In one simulation, all genes had the same

Although the pseudo-Newton algorithm requires slightly more iterations on average than true Newton-Raphson or the customary Fisher scoring algorithm for GLMs, the pseudo-Newton algorithm remains competitive in conjunction with our line-search strategy, and the computational gains that arise from the simplification are enormous. The algorithm is implemented in R in such a way that the iteration is progressed for all genes in parallel rather than for one gene at a time. Our pure R implementation fits GLMs to most RNA-Seq data sets in a few seconds, whereas genewise calls to the glm() function in R typically require minutes at least, and indeed may fail entirely due to iterative divergence for one or more genes.

Hypothesis tests

Our software allows users to test the significance of any coefficient in the linear model, or of any contrast or linear combination of the coefficients in the linear model. Genewise tests are conducted by computing likelihood-ratio statistics to compare the null hypothesis that the coefficient or contrast is equal to zero against the two-sided alternative that it is different from zero. The log-likelihood-ratio statistics are asymptotically chi square distributed under the null hypothesis that the coefficient or contrast is zero. Simulations show that the likelihood ratio tests hold their size relatively well and generally give a good approximation to the exact test (23) when the latter is available (data not shown). Any multiple testing adjustment method provided by the p.adjust function in R can be used. By default, *P*-values are adjusted to control the false discovery rate by the method of Benjamini and Hochberg (47).

Estimation of biological CV

The remaining issue is to obtain a reliable estimate of the BCV for each gene. An estimator that is approximately unbiased and performs well in small samples is required. Maximum likelihood estimation of the BCV would underestimate the BCV, because of the need to estimate the coefficients in the log-linear model from the same data. Our earlier work used exact conditional likelihood to estimate the BCV (22,23). This approach has excellent performance, but does not easily generalize to GLMs. Instead we use an approximate conditional likelihood approach known as APL (48). APL is a form of penalized likelihood. Again, we have implemented the APL computation in a vectorized and computationally efficient manner, rather than computing quantities gene by gene.

Estimating common dispersion

Estimating the BCV for each gene individually should not be considered unless a large number of biological replicates are available. When less replication is available, sharing information between genes is essential for reliable inference. Regardless of the amount of replication, appropriate information sharing methods should result in some benefits.

Let ϕ_g denote the squared BCV for gene g, which we call the *dispersion* of that gene. The dispersion is the coefficient of the quadratic term in the variance function. The simplest method of sharing information between genes is to assume that all genes share the same dispersion, so that $\phi_g = \phi$ (23). The common dispersion may be estimated by maximizing the shared likelihood function

$$\mathrm{APL}_S(\phi) = \frac{1}{G}\sum_{g=1}^{G} \mathrm{APL}_g(\phi).$$

where APL_g is the adjusted profile likelihood for gene g ('Materials and Methods' section). This maximization can be accomplished numerically in a number of ways, for example by a derivative-free approximate Newton algorithm (49).

Estimating trended dispersion

A generalization of the common dispersion is to model the dispersion ϕ_g as a smooth function of the average read count of each gene (25). Our software offers a number of methods to do this. A simple non-parametric method is to divide the genes into bins by average read count, estimate the common dispersion in each bin, then to fit a loess or spline curve through these bin-wise dispersions. A more sophisticated method is locally weighted APL. In this approach, each ϕ_g is estimated by making a local shared log-likelihood, which is a weighted average of the APLs for gene g and its neighbouring genes by average read count.

Estimating genewise dispersions

In real scientific applications, it is more likely that individual genes have individual BCVs depending on their genomic sequence, genomic length, expression level or biological function. We seek a compromise between entirely individual genewise dispersions ϕ_g and entirely shared values by extending the weighted likelihood empirical Bayes approach proposed by Robinson and Smyth (22). In this approach, ϕ_g is estimated by maximizing

$$\mathrm{APL}_g(\phi_g) + G_0\,\mathrm{APL}_{Sg}(\phi_g),$$

where G_0 is the weight given to the shared likelihood and $\mathrm{APL}_{Sg}(\phi_g)$ is the local shared log-likelihood. This weighted likelihood approach can be interpreted in empirical Bayes terms, with the shared likelihood as the prior distribution for ϕ_g and the weighted likelihood as the posterior. The prior distribution can be thought of as arising from prior observations on a set of G_0 genes. The number of prior genes G_0 therefore represents the weight assigned to the prior relative to the actual observed data for gene g. The optimal choice for G_0 depends on the variability of BCV between genes. Large values are best when the BCV is constant between genes. Smaller values are optimal when the BCVs vary considerably between genes. We have found that $G_0 = 20/\mathrm{df}$ gives good results over a wide range of real data sets, where df is the residual degrees of freedom for estimating the BCV. For multigroup experiments, df is the number of libraries minus the number of distinct treatment groups. The default setting implies that the prior has the weight of 20 degrees of freedom for estimating the BCV, regardless of

3 EdgeR - Quasi-Likelihood

Lund et al. 2012

For quasi-likelihood we do not specify the full distribution, only the first two moments: the mean and variance.

$$\begin{cases} E[y_{ig}|\mathbf{x}_{ig}] & = & \mu_{ig} \\ log(\mu_{ig}) & = & \eta_{ig} \\ \eta_{ig} & = & \mathbf{x}_i^T\beta + \log N_i \\ \text{Var}[y_{ig}|\mathbf{x}_{ig}] & = & \sigma_g^2\left(\mu_{ig} + \phi\mu_{ig}^2\right) \end{cases}$$

We will look-up the details in the paper.

4 Limma - Voom

Law et al. (2013). Genome Biology

- Count models vs transformation: Poisson counts, $\sqrt{(}y)$ stabilises the variance, insufficient for negative binomial. Log transformation: the transformed data are still heteroscedastic.→ limma-voom
- Use normalized log-cpm Limma pipeline for sequencing

Problem: counts have a mean variance relationship: heteroscedastic

How do we deal with heteroscedasticity in traditional linear models?

Two stage approach:

1. Stage I

- OLS
- Estimate variances at each data point
- Use variances as weights: $W = \text{diag}[1/\hat{\sigma}_i^2]$

2. Stage II WLS $\text{argmin}_\beta\{(\mathbf{y} - \mathbf{X}\beta)^T\mathbf{W}(\mathbf{y} - \mathbf{X}\beta)\}$

Port this idea to RNA-seq pipeline

Log-counts per million

We assume that an experiment has been conducted to generate a set of n RNA samples. Each RNA sample has been sequenced, and the sequence reads have been summarized by recording the number mapping to each gene. The RNA-seq data consist therefore of a matrix of read counts r_{gi}, for RNA samples $i = 1$ to n, and genes $g = 1$ to G. Write R_i for the total number of mapped reads for sample i, $R_i = \sum_{g=1}^{G} r_{gi}$. We define the log-counts per million (log-cpm) value for each count as

$$y_{ga} = \log_2 \left(\frac{r_{gi} + 0.5}{R_i + 1.0} \times 10^6 \right)$$

The counts are offset away from zero by 0.5 to avoid taking the log of zero, and to reduce the variability of log-cpm for low expression genes. The library size is offset by 1 to ensure that $(r_{gi} + 0.5)/(R_i + 1)$ is strictly less than 1 has well as strictly greater than zero.

Voom variance modelling

The above linear model is fitted, by ordinary least squares, to the log-cpm values y_{gi} for each gene. This yields regression coefficient estimates $\hat{\beta}^*_{gj}$, fitted values $\hat{\mu}_{gi} = x_i^T \hat{\beta}_g$ and residual standard deviations s_g.

Also computed is the average log-cpm $\bar{y}_g$ for each gene. The average log-cpm is converted to an average log-count value by

$$\tilde{r} = \bar{y}_g + \log_2(\tilde{R}) - \log_2(10^6)$$

where $\tilde{R}$ is the geometric mean of the library sizes plus one.

To obtain a smooth mean-variance trend, a loess curve is fitted to square-root standard deviations $s_g^{1/2}$ as a function of mean log-counts $\tilde{r}$ (Figure 2ab). Square-root standard deviations are used because they are roughly symmetrically distributed. The lowess curve [44] is statistically robust [45] and provides a trend line through the majority of the standard deviations. The lowess curve is used to define a piecewise linear function lo() by interpolating the curve between ordered values of $\tilde{r}$.

Next the fitted log-cpm values $\hat{\mu}_{gi}$ are converted to fitted counts by

$$\hat{\lambda}_{gi} = \hat{\mu}_{gi} + \log_2(R_i + 1) - \log_2(10^6).$$

The function value $\text{lo}(\hat{\lambda}_{ga})$ is then the predicted square-root standard deviation of y_{gi}.

Finally, the voom precision weights are the inverse variances $w_{gi} = \text{lo}(\hat{\lambda}_{gi})^{-4}$ (Figure 2c). The log-cpm values y_{gi} and associated weights w_{gj} are then input into the standard limma linear modeling and empirical Bayes differential expression analysis pipeline.

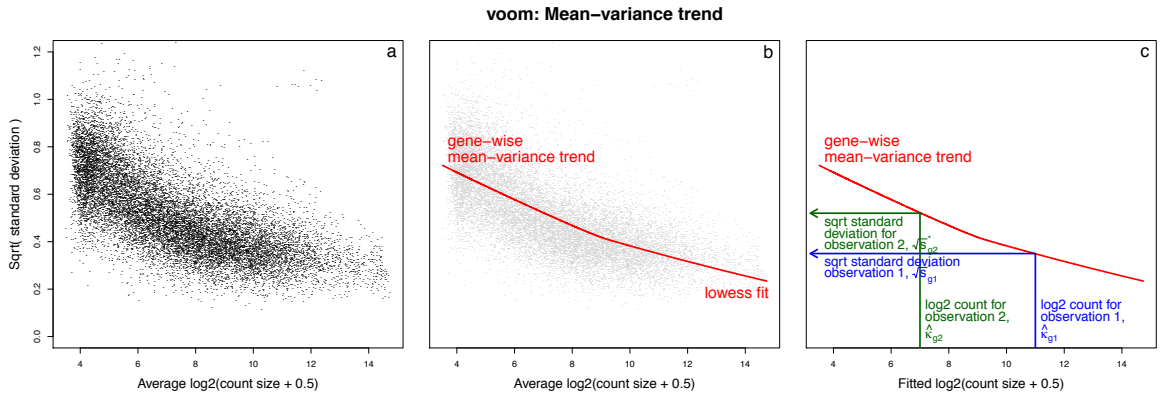


Figure 2: Voom mean-variance modelling. Panel (a), gene-wise square-root residual standard deviations are plotted against average log-count. Panel (b), a functional relationship between gene-wise means and variances is given by a robust lowess fit to the points. Panel (c), the mean-variance trend enables each observation to map to a square-root standard deviation value using its fitted value for log-count.

5 Independent Filtering

Independent filtering is a strategy to remove features (in this case, genes) prior to the analysis. Removal of these features may lower the multiple testing correction for other genes that pass the filter. We try to remove genes that have a low power to be found statistically significant, and/or that are biologically less relevant. A common filtering strategy is to remove genes with a generally low expression, as low counts have lower relative uncertainty (hence lower statistical power), and may be considered biologically less relevant.

Implementation in edgeR.

```
?filterByExpr
```

```
## Filter Genes By Expression Level
##
## Description:
##
##      Determine which genes have sufficiently large counts to be
##      retained in a statistical analysis.
##
## Usage:
##
##      ## S3 method for class 'DGEList'
##      filterByExpr(y, design = NULL, group = NULL, lib.size = NULL, ...)
##      ## S3 method for class 'SummarizedExperiment'
##      filterByExpr(y, design = NULL, group = NULL, lib.size = NULL, ...)
##      ## Default S3 method:
##      filterByExpr(y, design = NULL, group = NULL, lib.size = NULL,
##                   min.count = 10, min.total.count = 15, large.n = 10, min.prop = 0.7, ...)
##
## Arguments:
##
```

```
##        y: matrix of counts, or a 'DGEList' object, or a
##           'SummarizedExperiment' object.
##
##   design: design matrix. Ignored if 'group' is not 'NULL'.
##
##    group: vector or factor giving group membership for a oneway layout,
##           if appropriate.
##
## lib.size: library size, defaults to 'colSums(y)'.
##
## min.count: numeric. Minimum count required for at least some samples.
##
## min.total.count: numeric. Minimum total count required.
##
##  large.n: integer. Number of samples per group that is considered to be
##           "large".
##
## min.prop: numeric. Minimum proportion of samples in the smallest group
##           that express the gene.
##
##      ...: any other arguments. For the 'DGEList' and
##           'SummarizedExperiment' methods, other arguments will be
##           passed to the default method. For the default method, other
##           arguments are not currently used.
##
## Details:
##
##      This function implements the filtering strategy that was
##      intuitively described by Chen et al (2016). Roughly speaking, the
##      strategy keeps genes that have at least 'min.count' reads in a
##      worthwhile number samples. More precisely, the filtering keeps
##      genes that have count-per-million (CPM) above _k_ in _n_ samples,
##      where _k_ is determined by 'min.count' and by the sample library
##      sizes and _n_ is determined by the design matrix.
##
##      _n_ is essentially the smallest group sample size or, more
##      generally, the minimum inverse leverage of any fitted value. If
##      all the group sizes are larger than 'large.n', then this is
##      relaxed slightly, but with _n_ always greater than 'min.prop' of
##      the smallest group size (70% by default).
##
##      In addition, each kept gene is required to have at least
##      'min.total.count' reads across all the samples.
##
## Value:
##
##      Logical vector of length 'nrow(y)' indicating which rows of 'y' to
##      keep in the analysis.
##
## Author(s):
##
##      Gordon Smyth
##
## References:
```

```
##
##       Chen Y, Lun ATL, and Smyth, GK (2016). From reads to genes to
##       pathways: differential expression analysis of RNA-Seq experiments
##       using Rsubread and the edgeR quasi-likelihood pipeline.
##       _F1000Research_ 5, 1438.
##       <http://f1000research.com/articles/5-1438>
##
## Examples:
##
##       ## Not run:
##
##       keep <- filterByExpr(y, design)
##       y <- y[keep,]
##       ## End(Not run)
```

```
suppressPackageStartupMessages({
  library(limma)
  library(edgeR)
  library(DESeq2)
})

dds <- makeExampleDESeqDataSet()
simCounts <-counts(dds)
group <- dds$condition
dge <- edgeR::DGEList(simCounts)
design <- model.matrix(~group)
keep <- filterByExpr(dge, design)
table(keep)
```

FALSE	TRUE
373	627

```
lib.size <- dge$samples$lib.size * dge$samples$norm.factors
cpmMinCount <- 10/median(lib.size)*1e6
keep <- rowSums(cpm(dge) > cpmMinCount) >= 6
table(keep)
```

FALSE	TRUE
373	627

Independent filtering has been formalized by Bourgon *et al.* (2010).

The concept of independent filtering 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.

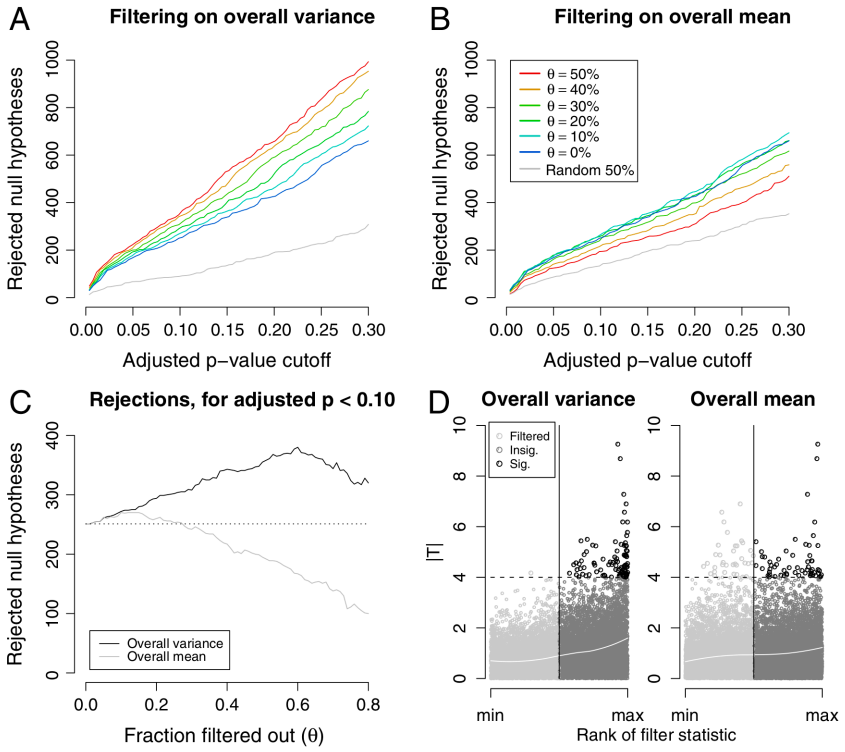


Fig. 1. Power assessment of filtering applied to the ALL data (12,625 genes). *R*, the number of genes called differentially expressed between the two cytogenetic groups, was computed for different stage-one filters, filtering stringencies, and FDR-adjusted *p*-value cutoffs. In all cases, a standard *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 2: Figure 1 from Bourgon *et al.* (2010).

- 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**.

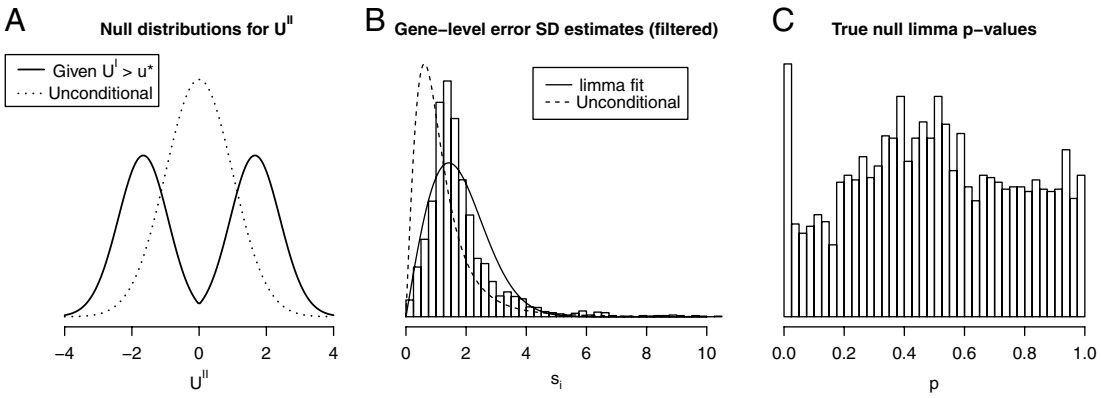


Fig. 2. (*A*) The null distribution of the test statistic is affected by filtering on the maximum of within-class averages. In this example, all genes have a known common variance, the filter statistic is the maximum of within-class means, and the test statistic is a *z*-score. The unconditional distribution 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 3: Figure 2 from Bourgon *et al.* (2010).

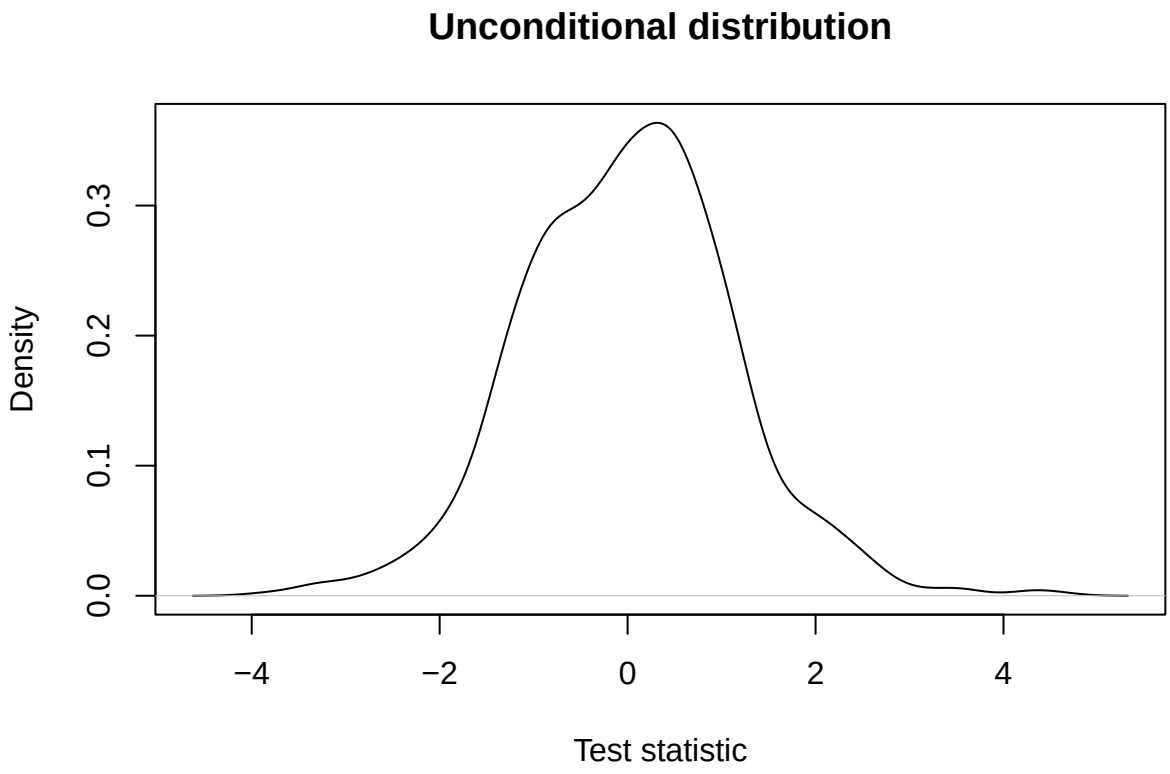
Let's try a couple of examples to get some intuition using simulated data.

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

5.1 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")
```

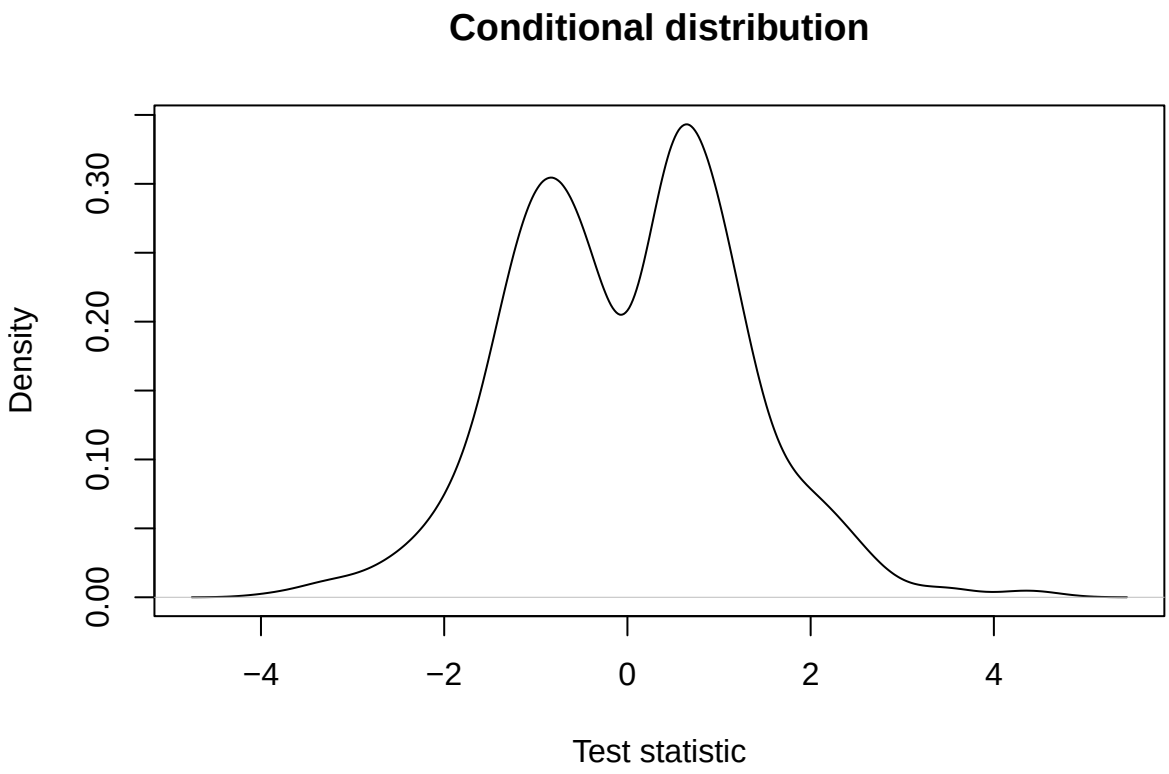



```
## conditional distribution: very different!
mean(filterStatEffectSize > 1)
```

```
## [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")
```

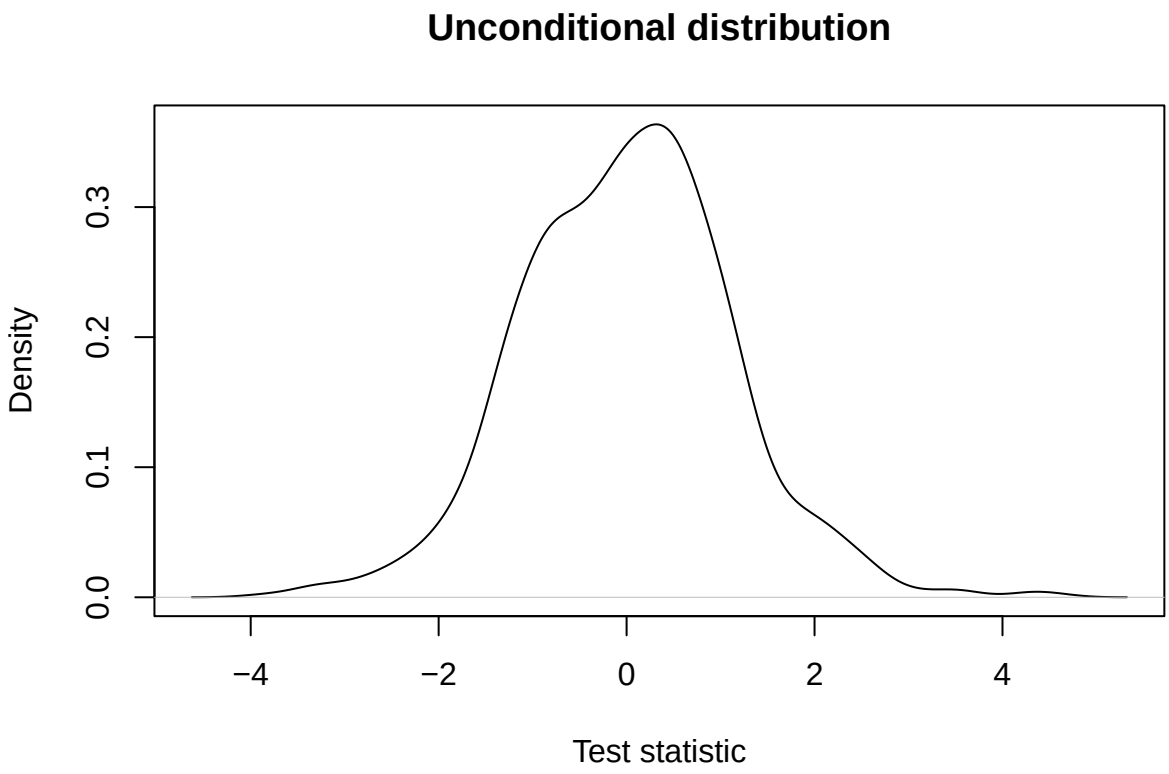


5.2 An Independent Test Statistic

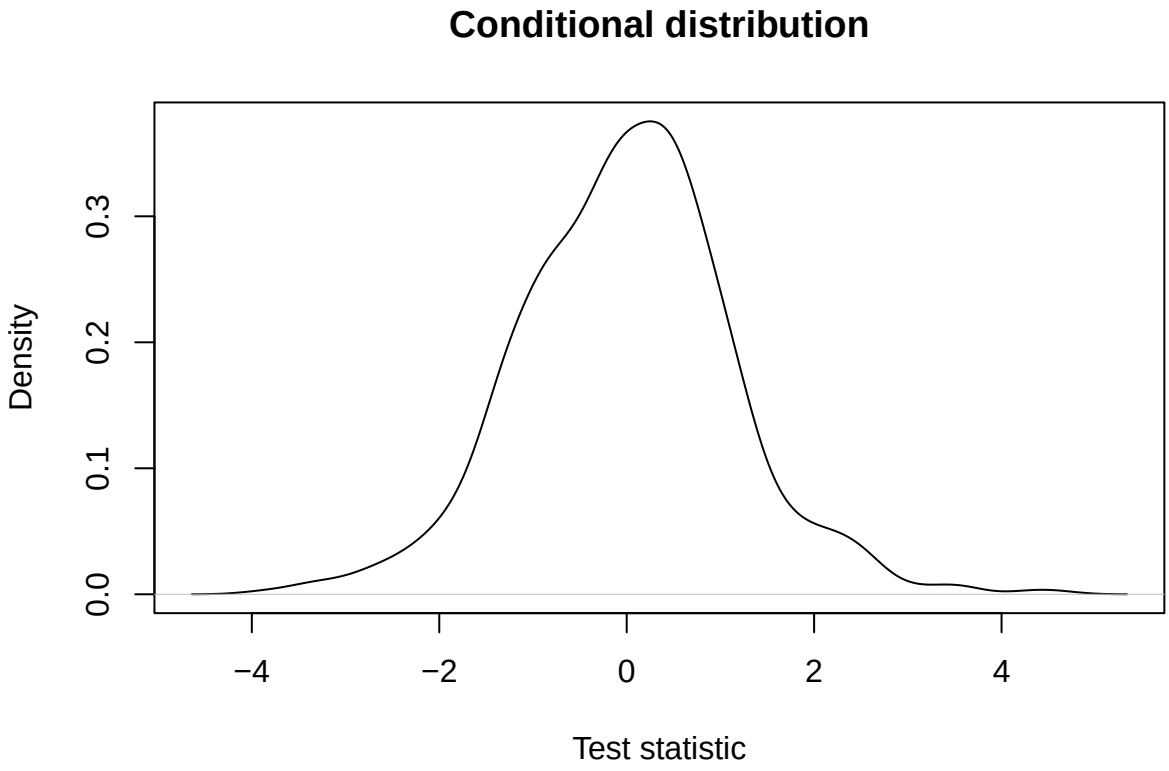
```
filterStatGlobalMean <- rowMeans(simCounts)
mean(filterStatGlobalMean > 5) # we remove a similar fraction
```

```
## [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")
```



6 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. As an extreme example, 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.

```
data("parathyroidGenesSE", package="parathyroidSE")
se1 <- parathyroidGenesSE
rm(parathyroidGenesSE)

colData(se1) %>%
```

```
  as.data.frame() %>%
  filter(duplicated(experiment))
```

run	experiment	patient	treatment	time	submission	study	sample
SRR479061	SRX140511	2	DPN	24h	SRA051611	SRP012167	SRS308873
SRR479064	SRX140513	2	OHT	24h	SRA051611	SRP012167	SRS308875
SRR479075	SRX140523	4	DPN	48h	SRA051611	SRP012167	SRS308885
SRR479078	SRX140525	4	OHT	48h	SRA051611	SRP012167	SRS308887

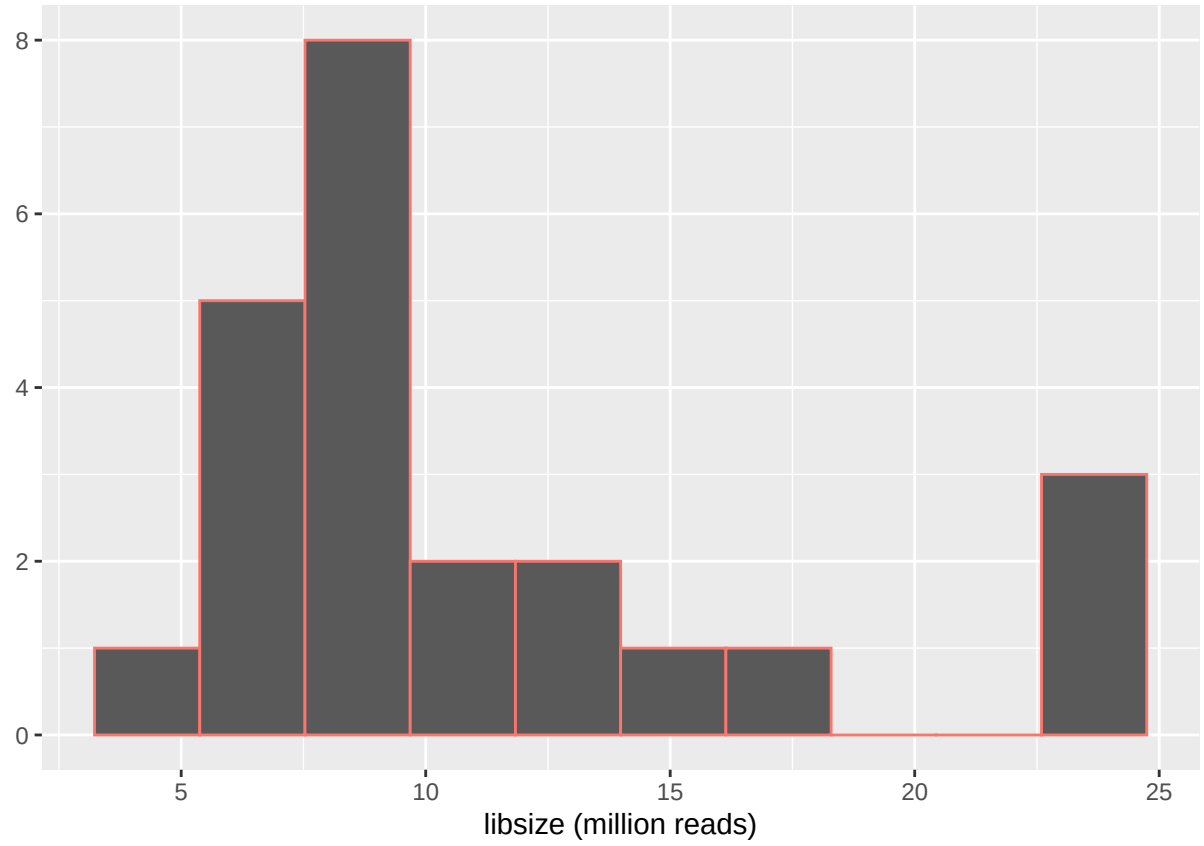
There are technical repeats in the data.

We mentioned previous lectures that we can sum over technical repeats, because techical repeats are Poisson and the sum of two Poisson variables is again Poisson.

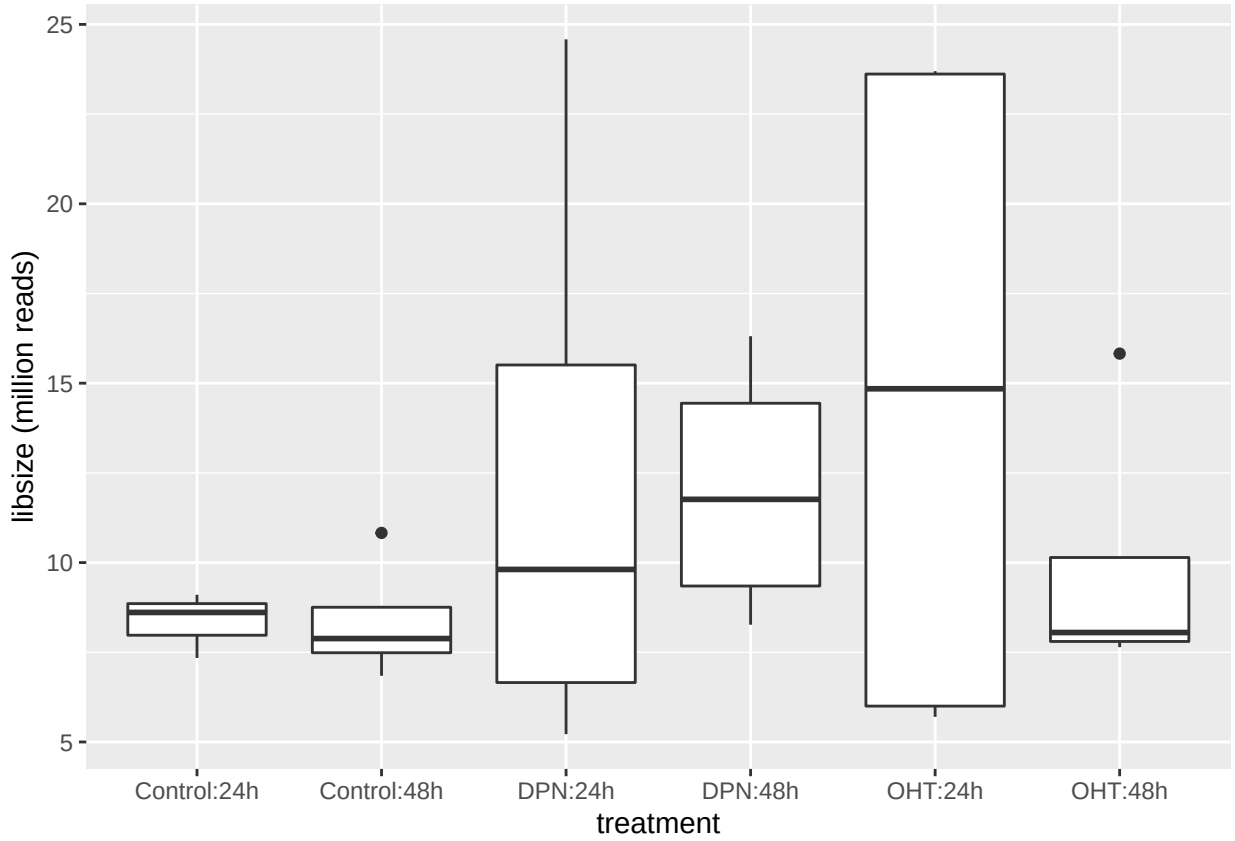
```
dupExps <- colData(se1) %>%
  as.data.frame() %>%
  filter(duplicated(experiment))  %>%
  pull(experiment)

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)
```

Control	DPN	OHT
7	8	8

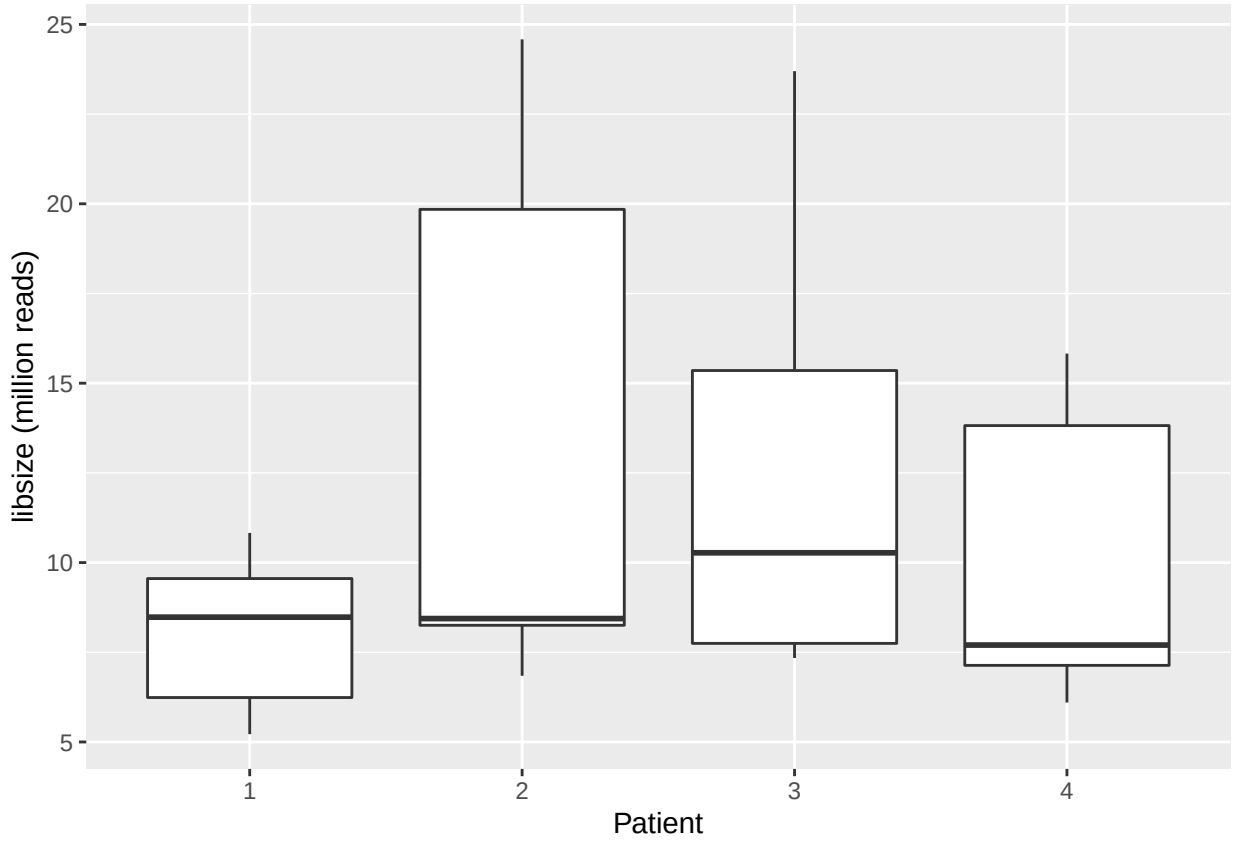
```
qplot(colSums(assays(se)$counts)/1e6, geom="histogram", bins=10,col="black") +
  theme(legend.position = "none") +
  xlab("libsize (million reads)")
```




```
qplot(
  colData(se)$treatment:colData(se)$time,
  colSums(assays(se)$counts)/1e6,geom="boxplot"
  ) +
  xlab("treatment")+
  ylab("libsize (million reads)")
```




```
qplot(
  colData(se)$patient,
  colSums(assays(se)$counts)/1e6,geom="boxplot"
  ) +
  xlab("Patient")+
  ylab("libsize (million reads)")
```




```
ma2Samp <- function(countMx) {
stopifnot("`countMx` is not a matrix with two columns" = ncol(countMx) == 2)
A <- countMx %>% log2 %>% rowMeans
M <- countMx %>% log2 %>% apply(.,1,diff)
w <- countMx[,1]==min(countMx[,1]) | countMx[,2]==min(countMx[,2])
if (any(w)) {
            A[w] <- runif(sum(w), min = -1, max = .1)
            M[w] <- log2(countMx[w,2] + 1) - log2(countMx[w,1] + 1)
}
MAplot <- qplot(A, M, col=w) +
  theme(legend.position = "none") +
  scale_color_manual(values = c("black","orange")) +
  xlab("A: log2 Average") +
  ylab("M: log2 Fold Change")

MAplot +
  geom_abline(intercept=0,slope=0,col="red")
}
```

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.

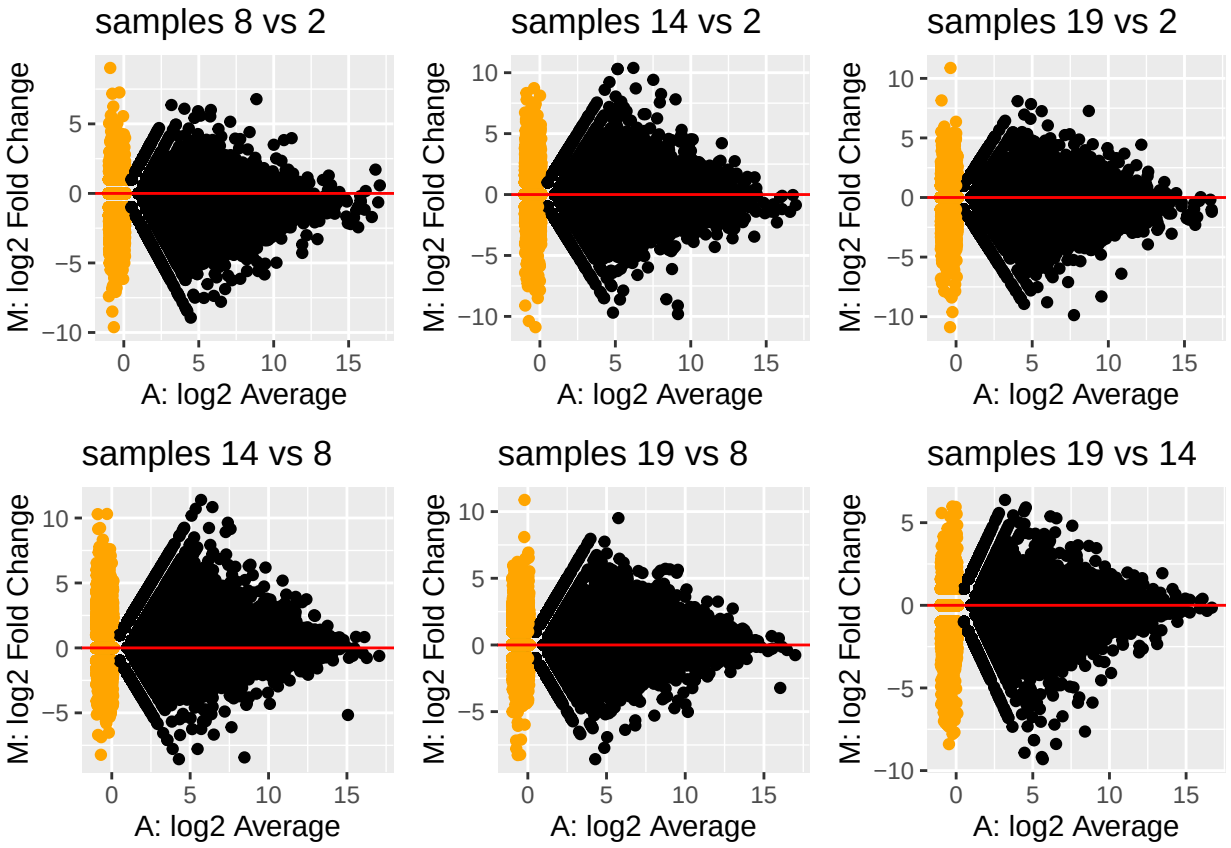
```
ids <- which(colData(se)$treatment =="Control" & colData(se)$time == "48h")
ids
```

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

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

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

```
pairComb <- combn(
  ids,
  m=2)
plots <- apply(pairComb,2,function(x) ma2Samp(assay(se)[,x]) + ggtitle(paste("samples",x[2],"vs", x[1]))
do.call("grid.arrange",c(plots,ncol=3))
```



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 a clear library size effect that we should take into account.

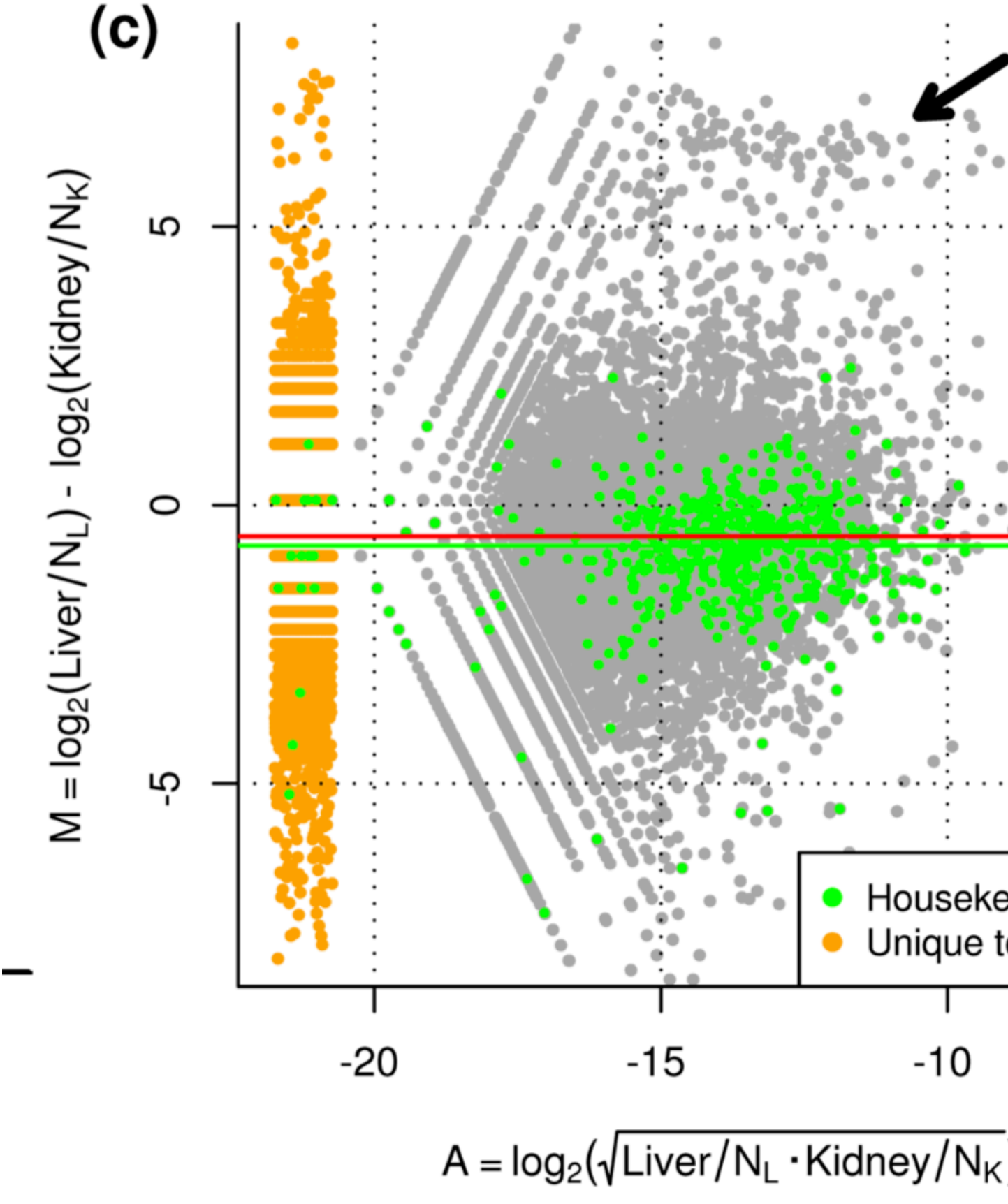
We can solve these issues by introducing offsets in our model.

$$\begin{cases} Y_{gi} & \sim & Poi(\mu_{gi}) \\ \log \mu_{gi} & = & \eta_{gi} \\ \eta_{gi} & = & \mathbf{X}_i^T \beta_g + log(O_{gi}) \end{cases}$$

6.1 TMM method (default of edgeR)

Robinson and Oshlack (2010). Genome Biology

```
knitr::include_graphics("./figs/edgeRNormIntro.png")
```


(c)
M = log2(Liver/NL) - log2(Kidney/NK)
5
0
-5
-20
-15
-10
A = log2(√Liver/NL · Kidney/NK
Houseke
Unique t

- On the plot we see a clear effect on all genes
- Correcting for library size tends to over correct.
- Some DE genes are highly abundant and determine the library size to a large extend

The trimmed mean of M-values (TMM) method introduced by Robinson & Oshlack (2010) is a normalization procedure that calculates a single normalization factor for each sample. 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.,

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

and w_{gi}^r represents a precision 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 genes after trimming those with the most extreme average expression. The weights serve to account for the fact that fold-changes for genes with lower read counts are more variable.

The procedure only takes genes into account where both $Y_{gi} > 0$ and $Y_{gr} > 0$. By default, TMM trims genes 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

$$N_i^s = \frac{N_i F_i^{(r)}}{\sum_{i=1}^n N_i F_i^{(r)}/n}.$$

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	time	submission	study	sample
Sample1	1	9102683	0.9782830	SRR479052	SRX140503	3	Control	24h	SRA051611	SRP012167	SRS308865
Sample2	1	10827109	0.9728700	SRR479053	SRX140504	4	Control	48h	SRA051611	SRP012167	SRS308866
Sample3	1	5217761	0.9898593	SRR479054	SRX140505	5	DPN	24h	SRA051611	SRP012167	SRS308867
Sample4	1	9706035	0.9930169	SRR479055	SRX140506	6	DPN	48h	SRA051611	SRP012167	SRS308868
Sample5	1	5700022	0.9850867	SRR479056	SRX140507	7	OHT	24h	SRA051611	SRP012167	SRS308869
Sample6	1	7854568	0.9897270	SRR479057	SRX140508	8	OHT	48h	SRA051611	SRP012167	SRS308870
Sample7	1	8610014	0.9266581	SRR479058	SRX140509	9	Control	24h	SRA051611	SRP012167	SRS308871
Sample8	1	6844144	0.9544240	SRR479059	SRX140510	10	Control	48h	SRA051611	SRP012167	SRS308872
Sample9	1	24584280	0.9188545	SRR479060	SRX140511	11	DPN	24h	SRA051611	SRP012167	SRS308873
Sample10	1	8267977	0.9398000	SRR479061	SRX140512	12	DPN	48h	SRA051611	SRP012167	SRS308874
Sample11	1	23590411	0.9096695	SRR479062	SRX140513	13	OHT	24h	SRA051611	SRP012167	SRS308875
Sample12	1	8247122	0.9369050	SRR479063	SRX140514	14	OHT	48h	SRA051611	SRP012167	SRS308876

	group	lib.size	norm.factors	run	experiment	patient	treatment	time	submission	study	sample
Sample13	1	7341000	1.0668032	SRR479068	SRX140513	3	Control	24h	SRA051611	SRP012167	SRS308877
Sample14	1	8064268	1.0552688	SRR479069	SRX140516	6	Control	48h	SRA051611	SRP012167	SRS308878
Sample15	1	124819581	1.0461698	SRR479066	SRX140517	7	DPN	24h	SRA051611	SRP012167	SRS308879
Sample16	1	163100901	1.0260056	SRR479067	SRX140518	8	DPN	48h	SRA051611	SRP012167	SRS308880
Sample17	1	236973291	1.0268459	SRR479070	SRX140519	9	OHT	24h	SRA051611	SRP012167	SRS308881
Sample18	1	7642648	1.0409451	SRR479071	SRX140520	10	OHT	48h	SRA051611	SRP012167	SRS308882
Sample19	1	7701432	1.0559132	SRR479072	SRX140521	1	Control	48h	SRA051611	SRP012167	SRS308883
Sample20	1	7135899	1.0675040	SRR479073	SRX140522	2	DPN	24h	SRA051611	SRP012167	SRS308884
Sample21	1	138183931	1.0327004	SRR479074	SRX140523	4	DPN	48h	SRA051611	SRP012167	SRS308885
Sample22	1	6099942	1.0890994	SRR479075	SRX140524	5	OHT	24h	SRA051611	SRP012167	SRS308886
Sample23	1	158252111	1.0286470	SRR479076	SRX140525	11	OHT	48h	SRA051611	SRP012167	SRS308887

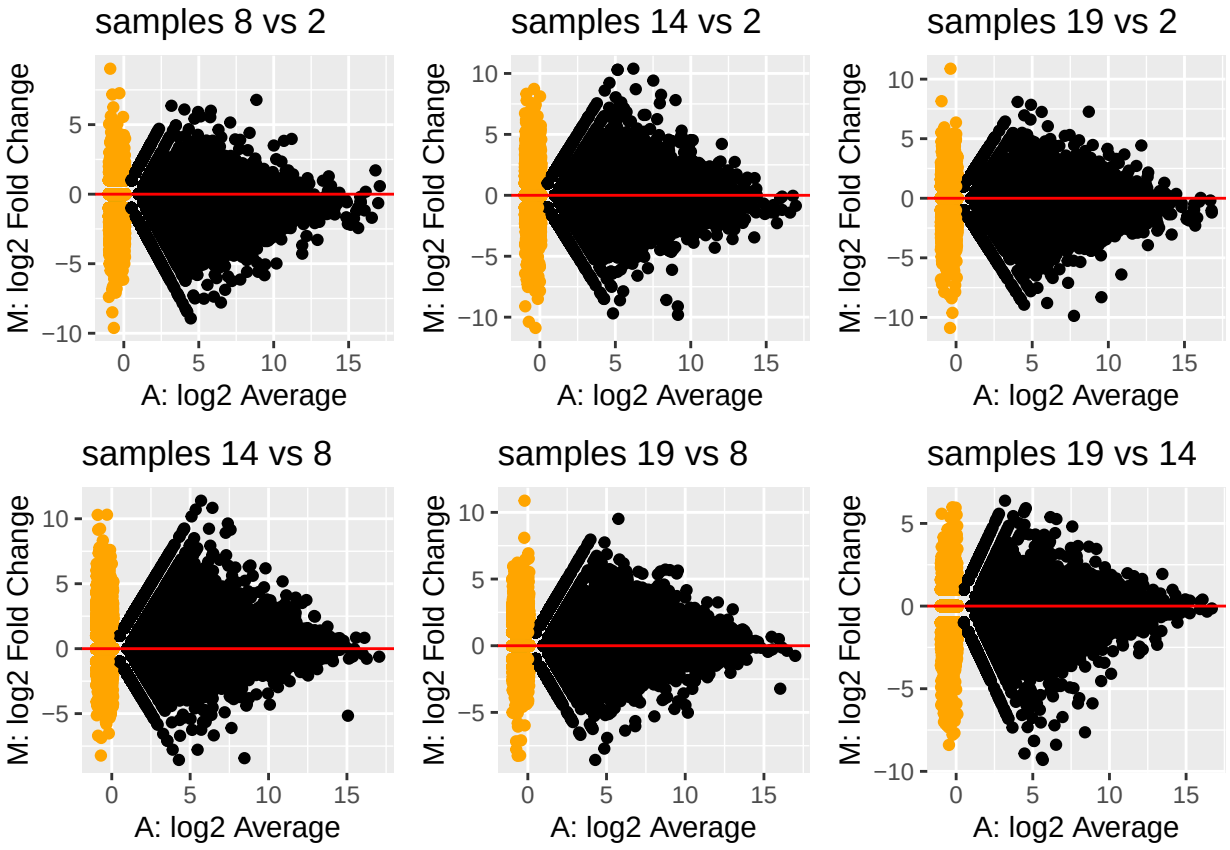
Let's check how our MD-plots look like after normalization. Note that, we can rewrite the GLM as

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

and so $\frac{\mu_{gi}}{N_i^s}$ 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)

plotsNorm <- apply(pairComb,2,function(x) ma2Samp(normCountTMM[,x]) + ggtitle(paste("samples",x[2],"vs"
do.call("grid.arrange",c(plots,ncol=3))
```



6.2 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

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

We can then use the size factors s_i as offsets to the GLM.

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)
```

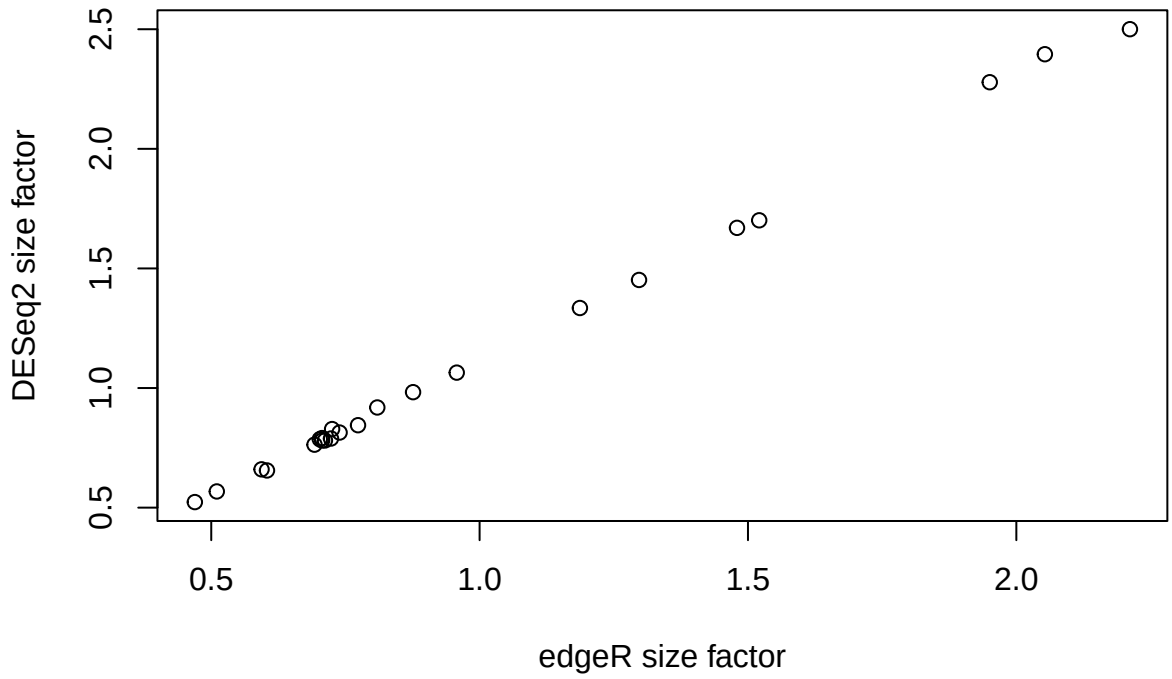
```
##  [1] 0.9187624 1.0644717 0.5232069 0.9826698 0.5676212 0.7807662 0.8284089
##  [8] 0.6600848 2.3955069 0.7906917 2.2784591 0.7860386 0.7807842 0.8445577
## [15] 1.3345483 1.7011973 2.5001652 0.7889061 0.8138720 0.7632306 1.4518320
## [22] 0.6553677 1.6696064
```

You may also want to check out the StatQuest video on DESeq2 normalization.

6.2.1 Comparing TMM with DESeq2 Normalization

We can compare the size factors for both normalizations to verify if they agree on the normalization factors. Note we need to scale the effective library sizes from `edgeR` to enforce a similar scale as the size factors from `DESeq2`. While below we are using an arithmetic mean, a geometric mean may be used as well, which will be more robust to outlying effective library sizes.

```
plot(effLibSize / mean(effLibSize), sizeFactors(dds),
     xlab = "edgeR size factor",
     ylab = "DESeq2 size factor")
```



7 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)
# 2 samples per patient for 8 patients
patient <- factor(rep(letters[1:8], each=2))
# first four are healthy, next four are diseased
disease <- factor(c(rep("healthy",8), rep("cancer",8)), levels=c("healthy", "cancer"))
# one before and one after sample for each
time <- factor(rep(c("before", "after"), 8), levels=c("before", "after"))

table(patient, disease, time)
```

patient	disease	time	Freq
a	cancer	after	0
		before	0
	healthy	after	1
		before	1
b	cancer	after	0
		before	0
	healthy	after	1
		before	1
c	cancer	after	0
		before	0
	healthy	after	1
		before	1
d	cancer	after	0
		before	0
	healthy	after	1
		before	1
e	cancer	after	1
		before	1
	healthy	after	0
		before	0
f	cancer	after	1
		before	1

patient	disease	time	Freq
	healthy	after	0
		before	0
g	cancer	after	1
		before	1
	healthy	after	0
		before	0
h	cancer	after	1
		before	1
	healthy	after	0
		before	0

```
## 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 ourselves. 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 as compared to 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	timeafter
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	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

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