Managing Batch Effects in Microbiome Data

Yiwen Wang

2019-06-20

Contents

Chapter 1

Examples of microbiome studies with batch effects

This vignette provides all the analyses performed in the paper 'Managing Batch Effects in Microbiome Data'.

Packages installation and loading

First, you will need to install then load the following packages:

```
# cran.packages = c('knitr', 'mixOmics', 'xtable', 'ggplot2', 'vegan', 'cluster',
#                   'gridExtra', 'pheatmap', 'ruv', 'lmerTest', 'bapred')
# install.packages(cran.packages)
# bioconductor.packages = c('sva', 'limma', 'AgiMicroRna',
#                           'variancePartition', 'pvca')
# if (!requireNamespace('BiocManager', quietly = TRUE))
#     install.packages('BiocManager')
# BiocManager::install(bioconductor.packages, version = '3.8')

library(knitr)
library(xtable) # table
library(mixOmics)
library(sva) # ComBat
library(ggplot2) # PCA sample plot with density
library(gridExtra) # PCA sample plot with density
library(limma) # removeBatchEffect (LIMMA)
library(vegan) # RDA
library(AgiMicroRna) # RLE plot
library(cluster) # silhouette coefficient
library(variancePartition) # variance calculation
library(pvca) # PVCA
library(pheatmap) # heatmap
library(ruv) # RUVIII
library(lmerTest) # lmer
library(bapred) # FAbatch
```

1.1 Study description

1.1.1 Sponge *Aplysina aerophoba* study

Sacristán-Soriano *et al.* studied the potential involvement of bacterial communities from the sponge species *A. aerophoba* in the biosynthesis of brominated alkaloids (BAs) (Sacristán-Soriano et al., 2011). They compared the microbial composition and BA concentration in two different tissues (ectosome and choanosome) to investigate the relationship between bacterial composition and BA concentration. The authors concluded that differences in bacterial profiles were not only due to tissue variation (the main effect of interest), but also because the samples were run on two separate denaturing gradient gels during processing. Gel thus acted as a technical batch effect as described in Table 1.

1.1.2 Anaerobic digestion study

Anaerobic Digestion (AD) is a microbiological process of organic matter degradation that produces a biogas used in electrical and thermal energy production. However, the AD bioprocess undergoes inhibition during its developmental stage that is not well characterised: Chapleur *et al.* explored microbial indicators that could improve the AD bioprocess's efficacy and prevent its failure (Chapleur et al., 2016). They profiled the microbiota of 75 AD samples in various conditions. Here we consider two different ranges of phenol concentration as treatments. The experiment was conducted at different dates (5), which constitutes a technical source of unwanted variation (Table 1).

1.1.3 Huntington's disease study

In their study, Kong *et al.* reported differences in microbial composition between Huntington's disease (HD) and wild-type (WT) mice (Kong et al., 2018). However, the establishment of microbial communities was also driven by biological batch effects: the cage environment and sex. Here we consider only female mice to illustrate a special case of a batch $\times$ treatment unbalanced design. The HD data include 13 faecal mice samples hosted across 4 cages (Table 1).

We load the data and functions that are provided *outside* the packages.

```
# load the data
load(file = './datasets/microbiome_datasets.RData')

# load the extra functions
source(file = './Functions.R')
dim(sponge.tss)
```

```
## [1] 32 24
```

```
dim(ad.count)
```

```
## [1]  75 567
```

```
dim(hd.count)
```

```
## [1]  13 368
```

Note: the AD data and HD data loaded are raw counts, while sponge data are total sum scaling (TSS) scaled data calculated on raw counts, with no offset.

1.2 Data processing

Data processing steps for microbiome data:

1. Prefilter the count data to remove features with excess zeroes across all samples
2. Add an offset of 1 to the whole data matrix
3. Total Sum Scaling transformation
4. Log-ratio transformation

1.2.1 Prefiltering

We use a prefiltering step to remove OTUs for which the sum of counts are below a set threshold (0.01%) compared to the total sum of all counts from (Arumugam et al., 2011).

```
# ad data
ad.index.keep = which(colSums(ad.count)*100/(sum(colSums(ad.count))) > 0.01)
ad.count.keep = ad.count[, ad.index.keep]
dim(ad.count.keep)
```

```
## [1]  75 231
```

```
# hd data
#hd.index.keep = which(colSums(hd.count)*100/(sum(colSums(hd.count))) > 0.001)
hd.count.keep = hd.count
dim(hd.count.keep)
```

```
## [1]  13 368
```

Compared with the raw counts, many OTUs below the threshold are removed. As HD data are small part of a big dataset and only have 13 samples, it is too strict to use the threshold 0.01% to filter the OTUs. We then maintained all the OTUs in HD data.

1.2.2 Total sum scaling

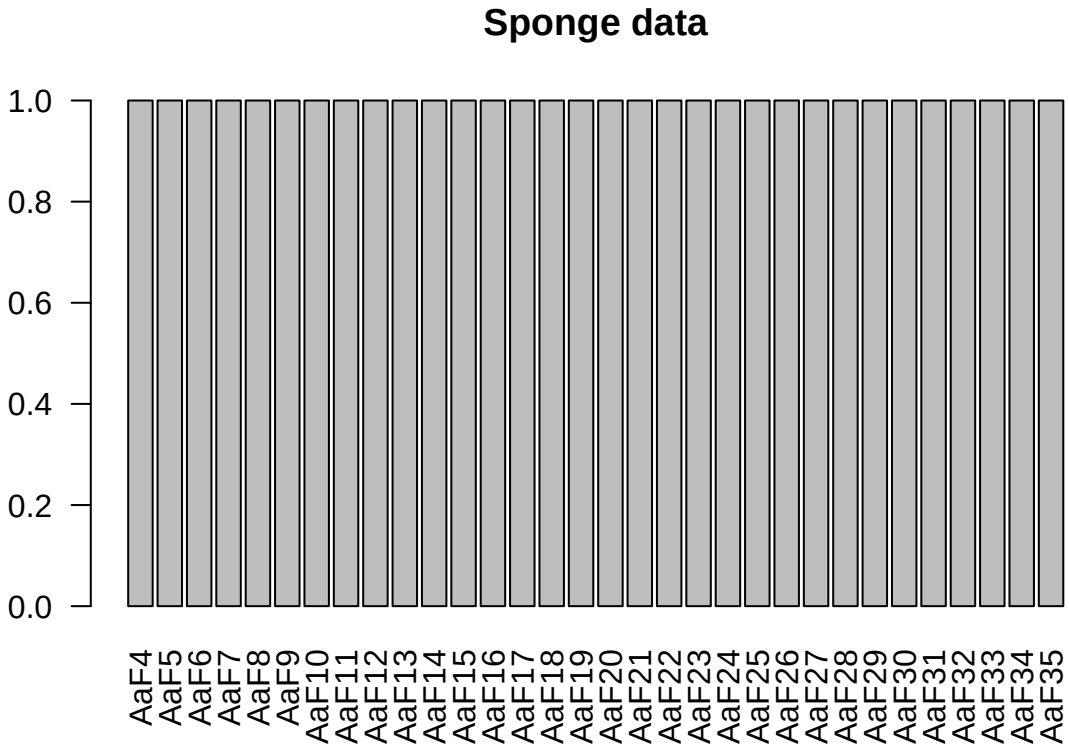
Now, let's apply a TSS scaling on the filtered data:

Note: We need to add an offset of 1 to all count data (i.e. count data = count.keep + 1). Although sponge data are already TSS data, a 'tiny' offset is added and then TSS is redone. Because the presence of zeroes will make the next step (log-ratio transformation) impossible.

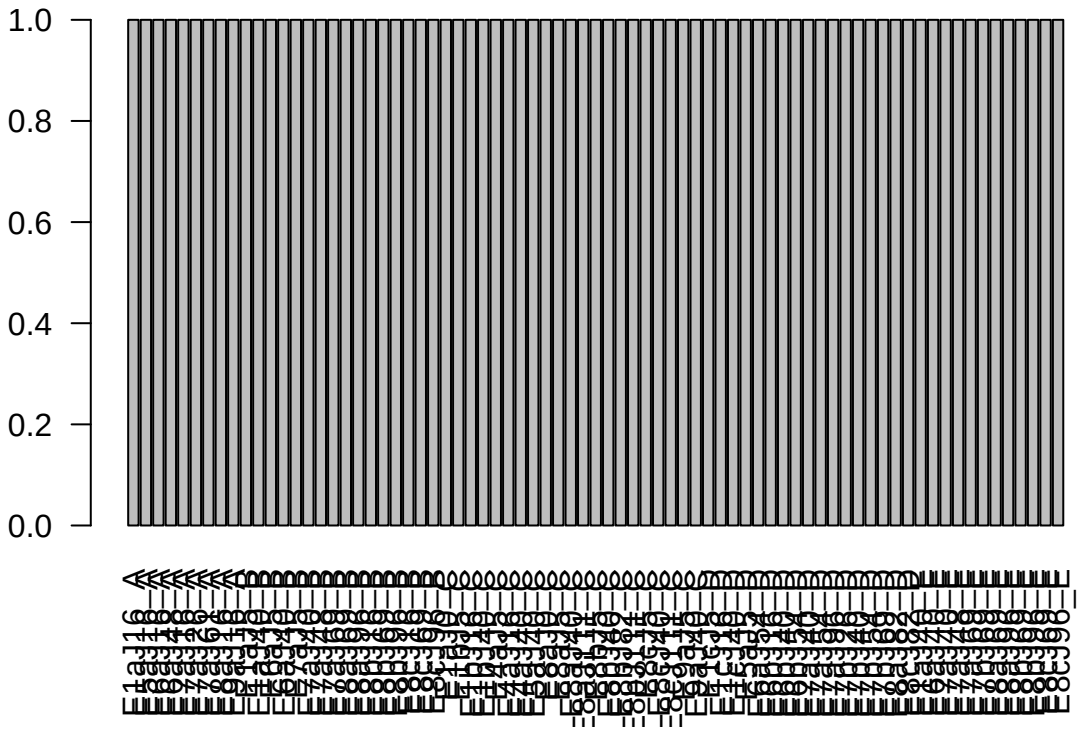
```
# sponge data
sponge.tss = t(apply(sponge.tss + 0.01, 1, TSS.divide))

# ad data
ad.tss = t(apply(ad.count.keep + 1, 1, TSS.divide))

# hd data
hd.tss = t(apply(hd.count.keep + 1, 1, TSS.divide))
```

We then check the library size that sums to 1 for each sample. We now have compositional data.

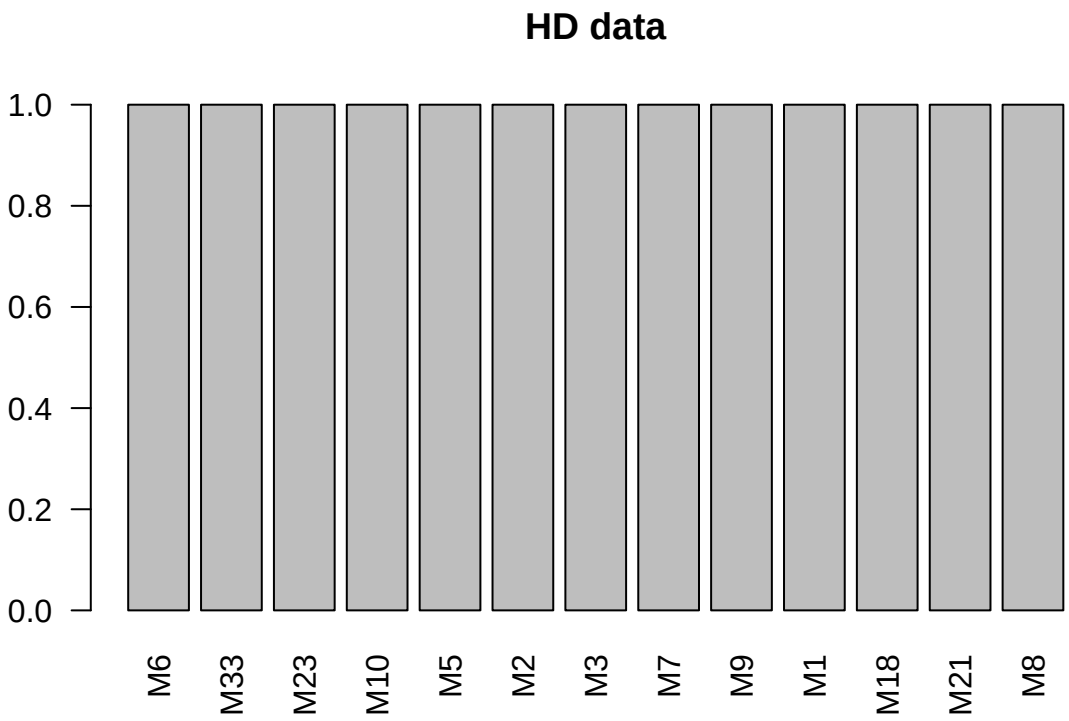
```
# sponge data
sponge.lib.size.tss <- apply(sponge.tss, 1, sum)
barplot(sponge.lib.size.tss, main =  'Sponge data', las = 2)
```




```
# ad data
ad.lib.size.tss <- apply(ad.tss, 1, sum)
barplot(ad.lib.size.tss, main =  'AD data', las = 2)
```



```
# hd data
hd.lib.size.tss <- apply(hd.tss, 1, sum)
barplot(hd.lib.size.tss, main =  'HD data', las = 2)
```



1.2.3 Centered log-ratio transformation

TSS results in compositional data (or proportions) that are restricted to a space where the sum of all OTU proportions for a given sample sums to 1. Using standard statistical methods on such data may lead to spurious results and therefore the data must be further transformed. The CLR is the transformation of choice.

```
# sponge data
sponge.tss.clr <- logratio.transfo(sponge.tss, logratio = 'CLR')
class(sponge.tss.clr) <- 'matrix'

# ad data
ad.tss.clr <- logratio.transfo(ad.tss, logratio = 'CLR')
class(ad.tss.clr) <- 'matrix'

# hd data
hd.tss.clr <- logratio.transfo(hd.tss, logratio = 'CLR')
class(hd.tss.clr) <- 'matrix'
```

The final TSS-CLR data of sponge study, AD study and HD study contain 32 samples and 24 OTUs, 75 samples and 231 OTUs, 13 samples and 368 OTUs, respectively as described in Table 1.

Table 1. Overview of exemplar datasets with batch effects. We considered microbiome studies from sponge *Aplysina aerophoba*; organic matter in anaerobic digestion (AD) and mice models with Huntington's disease (HD).

	Sponge data			**AD data**			**HD data**		
No. of OTUs	24			231			368		
No. of samples	32			75			13		
Design type	Balanced			Approx. balanced			Unbalanced		
Organism	Sponge samples			Organic matter			Fecal samples		
Batch sources	Gel (sample processing)			Date (sample processing and sequencing)			Cage (housing)		
		Ectosome	Choanosome		0-0.5	1-2		HD	WT
Batch × treatment design	Gel 1	8	8	09/04/2015	4	5	Cage F	0	3
				14/04/2016	4	12	Cage G	3	0
				01/07/2016	8	13	Cage H	3	0
	Gel 2	8	8	14/11/2016	8	9	Cage J	0	4
				21/09/2017	2	10			

Chapter 2

Batch effect detection

In this chapter, we apply qualitative methods and diagnostic plots to visually assess the presence of batch effects.

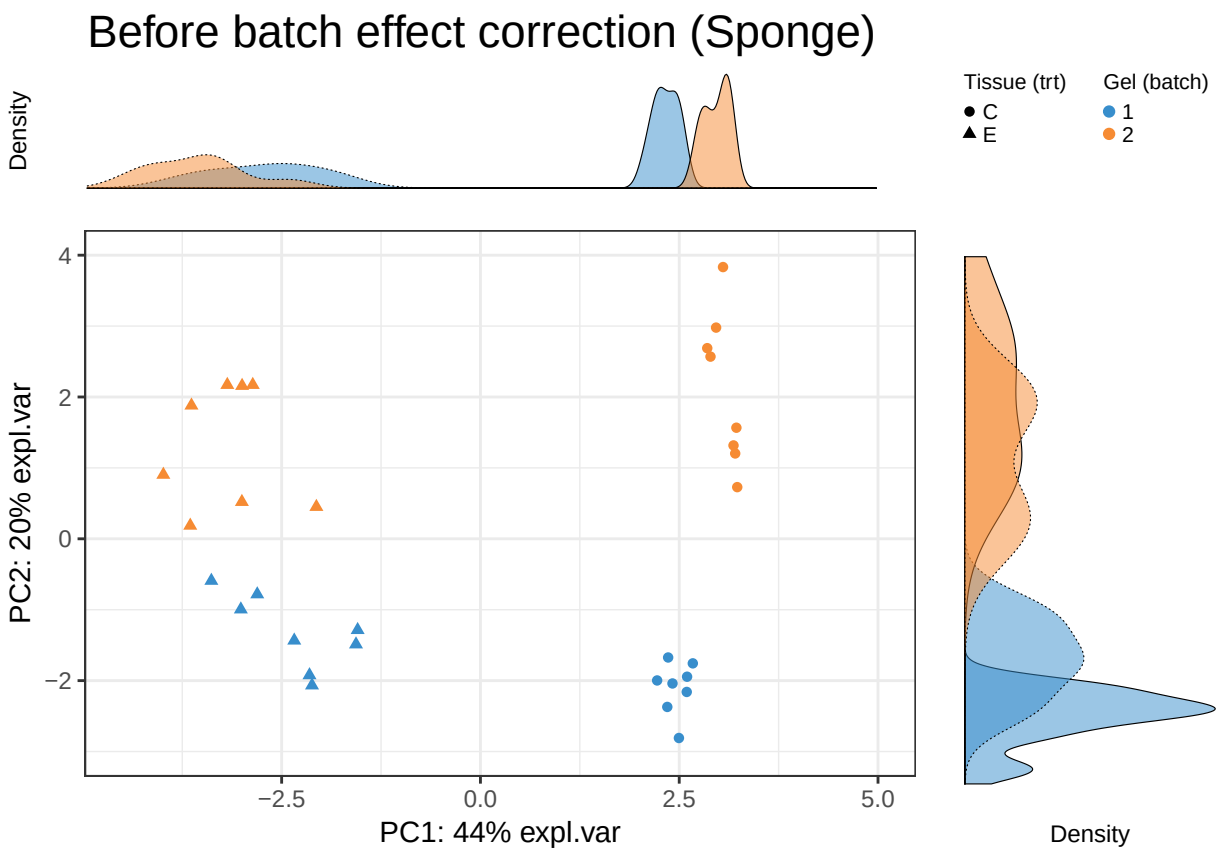
2.1 Principal component analysis (PCA) with density plot per component

PCA is an unsupervised method used to explore the data variance structure by reducing its dimensions to a few principal components (PC) that explain the greatest variation in the data. Density plots are a complementary way to visualise batch effects per PC through examining the distributions of all samples.

First, we run a good old PCA on the data, to assess whether major sources of variation can be explained by batch effects: in cases where batch effects account for a large source of variation in the data, the scatter plot of the top PCs should highlight a separation of the samples due to different batches. Plotting density plots on each component helps to visualise whether it is the case: samples within a batch will show similar distributions, and samples across different batches will show different distributions, if there is a batch effect.

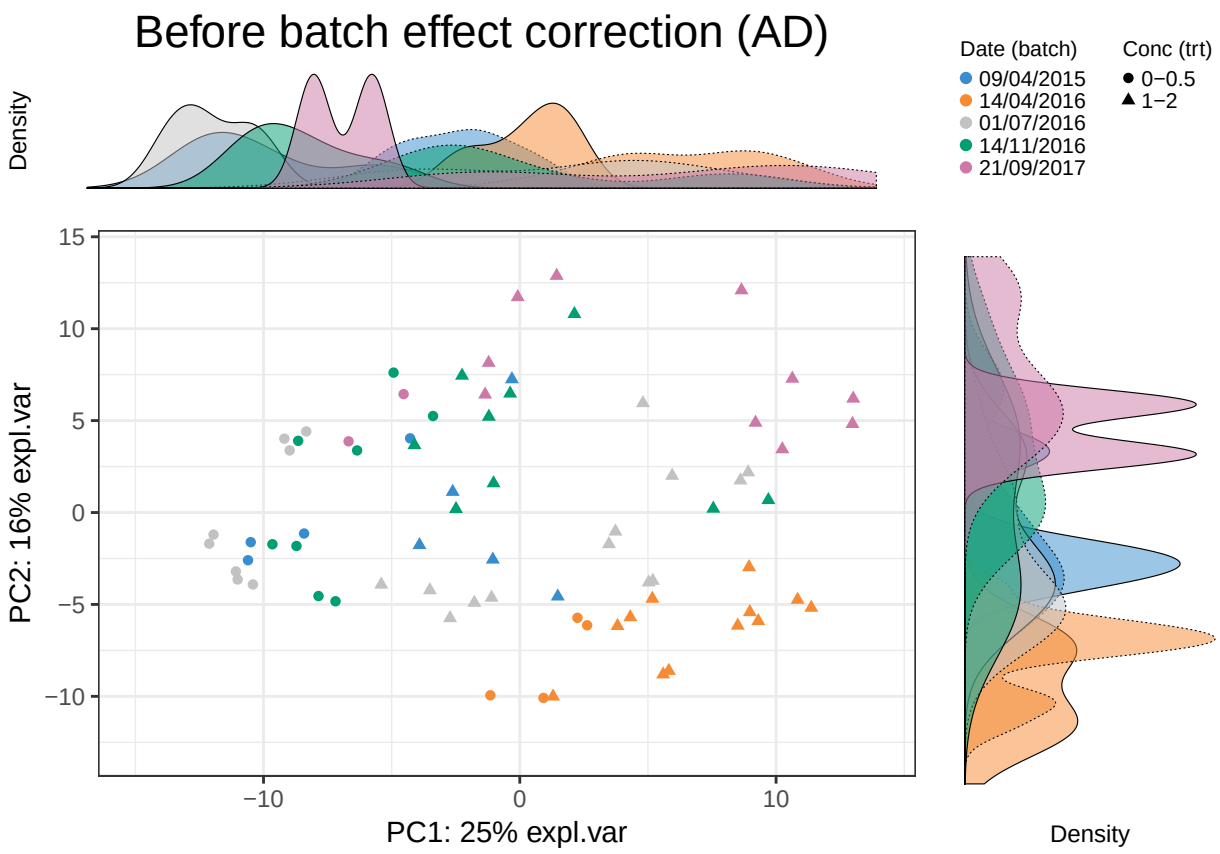
```
# sponge data
sponge.pca.before = pca(sponge.tss.clr, ncomp = 3)

# ad data
ad.pca.before = pca(ad.tss.clr, ncomp = 3)

# hd data
hd.pca.before = pca(hd.tss.clr, ncomp = 3)
```

```
# sponge data
Scatter_Density(data = sponge.pca.before$variates$X, batch = sponge.batch,
                trt = sponge.trt, expl.var = sponge.pca.before$explained_variance,
                xlim = c(-4.5,5), ylim = c(-3,4),
                batch.legend.title = 'Gel (batch)',
                trt.legend.title = 'Tissue (trt)',
                title = 'Before batch effect correction (Sponge)')
```



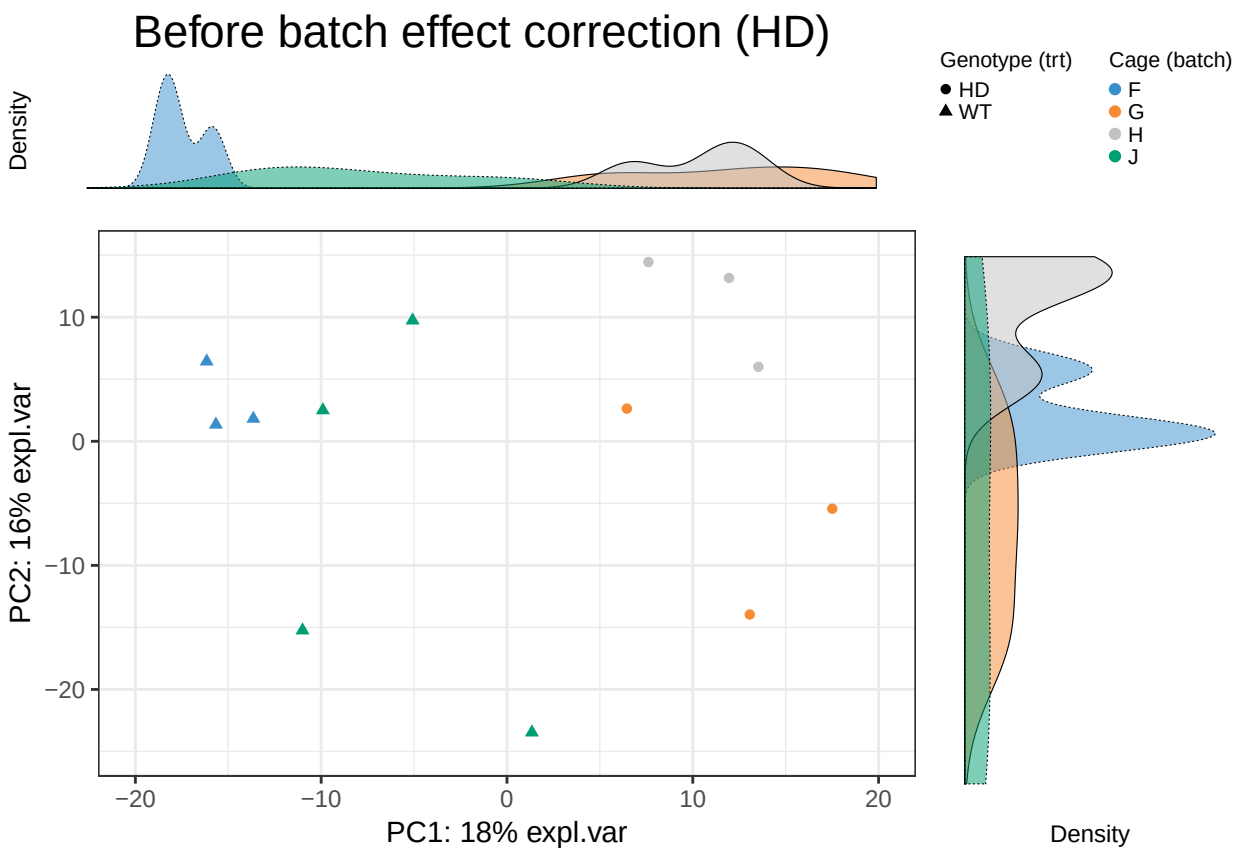
In sponge data, the first PC (explaining the largest source of variation) shows variation between samples from different tissues (the effect of interest), while the second PC (explaining the second largest source of variation) displays sample differences due to different batches, as also highlighted in the density plots per component. Therefore, PCA plots can inform not only of the presence of batch effects, but also which variation is the largest in the data. In this particular dataset, the effect of interest variation is larger than batch variation.

```
# ad data
Scatter_Density(data = ad.pca.before$variates$X, batch = ad.batch,
                trt = ad.trt, expl.var = ad.pca.before$explained_variance,
                xlim = c(-15,14), ylim = c(-13,14),
                batch.legend.title = 'Date (batch)',
                trt.legend.title = 'Conc (trt)',
                title = 'Before batch effect correction (AD)')
```



In AD data, we observe a separation of samples from batch 14/04/2016. The batch variation is mostly explained by the second PC.

```
# hd data
Scatter_Density(data = hd.pca.before$variates$X, batch = hd.batch,
                trt = hd.trt, expl.var = hd.pca.before$explained_variance,
                xlim = c(-20,20), ylim = c(-25,15),
                batch.legend.title = 'Cage (batch)',
                trt.legend.title = 'Genotype (trt)',
                title = 'Before batch effect correction (HD)')
```



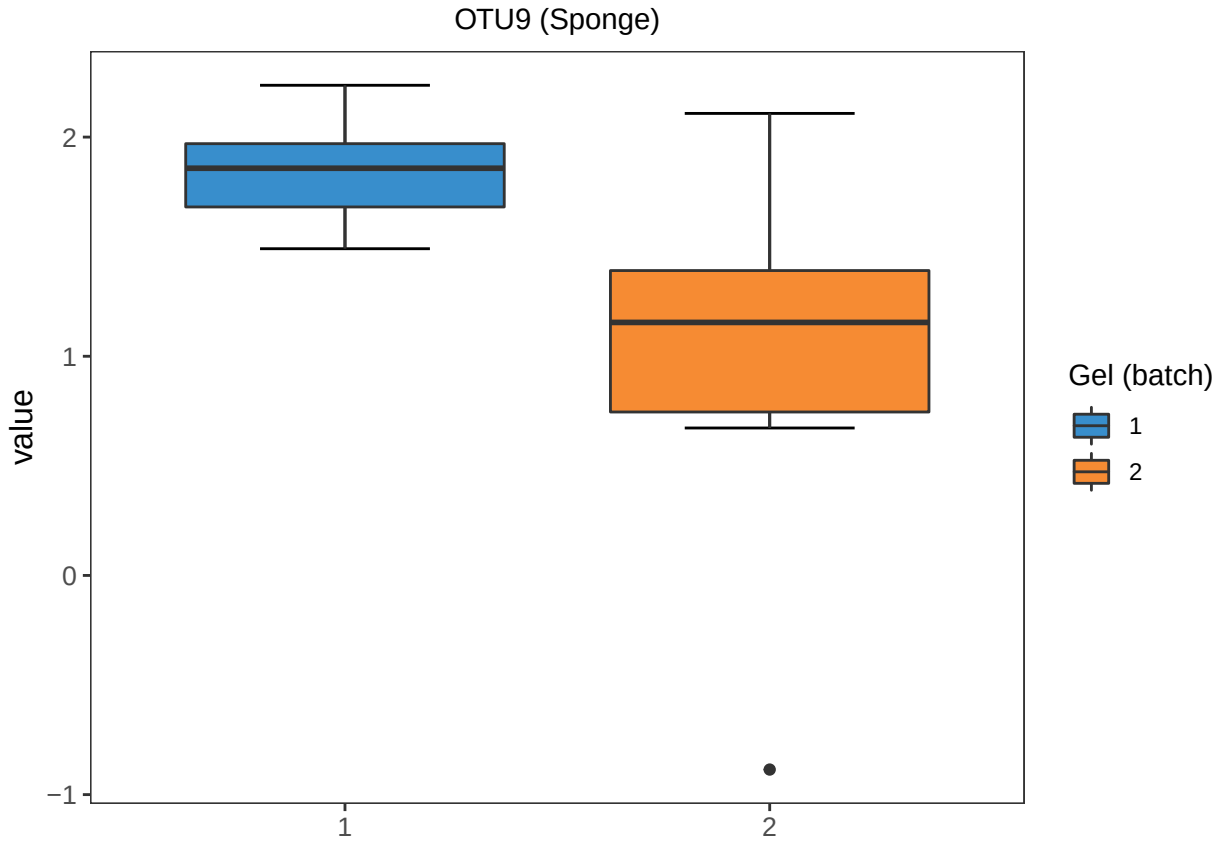
In HD data, batch effect is due to different cages and obvious. The batch variation is both explained by the first and second PC.

2.2 Density plot and box plot

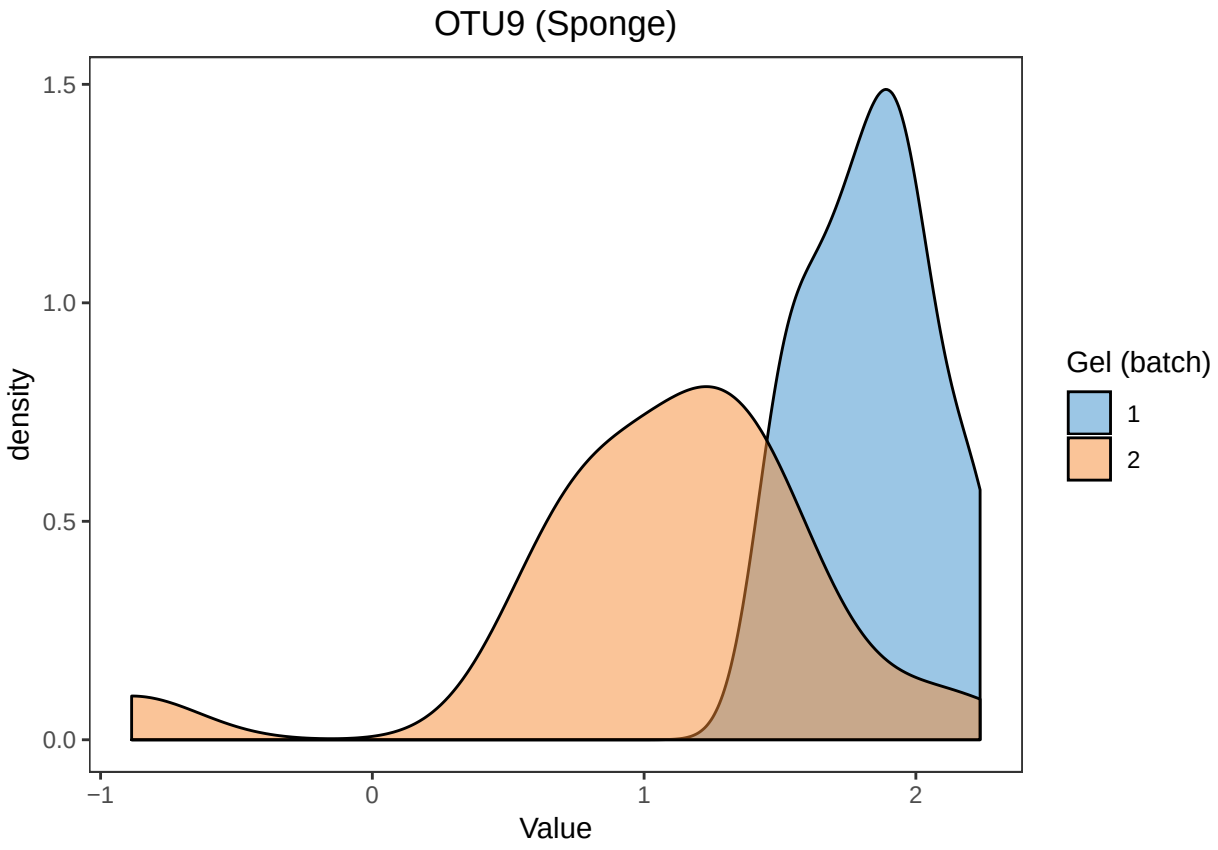
For non systematic batch effects, it is useful to visualise a few OTUs individually. We apply density plots and box plots on OTUs, one at a time from each dataset to visualise batch effects. But only one OTU each dataset is selected as examples.

We randomly select OTU9 in sponge data, and generate density plots and box plots separately across samples within each batch to observe whether batch effects serve as a major source of variation.

```
# sponge data
sponge.before.df = data.frame(value = sponge.tss.clr[,9], batch = sponge.batch)

box_plot_fun(data = sponge.before.df, x = sponge.before.df$batch,
             y = sponge.before.df$value, title = 'OTU9 (Sponge)',
             batch.legend.title = 'Gel (batch)')
```




```
ggplot(sponge.before.df, aes(x = value, fill = batch)) +
  geom_density(alpha = 0.5) + scale_fill_manual(values = color.mixo(1:10)) +
  labs(title = 'OTU9 (Sponge)', x = 'Value', fill = 'Gel (batch)') +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     panel.grid = element_blank())
```



For the abundance of OTU9, the samples within different batches are very distinct, indicating a strong batch effect in sponge data.

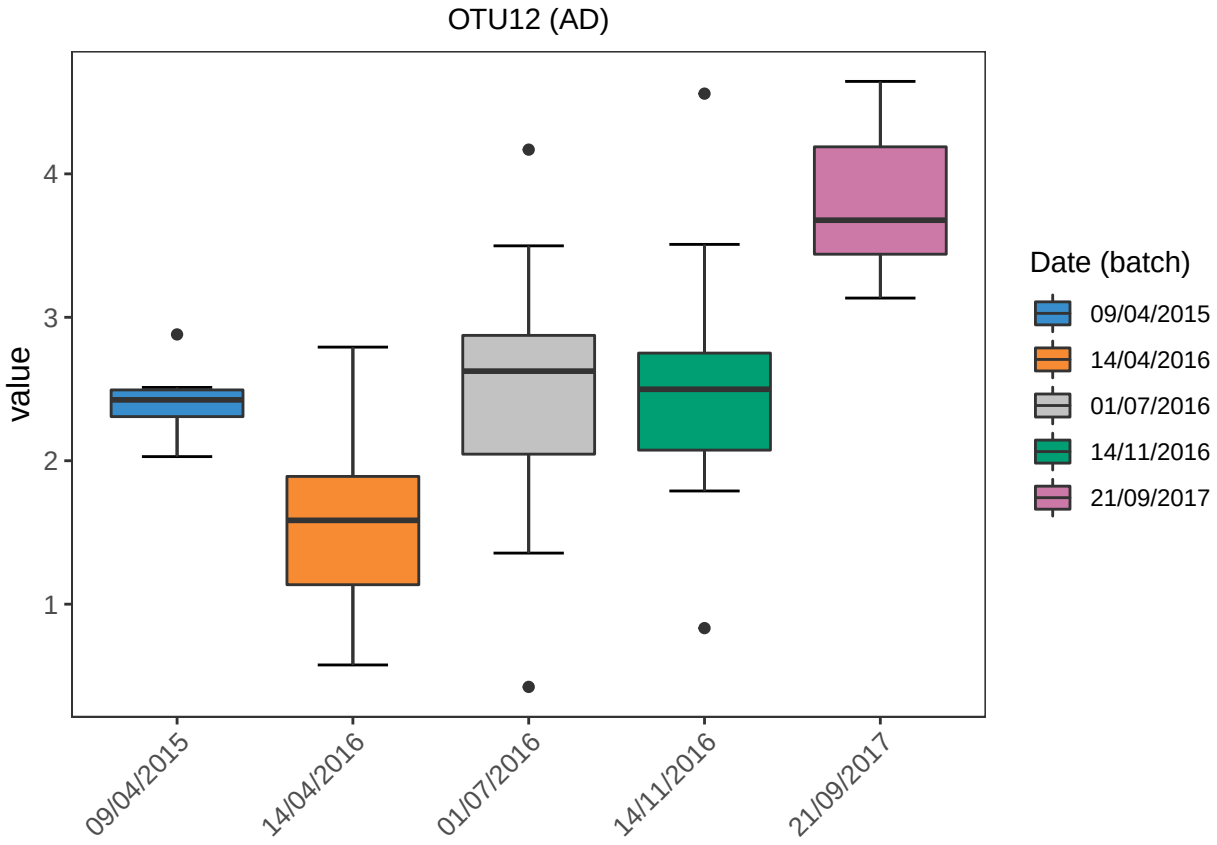
Assuming the data fit the distribution of the linear model (which is the case here after CLR transformation), we can also assess the effect of batch in a linear model on that particular OTU9:

```
sponge.lm = lm(sponge.tss.clr[,9] ~ sponge.trt + sponge.batch)
summary(sponge.lm)
```

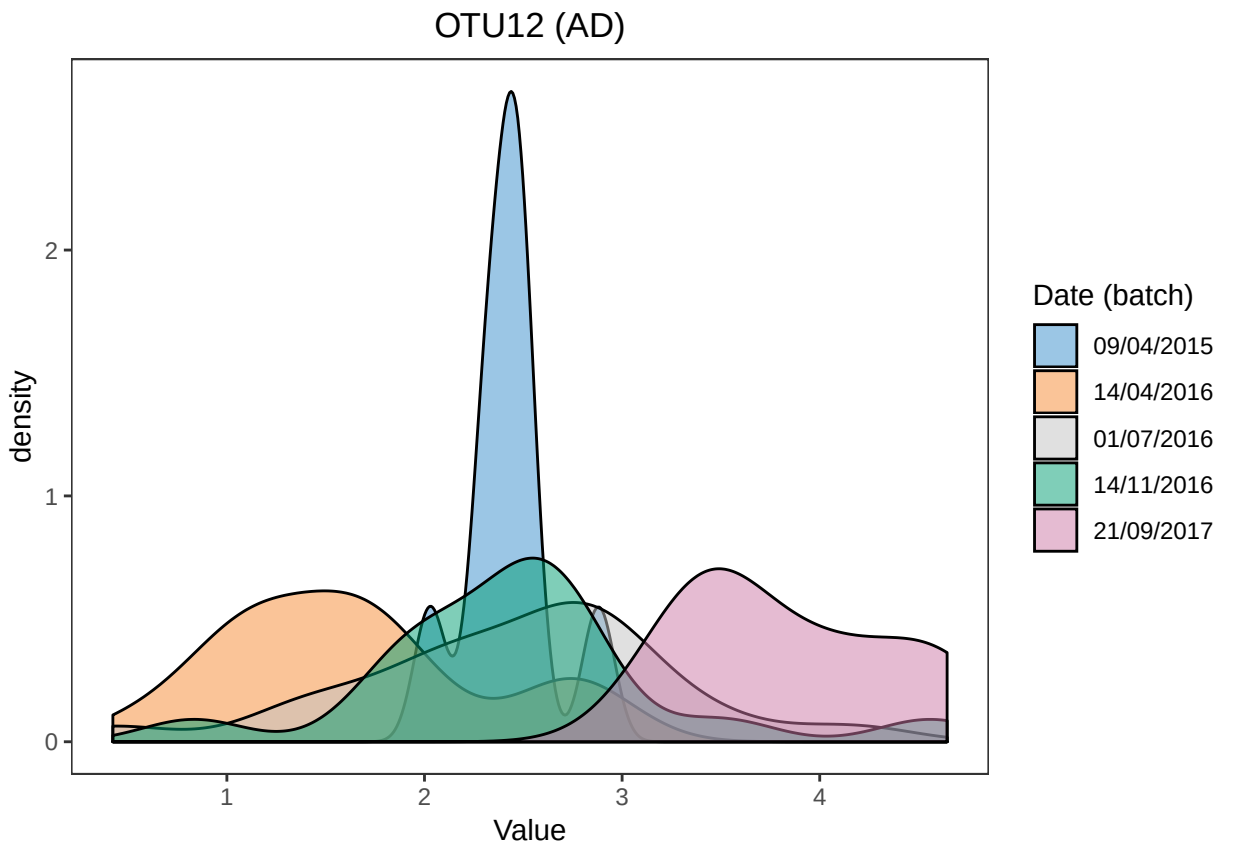
```
##
## Call:
## lm(formula = sponge.tss.clr[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.7849     0.1497  11.922 1.06e-12 ***
## sponge.trtE      0.1065     0.1729   0.616    0.543
## sponge.batch2   -0.7910     0.1729  -4.575 8.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.4236, Adjusted R-squared:  0.3839
## F-statistic: 10.66 on 2 and 29 DF,  p-value: 0.0003391
```

The batch (gel) effect is statistically significant ($P < 0.001$), as indicated in the **sponge.batch2** row.

```
# ad data
ad.before.df = data.frame(value = ad.tss.clr[,1], batch = ad.batch)

box_plot_fun(data = ad.before.df,x = ad.before.df$batch,
             y = ad.before.df$value, title = 'OTU12 (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)
```




```
ggplot(ad.before.df, aes(x = value, fill = batch)) +
  geom_density(alpha = 0.5) + scale_fill_manual(values = color.mixo(1:10)) +
  labs(title = 'OTU12 (AD)',x = 'Value',fill = 'Date (batch)') +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     panel.grid = element_blank())
```



Batch effects in AD data are also easily visualised.

```
ad.lm = lm(ad.tss.clr[,1] ~ ad.trt + ad.batch)
anova(ad.lm)
```

```
## Analysis of Variance Table
##
## Response: ad.tss.clr[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  1.460  1.4605  3.1001   0.08272 .
## ad.batch   4 32.889  8.2222 17.4532 6.168e-10 ***
## Residuals 69 32.506  0.4711
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

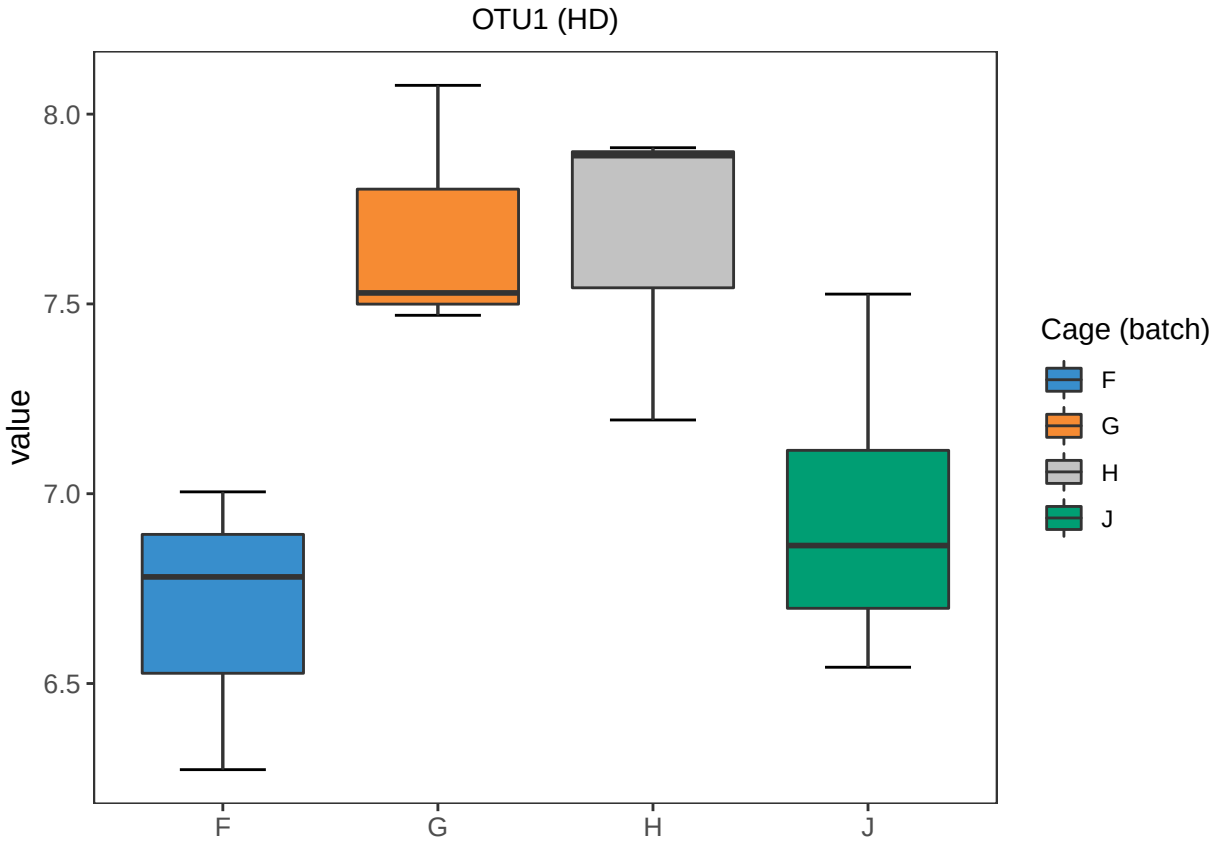
In AD data, the difference between batches (dates) is statistically significant ($P < 0.001$, as tested with ANOVA). We can also obtain P values between each two batch categories.

```
summary(ad.lm)
```

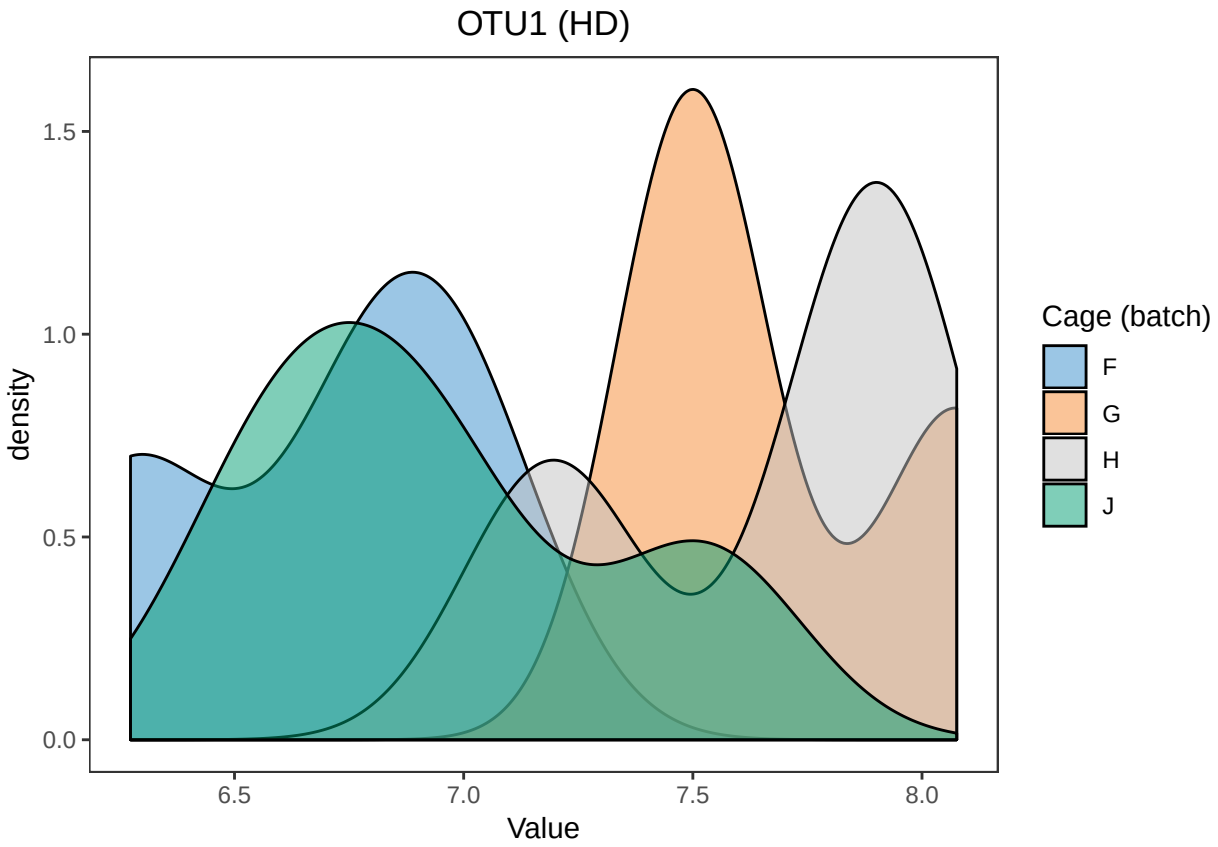
```
##
## Call:
## lm(formula = ad.tss.clr[, 1] ~ ad.trt + ad.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09885 -0.39613 -0.00381  0.36645  1.98185
##
## Coefficients:
```

```
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.311213   0.247768   9.328 7.57e-14 ***
## ad.trt1-2            0.203619   0.171183   1.189  0.23833
## ad.batch14/04/2016 -0.828100   0.287918  -2.876  0.00535 **
## ad.batch01/07/2016  0.007239   0.273672   0.026  0.97897
## ad.batch14/11/2016  0.062689   0.282978   0.222  0.82533
## ad.batch21/09/2017  1.361132   0.306373   4.443 3.30e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6864 on 69 degrees of freedom
## Multiple R-squared:  0.5138, Adjusted R-squared:  0.4786
## F-statistic: 14.58 on 5 and 69 DF,  p-value: 9.665e-10
```

```
# hd data
hd.before.df = data.frame(value = hd.tss.clr[,1], batch = hd.batch)

box_plot_fun(data = hd.before.df, x = hd.before.df$batch,
             y = hd.before.df$value,title = 'OTU1 (HD)',
             batch.legend.title = 'Cage (batch)')
```




```
ggplot(hd.before.df, aes(x = value, fill = batch)) +
  geom_density(alpha = 0.5) + scale_fill_manual(values = color.mixo(1:10)) +
  labs(title = 'OTU1 (HD)',x = 'Value',fill = 'Cage (batch)') +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     panel.grid = element_blank())
```



In HD data, we easily detect the differences between samples within different batches.

```
hd.lm = lm(hd.tss.clr[,1] ~ hd.batch)
anova(hd.lm)
```

```
## Analysis of Variance Table
##
## Response: hd.tss.clr[, 1]
##           Df Sum Sq Mean Sq F value  Pr(>F)
## hd.batch   3 2.4108 0.80359  5.2569 0.02276 *
## Residuals  9 1.3758 0.15286
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As the batch x treatment design of HD data is nested and unbalanced, the linear model with both treatment (genotype) and batch (cage) is unable to fit. We therefore fit a linear model with batch effect only. The difference between cages is statistically significant ($P < 0.05$). But the difference may also be influenced by treatment and we are unable to exclude treatment influence.

2.3 RLE plots

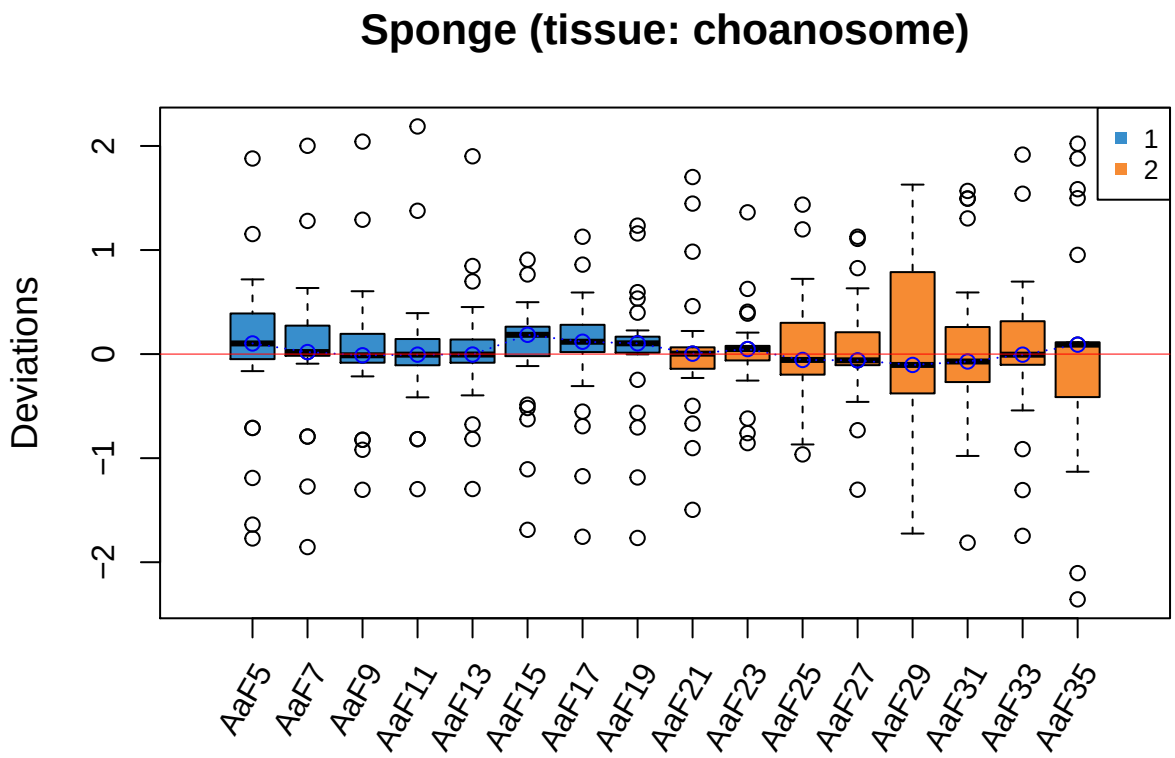
RLE plots can be plotted using 'RleMicroRna' in R package 'AgiMicroRna'. Here, we made some changes on the function 'RleMicroRna', thus we call it 'RleMicroRna2'.

RLE plots are based on the assumption that the majority of microbial variables are unaffected by the effect of interest, and therefore any sample heterogeneity observed - i.e. different distributions and their variances, and medians different from zero, should indicate the presence of batch effects. In our case studies, the treatment information is known, so we generate

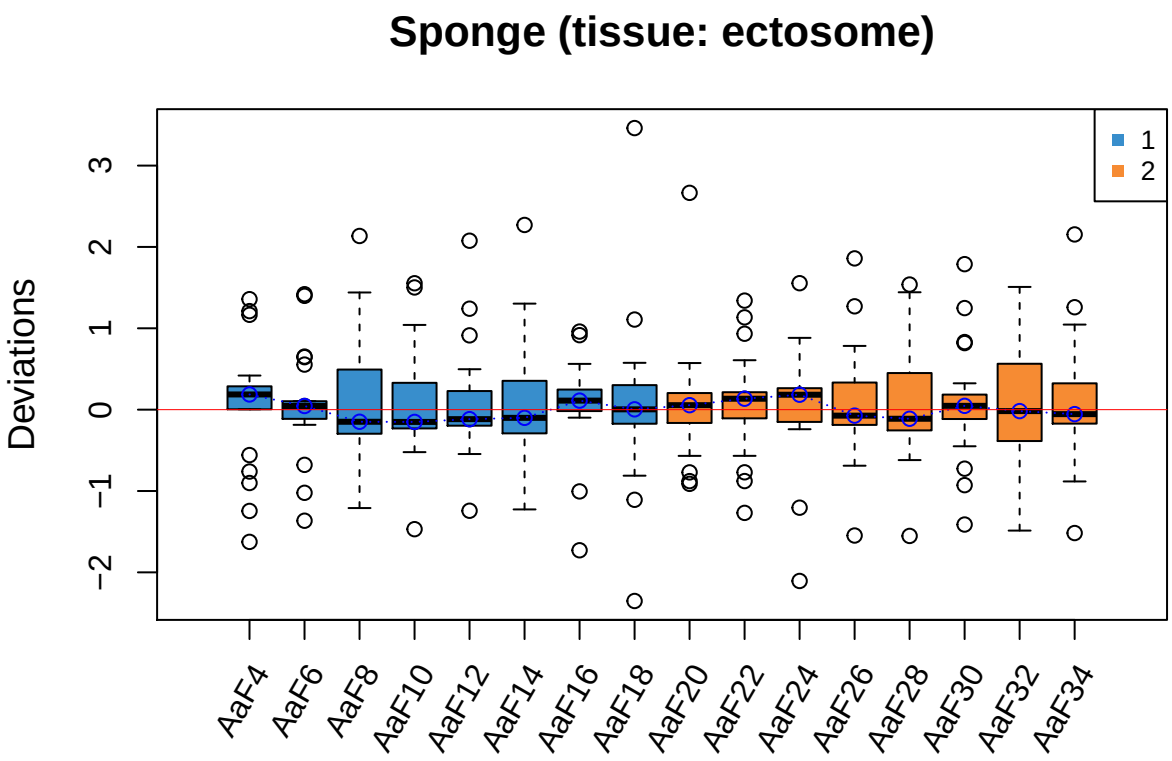
multiple RLE plots per treatment group, as suggested by (Lin et al., 2018).

In sponge data, we group the samples according to the tissue (choanosome / ectosome) and generate two RLE plots:

```
# sponge data
sponge.batch_c = sponge.batch[sponge.trt == 'C']
sponge.batch_e = sponge.batch[sponge.trt == 'E']

sponge.before_c = sponge.tss.clr[sponge.trt == 'C', ]
sponge.before_e = sponge.tss.clr[sponge.trt == 'E', ]


RleMicroRna2(object = t(sponge.before_c), batch = sponge.batch_c,
             maintitle = 'Sponge (tissue: choanosome)')
```

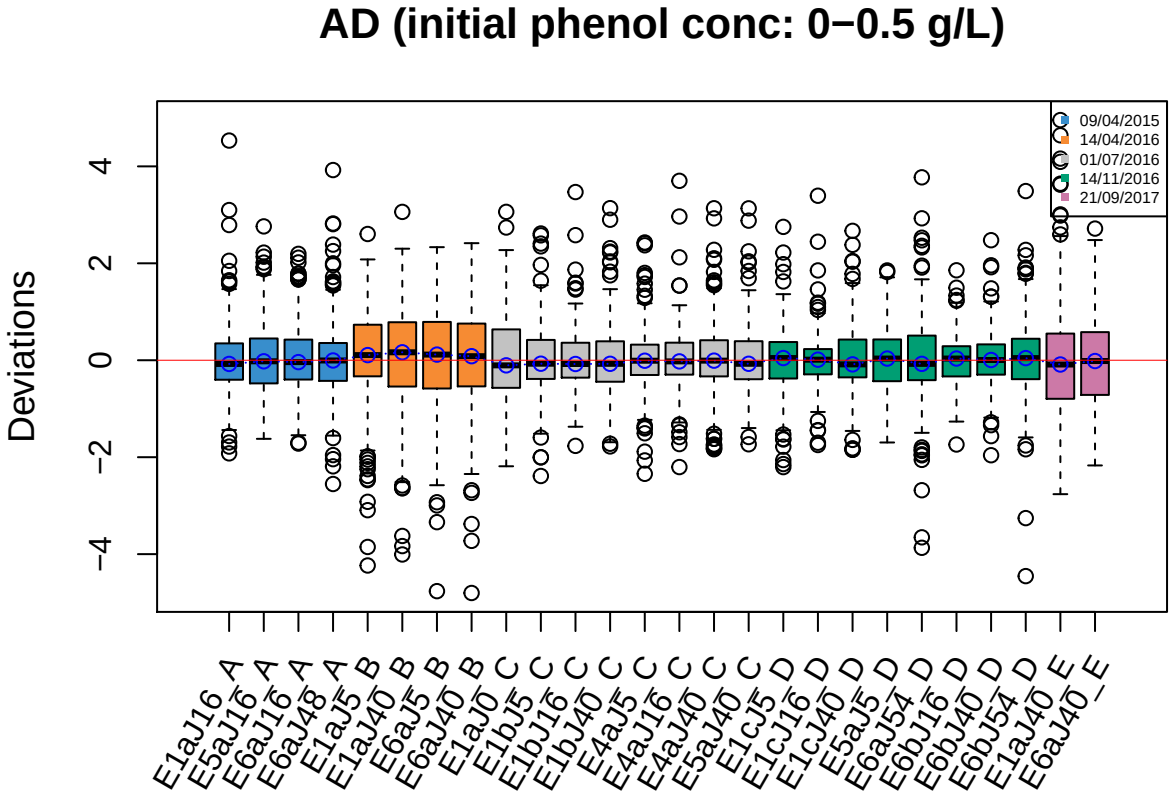



```
RleMicroRna2(object = t(sponge.before_e), batch = sponge.batch_e,
             maintitle = 'Sponge (tissue: ectosome)')
```

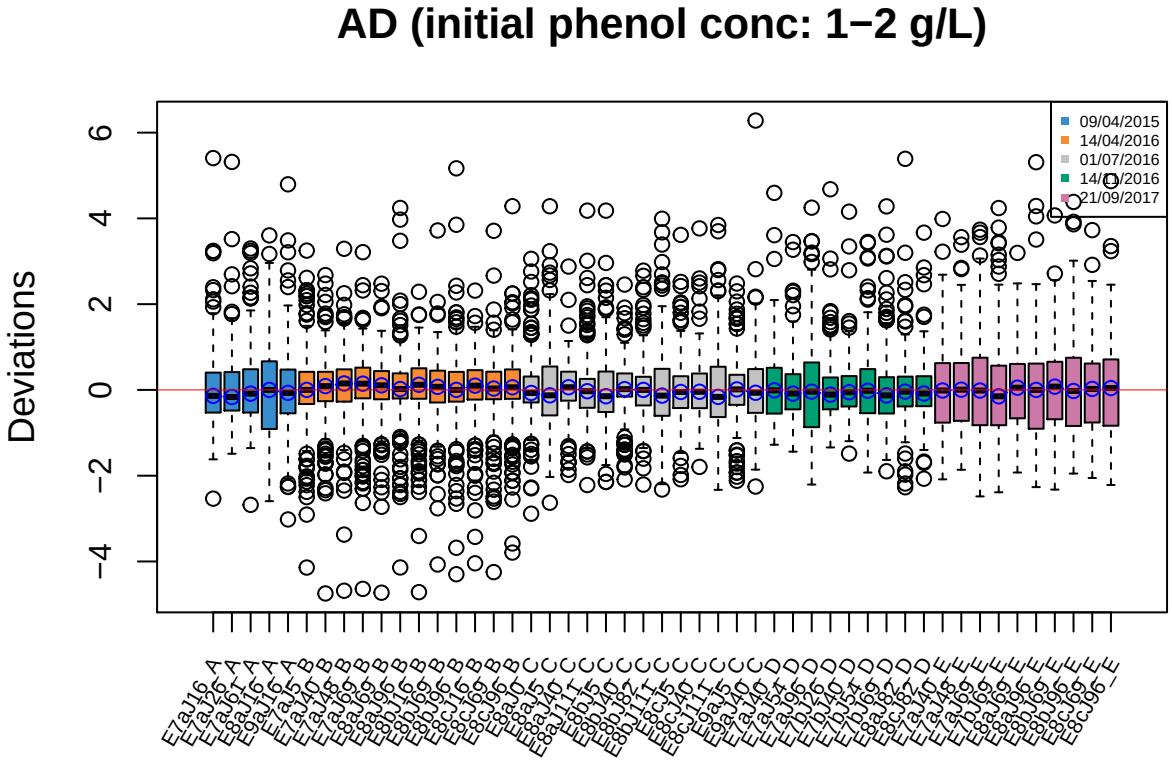


For both RLE plots with samples from different tissues, the batch effect is not obvious as all medians of samples are close to zero, but Gel2 has a greater interquartile range (IQR) than the other samples.

```
# ad data
ad.batch_05 = ad.batch[ad.trt == '0-0.5']
ad.batch_2 = ad.batch[ad.trt == '1-2']

ad.before_05 = ad.tss.clr[ad.trt == '0-0.5', ]
ad.before_2 = ad.tss.clr[ad.trt == '1-2', ]

RleMicroRna2(object = t(ad.before_05), batch = ad.batch_05,
             maintitle = 'AD (initial phenol conc: 0-0.5 g/L)',
             legend.cex = 0.5)
```



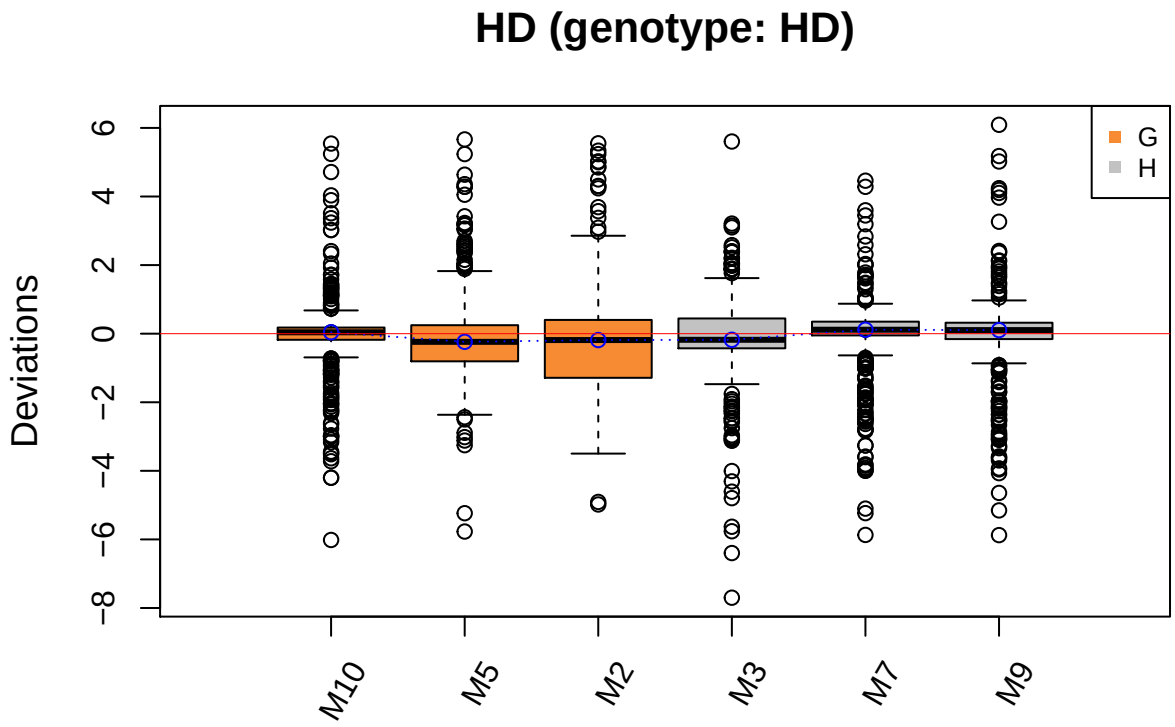

```
RleMicroRna2(object = t(ad.before_2), batch = ad.batch_2,
             maintitle = 'AD (initial phenol conc: 1-2 g/L)',
             cex.xaxis = 0.7, legend.cex = 0.5)
```



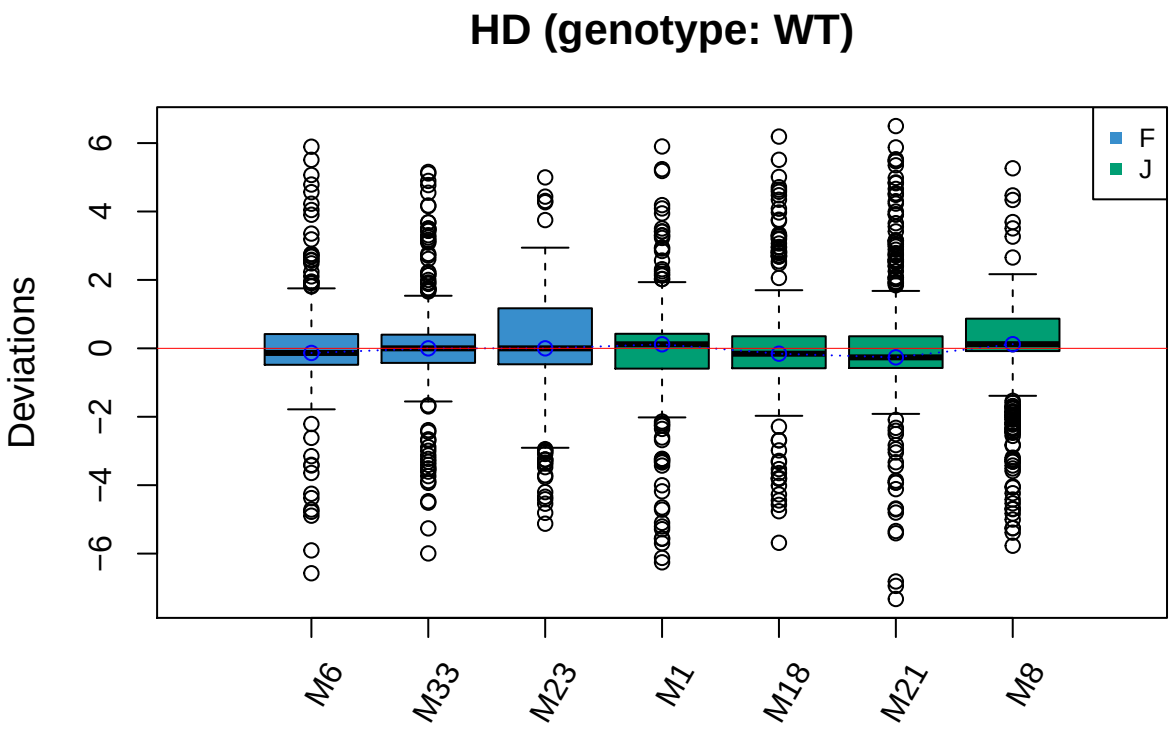
In RLE plots for the AD data, the batch effect is also not obvious as all medians of samples are close to zero, but the samples

dated 14/04/2016 may be affected by batch as they have a greater IQR than the other samples.

```
# hd data
hd.batch_h = hd.batch[hd.trt == 'HD']
hd.batch_w = hd.batch[hd.trt == 'WT']

hd.before_h = hd.tss.clr[hd.trt == 'HD', ]
hd.before_w = hd.tss.clr[hd.trt == 'WT', ]

RleMicroRna2(object = t(hd.before_h), batch = hd.batch_h,
             maintitle = 'HD (genotype: HD)')
```




```
RleMicroRna2(object = t(hd.before_w), batch = hd.batch_w,
             maintitle = 'HD (genotype: WT)')
```



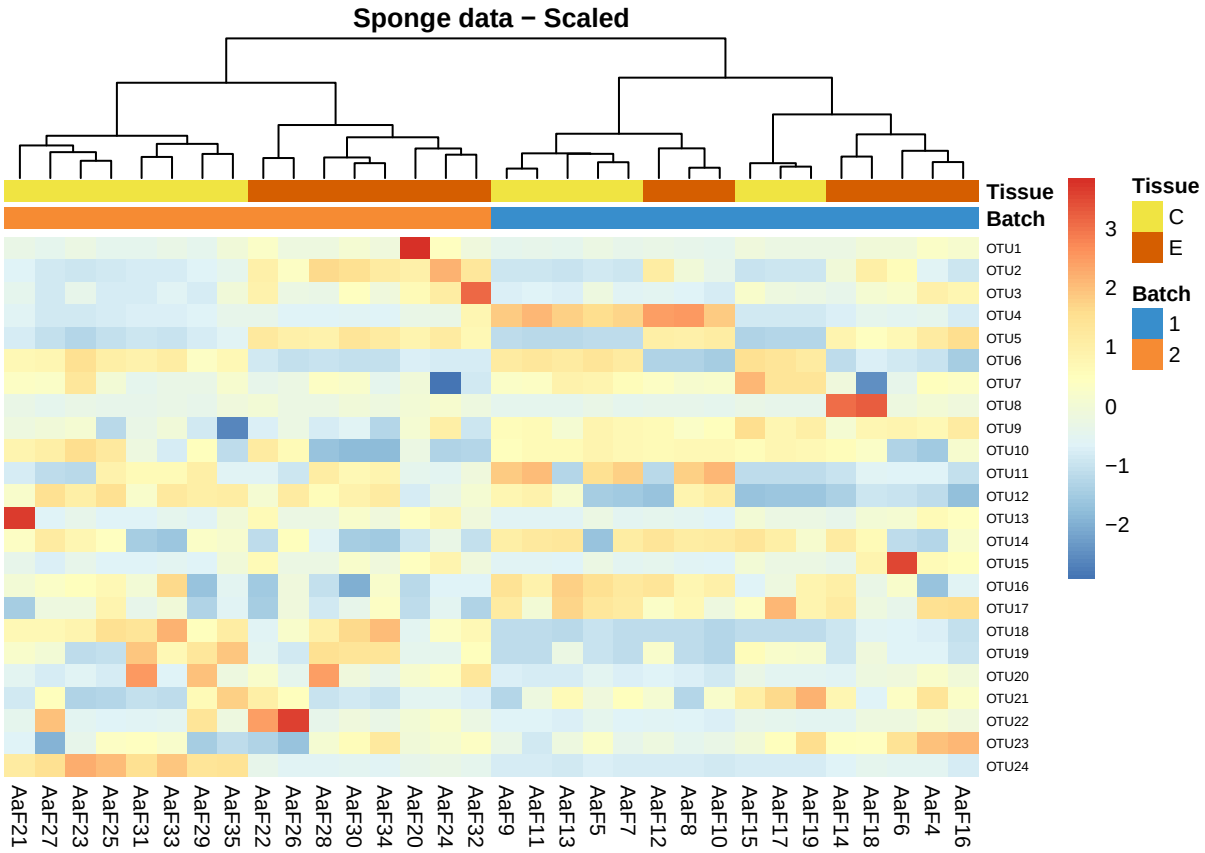
The batch effect in HD data is not easily detected, but Cage G has a greater IQR than the other samples, which may indicate a batch effect.

2.4 Heatmap

Clustering analysis can be used to detect batch effects. Ideally samples with the same treatment will be clustered together, data clustered by batches instead of treatments indicate a batch effect. Heatmaps and dendrograms are two common approaches to visualise the clusters.

```
# Sponge data
# scale on OTUs
sponge.tss.clr.scale = scale(sponge.tss.clr, center = T, scale = T)
# scale on samples
sponge.tss.clr.scale = scale(t(sponge.tss.clr.scale), center = T, scale = T)

sponge.anno_col = data.frame(Batch = sponge.batch, Tissue = sponge.trt)
sponge.anno_metabo_colors = list(Batch = c('1' = '#388ECC', '2' = '#F68B33'),
                                 Tissue = c(C = '#F0E442', E = '#D55E00'))


pheatmap(sponge.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row = 5, fontsize_col = 8,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
         clustering_method = 'ward.D',
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
```

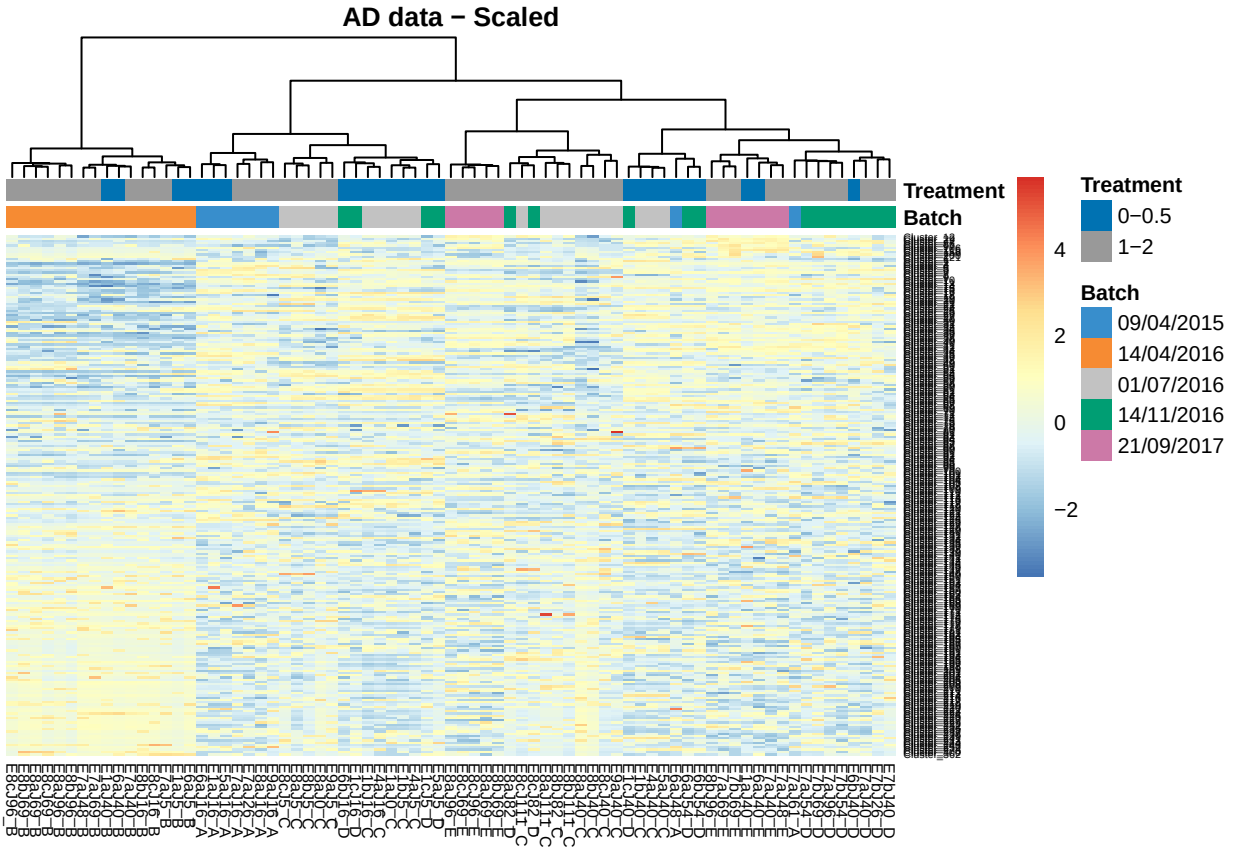
```
         annotation_colors = sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - Scaled')
```



In sponge data, samples are preferentially clustered by batch instead of tissue type, indicating a batch effect.

```
# AD data
ad.tss.clr.scale = scale(ad.tss.clr,center = T, scale = T)
ad.tss.clr.scale = scale(t(ad.tss.clr.scale), center = T, scale = T)

ad.anno_col = data.frame(Batch = ad.batch, Treatment = ad.trt)
ad.anno_metabo_colors = list(Batch = c('09/04/2015' = '#388ECC',
                                       '14/04/2016' = '#F68B33',
                                       '01/07/2016' = '#C2C2C2',
                                       '14/11/2016' = '#009E73',
                                       '21/09/2017' = '#CC79A7'),
                             Treatment = c('0-0.5' = '#0072B2', '1-2' = '#999999'))


pheatmap(ad.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row = 4, fontsize_col = 6,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
         clustering_method = 'ward.D',
```

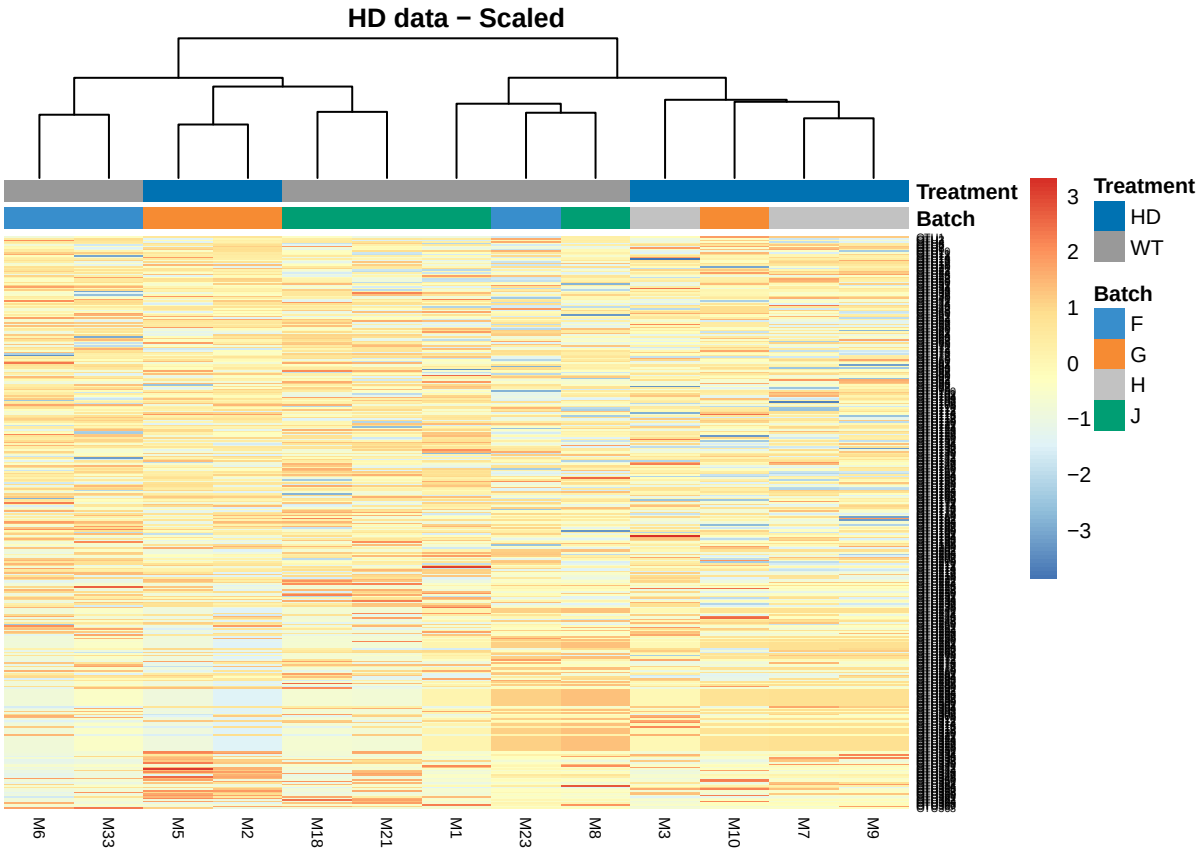
```
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - Scaled')
```



In AD data, samples within batch 14/04/2016 are clustered and distinct from other samples, also indicating a batch effect.

```
# HD data
hd.tss.clr.scale = scale(hd.tss.clr,center = T, scale = T)
hd.tss.clr.scale = scale(t(hd.tss.clr.scale), center = T, scale = T)

hd.anno_col = data.frame(Batch = hd.batch, Treatment = hd.trt)
hd.anno_metabo_colors = list(Batch = c('F' = '#388ECC', 'G' = '#F68B33',
                                       'H' = '#C2C2C2', 'J' = '#009E73'),
                             Treatment = c('HD' = '#0072B2', 'WT' = '#999999'))


pheatmap(hd.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row = 4, fontsize_col = 6,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
         clustering_method = 'ward.D',
         treeheight_row = 30,
```

```
        annotation_col = hd.anno_col,
        annotation_colors = hd.anno_metabo_colors,
        border_color = 'NA',
        main = 'HD data - Scaled')
```



The batch effect in HD data is not very obviously visualised through heatmap, because there is no batch with samples clustered together and separated from other samples.

Chapter 3

Batch effect adjustment

3.1 Accounting for batch effects

Methods that account for batch effects estimate unknown batch effects through matrix decomposition and / or assign a known or estimated batch as a covariate with linear models.

3.1.1 Linear model and linear mixed model

LM and LMM are suitable for known batch effects, and can consider batch x treatment interaction and deal with unbalanced batch x treatment design. But they are univariate and rely on a Gaussian likelihood assumption, which may not apply to zero-inflated microbiome data despite CLR transformation.

We fit a linear model for both sponge and AD data.

```
# Sponge data
sponge.trt_p <- apply(sponge.tss.clr, 2, FUN = function(x){
  res.lm <- lm(x ~ sponge.trt + sponge.batch)
  summary.res = summary(res.lm)
  p = summary.res$coefficients[2,4]
})

sponge.trt_q = p.adjust(sponge.trt_p,method = 'fdr')

# AD data
ad.trt_p <- apply(ad.tss.clr, 2, FUN = function(x){
  res.lm <- lm(x ~ ad.trt + ad.batch)
  summary.res = summary(res.lm)
  p = summary.res$coefficients[2,4]
})

ad.trt_q = p.adjust(ad.trt_p,method = 'fdr')
```

As the batch x treatment design of AD data is unbalaced, we fit a linear mixed model considering batch (cage) as random effects.

```
# HD data
hd.trt_p <- apply(hd.tss.clr, 2, FUN = function(x){
  res.lmm <- lmer(x ~ hd.trt + (1|hd.batch))
  summary.res = summary(res.lmm)
```

```
  p = summary.res$coefficients[2,5]
})

hd.trt_q = p.adjust(hd.trt_p,method = 'fdr')
```

3.1.2 SVA

SVA can account for unknown batch effects. But it is univariate, relies on a Gaussian likelihood assumption and implicitly introduces a correlation between treatment and batch.

The *sva* function performs two different steps. First it identifies the number of latent factors that need to be estimated. The number of factors can be estimated using the *num.sv*.

```
# sponge data
sponge.mod = model.matrix(~sponge.trt) # full model
sponge.mod0 = model.matrix(~1,data = sponge.trt) # null model
sponge.sva.n <- num.sv(dat = t(sponge.tss.clr), mod = sponge.mod)
```

Next we apply the *sva* function to estimate the surrogate variables:

```
sponge.sva = sva(t(sponge.tss.clr), sponge.mod, sponge.mod0, n.sv = sponge.sva.n)
```

```
## Number of significant surrogate variables is:  3
## Iteration (out of 5 ):1  2  3  4  5
```

We include the estimated surrogate variables in both the null and full models. The reason is that we want to adjust for the surrogate variables, so we treat them as adjustment variables that must be included in both models. The *f.pvalue* function is then used to calculate parametric F-test P-values and Q-values (adjusted P-values) for each OTU of sponge data.

```
sponge.mod.bat = cbind(sponge.mod,sponge.sva$sv)
sponge.mod0.bat = cbind(sponge.mod0,sponge.sva$sv)

sponge.sva.trt_p = f.pvalue(t(sponge.tss.clr),sponge.mod.bat,sponge.mod0.bat)
sponge.sva.trt_q = p.adjust(sponge.sva.trt_p,method="fdr")
```

Now these P-values and Q-values are accounting for surrogate variables (estimated batch effects).

We also apply SVA on both AD and HD data.

```
# ad data
ad.mod = model.matrix(~ad.trt)
ad.mod0 = model.matrix(~1,data = ad.trt)
ad.sva.n <- num.sv(dat = t(ad.tss.clr), mod = ad.mod)
ad.sva = sva(t(ad.tss.clr), ad.mod, ad.mod0, n.sv = ad.sva.n)
```

```
## Number of significant surrogate variables is:  6
## Iteration (out of 5 ):1  2  3  4  5
```

```
ad.mod.bat = cbind(ad.mod,ad.sva$sv)
ad.mod0.bat = cbind(ad.mod0,ad.sva$sv)
ad.sva.trt_p = f.pvalue(t(ad.tss.clr),ad.mod.bat,ad.mod0.bat)
ad.sva.trt_q = p.adjust(ad.sva.trt_p,method="fdr")

# hd data
hd.mod = model.matrix(~hd.trt)
hd.mod0 = model.matrix(~1,data = hd.trt)
```

```
hd.sva.n <- num.sv(dat = t(hd.tss.clr), mod = hd.mod)
hd.sva = sva(t(hd.tss.clr), hd.mod, hd.mod0, n.sv = hd.sva.n)

## Number of significant surrogate variables is:  3
## Iteration (out of 5 ):1  2  3  4  5
```

```
hd.mod.bat = cbind(hd.mod,hd.sva$sv)
hd.mod0.bat = cbind(hd.mod0,hd.sva$sv)
hd.sva.trt_p = f.pvalue(t(hd.tss.clr),hd.mod.bat,hd.mod0.bat)
hd.sva.trt_q = p.adjust(hd.sva.trt_p,method="fdr")
```

3.1.3 RUV2

RUV2 estimates and accounts for unknown batch effects. But it needs negative control variables that are affected by batch effects but not treatment effects.

In the real world, we design negative control variables that are not affected by treatment effects only, we are not sure but assume these controls are affected by batch effects. RUV2 can only account for the difference captured by these controls.

Since our three datasets do not have negative control variables, we use a linear model (or linear mixed model) to identify OTUs less likely to be affected by treatment effects as negative controls to fit the assumptions. The P-values of treatment effects calculated in section 'Linear model and linear mixed model' are therefore used here.

Note: it is not a optimal way to obtain negative control variables like what we did. Negative control variables should be known and designed before the experiment. The reason we obtained controls using this way is just to fit the assumption and demonstrate hwo to use RUV series.

```
# sponge data
sponge.nc = sponge.trt_q > 0.05

# AD data
ad.nc = ad.trt_q > 0.05

# HD data
hd.nc = hd.trt_p > 0.05
```

We then apply the *RUV2* function. This function needs to specify the number of unwanted factors (components of negative controls). We therefore specify k = 3, but the number is very subjective and can be changed to any number. After that, we extract the P-values of treatment effects considering unwanted batch effects and then Q-values after FDR adjustment.

```
# sponge data
sponge.ruv2 <- RUV2(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc, k = 3) # k is subjective
sponge.ruv2.trt_p <- sponge.ruv2$p
sponge.ruv2.trt_q <- p.adjust(sponge.ruv2.trt_p,method="fdr")

# AD data
ad.ruv2 <- RUV2(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc, k = 3) # k is subjective
ad.ruv2.trt_p <- ad.ruv2$p
ad.ruv2.trt_q <- p.adjust(ad.ruv2.trt_p,method="fdr")

# HD data
hd.ruv2 <- RUV2(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc, k = 3) # k is subjective
hd.ruv2.trt_p <- hd.ruv2$p
hd.ruv2.trt_q <- p.adjust(hd.ruv2.trt_p,method="fdr")
```

3.1.4 RUV4

RUV4 is an updated version of RUV2 that uses negative control variables and the residual matrix that has no treatment effect to estimate unwanted batch effects.

RUV4 also needs to specify the number of unwanted factors as *RUV2*, and here we use a function called *getK* to estimate this number. This function is only for *RUV4* and the estimated k is not always suitable. If the estimated k is 0, k can be forced to 1 and still to account for unwanted variation captured from negative controls.

```
# sponge data
sponge.k.obj = getK(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc)
sponge.k = sponge.k.obj$k
sponge.k = ifelse(sponge.k !=0, sponge.k, 1)
sponge.ruv4 <- RUV4(Y = sponge.tss.clr, X = sponge.trt, ctl = sponge.nc, k = sponge.k)
sponge.ruv4.trt_p <- sponge.ruv4$p
sponge.ruv4.trt_q <- p.adjust(sponge.ruv4.trt_p,method="fdr")

# AD data
ad.k.obj = getK(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc)
ad.k =ad.k.obj$k
ad.k = ifelse(ad.k !=0, ad.k, 1)
ad.ruv4 <- RUV4(Y = ad.tss.clr, X = ad.trt, ctl = ad.nc, k = ad.k)
ad.ruv4.trt_p <- ad.ruv4$p
ad.ruv4.trt_q <- p.adjust(ad.ruv4.trt_p,method="fdr")

# HD data
hd.k.obj = getK(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc)
hd.k = hd.k.obj$k
hd.k = ifelse(hd.k !=0, hd.k, 1)
hd.ruv4 <- RUV4(Y = hd.tss.clr, X = hd.trt, ctl = hd.nc, k = hd.k)
hd.ruv4.trt_p <- hd.ruv4$p
hd.ruv4.trt_q <- p.adjust(hd.ruv4.trt_p,method="fdr")
```

3.2 Correcting for batch effects

3.2.1 BMC (batch mean centering)

We centers data within a batch across all variables. As a result, each batch mean is standardised to zero. The disadvantages of BMC are it is univariate and not optimal for non-Gaussian distributed microbiome data.

```
# Sponge data
sponge.b1 = scale(sponge.tss.clr[sponge.batch== 1,],center = TRUE, scale = FALSE)
sponge.b2 = scale(sponge.tss.clr[sponge.batch== 2,],center = TRUE, scale = FALSE)
sponge.bmc = rbind(sponge.b1,sponge.b2)
sponge.bmc = sponge.bmc[rownames(sponge.tss.clr),]

##############
# AD data
ad.b1 = scale(ad.tss.clr[ad.batch=="09/04/2015",],center = TRUE, scale = FALSE)
ad.b2 = scale(ad.tss.clr[ad.batch=="14/04/2016",],center = TRUE, scale = FALSE)
ad.b3 = scale(ad.tss.clr[ad.batch=="14/11/2016",],center = TRUE, scale = FALSE)
ad.b4 = scale(ad.tss.clr[ad.batch=="01/07/2016",],center = TRUE, scale = FALSE)
ad.b5 = scale(ad.tss.clr[ad.batch=="21/09/2017",],center = TRUE, scale = FALSE)
```

```
ad.bmc = rbind(ad.b1,ad.b2,ad.b3,ad.b4,ad.b5)
ad.bmc = ad.bmc[rownames(ad.tss.clr),]
```

3.2.2 ComBat

ComBat works on known and systematic batch effects. If the treatment information is known, the option *mod* can be fitted with a full model having treatment informationn to efficiently maintain enough treatment variation, like what we do here on sponge and AD data. The *mod* can also be NULL if the treatment information is unknown.

```
# Sponge data
sponge.combat <- t(ComBat(t(sponge.tss.clr),batch=sponge.batch,mod = sponge.mod,par.prior=F,prior.plots
```

```
## Found2batches
## Adjusting for1covariate(s) or covariate level(s)
## Standardizing Data across genes
## Fitting L/S model and finding priors
## Finding nonparametric adjustments
## Adjusting the Data
```

```
##############
# AD data
ad.combat <- t(ComBat(t(ad.tss.clr),batch=ad.batch,mod = ad.mod, par.prior=F,prior.plots = F))
```

```
## Found5batches
## Adjusting for1covariate(s) or covariate level(s)
## Standardizing Data across genes
## Fitting L/S model and finding priors
## Finding nonparametric adjustments
## Adjusting the Data
```

3.2.3 removeBatchEffect

removeBatchEffect is a function implemented in the LIMMA package that fits a linear model for each variable given a series of conditions as explanatory variables, including the batch effect and treatment effect. Contrary to a standard linear or linear mixed models (see section 'linear model and linear mixed model') that simultaneously estimate treatment and batch effects, *removeBatchEffect* subtracts the batch effect from the original data, resulting in a residual matrix that contains any and only treatment effect. The option *design* is the same as option *mod* in ComBat.

```
# Sponge data
sponge.limma <- t(removeBatchEffect(t(sponge.tss.clr),batch = sponge.batch,design = sponge.mod))

##############
ad.limma <- t(removeBatchEffect(t(ad.tss.clr),batch = ad.batch,design = ad.mod))
```

3.2.4 FAbatch

FAbatch is a combination of location-scale adjustment and factor analysis. We fit *y* with the treatment information and *batch* with batch information. Since this method only accepts numeric variables, the levels of variables are changed to

be numeric (e.g. '1', '2' and so on). However, FAbatch is unable to converge on both sponge data and AD data. This may influence the effect of batch correction. We therefore do not compare the results from FAbatch with those from other methods.

```
# sponge data
sponge.fabatch.obj = fabatch(x = sponge.tss.clr,y = as.factor(as.numeric(sponge.trt)), batch = sponge.ba
sponge.fabatch <- sponge.fabatch.obj$xadj

# ad data
ad.fabatch.obj = fabatch(x = ad.tss.clr,y = as.factor(as.numeric(ad.trt)), batch = as.factor(as.numeric
ad.fabatch <- ad.fabatch.obj$xadj
```

3.2.5 percentile normalisation

Percentile normalisation (PN) is developed for microbiome data integration. For each batch, it transforms the relative abundance of control samples to their own percentiles while converting the relative abundance of case samples into the percentiles of their corresponding control distribution. The method thus only applies to case-control studies, and may lose a large amount of information as it uses percentiles rather than the original values.

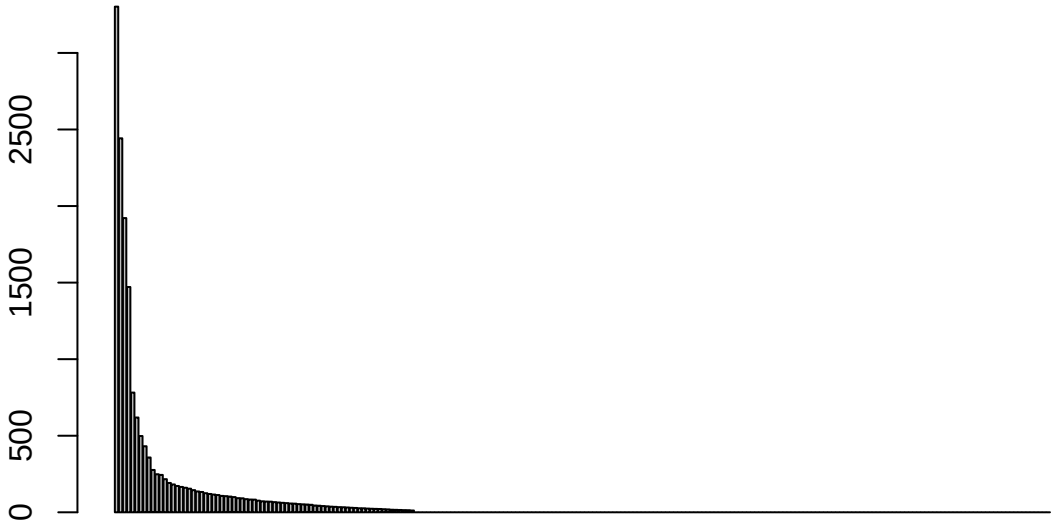
```
sponge.percentile = percentile_norm(data = sponge.tss, batch = sponge.batch, trt = sponge.trt)

# ad data
ad.percentile = percentile_norm(data = ad.tss, batch = ad.batch, trt = ad.trt)
```

3.2.6 SVD-based method

We center and scale the data before SVD. After SVD, we deflate the first component, which has the highest variation and is assumed to related to batch effects.

```
#################
# sponge data
sponge.sd = apply(sponge.tss.clr,2,sd)
sponge.mean = apply(sponge.tss.clr,2,mean)
sponge.X = scale(sponge.tss.clr,center = T,scale = T)

sponge.m = crossprod(sponge.X)
sponge.m.svd = svd(sponge.m)
#barplot(sponge.m.svd$d)

sponge.a1 = sponge.m.svd$u[,1]
sponge.b1 = sponge.m.svd$v[,1]

# component 1
sponge.t1 = sponge.X %*% sponge.a1 / drop(sqrt(crossprod(sponge.a1)))
sponge.c1 = crossprod(sponge.X, sponge.t1) / drop(crossprod(sponge.t1))
sponge.svd.defl.matrix1  = sponge.X - sponge.t1 %*% t(sponge.c1)

## add back mean and variance
sponge.svd = sponge.svd.defl.matrix1
sponge.svd[1:nrow(sponge.svd),1:ncol(sponge.svd)] = NA
for(i in 1:ncol(sponge.svd.defl.matrix1)){
  for(j in 1:nrow(sponge.svd.defl.matrix1)){
    sponge.svd[j,i] = sponge.svd.defl.matrix1[j,i]*sponge.sd[i] + sponge.mean[i]
```

```
  }
}

#####################
# ad data
ad.sd = apply(ad.tss.clr,2,sd)
ad.mean = apply(ad.tss.clr,2,mean)
ad.X = scale(ad.tss.clr,center = T,scale = T)

ad.m = crossprod(ad.X)
ad.m.svd = svd(ad.m)
barplot(ad.m.svd$d)
```




```
ad.a1 = ad.m.svd$u[,1]
ad.b1 = ad.m.svd$v[,1]

# component 1
ad.t1 = ad.X %*% ad.a1 / drop(sqrt(crossprod(ad.a1)))
ad.c1 = crossprod(ad.X, ad.t1) / drop(crossprod(ad.t1))
ad.svd.defl.matrix1  = ad.X - ad.t1 %*% t(ad.c1)

# #
ad.svd = ad.svd.defl.matrix1
ad.svd[1:nrow(ad.svd),1:ncol(ad.svd)] = NA
for(i in 1:ncol(ad.svd.defl.matrix1)){
  for(j in 1:nrow(ad.svd.defl.matrix1)){
    ad.svd[j,i] = ad.svd.defl.matrix1[j,i]*ad.sd[i] + ad.mean[i]
  }
}
```

3.2.7 RUVIII

RUVIII needs not only negative control variables as RUV2 and RUV4, but also technical sample replicates. As only AD data have sample replicates, RUVIII is only applied on AD data.

In contrast to RUV2 and RUV4, RUVIII is a multivariate method that accounts for the dependency between microbial

variables, but it is currently limited to the correction of technical and computational sources of batch effects.

Note: RUVIII requires the number of negative control variables to be larger than the sample size in order to fully use the sample information.

```
#####################
# ad data only
ad.replicates = ad.metadata$sample_name.data.extraction
ad.replicates.matrix = replicate.matrix(ad.replicates)

ad.ruvIII <- RUVIII(Y=ad.tss.clr,M = ad.replicates.matrix, ctl = ad.nc)
rownames(ad.ruvIII) = rownames(ad.tss.clr)
```

Chapter 4

Methods evaluation

After batch effect adjustment, it is essential to evaluate its effectiveness. For simplicity, we focus only on strategies to assess batch effect *correction* methods, as methods that account for the batch effect often imply it has been assessed internally in the statistical model.

4.1 Diagnostic plots

Diagnostic plots are useful to visually evaluate for post-correction batch effects, as presented earlier in section 'batch effect detection'.

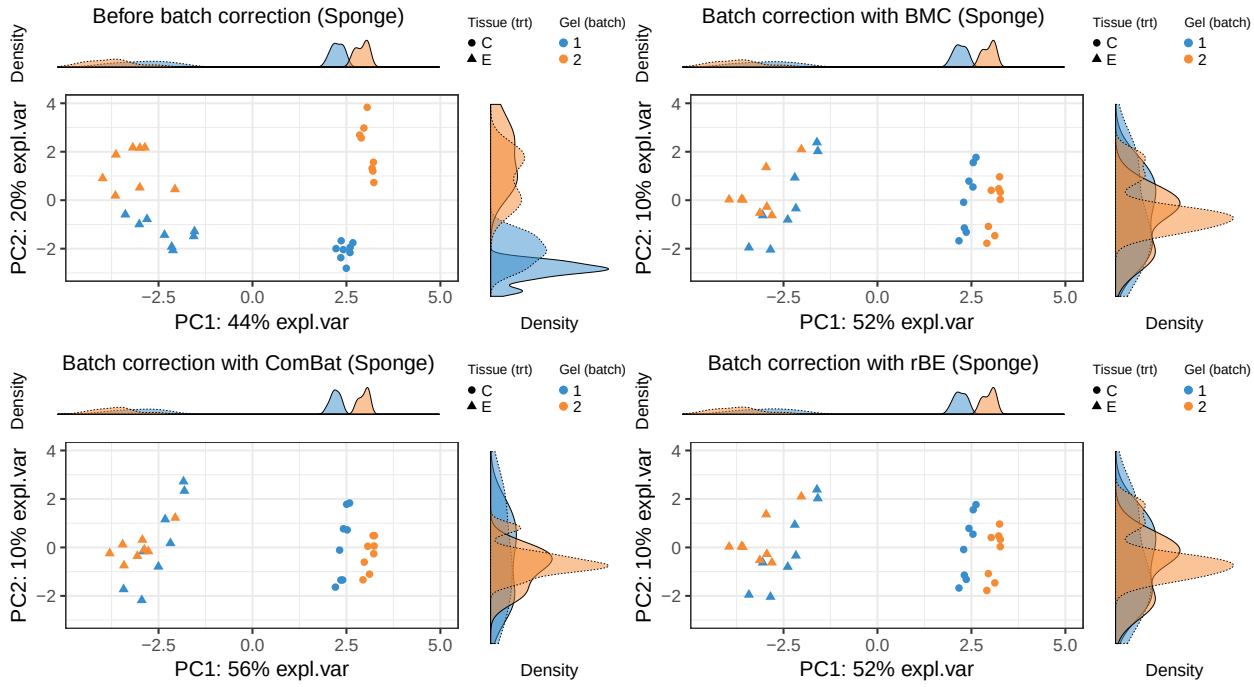
4.1.1 Principal component analysis (PCA) with density plot per component

We apply PCA on both sponge and AD data before and after correction with different methods.

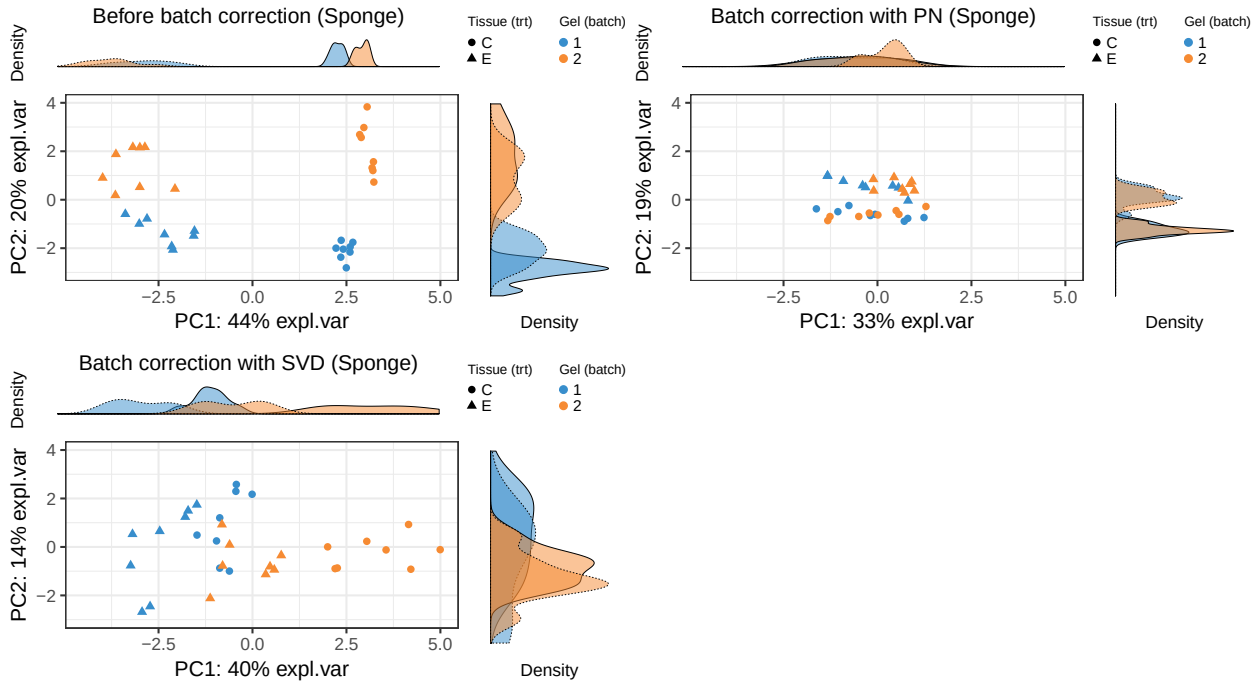
```
# sponge data
sponge.pca.before = pca(sponge.tss.clr, ncomp = 3)
sponge.pca.bmc = pca(sponge.bmc, ncomp = 3)
sponge.pca.combat = pca(sponge.combat, ncomp = 3)
sponge.pca.limma = pca(sponge.limma, ncomp = 3)
sponge.pca.percentile = pca(sponge.percentile, ncomp = 3)
sponge.pca.svd = pca(sponge.svd, ncomp = 3)

# ad data
ad.pca.before = pca(ad.tss.clr, ncomp = 3)
ad.pca.bmc = pca(ad.bmc, ncomp = 3)
ad.pca.combat = pca(ad.combat, ncomp = 3)
ad.pca.limma = pca(ad.limma, ncomp = 3)
ad.pca.percentile = pca(ad.percentile, ncomp = 3)
ad.pca.svd = pca(ad.svd, ncomp = 3)
ad.pca.ruv = pca(ad.ruvIII, ncomp = 3)
```

We then plot these PCA sample plots with denisty plots per PC.

```
grid.arrange(sponge.pca.plot.before, sponge.pca.plot.bmc, sponge.pca.plot.combat,sponge.pca.plot.limma,
```

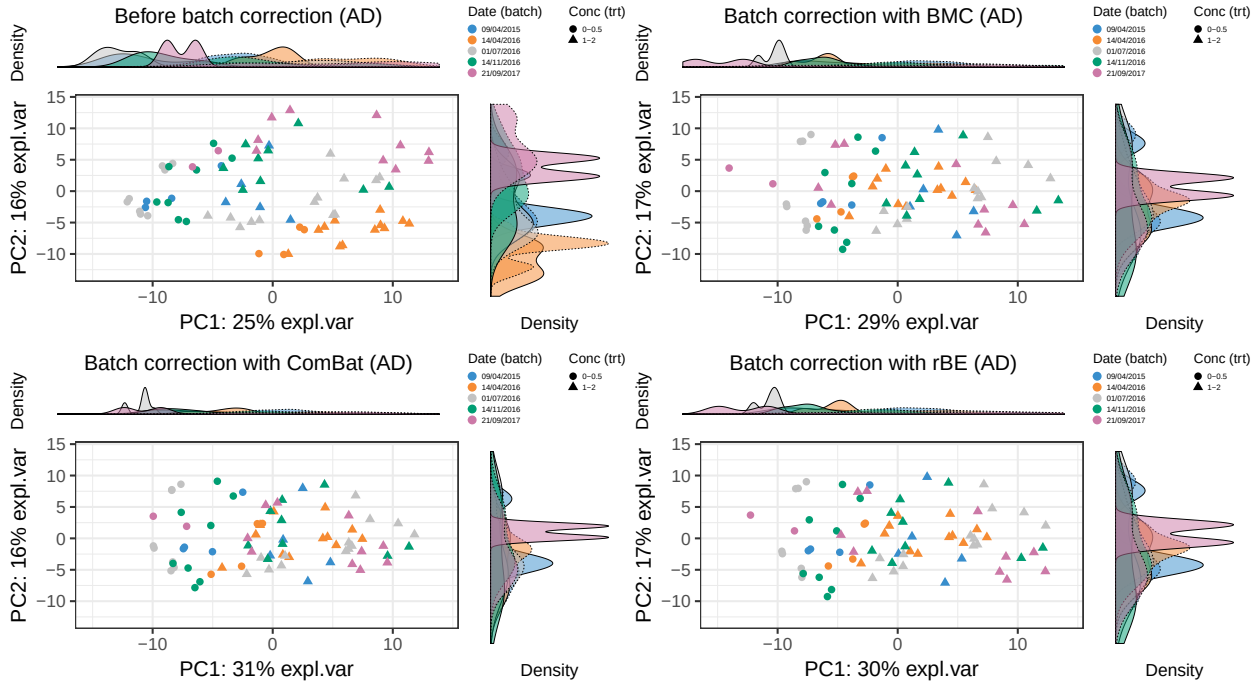



```
grid.arrange(sponge.pca.plot.before, sponge.pca.plot.percentile,sponge.pca.plot.svd,ncol=2)
```

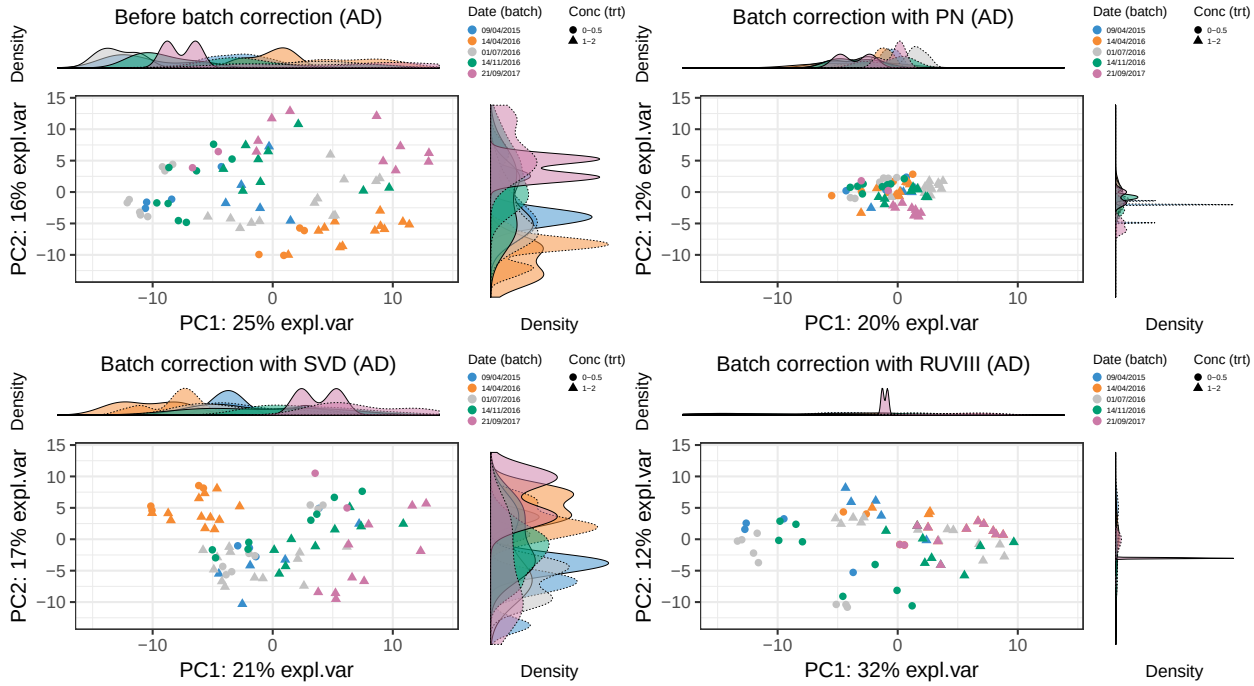


SVD performed the worse, compared to BMC, removeBatchEffect and percentile normalisation, as it did not remove batch effects but removed tissue variation instead.

```
grid.arrange(ad.pca.plot.before, ad.pca.plot.bmc, ad.pca.plot.combat,ad.pca.plot.limma,ncol=2)
```




```
grid.arrange(ad.pca.plot.before, ad.pca.plot.percentile,ad.pca.plot.svd,ad.pca.plot.ruv,ncol=2)
```



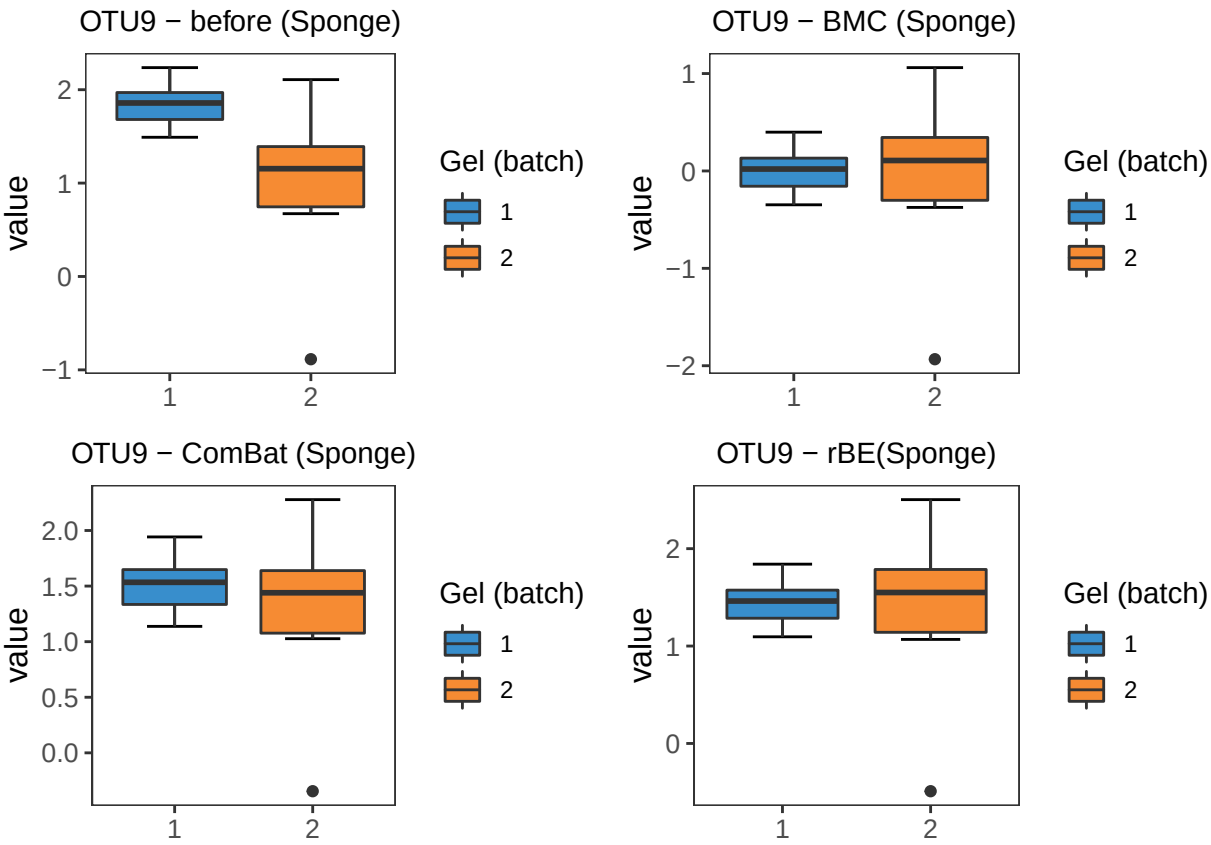
BMC, removeBatchEffect, percentile normalisation and RUVIII removed the batch effects, and maintained the treatment effects. SVD did not removed the batch effect but decreased the treatment effects.

4.1.2 Density plot and box plot

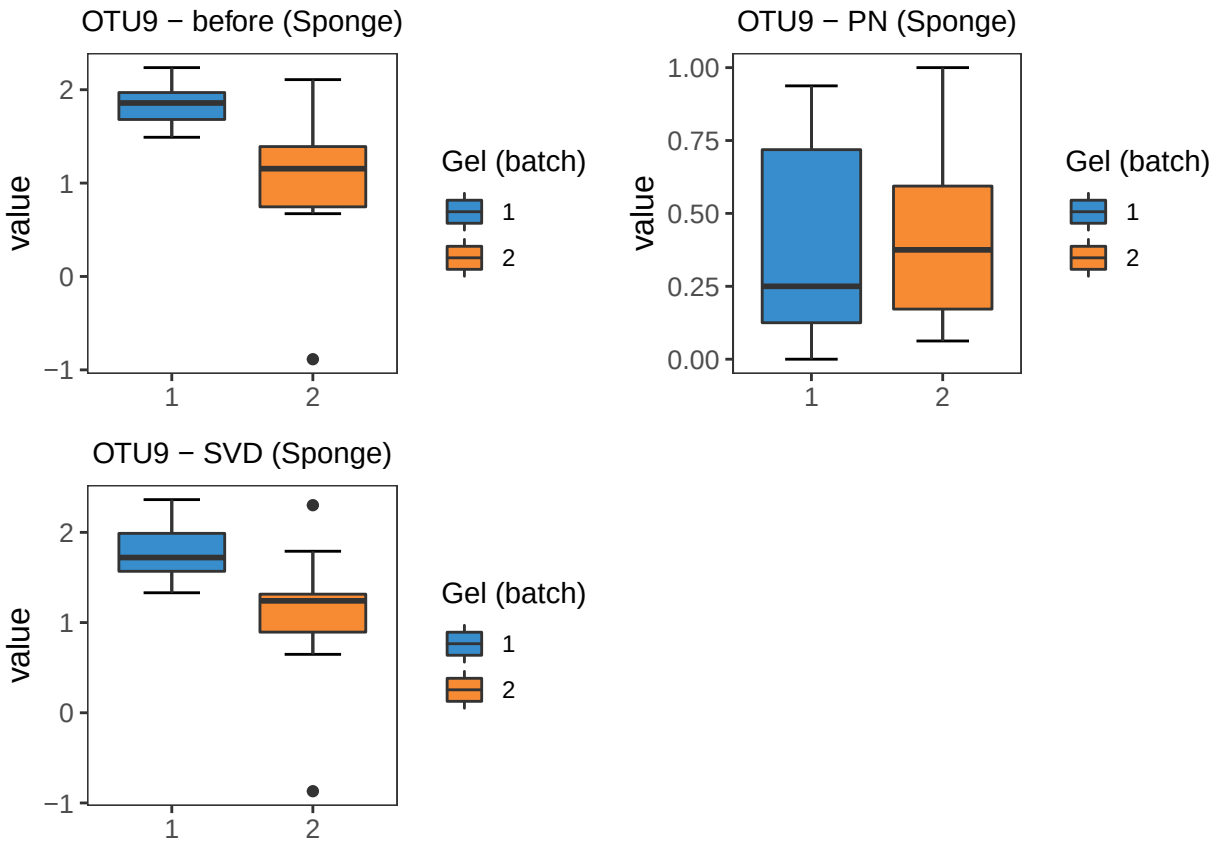
```
################
## sponge data
```

```
sponge.before.df = data.frame(value = sponge.tss.clr[,9], batch = sponge.batch)
sponge.boxplot.before <- box_plot_fun(data = sponge.before.df,x=sponge.before.df$batch,
             y=sponge.before.df$value,title = 'OTU9 - before (Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.bmc.df = data.frame(value = sponge.bmc[,9], batch = sponge.batch)
sponge.boxplot.bmc <-box_plot_fun(data = sponge.bmc.df,x=sponge.bmc.df$batch,
             y=sponge.bmc.df$value,title = 'OTU9 - BMC (Sponge)',
             batch.legend.title = 'Gel (batch)')


sponge.combat.df = data.frame(value = sponge.combat[,9], batch = sponge.batch)
sponge.boxplot.combat <-box_plot_fun(data = sponge.combat.df,x=sponge.combat.df$batch,
             y=sponge.combat.df$value,title = 'OTU9 - ComBat (Sponge)',
             batch.legend.title = 'Gel (batch)')


sponge.limma.df = data.frame(value = sponge.limma[,9], batch = sponge.batch)
sponge.boxplot.limma <-box_plot_fun(data = sponge.limma.df,x=sponge.limma.df$batch,
             y=sponge.limma.df$value,title = 'OTU9 - rBE(Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.percentile.df = data.frame(value = sponge.percentile[,9], batch = sponge.batch)
sponge.boxplot.percentile <-box_plot_fun(data = sponge.percentile.df,x=sponge.percentile.df$batch,
             y=sponge.percentile.df$value,title = 'OTU9 - PN (Sponge)',
             batch.legend.title = 'Gel (batch)')

sponge.svd.df = data.frame(value = sponge.svd[,9], batch = sponge.batch)
sponge.boxplot.svd <-box_plot_fun(data = sponge.svd.df,x=sponge.svd.df$batch,
             y=sponge.svd.df$value,title = 'OTU9 - SVD (Sponge)',
             batch.legend.title = 'Gel (batch)')
```

```
grid.arrange(sponge.boxplot.before, sponge.boxplot.bmc, sponge.boxplot.combat, sponge.boxplot.limma,nco
```



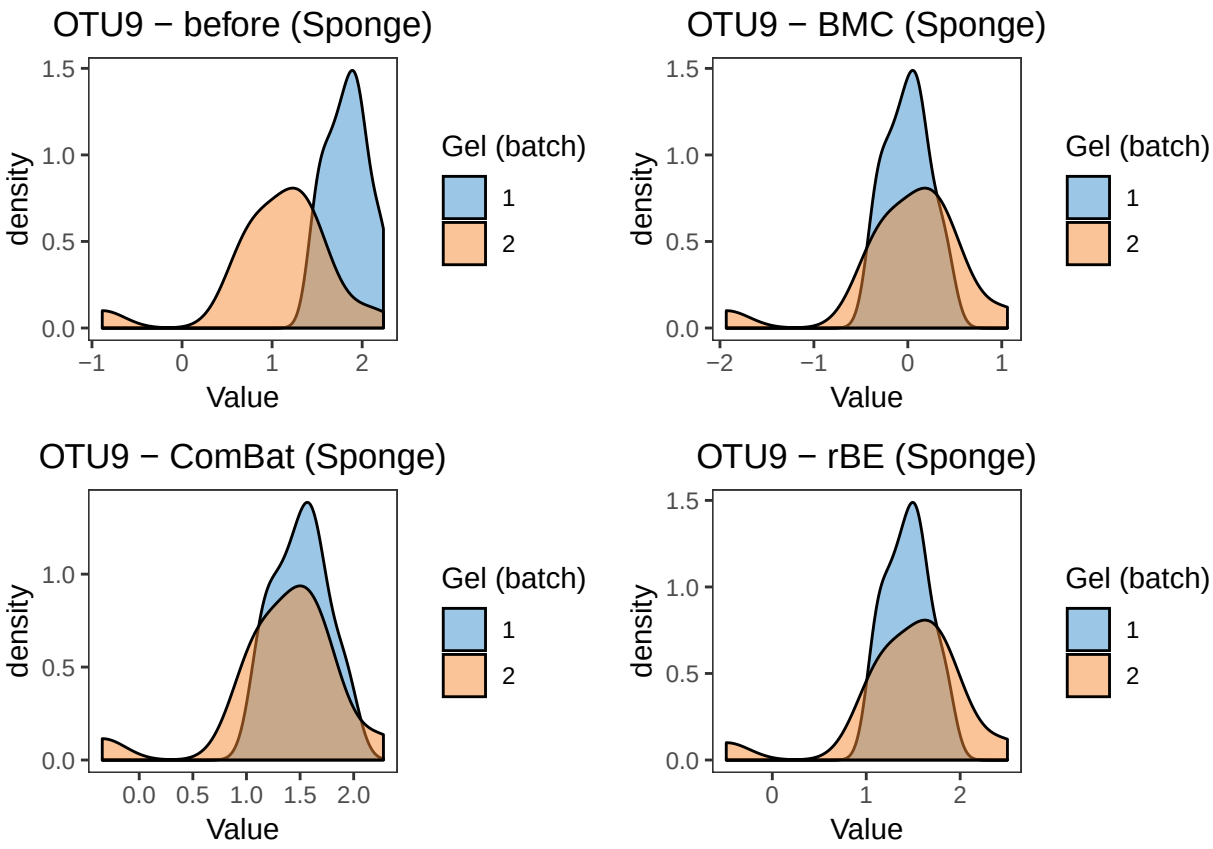

```
grid.arrange(sponge.boxplot.before, sponge.boxplot.percentile,sponge.boxplot.svd,ncol=2)
```



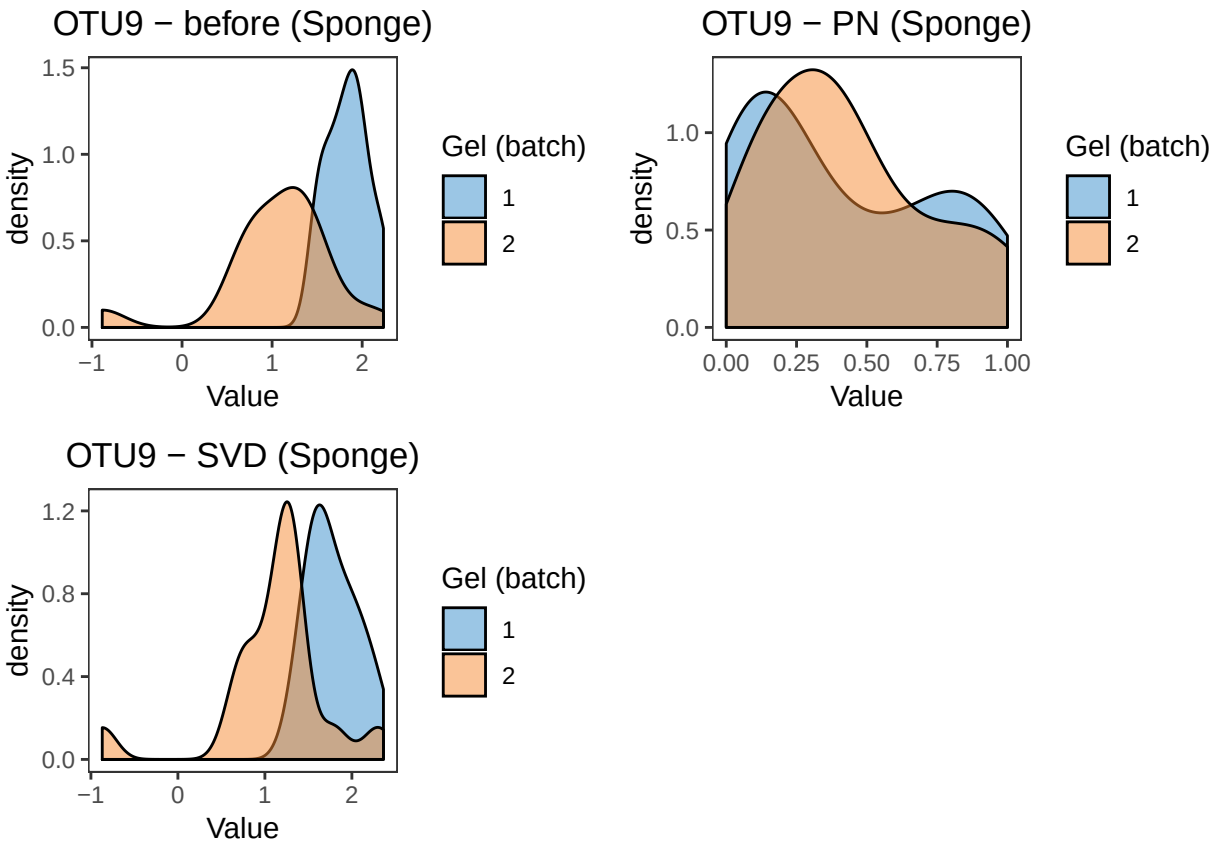
From the boxplots of OTU9 in sponge data, the difference between two batches are removed by BMC, ComBat, rBE and percentile normalisation correction. Among these methods, percentile normalisation does not remove as much batch difference as the other methods.

```
## density plot
# before
sponge.dens.before <- ggplot(sponge.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5)

# BMC
sponge.dens.bmc <- ggplot(sponge.bmc.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + sca


# ComBat
sponge.dens.combat <- ggplot(sponge.combat.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5)


# removeBatchEffect
sponge.dens.limma <- ggplot(sponge.limma.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) +


# percentile normal
sponge.dens.percentile <- ggplot(sponge.percentile.df, aes(x = value, fill = batch)) + geom_density(alpl


# SVD
sponge.dens.svd <- ggplot(sponge.svd.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + sca
```

```
grid.arrange(sponge.dens.before, sponge.dens.bmc, sponge.dens.combat,sponge.dens.limma,ncol=2)
```




```
grid.arrange(sponge.dens.before, sponge.dens.percentile, sponge.dens.svd,ncol=2)
```




```
######### p-values
sponge.lm.before = lm(sponge.tss.clr[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.before)
```

```
##
## Call:
## lm(formula = sponge.tss.clr[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.7849     0.1497  11.922 1.06e-12 ***
## sponge.trtE     0.1065     0.1729   0.616    0.543
## sponge.batch2  -0.7910     0.1729  -4.575 8.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.4236, Adjusted R-squared:  0.3839
## F-statistic: 10.66 on 2 and 29 DF,  p-value: 0.0003391
```

```
sponge.lm.bmc = lm(sponge.bmc[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.bmc)
```

```
##
```

```
## Call:
## lm(formula = sponge.bmc[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.323e-02  1.497e-01  -0.356    0.725
## sponge.trtE    1.065e-01  1.729e-01   0.616    0.543
## sponge.batch2  3.925e-17  1.729e-01   0.000    1.000
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.01291,    Adjusted R-squared:  -0.05517
## F-statistic: 0.1896 on 2 and 29 DF,  p-value: 0.8283
```

```
sponge.lm.combat = lm(sponge.combat[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.combat)
```

```
##
## Call:
## lm(formula = sponge.combat[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64583 -0.22414  0.05092  0.24065  0.88585
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.46291    0.13491  10.844 1.02e-11 ***
## sponge.trtE    0.09081    0.15578   0.583    0.564
## sponge.batch2 -0.16201    0.15578  -1.040    0.307
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4406 on 29 degrees of freedom
## Multiple R-squared:  0.04672,    Adjusted R-squared:  -0.01902
## F-statistic: 0.7107 on 2 and 29 DF,  p-value: 0.4996
```

```
sponge.lm.limma = lm(sponge.limma[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.limma)
```

```
##
## Call:
## lm(formula = sponge.limma[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87967 -0.24705  0.04588  0.24492  1.00757
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.389e+00  1.497e-01   9.280 3.49e-10 ***
## sponge.trtE   1.065e-01  1.729e-01   0.616    0.543
```

```
## sponge.batch2 2.355e-16  1.729e-01   0.000    1.000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.489 on 29 degrees of freedom
## Multiple R-squared:  0.01291,    Adjusted R-squared:  -0.05517
## F-statistic: 0.1896 on 2 and 29 DF,  p-value: 0.8283
```

```
sponge.lm.percentile = lm(sponge.percentile[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.percentile)
```

```
##
## Call:
## lm(formula = sponge.percentile[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4531 -0.2070 -0.0625  0.1797  0.6562
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.48438    0.09515   5.090 1.97e-05 ***
## sponge.trtE   -0.17187    0.10987  -1.564    0.129
## sponge.batch2  0.03125    0.10987   0.284    0.778
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3108 on 29 degrees of freedom
## Multiple R-squared:  0.08018,    Adjusted R-squared:  0.01674
## F-statistic: 1.264 on 2 and 29 DF,  p-value: 0.2976
```

```
sponge.lm.svd = lm(sponge.svd[,9]~ sponge.trt + sponge.batch)
summary(sponge.lm.svd)
```

```
##
## Call:
## lm(formula = sponge.svd[, 9] ~ sponge.trt + sponge.batch)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.82982 -0.27539  0.05228  0.28204  1.05932
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.6433     0.1526  10.766 1.21e-11 ***
## sponge.trtE     0.2817     0.1763   1.598  0.12085
## sponge.batch2  -0.6831     0.1763  -3.875  0.00056 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4985 on 29 degrees of freedom
## Multiple R-squared:  0.3773, Adjusted R-squared:  0.3344
## F-statistic: 8.787 on 2 and 29 DF,  p-value: 0.001039
```

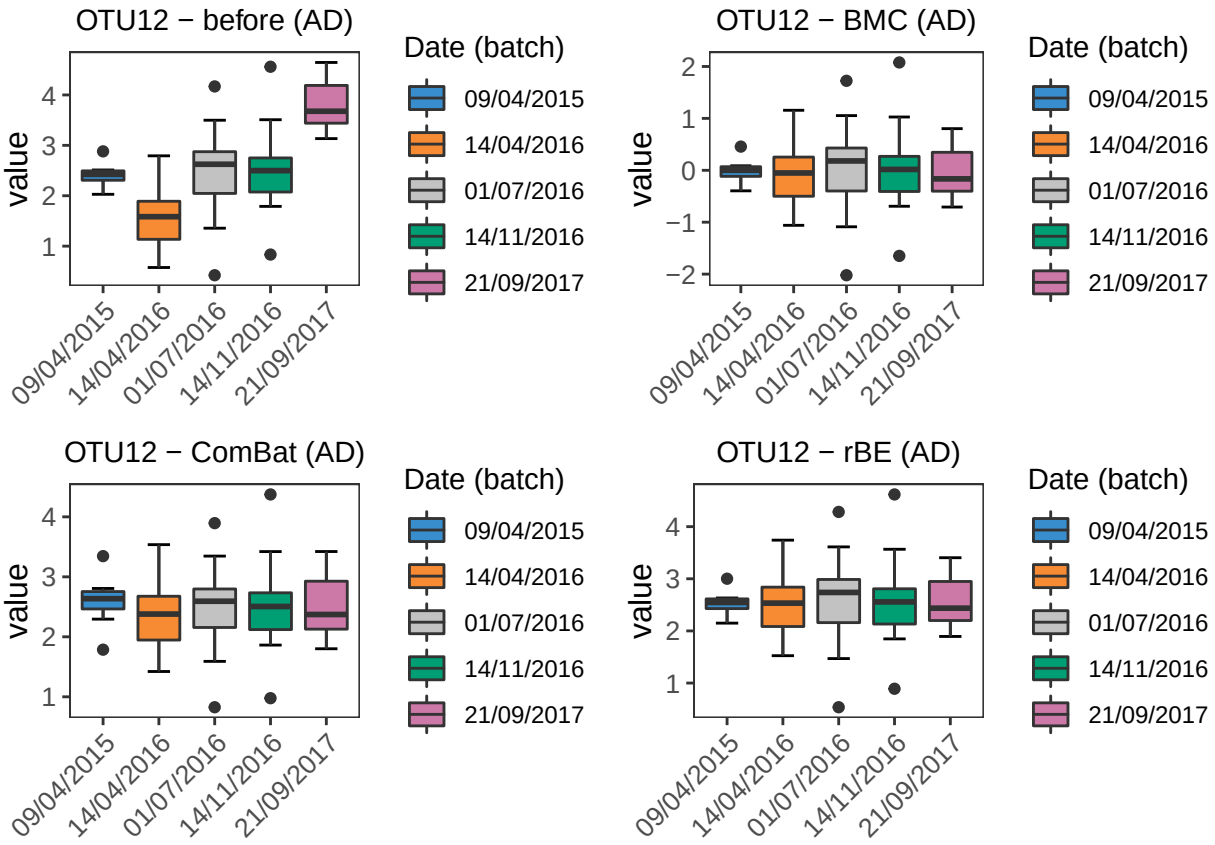
The results of density plots and P-values of batch effects from linear models reach a consesus with boxplots, the difference between two batches are removed by BMC, ComBat, rBE and percentile normalisation correction. But percentile

normalisation changes a lot of the distribution of samples from two batches.

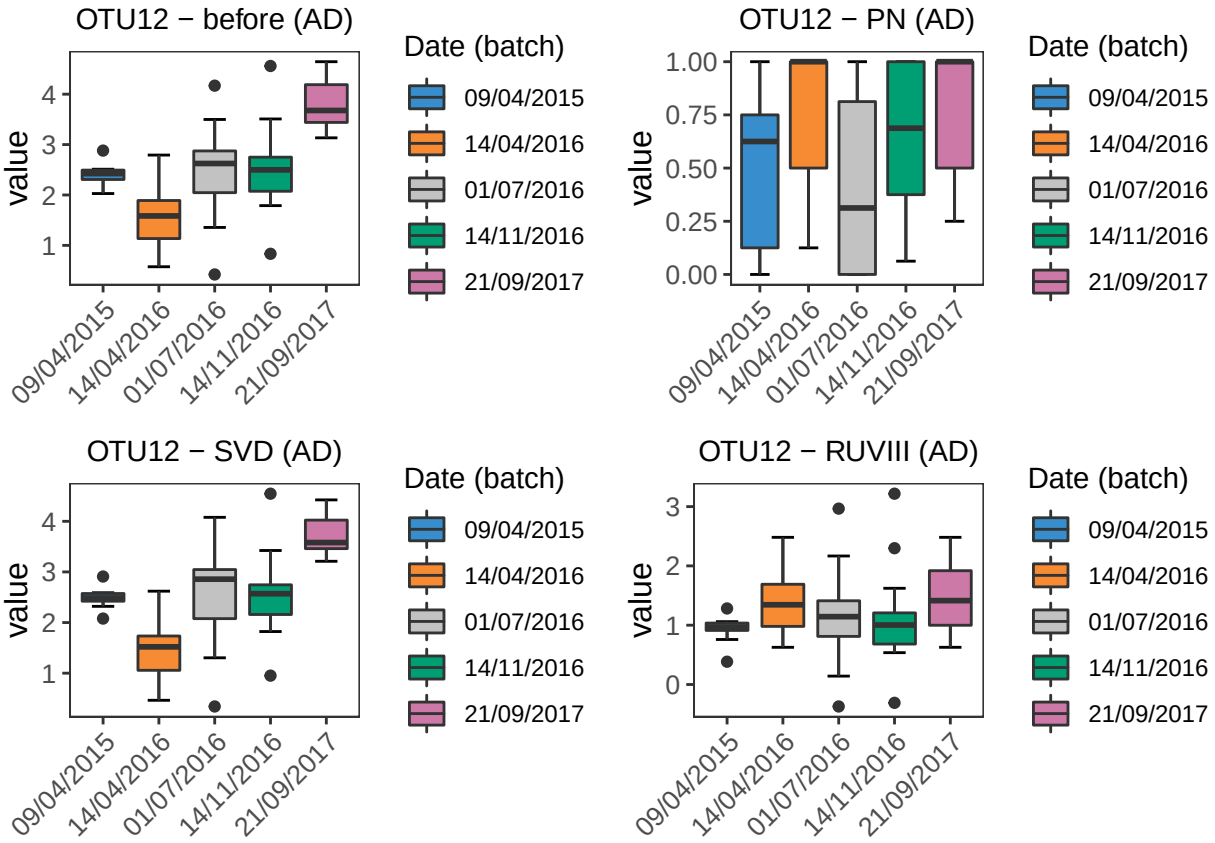
```
##################
# ad data

# boxplot
ad.before.df = data.frame(value = ad.tss.clr[,1], batch = ad.batch)

ad.boxplot.before <- box_plot_fun(data = ad.before.df,x=ad.before.df$batch,
             y=ad.before.df$value,title = 'OTU12 - before (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.bmc.df = data.frame(value = ad.bmc[,1], batch = ad.batch)

ad.boxplot.bmc <- box_plot_fun(data = ad.bmc.df,x=ad.bmc.df$batch,
             y=ad.bmc.df$value,title = 'OTU12 - BMC (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.combat.df = data.frame(value = ad.combat[,1], batch = ad.batch)

ad.boxplot.combat <- box_plot_fun(data = ad.combat.df,x=ad.combat.df$batch,
             y=ad.combat.df$value,title = 'OTU12 - ComBat (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.limma.df = data.frame(value = ad.limma[,1], batch = ad.batch)

ad.boxplot.limma <- box_plot_fun(data = ad.limma.df,x=ad.limma.df$batch,
             y=ad.limma.df$value,title = 'OTU12 - rBE (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)


ad.percentile.df = data.frame(value = ad.percentile[,1], batch = ad.batch)

ad.boxplot.percentile <- box_plot_fun(data = ad.percentile.df,x=ad.percentile.df$batch,
             y=ad.percentile.df$value,title = 'OTU12 - PN (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.svd.df = data.frame(value = ad.svd[,1], batch = ad.batch)

ad.boxplot.svd <- box_plot_fun(data = ad.svd.df,x=ad.svd.df$batch,
             y=ad.svd.df$value,title = 'OTU12 - SVD (AD)',
             batch.legend.title = 'Date (batch)',
             x.angle = 45, x.hjust = 1)

ad.ruv.df = data.frame(value = ad.ruvIII[,1], batch = ad.batch)

ad.boxplot.ruv <- box_plot_fun(data = ad.ruv.df,x=ad.ruv.df$batch,
```

```
            y=ad.ruv.df$value,title = 'OTU12 - RUVIII (AD)',
            batch.legend.title = 'Date (batch)',
            x.angle = 45, x.hjust = 1)
```

```
grid.arrange(ad.boxplot.before, ad.boxplot.bmc, ad.boxplot.combat, ad.boxplot.limma,ncol=2)
```




```
grid.arrange(ad.boxplot.before, ad.boxplot.percentile,ad.boxplot.svd,ad.boxplot.ruv,ncol=2)
```

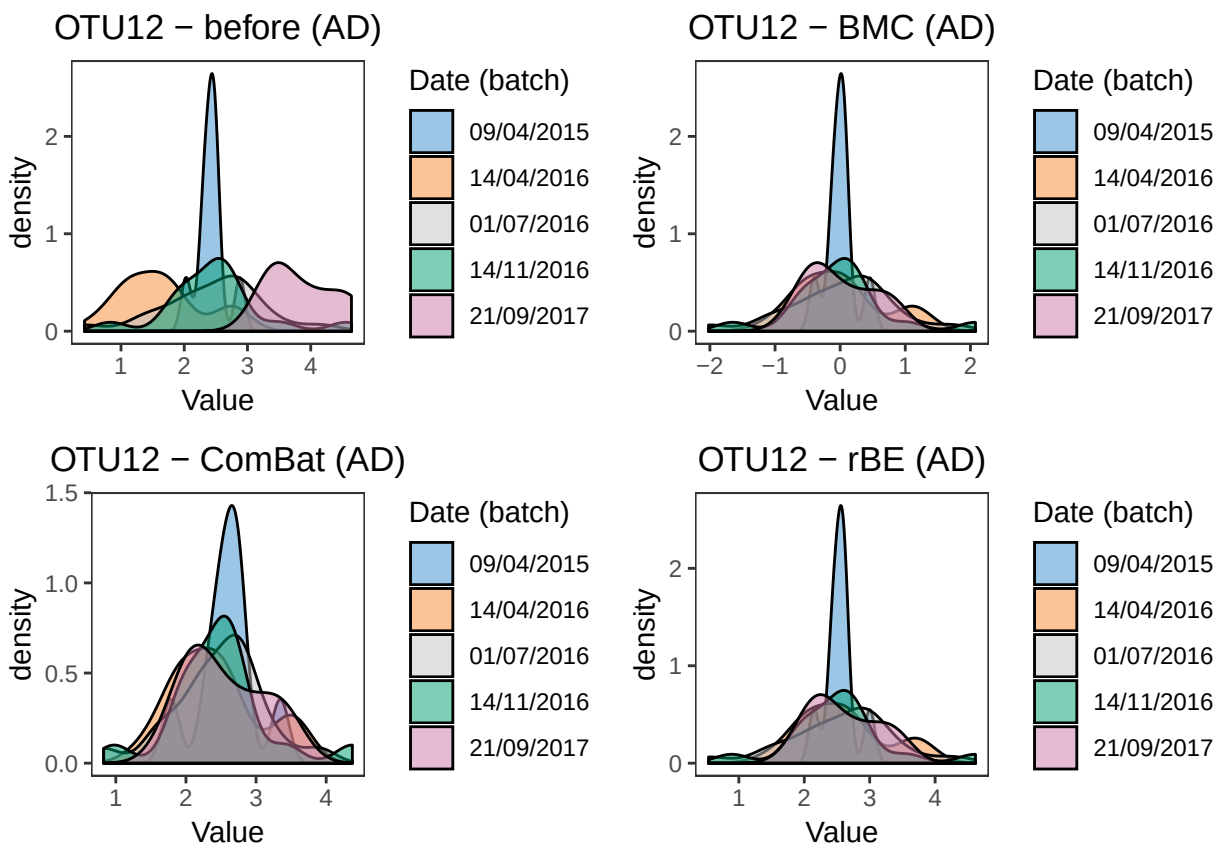


From the boxplots of OTU9 in AD data, the difference between five batches are removed by BMC, ComBat, rBE and RUVIII correction. Among these methods, RUVIII does not remove as much batch difference as the other methods.

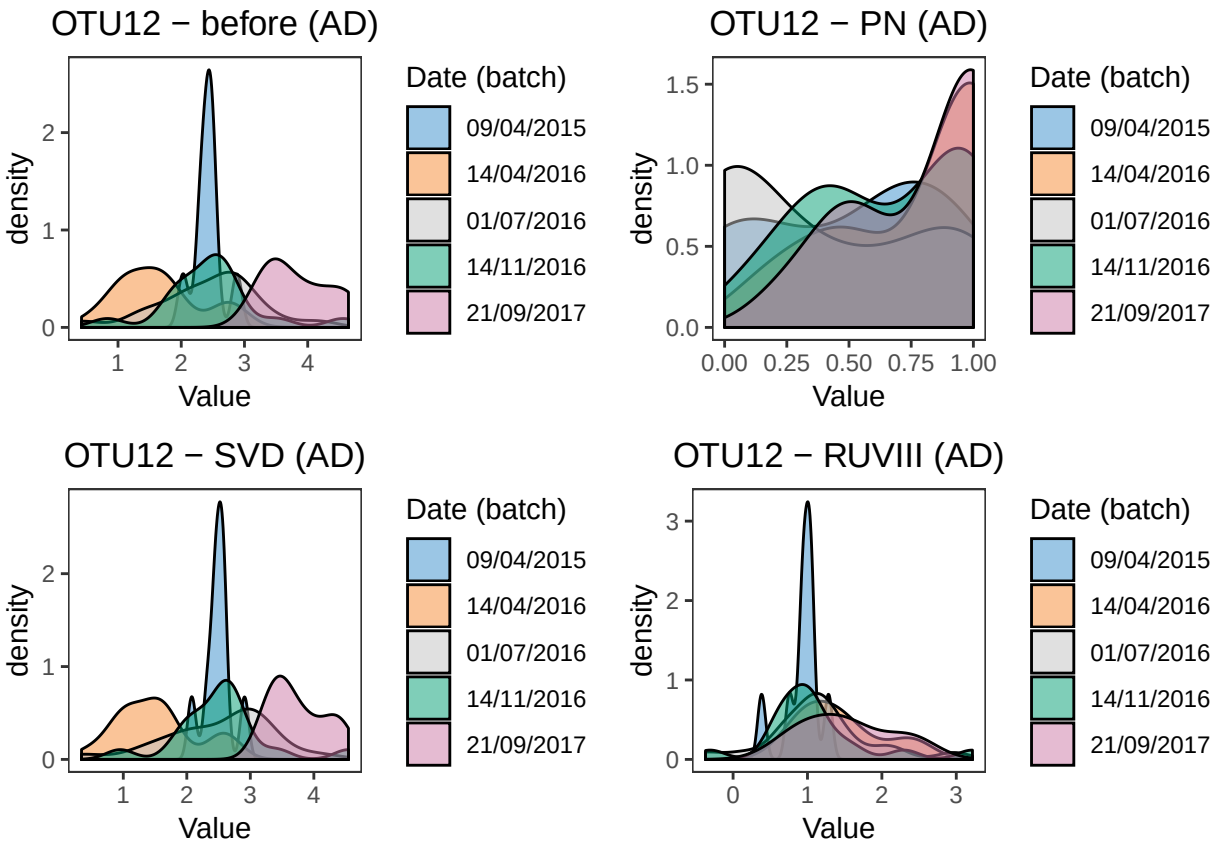
```
# density plot
# before
ad.dens.before = ggplot(ad.before.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_

# BMC
ad.dens.bmc = ggplot(ad.bmc.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_


# ComBat
ad.dens.combat = ggplot(ad.combat.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_


# removeBatchEffect
ad.dens.limma = ggplot(ad.limma.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_f


# percentile norm
ad.dens.percentile = ggplot(ad.percentile.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5)


# SVD
ad.dens.svd = ggplot(ad.svd.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_


# RUVIII
```

```
ad.dens.ruv = ggplot(ad.ruv.df, aes(x = value, fill = batch)) + geom_density(alpha = 0.5) + scale_fill_
```

```
grid.arrange(ad.dens.before, ad.dens.bmc, ad.dens.combat,ad.dens.limma,ncol=2)
```




```
grid.arrange(ad.dens.before, ad.dens.percentile, ad.dens.svd,ad.dens.ruv, ncol=2)
```




```
#p-values
ad.lm.before = lm(ad.tss.clr[,1]~ ad.trt + ad.batch)
anova(ad.lm.before)
```

```
## Analysis of Variance Table
##
## Response: ad.tss.clr[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  1.460  1.4605  3.1001   0.08272 .
## ad.batch   4 32.889  8.2222 17.4532 6.168e-10 ***
## Residuals 69 32.506  0.4711
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#summary(ad.lm.before)

ad.lm.bmc = lm(ad.bmc[,1]~ ad.trt + ad.batch)
anova(ad.lm.bmc)
```

```
## Analysis of Variance Table
##
## Response: ad.bmc[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.631 0.63084  1.3391 0.2512
## ad.batch   4  0.036 0.00893  0.0190 0.9993
## Residuals 69 32.506 0.47110
```

```
#summary(ad.lm.bmc)
```

```r
ad.lm.combat = lm(ad.combat[,1]~ ad.trt + ad.batch)
anova(ad.lm.combat)
```

```
## Analysis of Variance Table
##
## Response: ad.combat[, 1]
##           Df  Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.6695 0.66954  1.6980 0.1969
## ad.batch   4  0.2373 0.05932  0.1504 0.9622
## Residuals 69 27.2080 0.39432
```

```r
#summary(ad.lm.combat)

ad.lm.limma = lm(ad.limma[,1]~ ad.trt + ad.batch)
anova(ad.lm.limma)
```

```
## Analysis of Variance Table
##
## Response: ad.limma[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## ad.trt     1  0.704 0.70428   1.495 0.2256
## ad.batch   4  0.000 0.00000   0.000 1.0000
## Residuals 69 32.506 0.47110
```

```r
#summary(ad.lm.limma)

ad.lm.percentile = lm(ad.percentile[,1]~ ad.trt + ad.batch)
anova(ad.lm.percentile)
```

```
## Analysis of Variance Table
##
## Response: ad.percentile[, 1]
##           Df Sum Sq Mean Sq F value  Pr(>F)
## ad.trt     1 0.4670 0.46705  3.8934 0.05248 .
## ad.batch   4 1.7037 0.42592  3.5506 0.01081 *
## Residuals 69 8.2772 0.11996
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#summary(ad.lm.percentile)

ad.lm.svd = lm(ad.svd[,1]~ ad.trt + ad.batch)
anova(ad.lm.svd)
```

```
## Analysis of Variance Table
##
## Response: ad.svd[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## ad.trt     1  0.222  0.2218  0.4841    0.4889
## ad.batch   4 33.914  8.4784 18.5081 2.256e-10 ***
## Residuals 69 31.608  0.4581
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#summary(ad.lm.svd)
```

```
ad.lm.ruv = lm(ad.ruvIII[,1]~ ad.trt + ad.batch)
anova(ad.lm.ruv)
```

```
## Analysis of Variance Table
##
## Response: ad.ruvIII[, 1]
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## ad.trt     1  2.1333 2.13330  5.4144 0.02291 *
## ad.batch   4  2.0222 0.50554  1.2831 0.28512
## Residuals 69 27.1862 0.39400
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#summary(ad.lm.ruv)
```

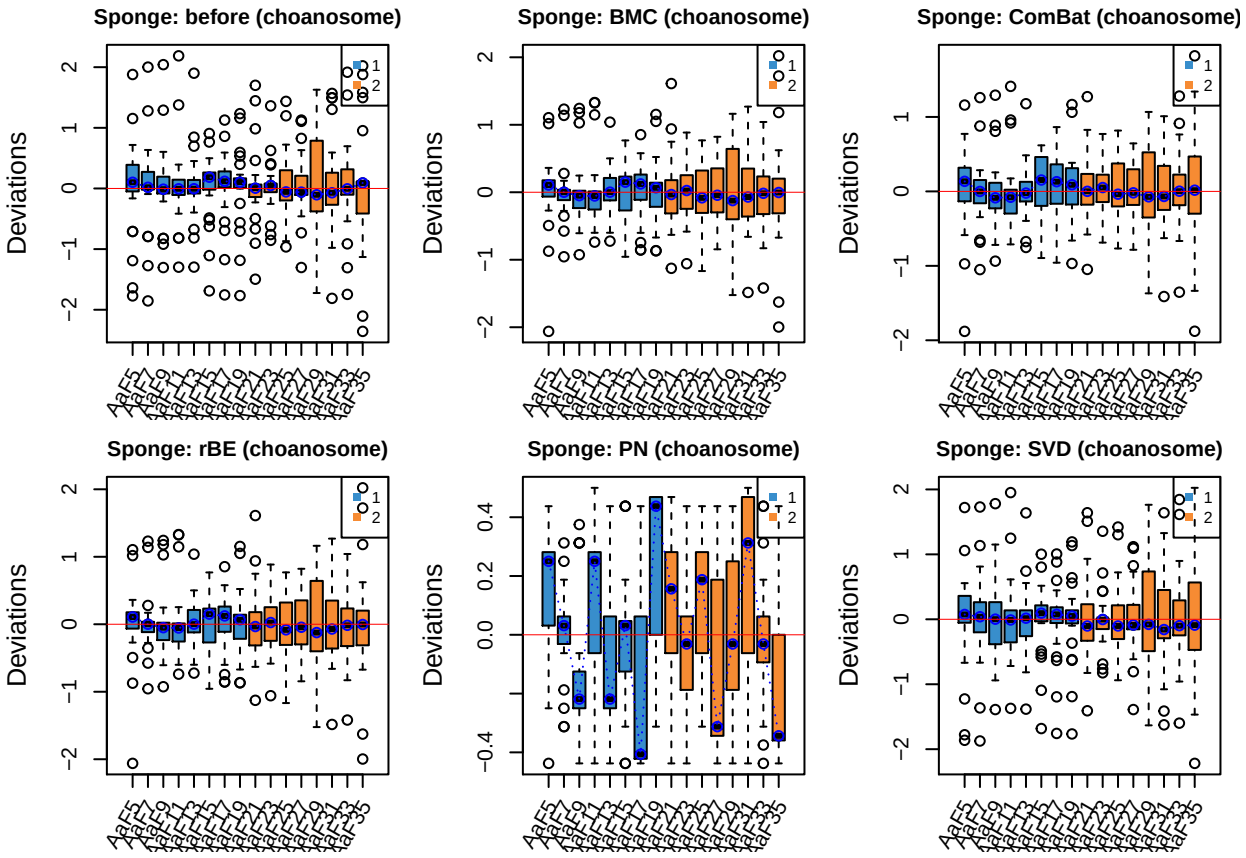
The results of density plots and P-values of batch effects from linear models reach a consesus with boxplots, the difference between two batches are removed by BMC, ComBat, rBE and RUVIII correction.

4.1.3 RLE plots

```
# sponge data
# before

###### BMC
sponge.bmc_c = sponge.bmc[sponge.trt == 'C',]
sponge.bmc_e = sponge.bmc[sponge.trt == 'E',]

###### ComBat
sponge.combat_c = sponge.combat[sponge.trt == 'C',]
sponge.combat_e = sponge.combat[sponge.trt == 'E',]

###### rBE
sponge.limma_c = sponge.limma[sponge.trt == 'C',]
sponge.limma_e = sponge.limma[sponge.trt == 'E',]

###### PN
sponge.percentile_c = sponge.percentile[sponge.trt == 'C',]
sponge.percentile_e = sponge.percentile[sponge.trt == 'E',]

###### SVD
sponge.svd_c = sponge.svd[sponge.trt == 'C',]
sponge.svd_e = sponge.svd[sponge.trt == 'E',]
```

```
par(mfrow = c(2,3), mai=c(0.4,0.6,0.3,0.1))

RleMicroRna2(object = t(sponge.before_c),batch = sponge.batch_c,maintitle = 'Sponge: before (choanosome)

RleMicroRna2(object = t(sponge.bmc_c), batch = sponge.batch_c,maintitle = 'Sponge: BMC (choanosome)',ti

RleMicroRna2(object = t(sponge.combat_c),batch = sponge.batch_c,maintitle = 'Sponge: ComBat (choanosome)

RleMicroRna2(object = t(sponge.limma_c),batch = sponge.batch_c,maintitle = 'Sponge: rBE (choanosome)',t
```

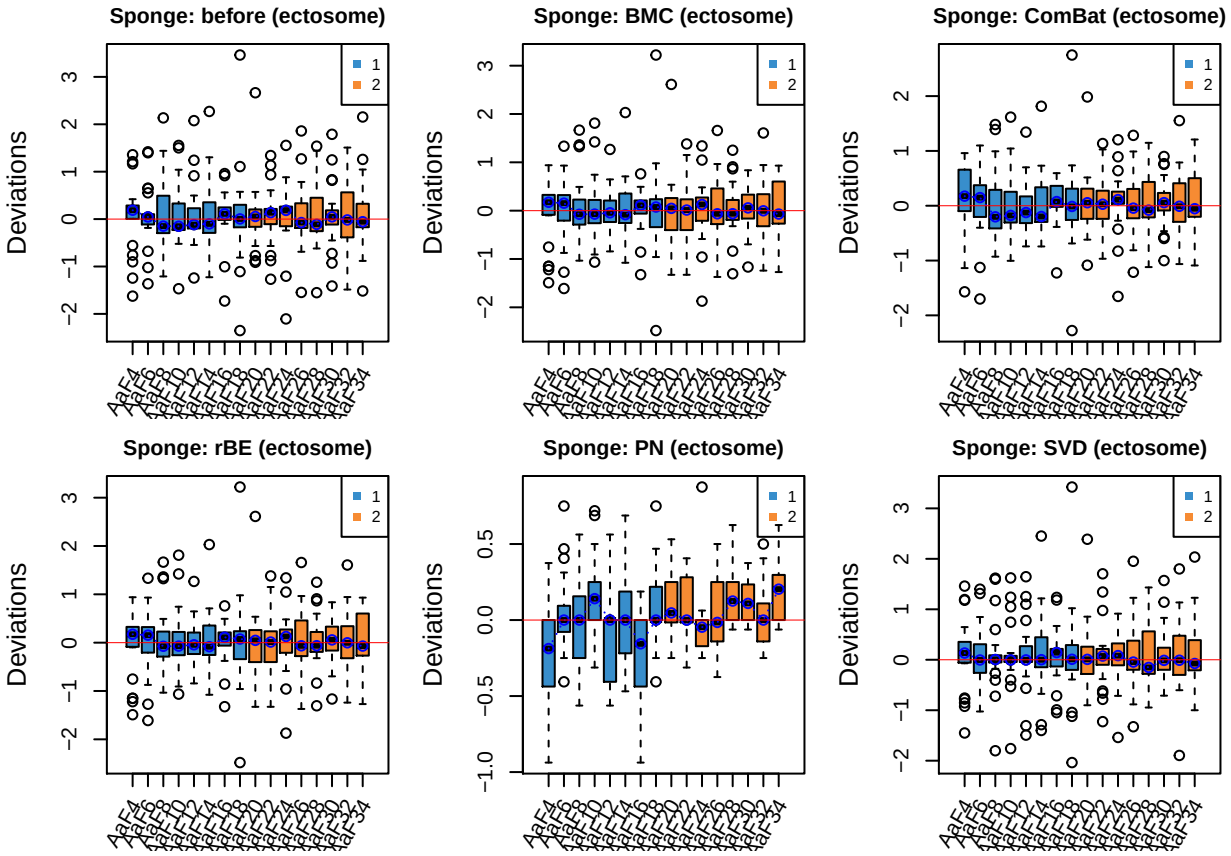
```
RleMicroRna2(object = t(sponge.percentile_c),batch = sponge.batch_c,maintitle = 'Sponge: PN (choanosome)

RleMicroRna2(object = t(sponge.svd_c),batch = sponge.batch_c,maintitle = 'Sponge: SVD (choanosome)',titl
```




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

The batch effect before correction is not obvious as all medians of samples are close to zero, but gel2 has a greater interquartile range (IQR) than the other samples. Percentile normalisation increased batch effects, as the samples after correction are deviated from zero and the IQR is increased. Among other methods, the IQR of all the box plots (samples) from different batches is consistent after ComBat correction.

```
par(mfrow = c(2,3), mai=c(0.4,0.6,0.3,0.1))

RleMicroRna2(object = t(sponge.before_e),batch = sponge.batch_e,maintitle = 'Sponge: before (ectosome)'

RleMicroRna2(object = t(sponge.bmc_e), batch = sponge.batch_e,maintitle = 'Sponge: BMC (ectosome)',titl

RleMicroRna2(object = t(sponge.combat_e),batch = sponge.batch_e,maintitle = 'Sponge: ComBat (ectosome)'

RleMicroRna2(object = t(sponge.limma_e),batch = sponge.batch_e,maintitle = 'Sponge: rBE (ectosome)',titl

RleMicroRna2(object = t(sponge.percentile_e),batch = sponge.batch_e,maintitle = 'Sponge: PN (ectosome)'

RleMicroRna2(object = t(sponge.svd_e),batch = sponge.batch_e,maintitle = 'Sponge: SVD (ectosome)',title
```



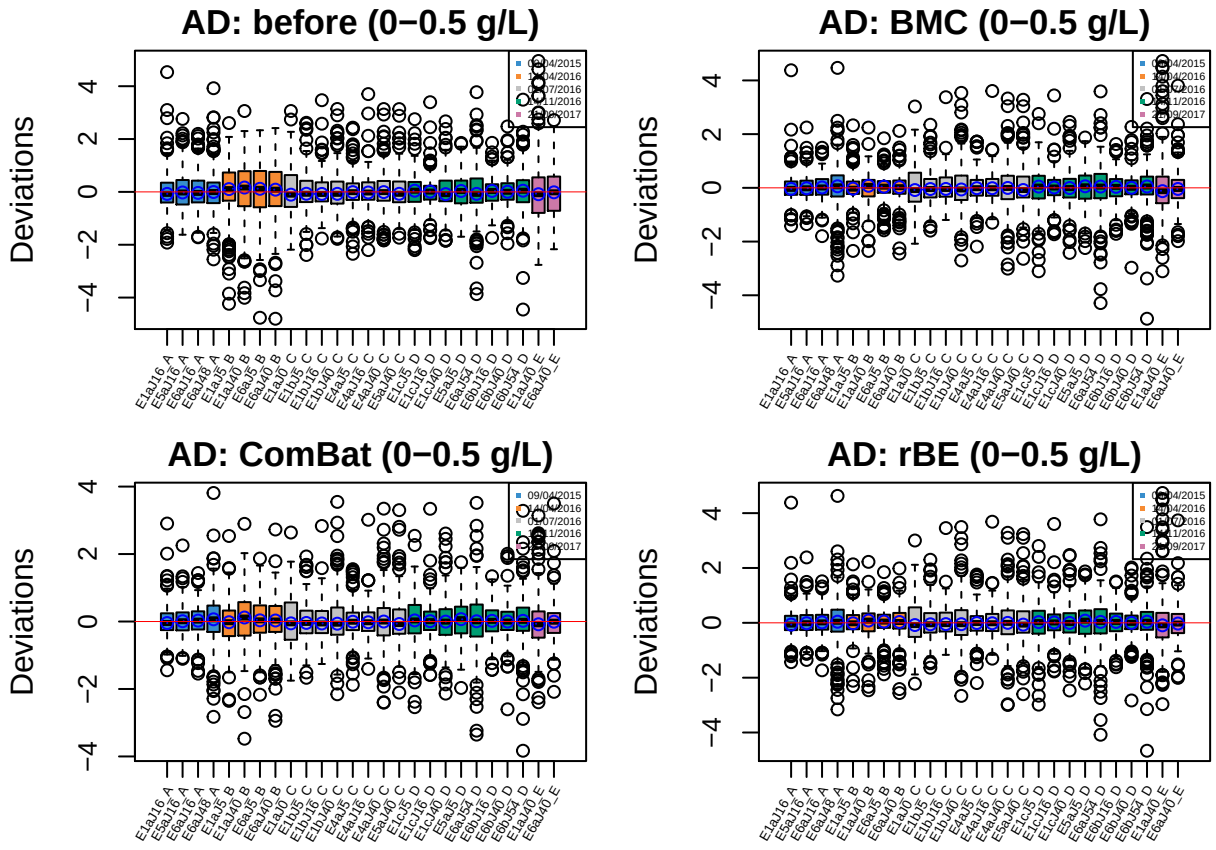

```
par(mfrow = c(1,1))
```

Batch effect is not easily detected, but percentile normalisation increased batch variation.

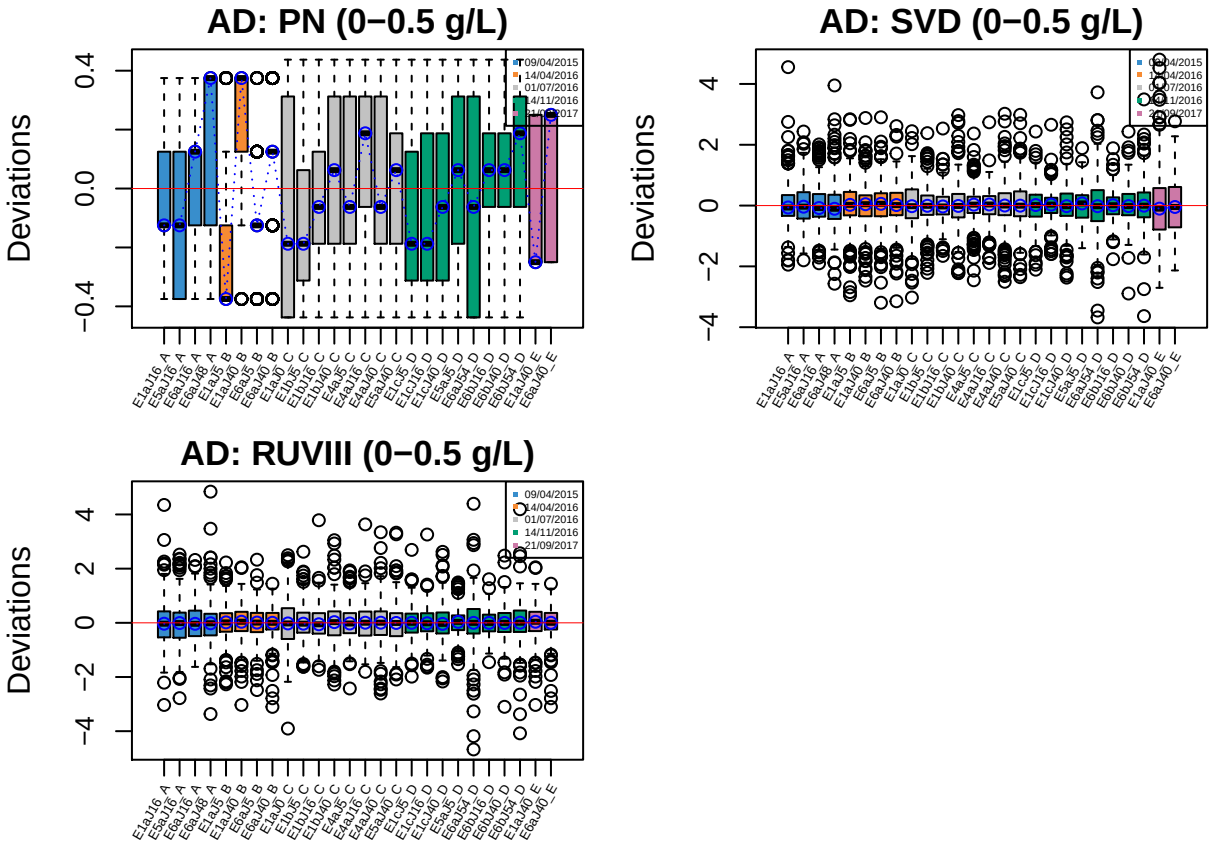
```
#################
# ad data
# before

###### BMC
ad.bmc_05 = ad.bmc[ad.trt == '0-0.5',]
ad.bmc_2 = ad.bmc[ad.trt == '1-2',]

###### ComBat
ad.combat_05 = ad.combat[ad.trt == '0-0.5',]
ad.combat_2 = ad.combat[ad.trt == '1-2',]

###### rBE
ad.limma_05 = ad.limma[ad.trt == '0-0.5',]
ad.limma_2 = ad.limma[ad.trt == '1-2',]

###### PN
ad.percentile_05 = ad.percentile[ad.trt == '0-0.5',]
ad.percentile_2 = ad.percentile[ad.trt == '1-2',]

###### SVD
ad.svd_05 = ad.svd[ad.trt == '0-0.5',]
ad.svd_2 = ad.svd[ad.trt == '1-2',]
```

```
###### RUVIII
ad.ruv_05 = ad.ruvIII[ad.trt == '0-0.5',]
ad.ruv_2 = ad.ruvIII[ad.trt == '1-2',]
```

```
par(mfrow=c(2,2), mai=c(0.5,0.8,0.3,0.1))

RleMicroRna2(object = t(ad.before_05),batch = ad.batch_05,maintitle = 'AD: before (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.bmc_05),batch = ad.batch_05,maintitle = 'AD: BMC (0-0.5 g/L)',legend.cex = 0

RleMicroRna2(object = t(ad.combat_05),batch = ad.batch_05,maintitle = 'AD: ComBat (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.limma_05),batch = ad.batch_05,maintitle = 'AD: rBE (0-0.5 g/L)',legend.cex =
```

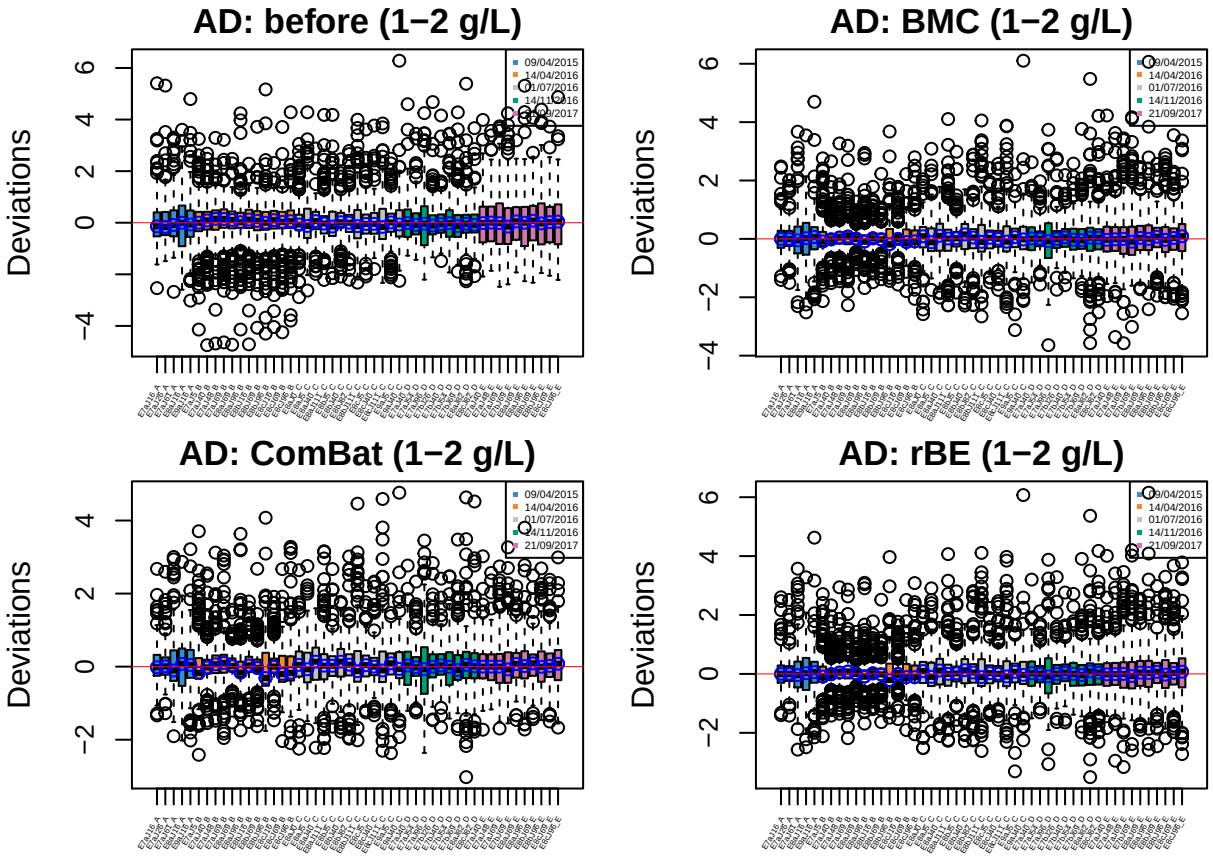



```
RleMicroRna2(object = t(ad.percentile_05),batch = ad.batch_05,maintitle = 'AD: PN (0-0.5 g/L)',legend.c

RleMicroRna2(object = t(ad.svd_05),batch = ad.batch_05,maintitle = 'AD: SVD (0-0.5 g/L)',legend.cex = 0

RleMicroRna2(object = t(ad.ruv_05),batch = ad.batch_05,maintitle = 'AD: RUVIII (0-0.5 g/L)',legend.cex =

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

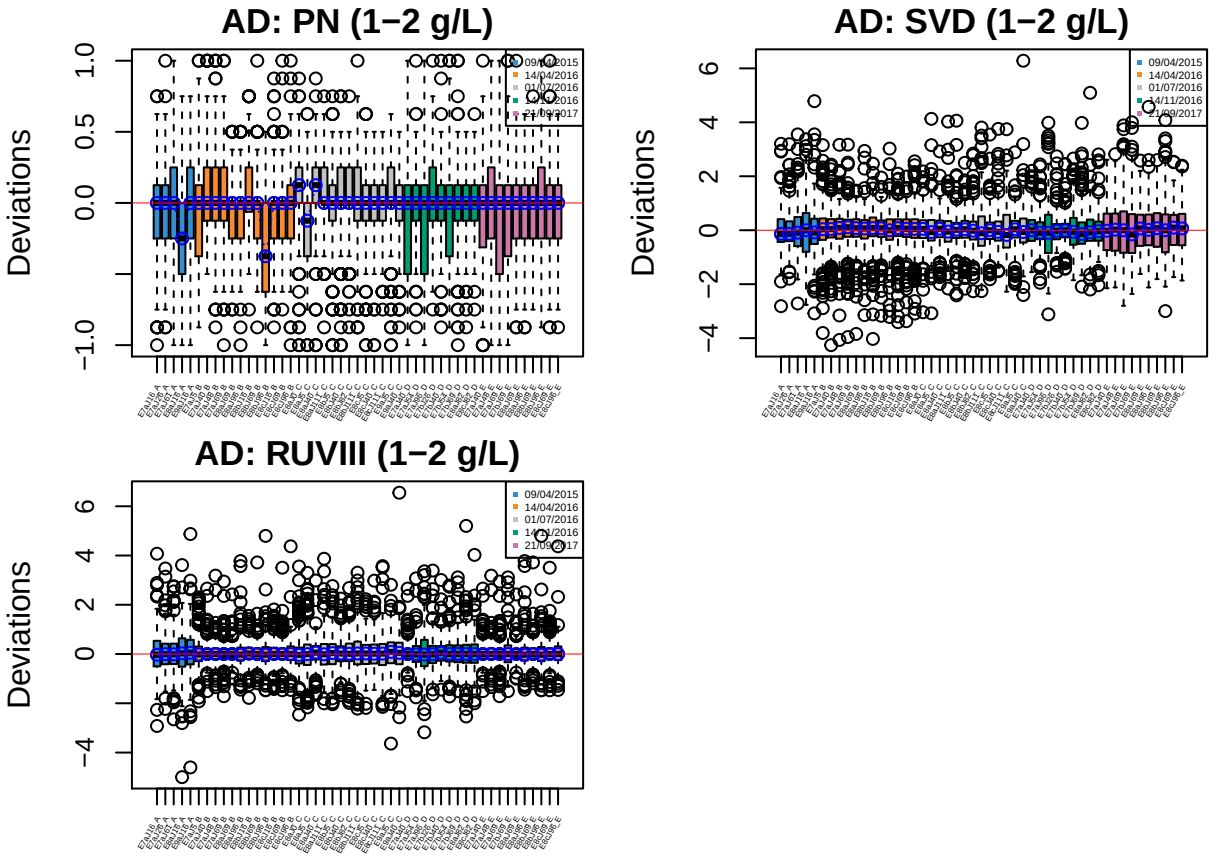


Batch effect is not easily detected, but percentile normalisation increased batch variation.

```
par(mfrow=c(2,2), mai=c(0.35,0.8,0.3,0.1))

RleMicroRna2(object = t(ad.before_2),batch = ad.batch_2,maintitle = 'AD: before (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.bmc_2),batch = ad.batch_2,maintitle = 'AD: BMC (1-2 g/L)',legend.cex = 0.4,

RleMicroRna2(object = t(ad.combat_2),batch = ad.batch_2,maintitle = 'AD: ComBat (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.limma_2),batch = ad.batch_2,maintitle = 'AD: rBE (1-2 g/L)',legend.cex = 0.4
```




```
RleMicroRna2(object = t(ad.percentile_2),batch = ad.batch_2,maintitle = 'AD: PN (1-2 g/L)',legend.cex =

RleMicroRna2(object = t(ad.svd_2),batch = ad.batch_2,maintitle = 'AD: SVD (1-2 g/L)',legend.cex = 0.4, 

RleMicroRna2(object = t(ad.ruv_2),batch = ad.batch_2,maintitle = 'AD: RUVIII (1-2 g/L)',legend.cex = 0.4

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

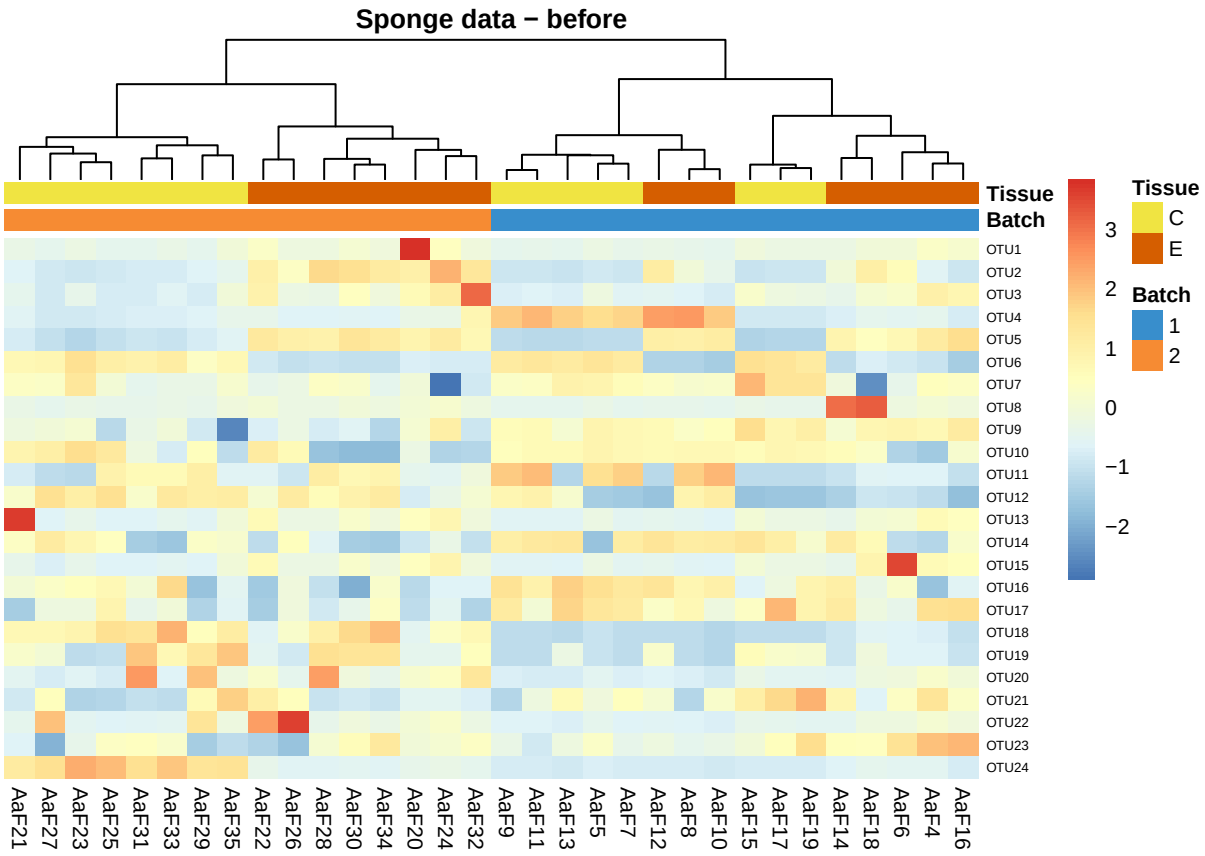


Batch effect is not easily detected, but percentile normalisation increased batch variation.

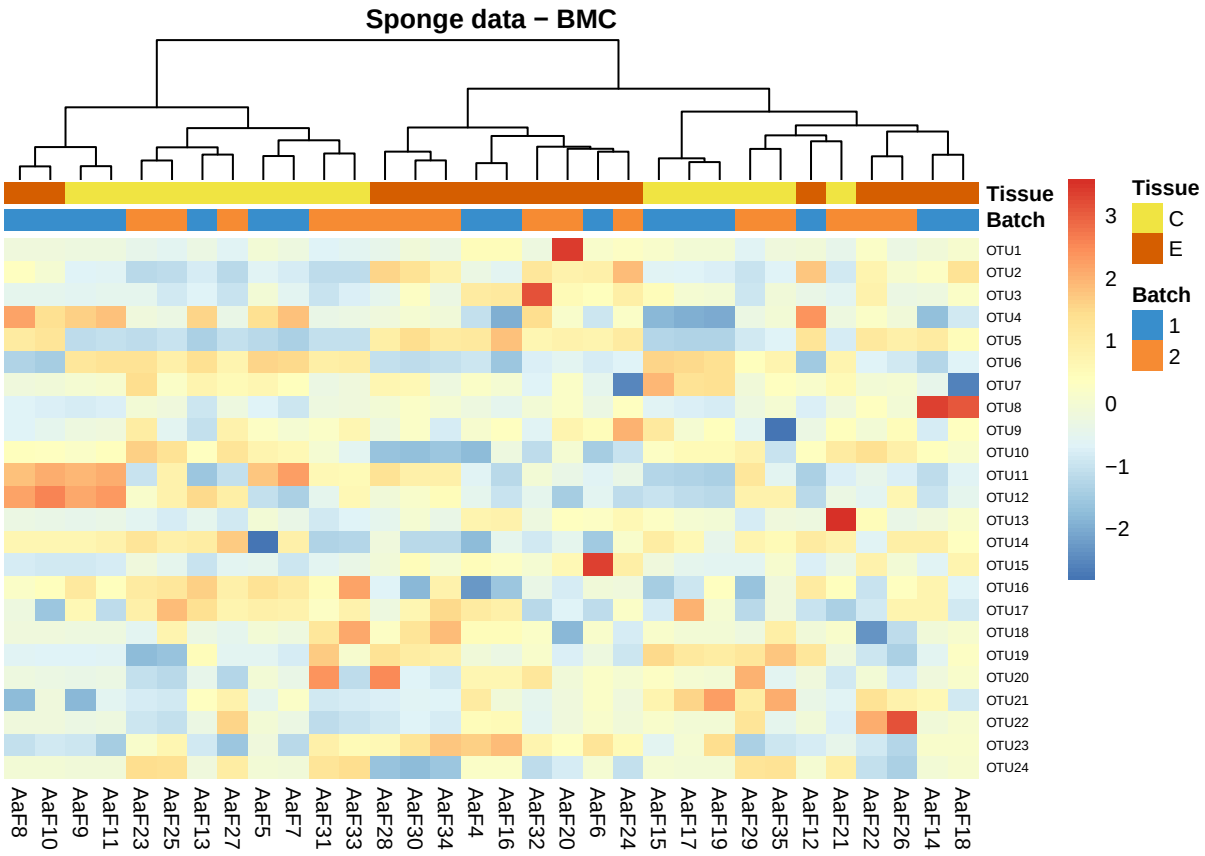
4.1.4 Heatmap

```
# Sponge data
# before
sponge.tss.clr.scale = scale(sponge.tss.clr,center = T, scale = T) # scale on OTUs
sponge.tss.clr.scale = scale(t(sponge.tss.clr.scale), center = T, scale = T) # scale on samples

sponge.anno_col = data.frame(Batch = sponge.batch, Tissue = sponge.trt)
sponge.anno_metabo_colors = list(Batch = c('1'="#388ECC",'2'="#F68B33"),Tissue = c(C="#F0E442",E="#D55E


pheatmap(sponge.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors = sponge.anno_metabo_colors,
         border_color = 'NA',
```

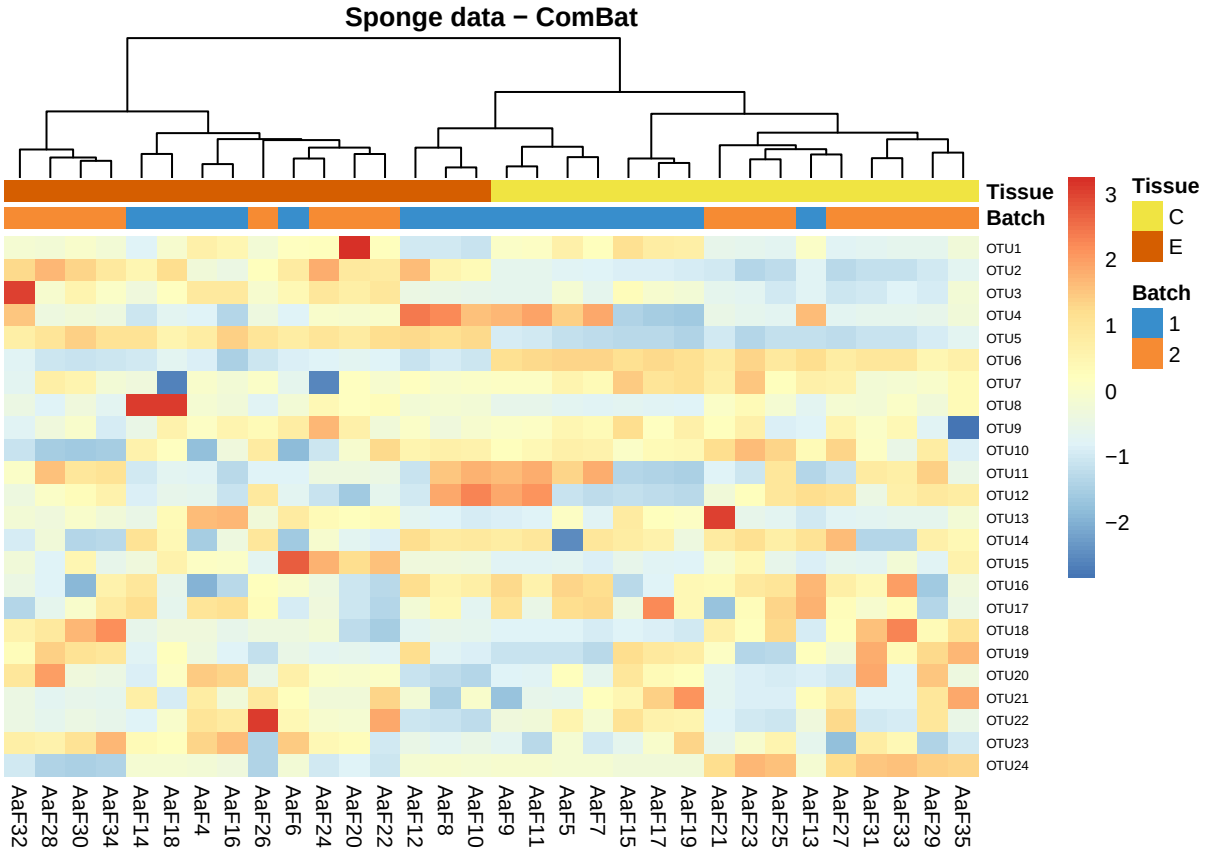
```
         main = 'Sponge data - before')
```



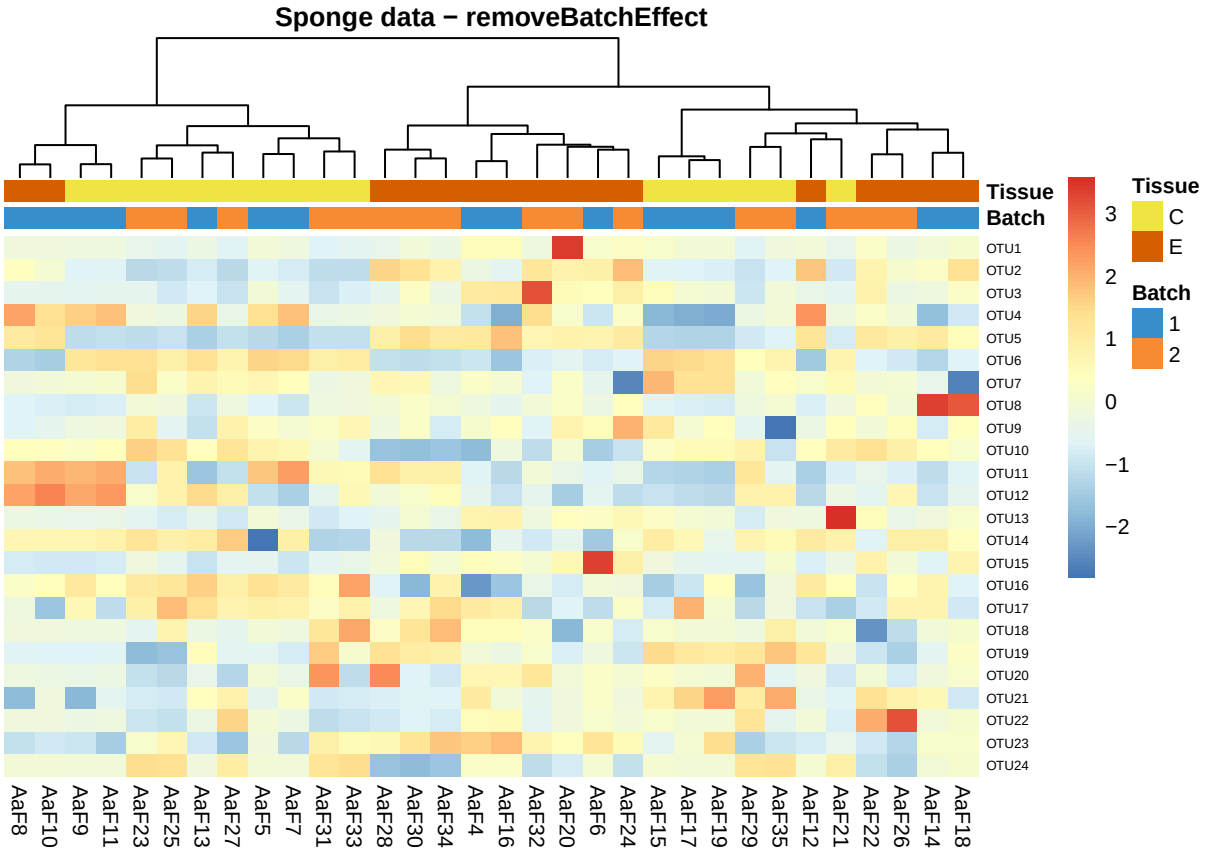

```
# BMC
sponge.bmc.scale = scale(sponge.bmc,center = T, scale = T) # scale on OTUs
sponge.bmc.scale = scale(t(sponge.bmc.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.bmc.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors=sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - BMC')
```



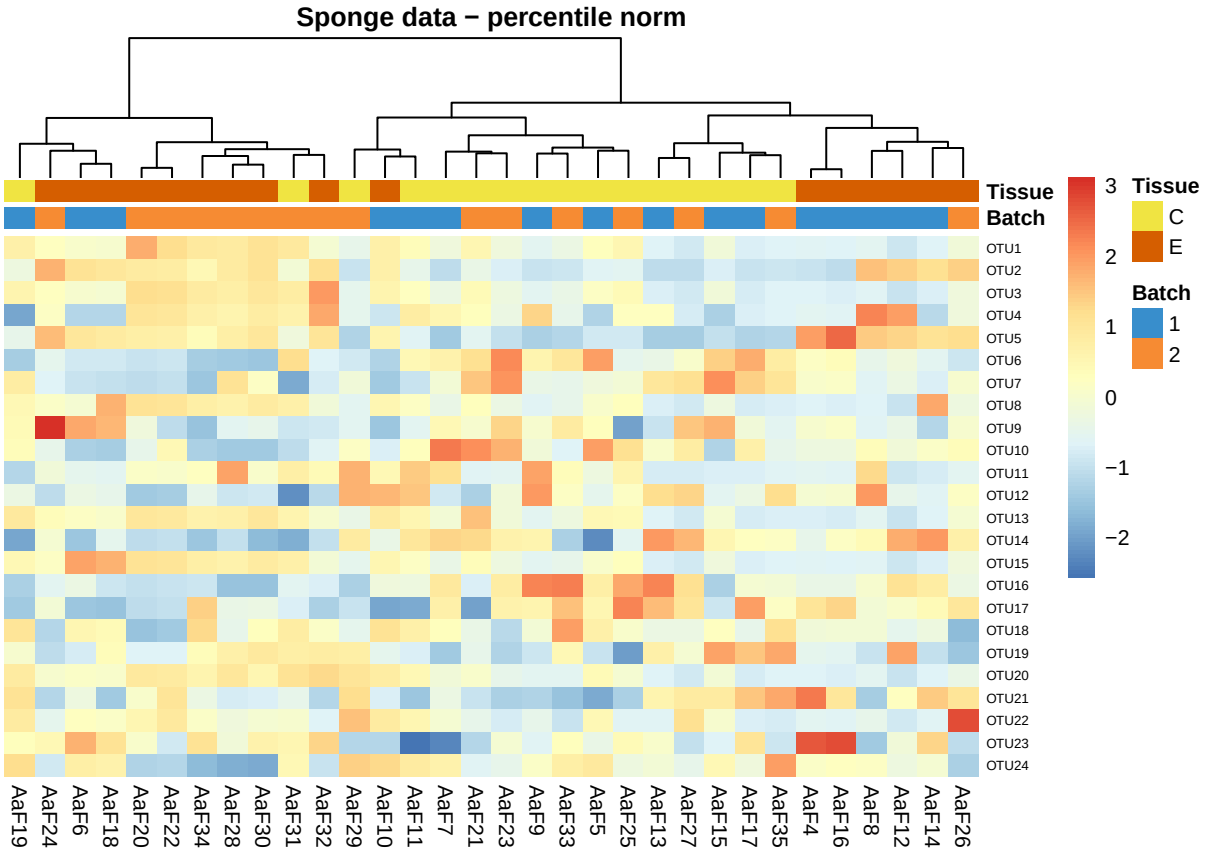

```
# ComBat
sponge.combat.scale = scale(sponge.combat,center = T, scale = T) # scale on OTUs
sponge.combat.scale = scale(t(sponge.combat.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.combat.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors=sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - ComBat')
```



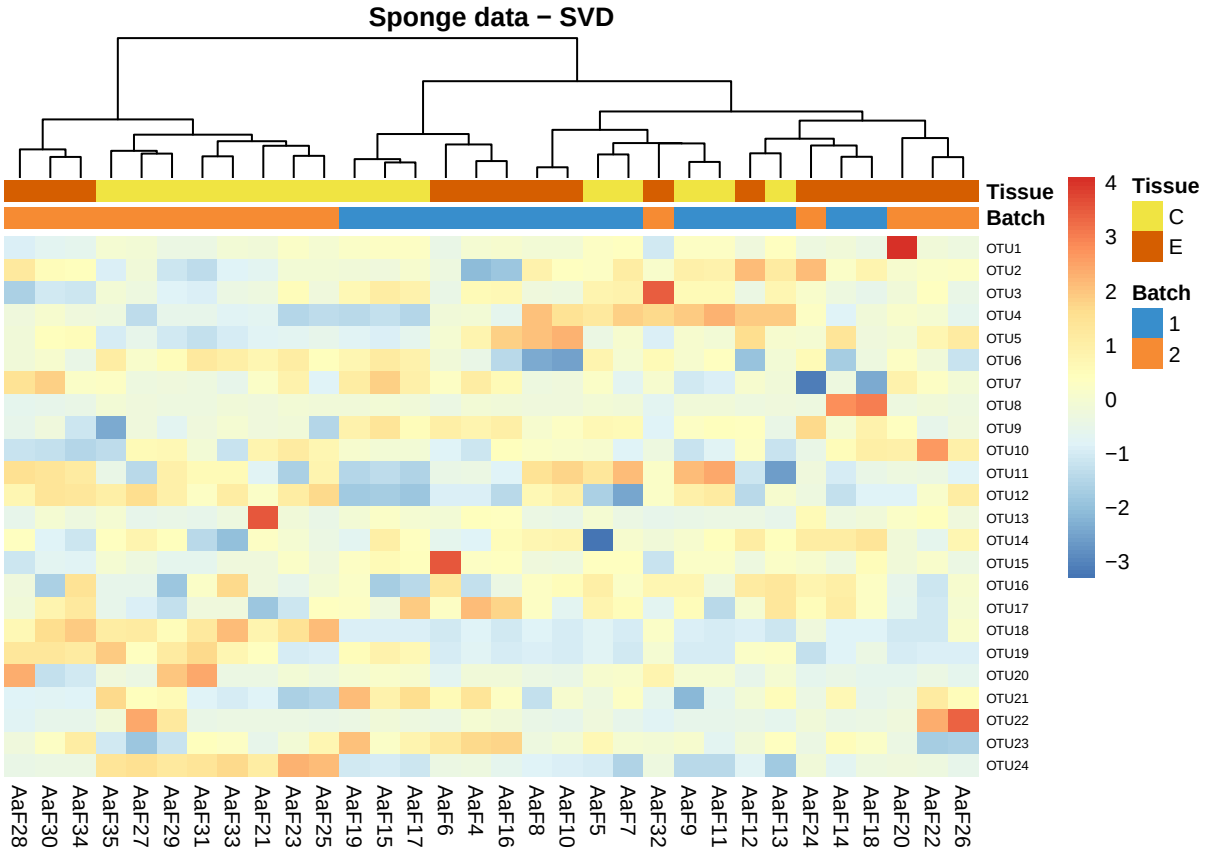

```
# removeBatchEffect
sponge.limma.scale = scale(sponge.limma,center = T, scale = T) # scale on OTUs
sponge.limma.scale = scale(t(sponge.limma.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.limma.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors= sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - removeBatchEffect')
```




```
# percentile normalisation
sponge.percentile.scale = scale(sponge.percentile,center = T, scale = T) # scale on OTUs
sponge.percentile.scale = scale(t(sponge.percentile.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.percentile.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors = sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - percentile norm')
```

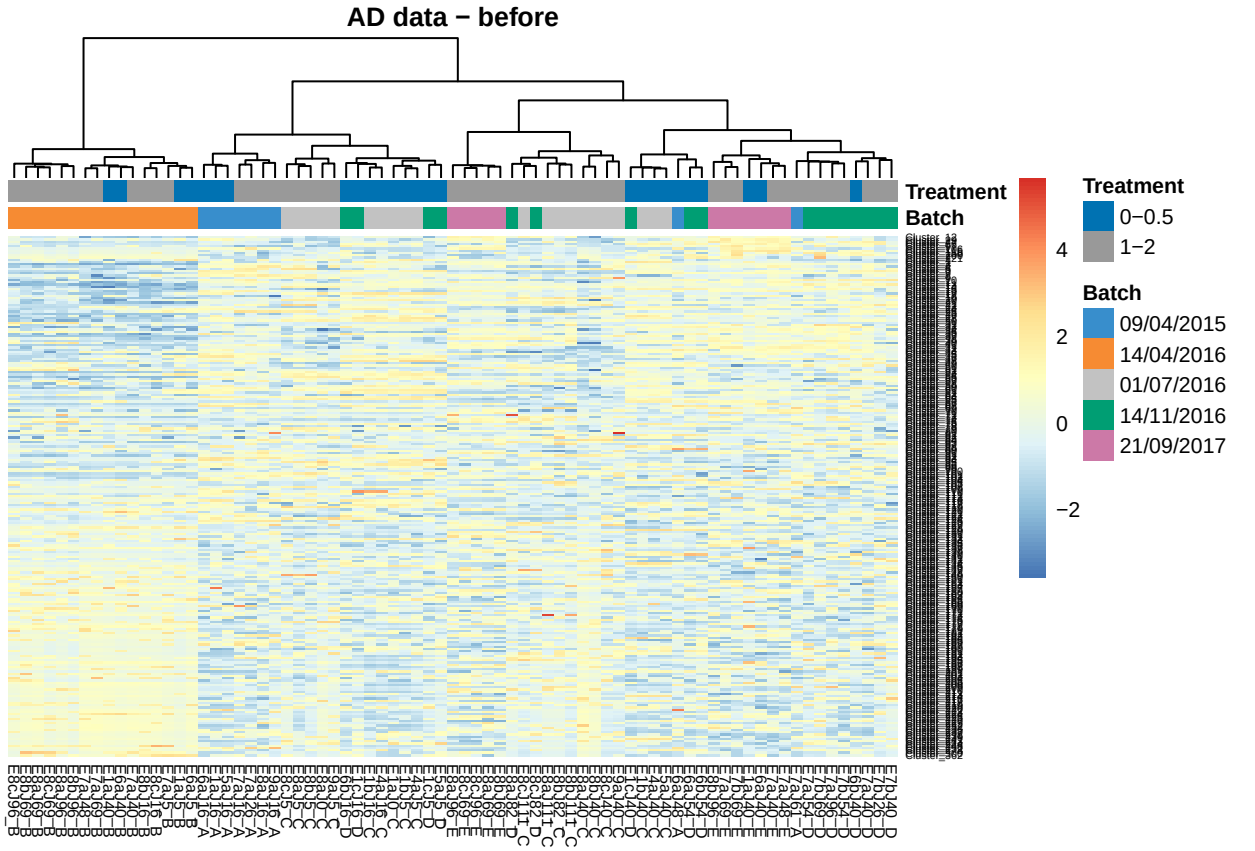



```
# SVD
sponge.svd.scale = scale(sponge.svd,center = T, scale = T) # scale on OTUs
sponge.svd.scale = scale(t(sponge.svd.scale), center = T, scale = T) # scale on samples

pheatmap(sponge.svd.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=5, fontsize_col=8,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors = sponge.anno_metabo_colors,
         border_color = 'NA',
         main = 'Sponge data - SVD')
```

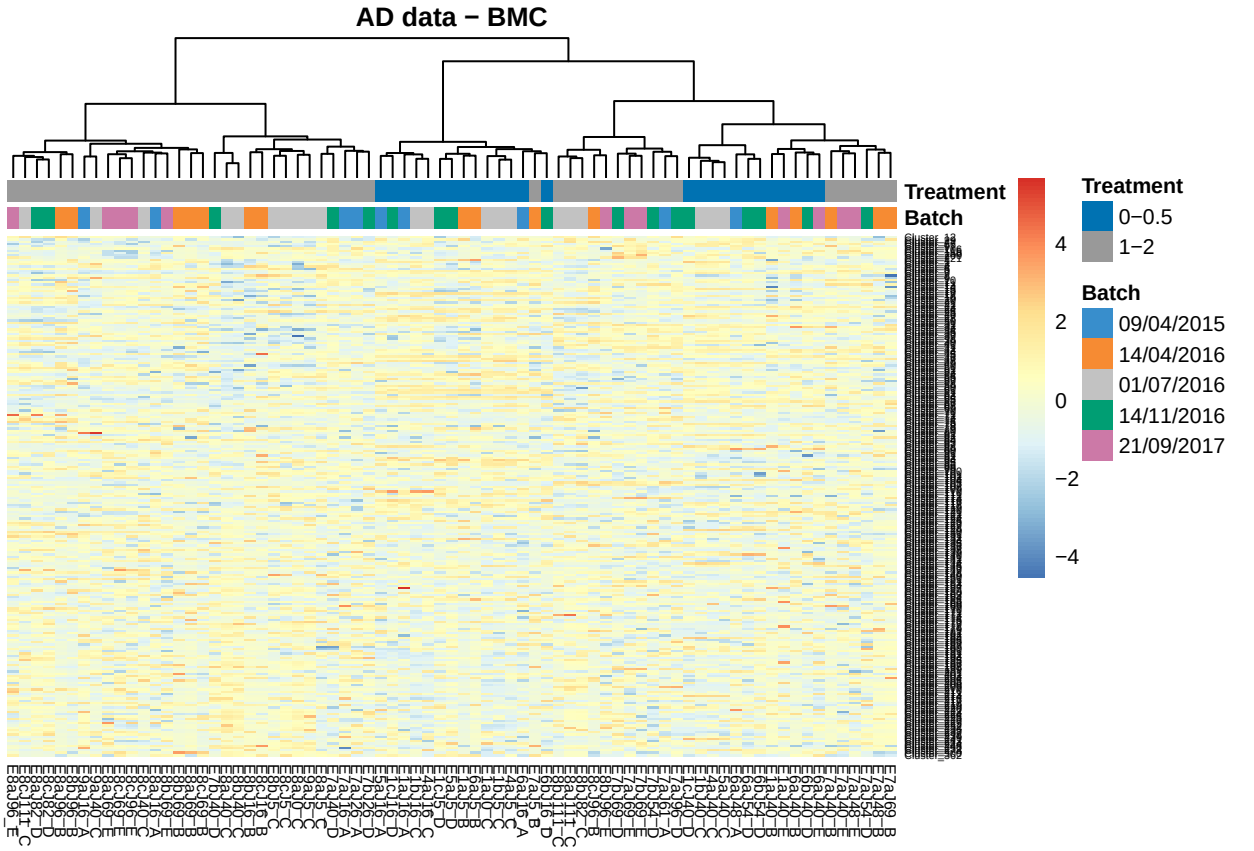


From heatmaps, the sponge data are clustered by batches instead of tissues indicating a batch effect. After batch correction with ComBat, the data are clustered by tissues rather than batches, therefore, ComBat not only removes batch effects, but also disengages the effect of interest.

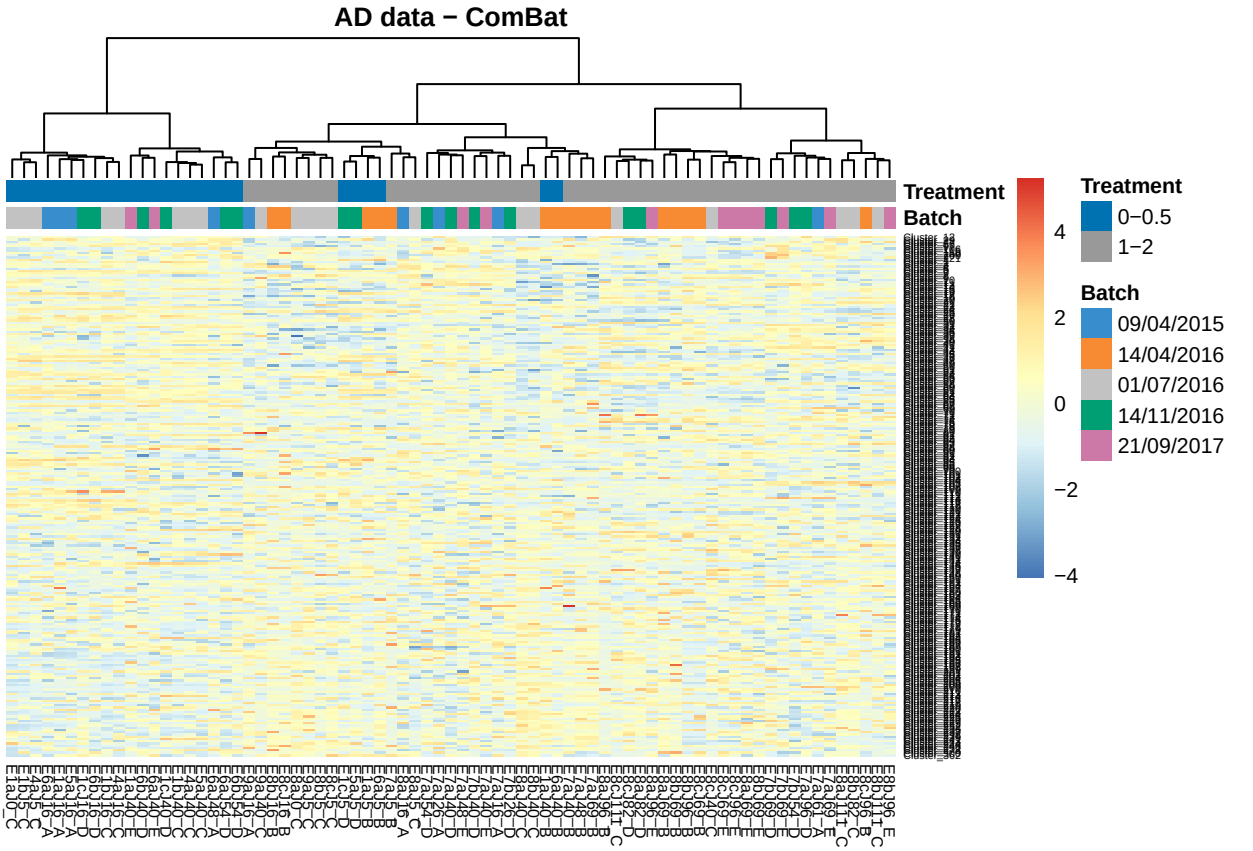
```
##################
# AD data
# before
ad.tss.clr.scale = scale(ad.tss.clr,center = T, scale = T) # scale on OTUs
ad.tss.clr.scale = scale(t(ad.tss.clr.scale), center = T, scale = T) # scale on samples

ad.anno_col = data.frame(Batch = ad.batch, Treatment = ad.trt)
ad.anno_metabo_colors = list(Batch = c('09/04/2015'="#388ECC",'14/04/2016'="#F68B33",'01/07/2016'="#C2C


pheatmap(ad.tss.clr.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - before')
```



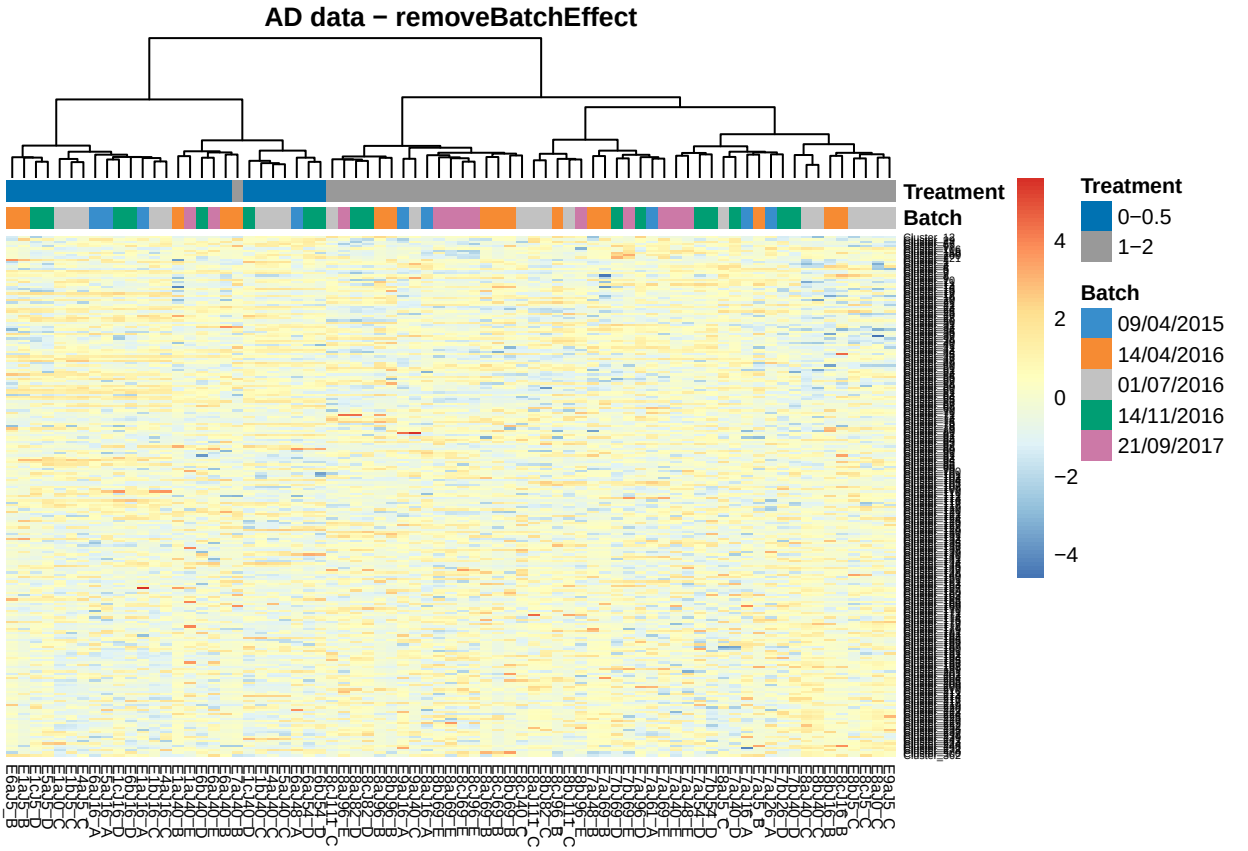

```
# BMC
ad.bmc.scale = scale(ad.bmc,center = T, scale = T) # scale on OTUs
ad.bmc.scale = scale(t(ad.bmc.scale), center = T, scale = T) # scale on samples

pheatmap(ad.bmc.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - BMC')
```



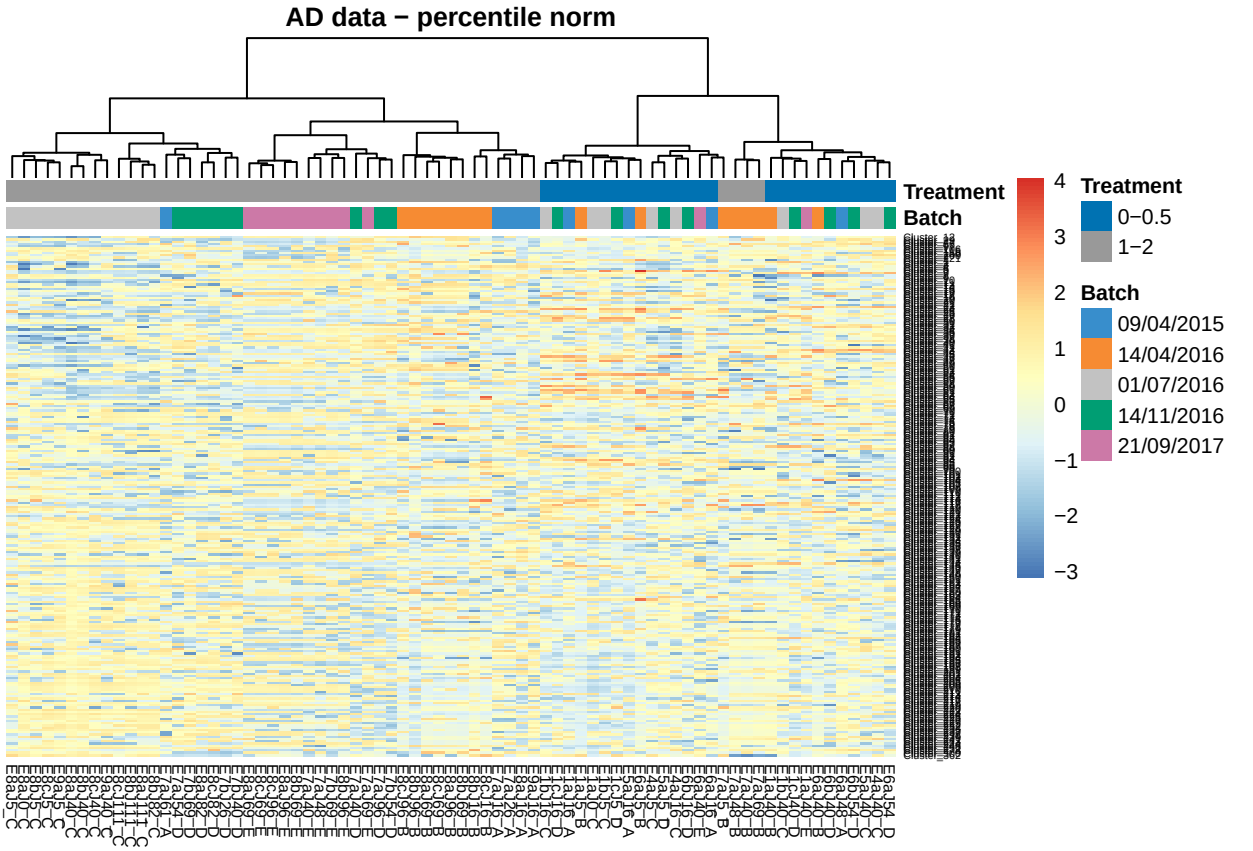

```
# ComBat
ad.combat.scale = scale(ad.combat,center = T, scale = T) # scale on OTUs
ad.combat.scale = scale(t(ad.combat.scale), center = T, scale = T) # scale on samples

pheatmap(ad.combat.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - ComBat')
```



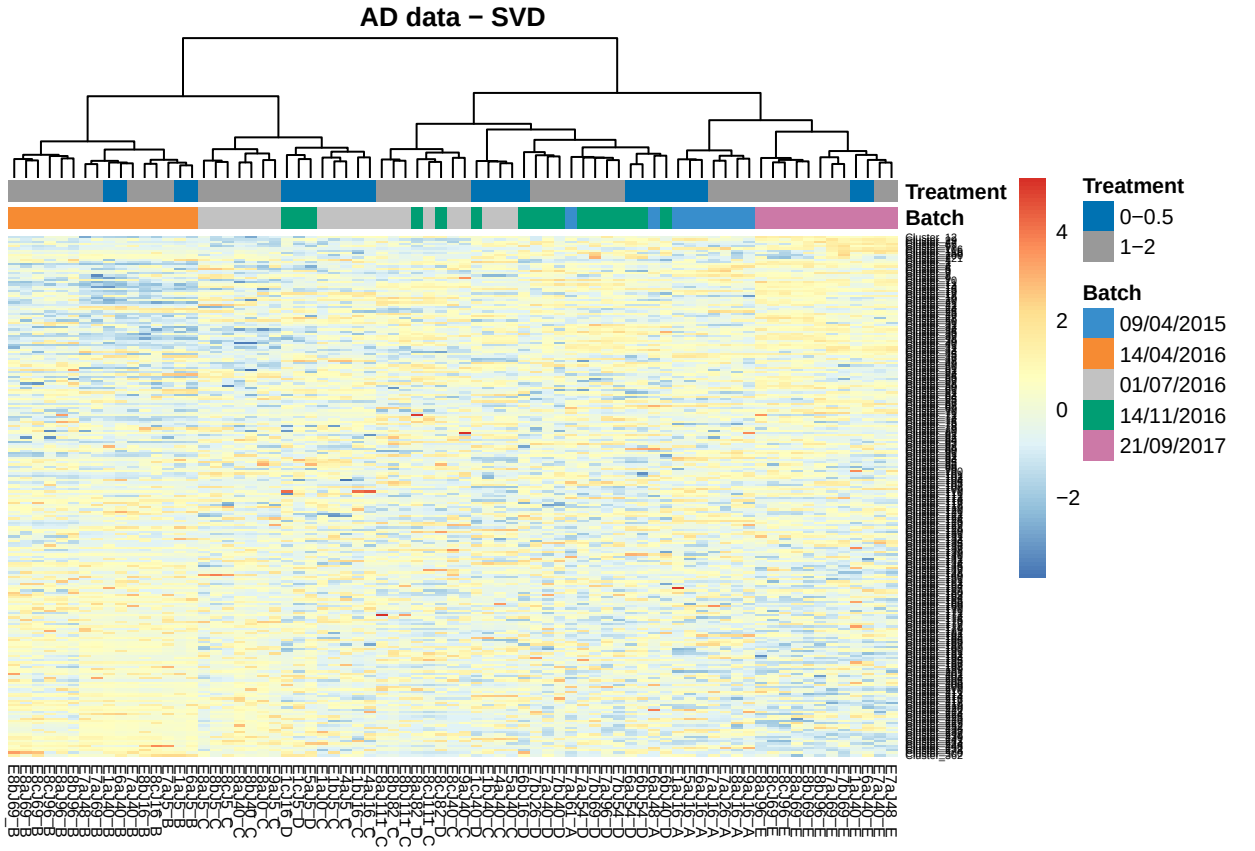

```
# removeBatchEffect
ad.limma.scale = scale(ad.limma,center = T, scale = T) # scale on OTUs
ad.limma.scale = scale(t(ad.limma.scale), center = T, scale = T) # scale on samples

pheatmap(ad.limma.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - removeBatchEffect')
```



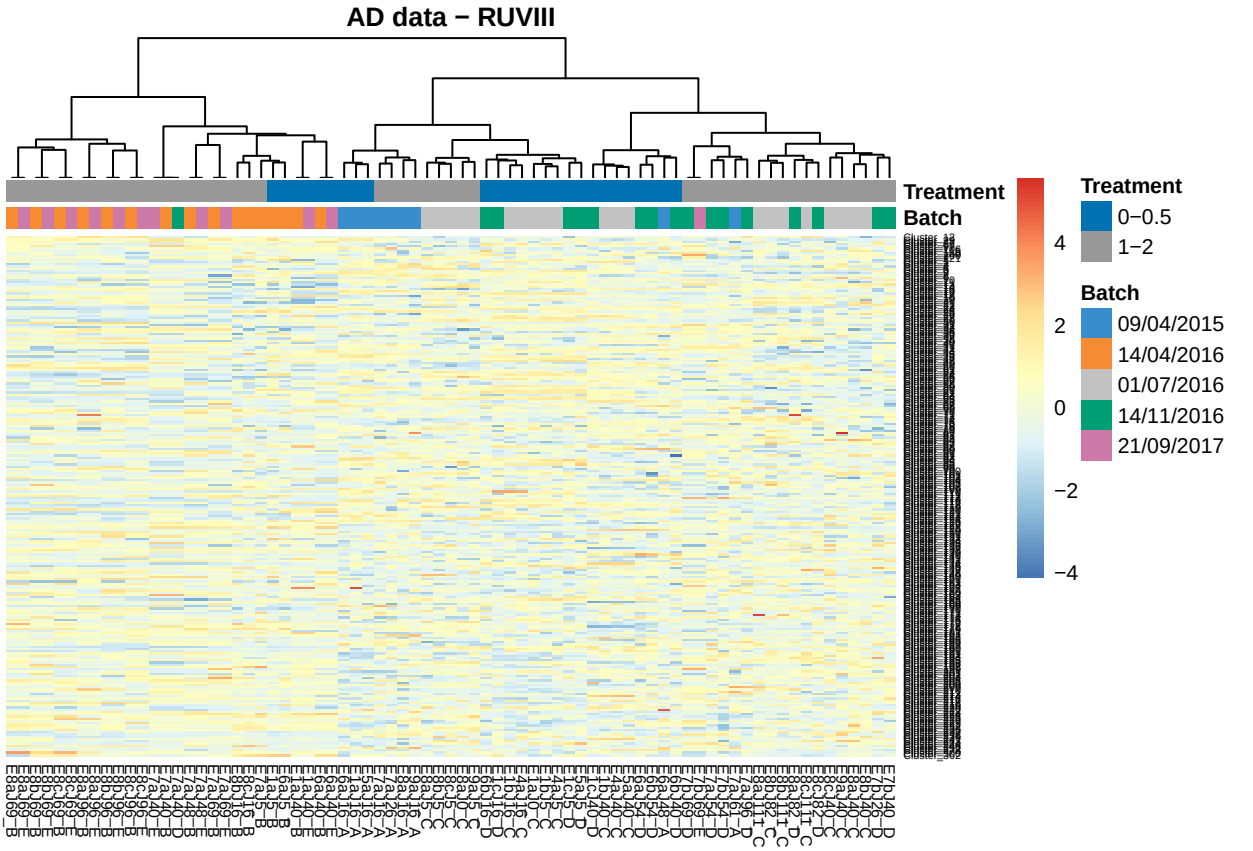

```
# percentile normalisation
ad.percentile.scale = scale(ad.percentile,center = T, scale = T) # scale on OTUs
ad.percentile.scale = scale(t(ad.percentile.scale), center = T, scale = T) # scale on samples

pheatmap(ad.percentile.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - percentile norm')
```




```
# SVD
ad.svd.scale = scale(ad.svd,center = T, scale = T) # scale on OTUs
ad.svd.scale = scale(t(ad.svd.scale), center = T, scale = T) # scale on samples

pheatmap(ad.svd.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - SVD')
```




```
# RUVIII
ad.ruv.scale = scale(ad.ruvIII,center = T, scale = T) # scale on OTUs
ad.ruv.scale = scale(t(ad.ruv.scale), center = T, scale = T) # scale on samples

pheatmap(ad.ruv.scale,
         scale = 'none',
         cluster_rows = F,
         cluster_cols = T,
         fontsize_row=4, fontsize_col=6,
         fontsize = 8,
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D",
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_metabo_colors,
         border_color = 'NA',
         main = 'AD data - RUVIII')
```



In the heatmaps of AD data, samples within batch 14/04/2016 are clustered and distinct from other samples, indicating a batch effect. After batch correction with removeBatchEffect and percentile normalisation, the data are almost clustered by treatments rather than batches, therefore, removeBatchEffect and percentile normalisation not only remove batch effects, but also disengage the treatment effects.

4.2 Variance calculation

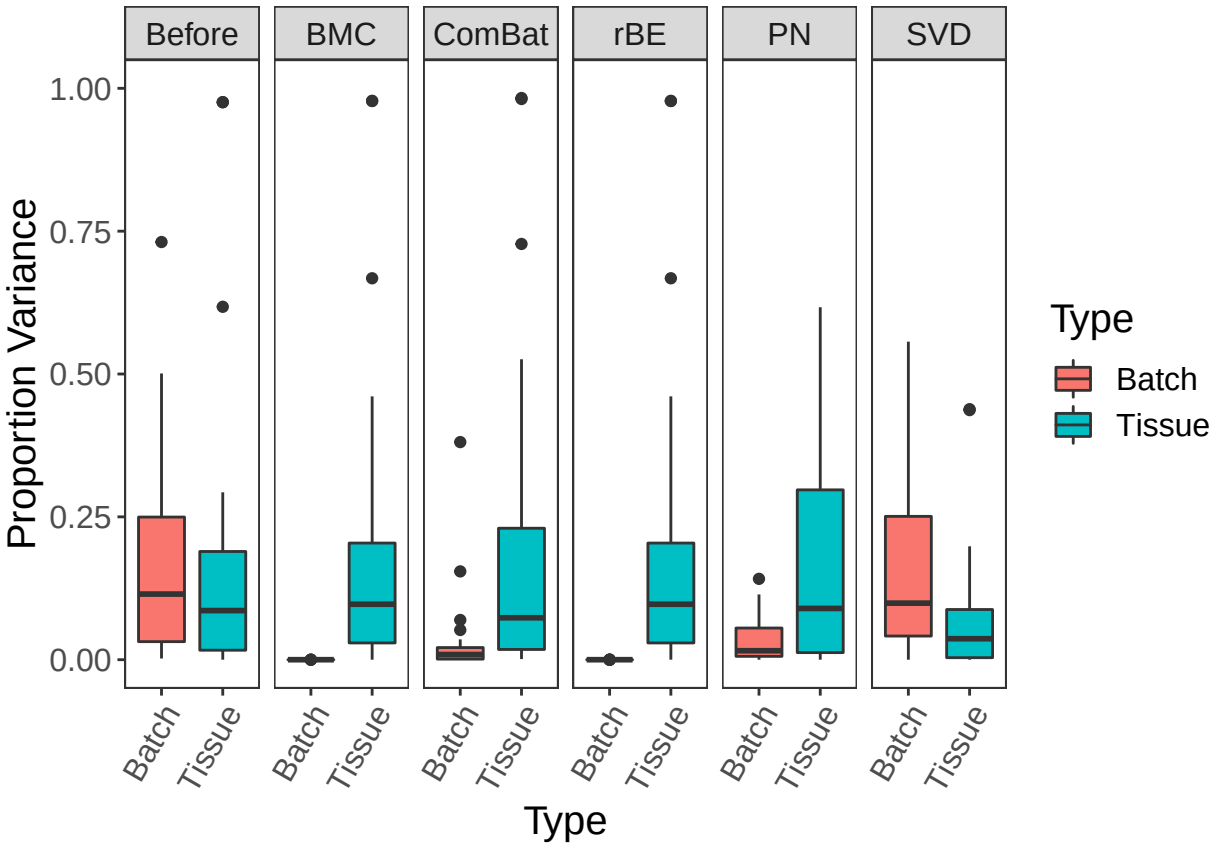
Compared to diagnostic plots, quantitative approaches are more objective and direct. The evaluation is done quantitatively before and after batch effect removal and the biological (treatment) effect must also be assessed before and after the batch effect correction to ensure it has been preserved.

4.2.1 Linear model per variable

The variance explained by batch or treatment per variable can be calculated by a linear model. Density plots or box plots can then be used to visualise the variance calculated from each variable. Following batch effect correction, the percentage of variance explained by the treatment should be greater than the batch.

```
## Sponge data
sponge.form <- ~ sponge.trt + sponge.batch
sponge.info = as.data.frame(cbind(rownames(sponge.tss.clr),sponge.trt,sponge.batch))
rownames(sponge.info) = rownames(sponge.tss.clr)

# before
sponge.varPart.before <- fitExtractVarPartModel(t(sponge.tss.clr), sponge.form, sponge.info)
```

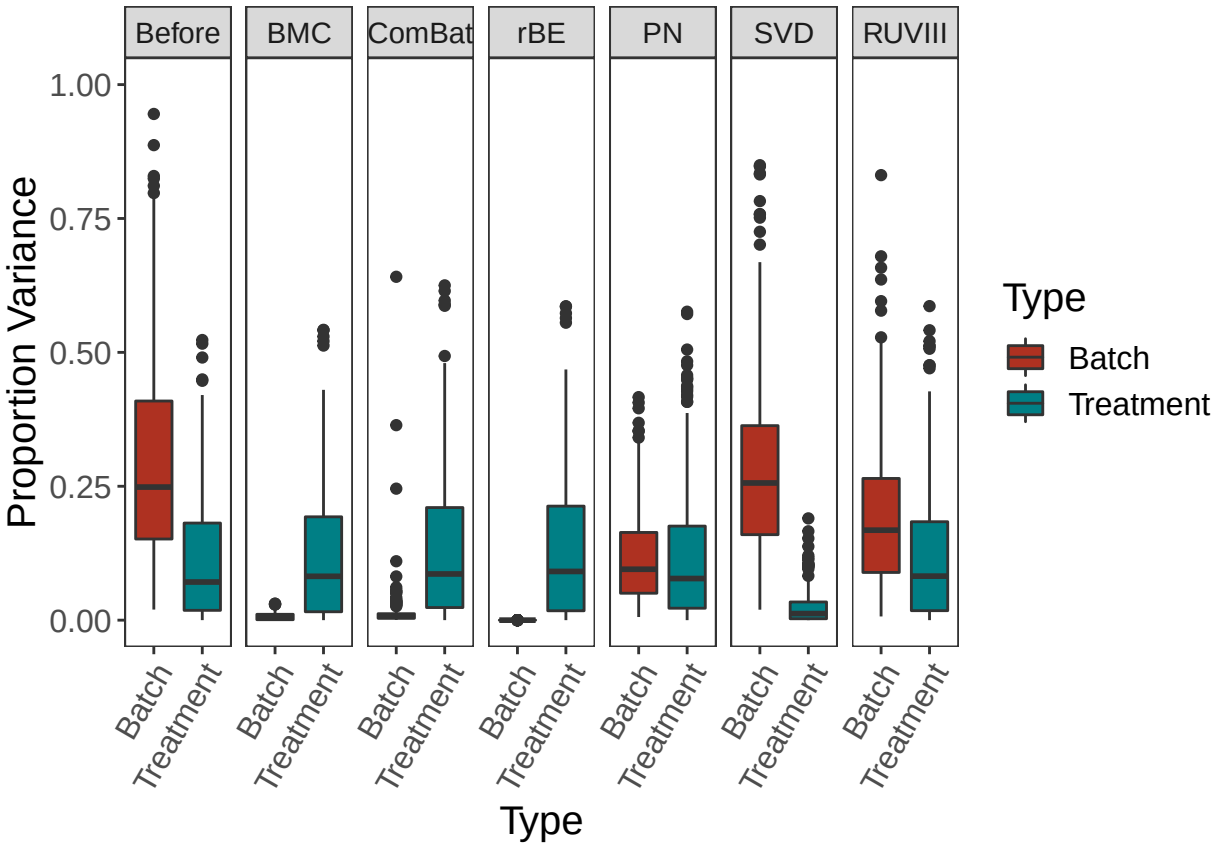
```
# BMC
sponge.varPart.bmc <- fitExtractVarPartModel(t(sponge.bmc), sponge.form, sponge.info)

# combat
sponge.varPart.combat <- fitExtractVarPartModel(t(sponge.combat), sponge.form, sponge.info)

# removeBatchEffect
sponge.varPart.limma <- fitExtractVarPartModel(t(sponge.limma), sponge.form, sponge.info)

# percentile normalisation
sponge.varPart.percentile <- fitExtractVarPartModel(t(sponge.percentile), sponge.form, sponge.info)

# svd
sponge.varPart.svd <- fitExtractVarPartModel(t(sponge.svd), sponge.form, sponge.info)


################
#merge them
sponge.variance =  rbind(cbind(variance = sponge.varPart.before$sponge.batch,Type = rep('Batch',24), me
                               cbind(variance = sponge.varPart.before$sponge.trt,Type = rep('Tissue',24),
                               cbind(variance = sponge.varPart.bmc$sponge.batch,Type = rep('Batch',24), m
                               cbind(variance = sponge.varPart.bmc$sponge.trt,Type = rep('Tissue',24), me
                               cbind(variance = sponge.varPart.combat$sponge.batch,Type = rep('Batch',24)
                               cbind(variance = sponge.varPart.combat$sponge.trt,Type = rep('Tissue',24),
                               cbind(variance = sponge.varPart.limma$sponge.batch,Type = rep('Batch',24),
                               cbind(variance = sponge.varPart.limma$sponge.trt,Type = rep('Tissue',24), r
                               cbind(variance = sponge.varPart.percentile$sponge.batch,Type = rep('Batch'
                               cbind(variance = sponge.varPart.percentile$sponge.trt,Type = rep('Tissue',
                               cbind(variance = sponge.varPart.svd$sponge.batch,Type = rep('Batch',24), m
                               cbind(variance = sponge.varPart.svd$sponge.trt,Type = rep('Tissue',24), me

sponge.variance = as.data.frame(sponge.variance)
sponge.variance$Type = factor(sponge.variance$Type,levels = unique(sponge.variance$Type))
sponge.variance$method = factor(sponge.variance$method,levels = unique(sponge.variance$method))
sponge.variance$variance = as.numeric(as.character(sponge.variance$variance))

ggplot(sponge.variance, aes(x=Type, y=variance,fill=Type)) + geom_boxplot() + facet_grid(cols = vars(me
```



BMC, ComBat and removeBatchEffect successfully removed batch variation while preserving treatment variation, but SVD performed poorly. Percentile normalisation preserved sufficient treatment variation, but removed less batch effect variation than BMC, ComBat and removeBatchEffect. Percentile normalisation used percentiles instead of actual values. This leads to information loss. Moreover, it was designed for case-control microbiome studies, which differ from our example studies. This method would therefore be more effective with control samples in each batch. SVD was inefficient at targeting batch effects, as it assumes that batch effects cause the largest variation in the data and should therefore appear as the first component. However, in sponge data the batch effect is mainly present on the second component as was shown in PCA sample plots with density, which results in a miscorrection of batch effects.

```
##########
# AD data
ad.form <- ~ ad.trt + ad.batch
ad.info = as.data.frame(cbind(rownames(ad.tss.clr),ad.trt,ad.batch))
rownames(ad.info) = rownames(ad.tss.clr)

# before
ad.varPart.before <- fitExtractVarPartModel(t(ad.tss.clr), ad.form, ad.info)

# BMC
ad.varPart.bmc <- fitExtractVarPartModel(t(ad.bmc), ad.form, ad.info)

# combat
ad.varPart.combat <- fitExtractVarPartModel(t(ad.combat), ad.form, ad.info)

# removeBatchEffect
ad.varPart.limma <- fitExtractVarPartModel(t(ad.limma), ad.form, ad.info)
```

```
# percentile normalisation
ad.varPart.percentile <- fitExtractVarPartModel(t(ad.percentile), ad.form, ad.info)

# svd
ad.varPart.svd <- fitExtractVarPartModel(t(ad.svd), ad.form, ad.info)

# ruv
ad.varPart.ruv <- fitExtractVarPartModel(t(ad.ruvIII), ad.form, ad.info)


#################
#merge them
ad.variance =  rbind(cbind(variance = ad.varPart.before$ad.batch,Type = rep('Batch',231), method = rep(
                           cbind(variance = ad.varPart.before$ad.trt,Type = rep('Treatment',231), meth
                           cbind(variance = ad.varPart.bmc$ad.batch,Type = rep('Batch',231), method =
                           cbind(variance = ad.varPart.bmc$ad.trt,Type = rep('Treatment',231), method
                           cbind(variance = ad.varPart.combat$ad.batch,Type = rep('Batch',231), metho
                           cbind(variance = ad.varPart.combat$ad.trt,Type = rep('Treatment',231), meth
                           cbind(variance = ad.varPart.limma$ad.batch,Type = rep('Batch',231), method
                           cbind(variance = ad.varPart.limma$ad.trt,Type = rep('Treatment',231), metho
                           cbind(variance = ad.varPart.percentile$ad.batch,Type = rep('Batch',231), m
                           cbind(variance = ad.varPart.percentile$ad.trt,Type = rep('Treatment',231),
                           cbind(variance = ad.varPart.svd$ad.batch,Type = rep('Batch',231), method =
                           cbind(variance = ad.varPart.svd$ad.trt,Type = rep('Treatment',231), method
                           cbind(variance = ad.varPart.ruv$ad.trt,Type = rep('Treatment',231), method

ad.variance = as.data.frame(ad.variance)
ad.variance$Type = factor(ad.variance$Type,levels = unique(ad.variance$Type))
ad.variance$method = factor(ad.variance$method,levels = unique(ad.variance$method))
ad.variance$variance = as.numeric(as.character(ad.variance$variance))

ggplot(ad.variance, aes(x=Type, y=variance,fill=Type)) + geom_boxplot() + facet_grid(cols = vars(method
```



Similar results as in sponge data, BMC, ComBat and removeBatchEffect successfully removed batch variation while preserving treatment variation, but SVD performed poorly. Percentile normalisation and RUVIII preserved sufficient treatment variation, but removed less batch effect variation than BMC, ComBat and removeBatchEffect. It is possible the negative control variables or technical replicate samples did not entirely capture the batch effects. SVD was also inefficient at targeting batch effects, as in AD data the batch effect is present on both first and second components, which results in a miscorrection of batch effects.

4.2.2 RDA

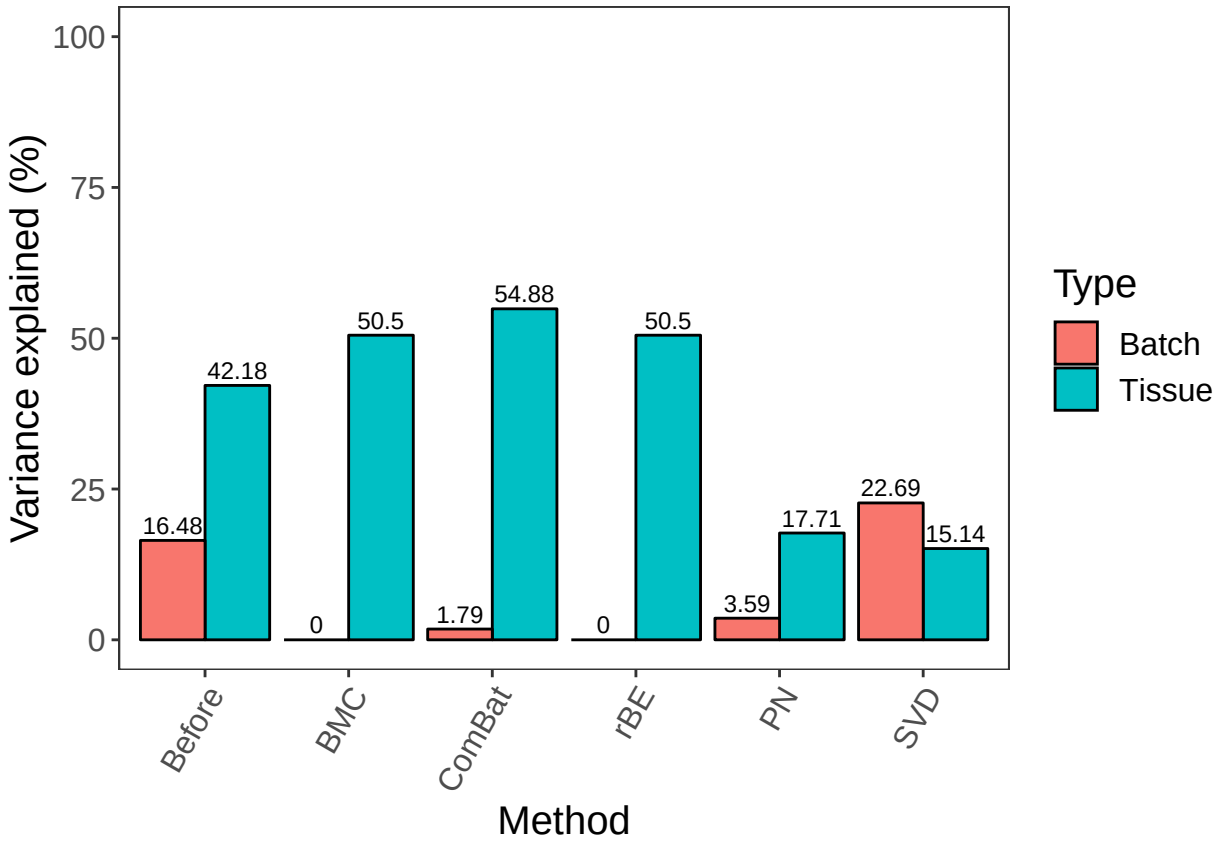
The multivariate method pRDA can be applied to calculate the variance of batch and treatment effects for all variables at once.

```
# Sponge data
sponge.data.design = numeric()
sponge.data.design$group = sponge.trt
sponge.data.design$batch = sponge.batch

# before
# conditioning on a batch effect
sponge.rda.before1 = rda(sponge.tss.clr ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.before2 = rda(sponge.tss.clr ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.before = sponge.rda.before1$pCCA$tot.chi*100/sponge.rda.before1$tot.chi
sponge.rda.trt_prop.before = sponge.rda.before2$pCCA$tot.chi*100/sponge.rda.before2$tot.chi

# BMC
```

```
# conditioning on a batch effect
sponge.rda.bmc1 = rda(sponge.bmc ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.bmc2 = rda(sponge.bmc ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.bmc = sponge.rda.bmc1$pCCA$tot.chi*100/sponge.rda.bmc1$tot.chi
sponge.rda.trt_prop.bmc = sponge.rda.bmc2$pCCA$tot.chi*100/sponge.rda.bmc2$tot.chi


# combat
# conditioning on a batch effect
sponge.rda.combat1 = rda(sponge.combat ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.combat2 = rda(sponge.combat ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.combat = sponge.rda.combat1$pCCA$tot.chi*100/sponge.rda.combat1$tot.chi
sponge.rda.trt_prop.combat = sponge.rda.combat2$pCCA$tot.chi*100/sponge.rda.combat2$tot.chi


# limma
# conditioning on a batch effect
sponge.rda.limma1 = rda(sponge.limma ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.limma2 = rda(sponge.limma ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.limma = sponge.rda.limma1$pCCA$tot.chi*100/sponge.rda.limma1$tot.chi
sponge.rda.trt_prop.limma = sponge.rda.limma2$pCCA$tot.chi*100/sponge.rda.limma2$tot.chi


# percentile
# conditioning on a batch effect
sponge.rda.percentile1 = rda(sponge.percentile ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.percentile2 = rda(sponge.percentile ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.percentile = sponge.rda.percentile1$pCCA$tot.chi*100/sponge.rda.percentile1$tot.chi
sponge.rda.trt_prop.percentile = sponge.rda.percentile2$pCCA$tot.chi*100/sponge.rda.percentile2$tot.chi


# SVD
# conditioning on a batch effect
sponge.rda.svd1 = rda(sponge.svd ~ group + Condition(batch), data = sponge.data.design)
sponge.rda.svd2 = rda(sponge.svd ~ batch + Condition(group), data = sponge.data.design)

# amount of variance
sponge.rda.bat_prop.svd = sponge.rda.svd1$pCCA$tot.chi*100/sponge.rda.svd1$tot.chi
sponge.rda.trt_prop.svd = sponge.rda.svd2$pCCA$tot.chi*100/sponge.rda.svd2$tot.chi
```

```
# proportion
sponge.rda.prop.before = c(sponge.rda.bat_prop.before,sponge.rda.trt_prop.before)
sponge.rda.prop.bmc = c(sponge.rda.bat_prop.bmc,sponge.rda.trt_prop.bmc)
sponge.rda.prop.combat = c(sponge.rda.bat_prop.combat,sponge.rda.trt_prop.combat)
sponge.rda.prop.limma = c(sponge.rda.bat_prop.limma,sponge.rda.trt_prop.limma)
sponge.rda.prop.percentile = c(sponge.rda.bat_prop.percentile,sponge.rda.trt_prop.percentile)
```

```
sponge.rda.prop.svd= c(sponge.rda.bat_prop.svd,sponge.rda.trt_prop.svd)

sponge.rda.prop.val = c(sponge.rda.prop.before,sponge.rda.prop.bmc,sponge.rda.prop.combat,sponge.rda.pr
sponge.rda.prop = data.frame(prop = sponge.rda.prop.val, prop.r = round(sponge.rda.prop.val,2), Method =

sponge.rda.prop$Method =  factor(sponge.rda.prop$Method, levels = unique(sponge.rda.prop$Method))

ggplot(data = sponge.rda.prop, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position =
```



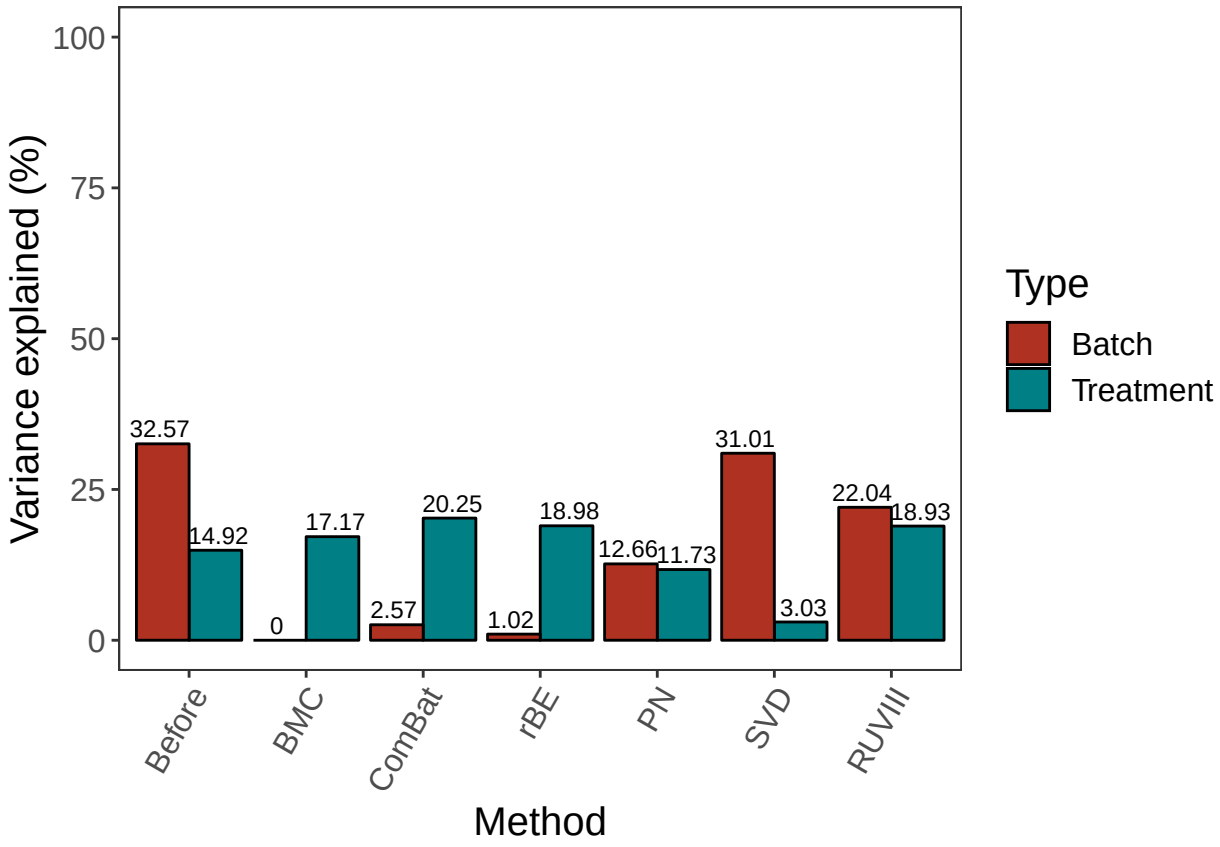
pRDA shows that BMC, ComBat and removeBatchEffect were more efficient at removing batch variation while preserving treatment variation. This result is in agreement with the proportional variance calculated using a linear model in the previous section 'linear model per variable'. ComBat removed relatively less batch variation compared with the other two methods, which implies that the batch effect is not purely systematic. The low efficiency of percentile normalisation is more obvious in the variance calculated with pRDA. It did not remove enough batch variation, nor preserve enough treatment variation in sponge data.

```
# AD data
ad.data.design = numeric()
ad.data.design$group = ad.trt
ad.data.design$batch = ad.batch

# before
# conditioning on a batch effect
ad.rda.before1 = rda(ad.tss.clr ~ group + Condition(batch), data = ad.data.design)
ad.rda.before2 = rda(ad.tss.clr ~ batch + Condition(group), data = ad.data.design)

# amount of variance
```

```
ad.rda.bat_prop.before = ad.rda.before1$pCCA$tot.chi*100/ad.rda.before1$tot.chi
ad.rda.trt_prop.before = ad.rda.before2$pCCA$tot.chi*100/ad.rda.before2$tot.chi

# BMC
# conditioning on a batch effect
ad.rda.bmc1 = rda(ad.bmc ~ group + Condition(batch), data = ad.data.design)
ad.rda.bmc2 = rda(ad.bmc ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.bmc = ad.rda.bmc1$pCCA$tot.chi*100/ad.rda.bmc1$tot.chi
ad.rda.trt_prop.bmc = ad.rda.bmc2$pCCA$tot.chi*100/ad.rda.bmc2$tot.chi


# combat
# conditioning on a batch effect
ad.rda.combat1 = rda(ad.combat ~ group + Condition(batch), data = ad.data.design)
ad.rda.combat2 = rda(ad.combat ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.combat = ad.rda.combat1$pCCA$tot.chi*100/ad.rda.combat1$tot.chi
ad.rda.trt_prop.combat = ad.rda.combat2$pCCA$tot.chi*100/ad.rda.combat2$tot.chi


# limma
# conditioning on a batch effect
ad.rda.limma1 = rda(ad.limma ~ group + Condition(batch), data = ad.data.design)
ad.rda.limma2 = rda(ad.limma ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.limma = ad.rda.limma1$pCCA$tot.chi*100/ad.rda.limma1$tot.chi
ad.rda.trt_prop.limma = ad.rda.limma2$pCCA$tot.chi*100/ad.rda.limma2$tot.chi


# percentile
# conditioning on a batch effect
ad.rda.percentile1 = rda(ad.percentile ~ group + Condition(batch), data = ad.data.design)
ad.rda.percentile2 = rda(ad.percentile ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.percentile = ad.rda.percentile1$pCCA$tot.chi*100/ad.rda.percentile1$tot.chi
ad.rda.trt_prop.percentile = ad.rda.percentile2$pCCA$tot.chi*100/ad.rda.percentile2$tot.chi


# SVD
# conditioning on a batch effect
ad.rda.svd1 = rda(ad.svd ~ group + Condition(batch), data = ad.data.design)
ad.rda.svd2 = rda(ad.svd ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.svd = ad.rda.svd1$pCCA$tot.chi*100/ad.rda.svd1$tot.chi
ad.rda.trt_prop.svd = ad.rda.svd2$pCCA$tot.chi*100/ad.rda.svd2$tot.chi
```

```
# RUVIII
# conditioning on a batch effect
ad.rda.ruv1 = rda(ad.ruvIII ~ group + Condition(batch), data = ad.data.design)
ad.rda.ruv2 = rda(ad.ruvIII ~ batch + Condition(group), data = ad.data.design)

# amount of variance
ad.rda.bat_prop.ruv = ad.rda.ruv1$pCCA$tot.chi*100/ad.rda.ruv1$tot.chi
ad.rda.trt_prop.ruv = ad.rda.ruv2$pCCA$tot.chi*100/ad.rda.ruv2$tot.chi
```

```
# proportion
ad.rda.prop.before = c(ad.rda.bat_prop.before,ad.rda.trt_prop.before)
ad.rda.prop.bmc = c(ad.rda.bat_prop.bmc,ad.rda.trt_prop.bmc)
ad.rda.prop.combat = c(ad.rda.bat_prop.combat,ad.rda.trt_prop.combat)
ad.rda.prop.limma = c(ad.rda.bat_prop.limma,ad.rda.trt_prop.limma)
ad.rda.prop.percentile = c(ad.rda.bat_prop.percentile,ad.rda.trt_prop.percentile)
ad.rda.prop.svd= c(ad.rda.bat_prop.svd,ad.rda.trt_prop.svd)
ad.rda.prop.ruv= c(ad.rda.bat_prop.ruv,ad.rda.trt_prop.ruv)

#############
ad.rda.prop.val = c(ad.rda.prop.before,ad.rda.prop.bmc,ad.rda.prop.combat,ad.rda.prop.limma,ad.rda.prop
ad.rda.prop = data.frame(prop = ad.rda.prop.val, prop.r = round(ad.rda.prop.val,2), Method = rep(c('Bef

ad.rda.prop$Method =  factor(ad.rda.prop$Method, levels = unique(ad.rda.prop$Method))

ggplot(data = ad.rda.prop, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position = 'dod
```



Similar as results from sponge data, BMC, ComBat and removeBatchEffect were more efficient at removing batch vari-

ation while preserving treatment variation. ComBat removed relatively less batch variation compared with the other two methods. Percentile normalisation did not remove enough batch variation, nor preserve enough treatment variation in AD data. RUVIII preserved enough treatment variation but did not remove enough batch variation.

In sponge data, the results of BMC and removeBatchEffect are the same, while in AD data, removeBatchEffect removed less batch variation but preserved more treatment variation. This indicates some linear correlation exists between the batch and treatment effects, which might originate from the batch x treatment design.

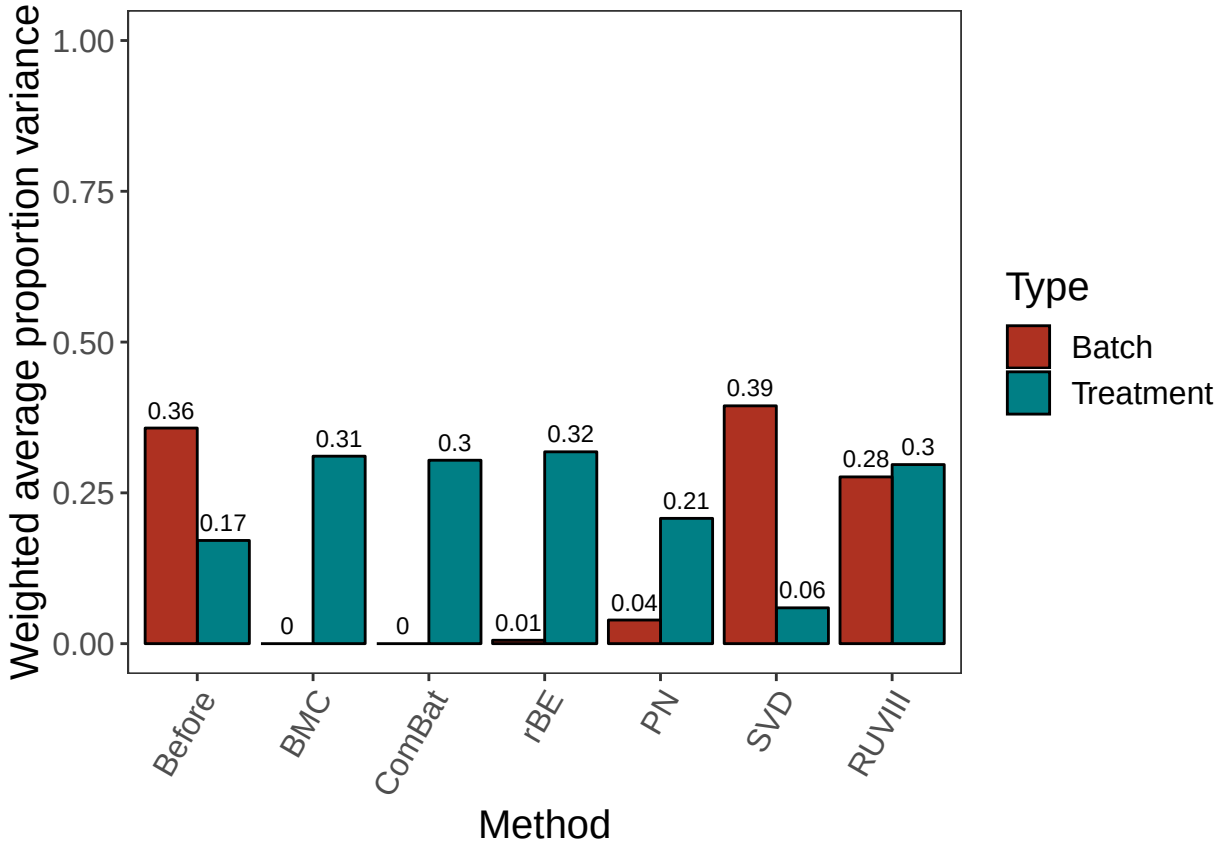
4.2.3 PVCA

PVCA can be applied as a validation method that complements pRDA, but it does not work for dataset with only 24 samples, therefore it cannot be applied on sponge data.

```
# AD data
ad.PVCA.score = data.frame(Interaction = NA, Batch = NA,Treatment = NA,Residuals = NA)

ad.Bat_Int.factors = data.frame(Batch = ad.batch, Treatment = ad.trt)
rownames(ad.Bat_Int.factors) = rownames(ad.tss.clr)
pdata <- AnnotatedDataFrame(ad.Bat_Int.factors)

# before
ad.eset.X.before <- new("ExpressionSet", exprs = t(ad.tss.clr), phenoData = pdata)
ad.pvcaObj.before <- pvcaBatchAssess(ad.eset.X.before, c('Batch','Treatment'), 0.6)
ad.values.before = ad.pvcaObj.before$dat
ad.PVCA.score[1,] = ad.values.before

# bmc
ad.eset.X.bmc <- new("ExpressionSet", exprs = t(ad.bmc), phenoData = pdata)
ad.pvcaObj.bmc <- pvcaBatchAssess(ad.eset.X.bmc, c('Batch','Treatment'), 0.6)
ad.values.bmc = ad.pvcaObj.bmc$dat
ad.PVCA.score[2,] = ad.values.bmc


# combat

ad.eset.X.combat <- new("ExpressionSet", exprs = t(ad.combat), phenoData = pdata)
ad.pvcaObj.combat <- pvcaBatchAssess(ad.eset.X.combat, c('Batch','Treatment'), 0.6)
ad.values.combat = ad.pvcaObj.combat$dat
ad.PVCA.score[3,] = ad.values.combat

# PN

ad.eset.X.percentile <- new("ExpressionSet", exprs = t(ad.percentile), phenoData = pdata)
ad.pvcaObj.percentile <- pvcaBatchAssess(ad.eset.X.percentile, c('Batch','Treatment'), 0.6)
ad.values.percentile = ad.pvcaObj.percentile$dat
ad.PVCA.score[5,] = ad.values.percentile


# limma

ad.eset.X.limma <- new("ExpressionSet", exprs = t(ad.limma), phenoData = pdata)
ad.pvcaObj.limma <- pvcaBatchAssess(ad.eset.X.limma, c('Batch','Treatment'), 0.6)
ad.values.limma = ad.pvcaObj.limma$dat
ad.PVCA.score[4,] = ad.values.limma
```

```
# svd

ad.eset.X.svd <- new("ExpressionSet", exprs = t(ad.svd), phenoData = pdata)
ad.pvcaObj.svd <- pvcaBatchAssess(ad.eset.X.svd, c('Batch','Treatment'), 0.6)
ad.values.svd = ad.pvcaObj.svd$dat
ad.PVCA.score[6,] = ad.values.svd


# RUVIII

ad.eset.X.ruv <- new("ExpressionSet", exprs = t(ad.ruvIII), phenoData = pdata)
ad.pvcaObj.ruv <- pvcaBatchAssess(ad.eset.X.ruv, c('Batch','Treatment'), 0.6)
ad.values.ruv = ad.pvcaObj.ruv$dat
ad.PVCA.score[7,] = ad.values.ruv

rownames(ad.PVCA.score) =c('Before','BMC','ComBat','rBE','PN','SVD','RUVIII')
```

```
#############
ad.pvca.prop.val = c(ad.PVCA.score$Batch,ad.PVCA.score$Treatment)
ad.pvca.prop = data.frame(prop = ad.pvca.prop.val, prop.r = round(ad.pvca.prop.val,2), Method = rep(c('

ad.pvca.prop$Method =  factor(ad.pvca.prop$Method, levels = unique(ad.pvca.prop$Method))

ggplot(data = ad.pvca.prop, aes(x=Method,y=prop,fill = Type)) + geom_bar(stat="identity",position = 'do
```



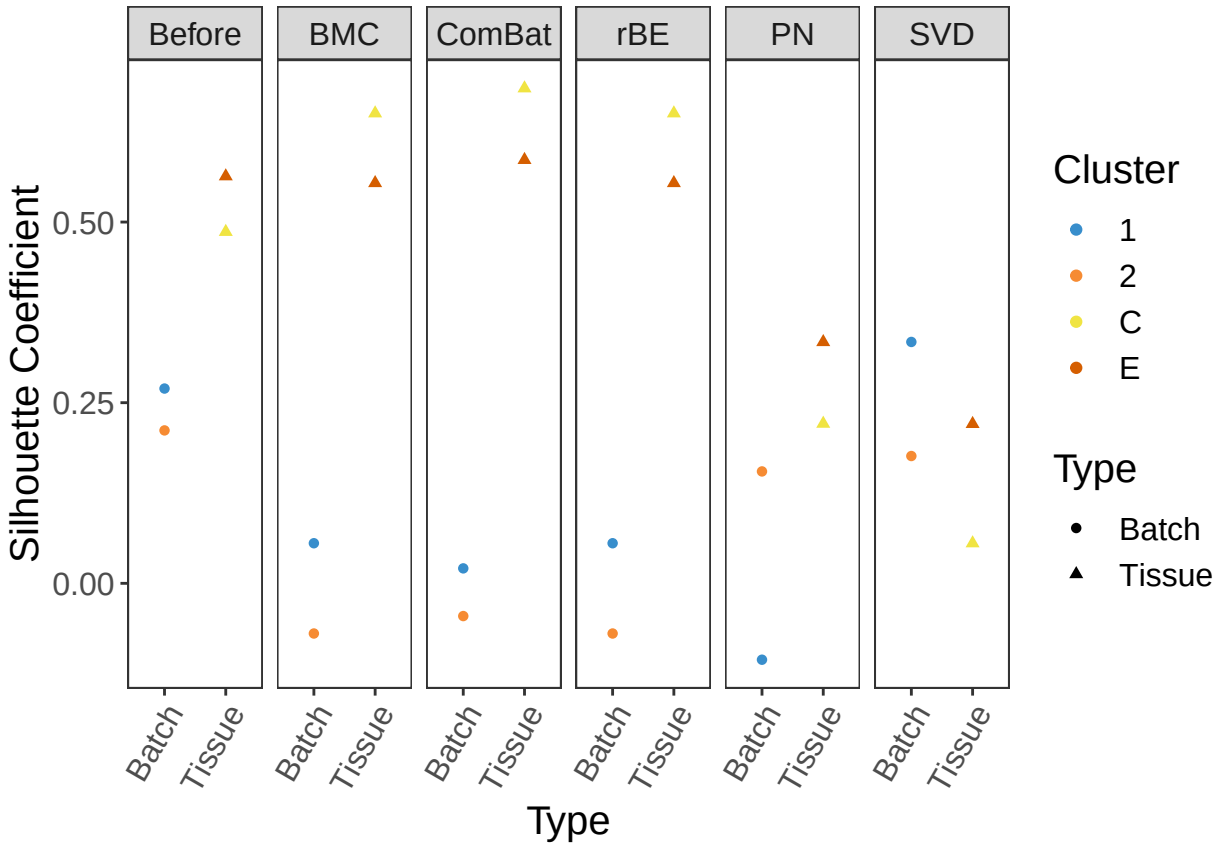
Proportionally, BMC, ComBat and rBE removed more of the variance explained by batch and maintained more variance explained by treatment than other methods.

4.2.4 Silhouette coefficient

This coefficient measures how cohesive a sample is to its own cluster compared to other clusters. However, it has been adapted to assess the consistency of sample groups based on treatment or batch effects.

The average silhouette coefficient width across all samples in one group (batch $\bar{s}_b$ or treatment $\bar{s}_t$) is calculated and plotted here. If $\bar{s}_b$ is close to 0 there is no batch effect, and if $\bar{s}_t$ is close to 1 or -1 there is a treatment effect.

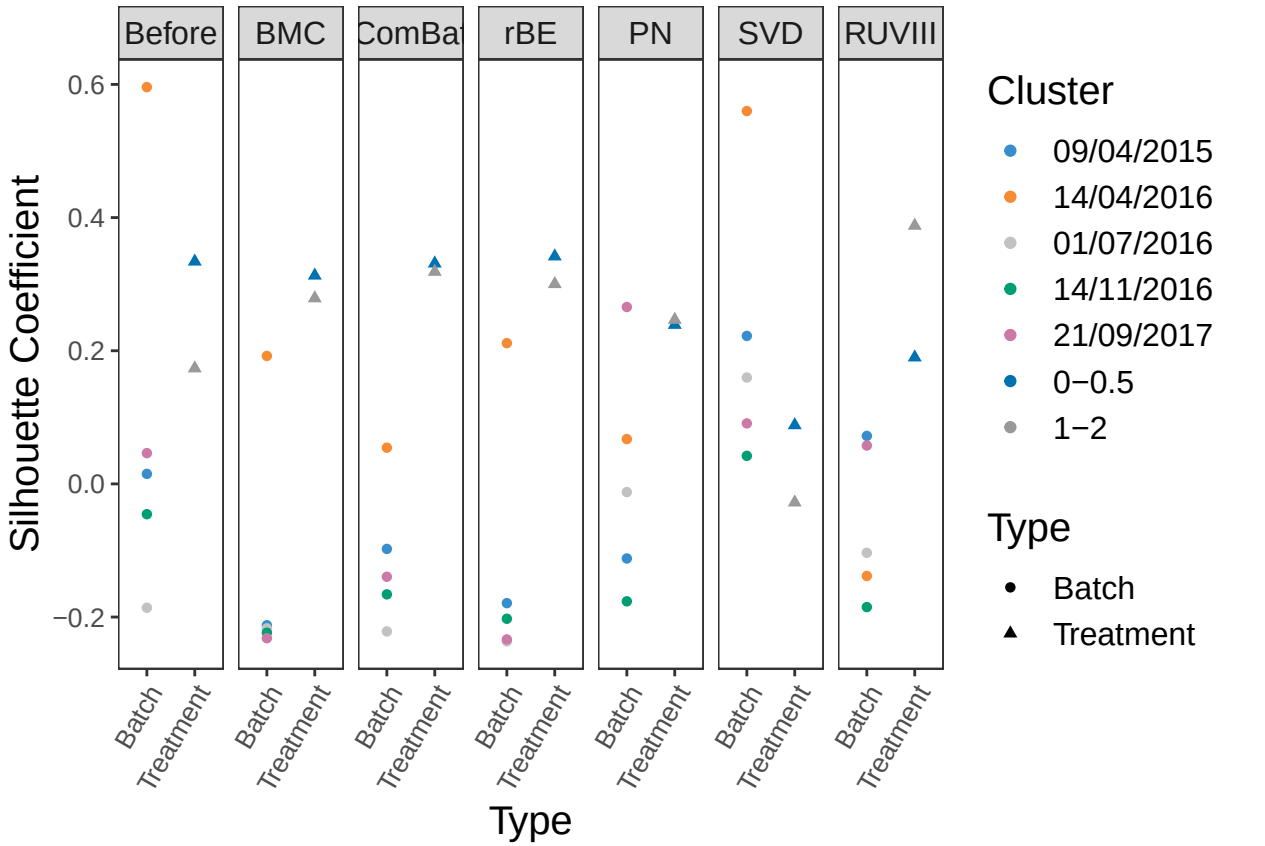
```
###################
# Sponge data

sponge.silh.before = calc.sil(sponge.pca.before$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 =
sponge.silh.bmc = calc.sil(sponge.pca.bmc$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = 'Batcl
sponge.silh.combat = calc.sil(sponge.pca.combat$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 =
sponge.silh.limma = calc.sil(sponge.pca.limma$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = '
sponge.silh.percentile = calc.sil(sponge.pca.percentile$variates$X,y1 = sponge.batch, y2= sponge.trt, na
sponge.silh.svd = calc.sil(sponge.pca.svd$variates$X,y1 = sponge.batch, y2= sponge.trt, name.y1 = 'Batcl


sponge.silh.plot = rbind(sponge.silh.before, sponge.silh.bmc, sponge.silh.combat, sponge.silh.limma, sp
sponge.silh.plot$method = c(rep('Before', nrow(sponge.silh.before)),
                            rep('BMC', nrow(sponge.silh.bmc)),
                            rep('ComBat', nrow(sponge.silh.combat)),
                            rep('rBE', nrow(sponge.silh.limma)),
                            rep('PN', nrow(sponge.silh.percentile)),
                            rep('SVD', nrow(sponge.silh.svd))
)
sponge.silh.plot$method = factor(sponge.silh.plot$method,levels = unique(sponge.silh.plot$method))
sponge.silh.plot$Cluster =  factor(sponge.silh.plot$Cluster, levels = unique(sponge.silh.plot$Cluster))
sponge.silh.plot$Type =  factor(sponge.silh.plot$Type, levels = unique(sponge.silh.plot$Type))


ggplot(sponge.silh.plot, aes(x=Type, y=silh.coeff, color = Cluster, shape = Type)) + geom_point() + face
```



$\bar{s}_b$ of different batches in sponge data decreased to 0 after correction with BMC, ComBat and removeBatchEffect, while $\bar{s}_t$ of different tissues increased or maintained at the same level as before correction.

```
############
# AD data
ad.silh.before = calc.sil(ad.pca.before$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2
ad.silh.bmc = calc.sil(ad.pca.bmc$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr
ad.silh.combat = calc.sil(ad.pca.combat$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2
ad.silh.limma = calc.sil(ad.pca.limma$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 =
ad.silh.percentile = calc.sil(ad.pca.percentile$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch'
ad.silh.svd = calc.sil(ad.pca.svd$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr
ad.silh.ruv = calc.sil(ad.pca.ruv$variates$X,y1 = ad.batch, y2= ad.trt, name.y1 = 'Batch',name.y2 = 'Tr


ad.silh.plot = rbind(ad.silh.before, ad.silh.bmc, ad.silh.combat, ad.silh.limma, ad.silh.percentile, ad
ad.silh.plot$method = c(rep('Before', nrow(ad.silh.before)),
                        rep('BMC', nrow(ad.silh.bmc)),
                        rep('ComBat', nrow(ad.silh.combat)),
                        rep('rBE', nrow(ad.silh.limma)),
                        rep('PN', nrow(ad.silh.percentile)),
                        rep('SVD', nrow(ad.silh.svd)),
                    rep('RUVIII', nrow(ad.silh.ruv))
)
ad.silh.plot$method = factor(ad.silh.plot$method, levels = unique(ad.silh.plot$method))
ad.silh.plot$Cluster =  factor(ad.silh.plot$Cluster, levels = unique(ad.silh.plot$Cluster))
ad.silh.plot$Type =  factor(ad.silh.plot$Type, levels = unique(ad.silh.plot$Type))
```

```
ggplot(ad.silh.plot, aes(x=Type, y=silh.coeff, color = Cluster, shape = Type)) + geom_point() + facet_g
```



In AD data that includes five batches, the interpretation of the results is challenging. After correction, BMC, ComBat, removeBatchEffect, percentile normalisation and RUVIII decreased the batch silhouette coefficients for the batch dated 14/04/2016, but increased the coefficients for the other batches. Therefore it is difficult to assess the efficiency of these methods in this particular case.

Chapter 5

Simulations of systematic and non-systematic batch effects

5.1 Mean=5,unequal variance

'Systematic' refers to homogeneous change amongst all microbial variables (OTUs) due to having the same source of variation. We illustrate this concept in a linear model framework where batch and treatment regression coefficients are estimated simultaneously on each OTU. For a given batch effect and a given OTU, we can formulate the systematic assumption as:

$$\beta_j \sim N(\mu, \sigma^2)$$

where β_j is the batch regression coefficient of OTU$_j$ ($j = 1, ..., p$). Here we consider the simplest case of a linear model with one batch predictor, but this formulation could be extended to a model with multiple batch predictors where the batch regression coefficients can represent more than two batch levels. In a univariate model that tests each OTU idividually, then the distribution of the batch coefficients of all OTUs is Gaussian with a mean μ, and standard deviation σ. This indicates that the batch effect has a similar, though not necessarily identical, influence on all OTUs.

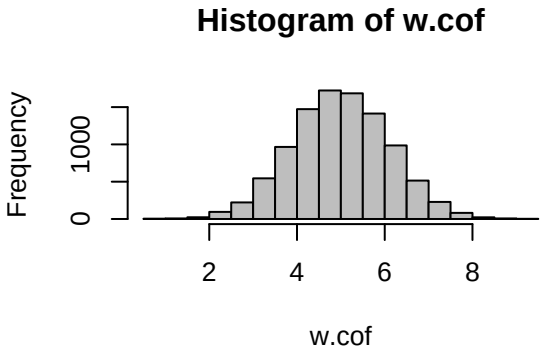
To illustrate the type of batch effects, we simulated a set of data with 50 samples and 10,000 OTUs each, based on the simulation approach from **?**.

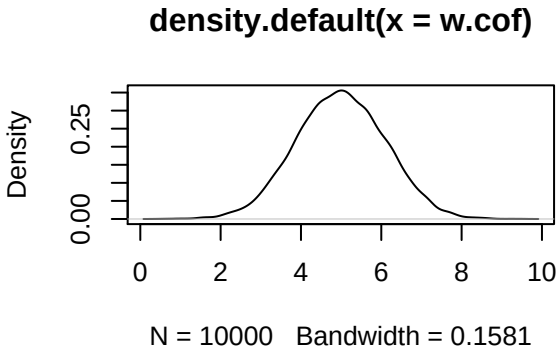
The dataset with a systematic batch effect:

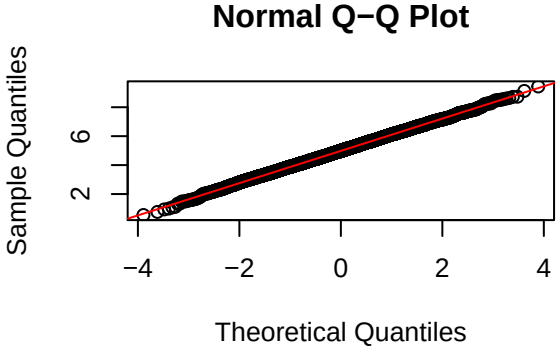
- $\beta_j \sim N(5, 1^2)$ for $j = 1, ..., p$ OTUs;
- $\sigma_j \sim N(0, 2^2)$ for $j = 1, ..., p$ OTUs; This variance per OTU is aimed to simulate data as realistically as possible.
- $\beta_{ij} \sim N(\beta_j, \sigma_j^2)$ for $i = 1, ..., n$ samples.

```
## Create the simulated data
m = 50
n = 10000
nc = 1000 #negative controls without treatment effects
p = 1
k = 1
ctl = rep(FALSE, n)
ctl[1:nc] = TRUE
# treatment effect
X = matrix(c(rep(0,floor(m/2)), rep(1,ceiling(m/2))), m, p)
beta = matrix(rnorm(p*n,5,1), p, n) #treatment coefficients
```

```
beta[,ctl] = 0
# batch effect
W = as.matrix(rep(0,m),m,k)
W[c(1:12,38:50),1] =  1
alpha = matrix(rnorm(k*n,5,1),k,n)
Y_alpha = sapply(alpha, function(alpha){rnorm(m, mean =  alpha, abs(rnorm(1, mean = 0, sd = 2)))})
YY_alpha = apply(Y_alpha,2,function(x){x*W})

epsilon = matrix(rnorm(m*n,0,1),m,n)
Y = X%*%beta + YY_alpha + epsilon


# estimate batch coefficient for each OTU
w.cof = c()
for(i in 1:ncol(Y)){
  res = lm(Y[,i] ~ X + W)
  sum.res = summary(res)
  w.cof[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(w.cof,col = 'gray')
plot(density(w.cof))
qqnorm(w.cof)
qqline(w.cof, col='red')
par(mfrow=c(1,1))
```







The histogram and density plots are plotted using estimated batch regression coefficients $\hat{\beta}_j$ for each OTU j.

5.2 Mean=0&5,unequal variance

Non-systematic batch effects have a heterogeneous influence on microbial variables. Using a linear model framework, as described previously, we can formulate this non-systematic assumption as:

$$\beta'_j \sim \begin{cases} N(0, \delta^2) & \text{for OTUs with no batch effect,} \\ N(\mu, \sigma^2) & \text{for OTUs with batch effect.} \end{cases}$$

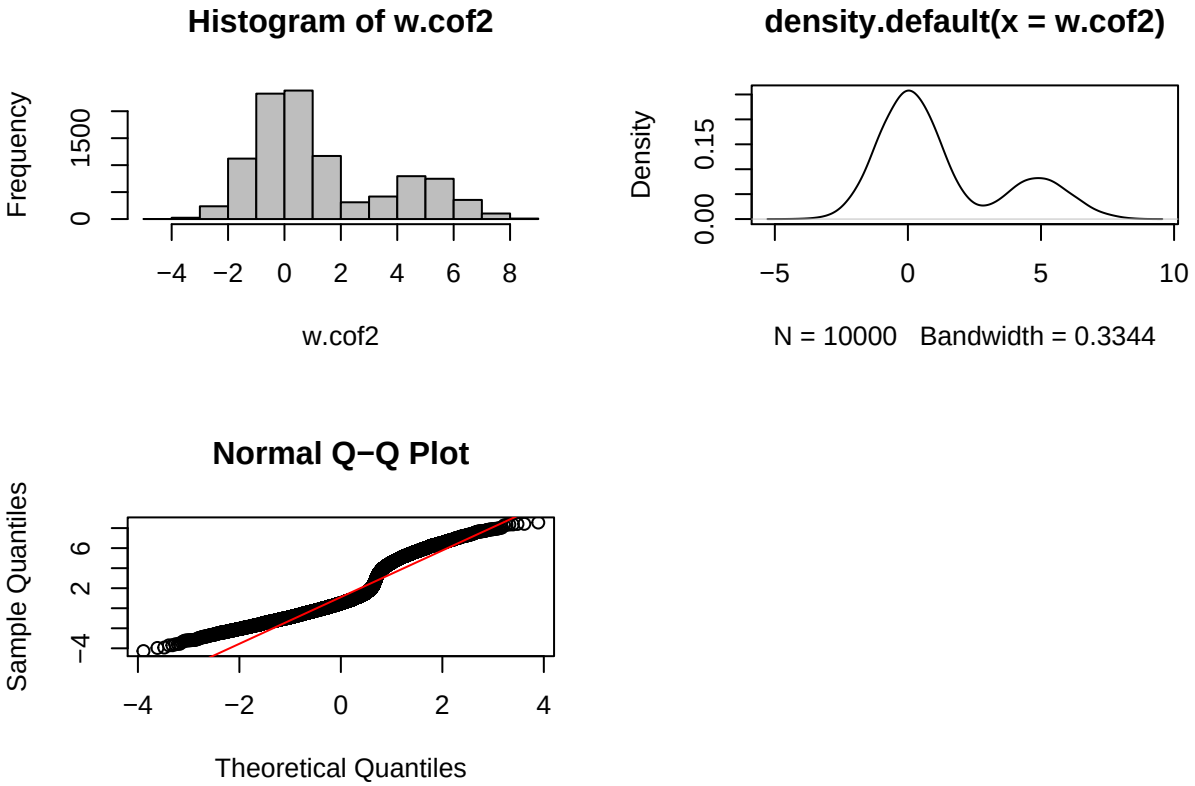
Therefore, the batch regression coefficients β'_j may follow skewed distributions with several modes.

The dataset with non-systematic batch effect:

- $\beta'_t \sim N(0, 1^2)$ and $\beta'_k \sim N(5, 1^2)$ for $t = 1, ..., T$ OTUs, $k = 1, ..., K$ OTUs and $T = \frac{3}{4}p$, $K = \frac{1}{4}p$;
- $\sigma'_j \sim N(0, 2^2)$ for $j = 1, ..., p$ OTUs;
- $\beta'_{ij} \sim N(\beta'_j, \sigma_j^{'2})$ for $i = 1, ..., n$ samples.

```
## Create the simulated data
m = 50
n = 10000
nc = 1000 #negative controls without treatment effects
p = 1
k = 1
ctl = rep(FALSE, n)
ctl[1:nc] = TRUE
# treatment effect
X = matrix(c(rep(0,floor(m/2)), rep(1,ceiling(m/2))), m, p)
beta = matrix(rnorm(p*n,5,1), p, n) #treatment coefficients
beta[,ctl] = 0
# batch effect
W = as.matrix(rep(0,m),m,k)
W[c(1:12,38:50),1] =  1
alpha2 = matrix(sample(c(rnorm(k*(3*n/4),0,1),rnorm(k*(n/4),5,1)),n),k,n)
Y_alpha2 = sapply(alpha2, function(alpha){rnorm(m, mean =  alpha, sd = abs(rnorm(1, mean = 0, sd = 2)) )
YY_alpha2 = apply(Y_alpha2,2,function(x){x*W})

epsilon = matrix(rnorm(m*n,0,1),m,n)
Y2 = X%*%beta + YY_alpha2 + epsilon




w.cof2 = c()
for(i in 1:ncol(Y2)){
  res = lm(Y2[,i] ~ X + W)
  sum.res = summary(res)
  w.cof2[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(w.cof2,col = 'gray')
plot(density(w.cof2))
qqnorm(w.cof2)
```

```
qqline(w.cof2, col='red')
par(mfrow=c(1,1))
```

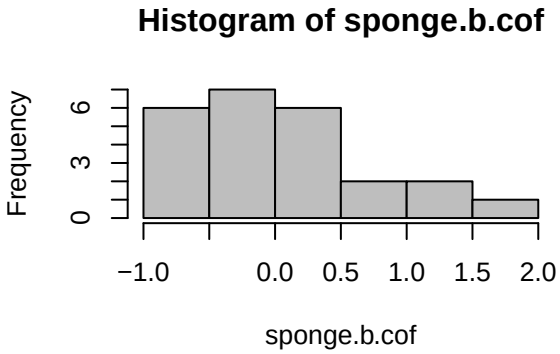


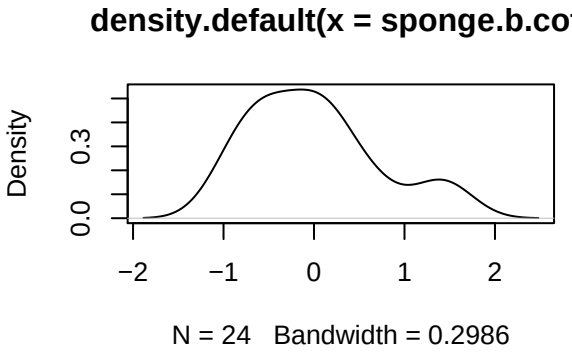
The histogram and density plots are plotted using estimated batch regression coefficients $\hat{\beta}_j$ for each OTU j, showing a bi-modal distribution.

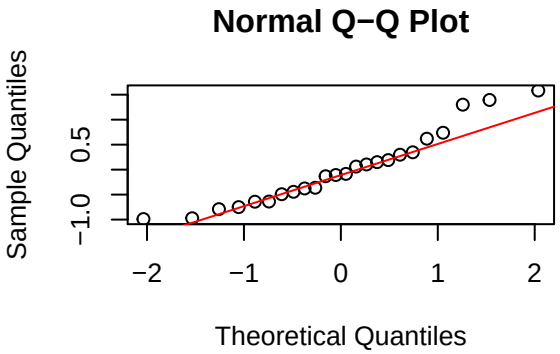
We observed similar patterns in our real case studies, suggesting that the batch effects are mixed with multiple sources and are non-systematic.

5.3 Sponge data

```
sponge.b.cof = c()
for(i in 1:ncol(sponge.tss.clr)){
  res = lm(sponge.tss.clr[,i] ~ sponge.trt + sponge.batch)
  sum.res = summary(res)
  sponge.b.cof[i] = sum.res$coefficients[3,1]
}

par(mfrow=c(2,2))
hist(sponge.b.cof,col = 'gray')
plot(density(sponge.b.cof))
qqnorm(sponge.b.cof)
qqline(sponge.b.cof, col='red')
par(mfrow=c(1,1))
```

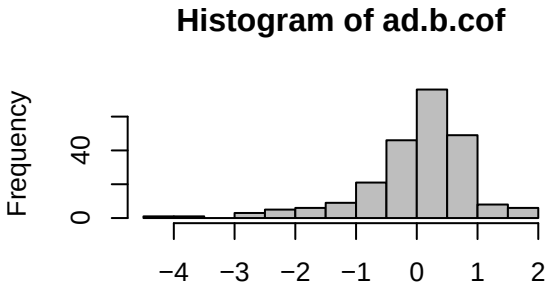


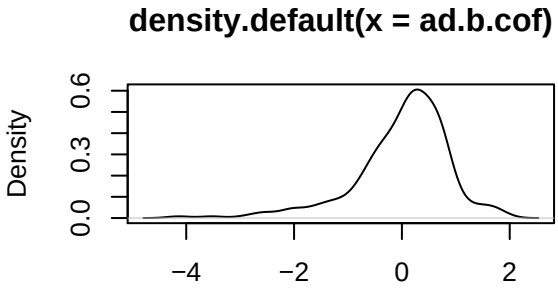


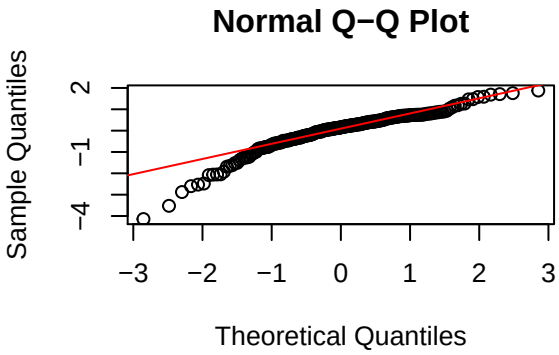


5.4 AD data

```
ad.b.cof = c()
ad.batch.relevel = relevel(ad.batch, '01/07/2016')
for(i in 1:ncol(ad.tss.clr)){
  res = lm(ad.tss.clr[,i] ~ ad.trt + ad.batch.relevel)
  sum.res = summary(res)
  ad.b.cof[i] = sum.res$coefficients[4,1]
}

par(mfrow=c(2,2))
hist(ad.b.cof,col = 'gray')
plot(density(ad.b.cof))
qqnorm(ad.b.cof)
qqline(ad.b.cof, col='red')
par(mfrow=c(1,1))
```


Histogram of ad.b.cof
Frequency
0
40
−4
−3
−2
−1
0
1
2
ad.b.cof


density.default(x = ad.b.cof)
Density
0.0
0.3
0.6
−4
−2
0
2
N = 231 Bandwidth = 0.217


Normal Q−Q Plot
Sample Quantiles
−4
−1
2
−3
−2
−1
0
1
2
3
Theoretical Quantiles

Bibliography

Arumugam, M., Raes, J., Pelletier, E., Le Paslier, D., Yamada, T., Mende, D. R., Fernandes, G. R., Tap, J., Bruls, T., Batto, J.-M., et al. (2011). Enterotypes of the human gut microbiome. *nature*, 473(7346):174.

Chapleur, O., Madigou, C., Civade, R., Rodolphe, Y., Mazéas, L., and Bouchez, T. (2016). Increasing concentrations of phenol progressively affect anaerobic digestion of cellulose and associated microbial communities. *Biodegradation*, 27(1):15–27.

Kong, G., Lê Cao, K.-A., Judd, L. M., Li, S., Renoir, T., and Hannan, A. J. (2018). Microbiome profiling reveals gut dysbiosis in a transgenic mouse model of Huntington's disease. *Neurobiology of Disease*.

Lin, Y., Ghazanfar, S., Wang, K., Gagnon-Bartsch, J. A., Lo, K. K., Su, X., Han, Z.-G., Ormerod, J. T., Speed, T. P., Yang, P., et al. (2018). scMerge: Integration of multiple single-cell transcriptomics datasets leveraging stable expression and pseudo-replication. *BioRxiv*, page 393280.

Sacristán-Soriano, O., Banaigs, B., Casamayor, E. O., and Becerro, M. A. (2011). Exploring the links between natural products and bacterial assemblages in the sponge Aplysina aerophoba. *Applied and Environmental Microbiology*, 77(3):862–870.