BIOS 7659 Homework 6

Tim Vigers

07 November 2020

data("montgomery.subset")  
data("uCovar")

# 1. Differential Expression

## a) Calculate RPKM and perform a t test

RPKM stands for reads per kilobases per million reads and is calculated by

where the total mapped reads refers to column-wise sums.

# Divide by column sums (in millions)  
rpkm = as.data.frame(lapply(montgomery.subset, function(c){  
 c / (sum(c)/1e6)  
}))  
# Divide rows by gene length (in Kb)  
rpkm = as.matrix(rpkm) / c(uCovar$length / 1e3)  
rownames(rpkm) = rownames(montgomery.subset)

Do a standard t test between the groups for each gene (using RPKM):

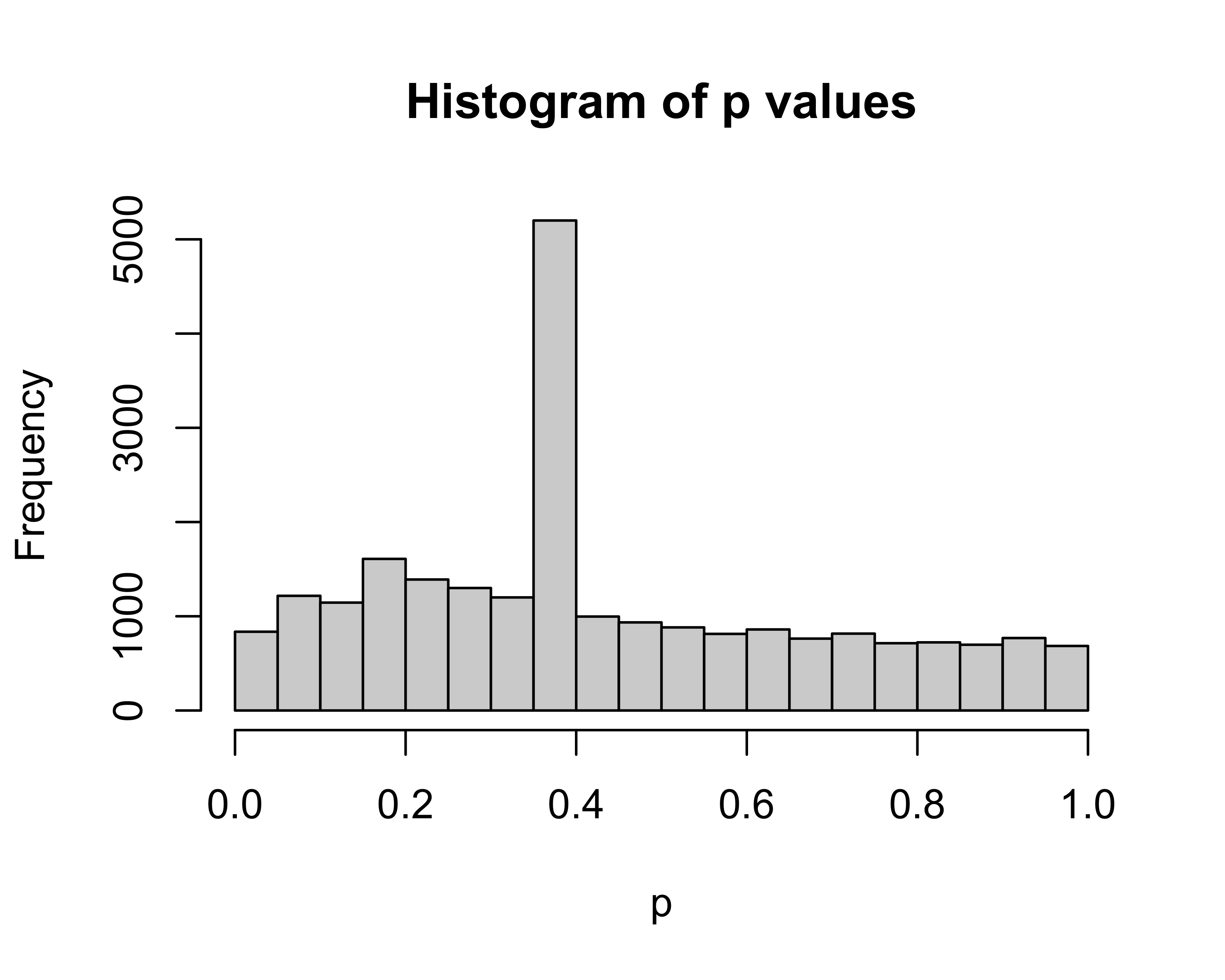
# T test for each row  
t\_tests = apply(rpkm,1,function(x){  
 group1 = as.numeric(x[1:5])  
 group2 = as.numeric(x[6:10])  
 t = t.test(group1,group2)  
 return(c(t$statistic,t$p.value))  
})  
# Format and print  
t\_tests = as.data.frame(t(t\_tests))  
t\_tests = t\_tests %>% rownames\_to\_column() %>%   
 set\_names(c("Gene","T","p")) %>% arrange(desc(abs(T)))   
  
t\_tests %>% head(10) %>% flextable(.) %>%   
 set\_caption("Top 10 Genes by t-statistic") %>% autofit(.)

Top 10 Genes by t-statistic

| Gene | T | p |
| --- | --- | --- |
| ENSG00000105723 | 9.240458 | 5.649871e-05 |
| ENSG00000131116 | 7.009775 | 1.284405e-04 |
| ENSG00000228109 | 6.760059 | 1.675717e-04 |
| ENSG00000231615 | -6.704669 | 3.753160e-04 |
| ENSG00000232818 | 6.633703 | 3.615860e-04 |
| ENSG00000162889 | 6.406242 | 2.333907e-04 |
| ENSG00000133103 | -6.148776 | 2.351005e-03 |
| ENSG00000134686 | 6.029173 | 4.295015e-04 |
| ENSG00000135677 | -5.995361 | 3.597591e-04 |
| ENSG00000175274 | 5.920333 | 1.726563e-03 |

## b) Plot the histogram of p-values

hist(t\_tests$p,main = "Histogram of p values",xlab = "p")



Normally we would expect a uniform distribution of p values (or hopefully one with a slight tail around 0.05, suggesting some sort of signal in the data), but this distribution has a large peak around 0.4. Looking at the genes that produce these p values, it seems that they all have 0 RPKM across most samples, and one sample with a low value. This results in an absolute value of the t statistic of 1, which corresponds to a p value of 0.374.

t1 = t\_tests$Gene[which(abs(t\_tests$T)==1)]  
t1 = as.data.frame(rpkm[t1,]) %>% rownames\_to\_column(var="Gene")  
set\_table\_properties(flextable(head(t1,10)),  
 width = .5, layout = "autofit")

| Gene | NA06985 | NA06994 | NA07037 | NA10847 | NA11920 | NA11918 | NA11931 | NA12003 | NA12006 | NA12287 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ENSG00000000971 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.00000000 | 0 | 0.03213093 |
| ENSG00000002079 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.04853275 | 0.00000000 | 0 | 0.00000000 |
| ENSG00000004809 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.05401735 | 0.00000000 | 0 | 0.00000000 |
| ENSG00000004846 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.04454858 | 0 | 0.00000000 |
| ENSG00000006071 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.00000000 | 0 | 0.05298604 |
| ENSG00000006468 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.02815965 | 0 | 0.00000000 |
| ENSG00000007062 | 0 | 0.1250505 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.00000000 | 0 | 0.00000000 |
| ENSG00000007908 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.19604966 | 0 | 0.00000000 |
| ENSG00000008438 | 0 | 0.0000000 | 0 | 0.5619036 | 0 | 0 | 0.00000000 | 0.00000000 | 0 | 0.00000000 |
| ENSG00000009950 | 0 | 0.0000000 | 0 | 0.0000000 | 0 | 0 | 0.00000000 | 0.22904355 | 0 | 0.00000000 |

## c) Filter genes by total counts

# Remove genes with low counts  
filtered = montgomery.subset[rowSums(montgomery.subset)>=10,]  
# Create edgeR object  
filtered\_dge = DGEList(filtered,group = rep(c(1,2),each=5))

There are a total of 15620 genes with at least 10 counts across all samples. By default, the DGEList() function uses column sums for the lib.size argument. This makes the most sense for this dataset, as the provided sizeFactors.subset data appears to be incorrect.

## d) Calculate TMM normalization factors

filtered\_dge = calcNormFactors(filtered\_dge)  
autofit(flextable(filtered\_dge$samples))

| group | lib.size | norm.factors |
| --- | --- | --- |
| 1 | 2817342 | 0.9420009 |
| 1 | 2168185 | 1.0077134 |
| 1 | 2782270 | 0.9975042 |
| 1 | 2551303 | 0.9790312 |
| 1 | 4259582 | 0.9025138 |
| 2 | 1461731 | 1.1474002 |
| 2 | 5013427 | 1.0833630 |
| 2 | 3623044 | 0.9307130 |
| 2 | 6332824 | 0.9784469 |
| 2 | 3786172 | 1.0558556 |

The effective library sizes are generally similar to the column sums because the normalization factors are fairly close to 1. A normalization factor > 1 increases the library size, which is similar to downscaling the counts (and vice versa for factors < 1). So, the effective library size for sample 6 is increased by about 115%. Conversely, the effective library size for sample 5 is decreased by approximately 90%, which suggests that there are a small number of high-count sequences that need to be adjusted for.

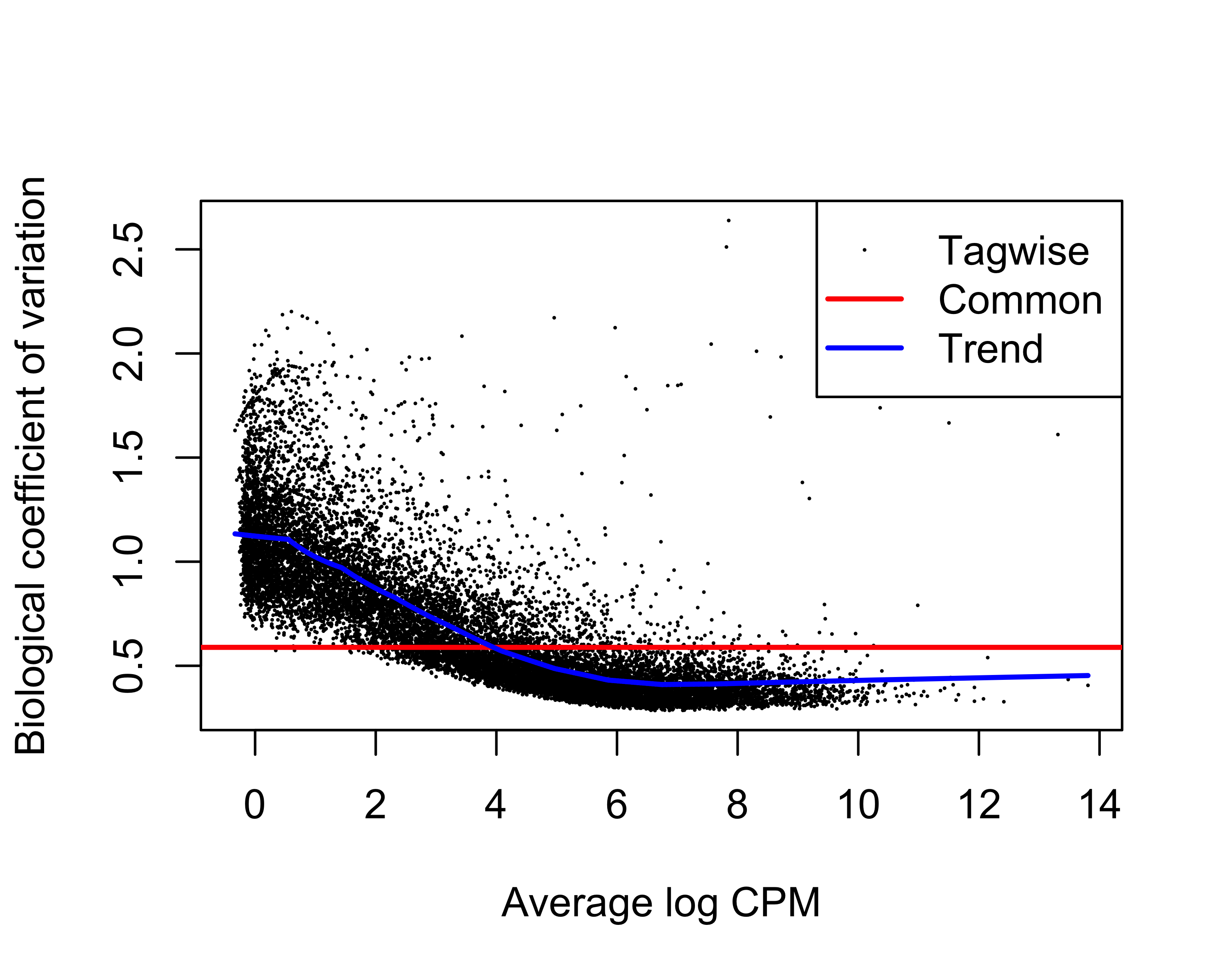
## e) Use the estimateDisp() function to calculate the common, trended and tagwise dispersions

filtered\_dge = estimateDisp(filtered\_dge)

## Design matrix not provided. Switch to the classic mode.

The common dispersion estimate is approximately 0.346. Plot the tagwise dispersion estimate for each gene vs. the average log counts per million:

plotBCV(filtered\_dge)



The common dispersion estimate only seems to work well for a narrow range of log CPM (around 4). It appears to underestimate dispersion for the lower count genes and overestimate for the higher average count genes. Note that the y axis of the above plot is the biological CV, which is the square root of the dispersion factor.

## f) Fit the negative binomial model

### Using the common dispersion estimate

et = exactTest(filtered\_dge,dispersion = "common")  
top\_common = topTags(et) %>% as.data.frame(.) %>%   
 rownames\_to\_column(.,var = "Gene")  
autofit(flextable(top\_common))

| Gene | logFC | logCPM | PValue | FDR |
| --- | --- | --- | --- | --- |
| ENSG00000211642 | -11.041227 | 6.496593 | 3.128081e-30 | 4.886063e-26 |
| ENSG00000211660 | -10.845016 | 6.306155 | 2.177168e-29 | 1.700368e-25 |
| ENSG00000211890 | -7.817087 | 10.362693 | 1.678486e-26 | 8.739317e-23 |
| ENSG00000211937 | -7.225483 | 6.153042 | 7.628122e-23 | 2.978782e-19 |
| ENSG00000211638 | 7.639730 | 5.092701 | 1.166973e-20 | 3.606106e-17 |
| ENSG00000243063 | -7.502970 | 5.968965 | 1.385188e-20 | 3.606106e-17 |
| ENSG00000211651 | 7.176541 | 3.774310 | 1.011370e-16 | 2.256801e-13 |
| ENSG00000238649 | 6.412053 | 4.139913 | 3.306839e-16 | 6.456603e-13 |
| ENSG00000239951 | 5.742261 | 4.411399 | 2.561470e-15 | 4.445573e-12 |
| ENSG00000253701 | 6.243808 | 3.734188 | 7.096301e-15 | 1.108442e-11 |

### Using the tag-wise dispersion estimates

et = exactTest(filtered\_dge,dispersion = "tagwise")  
top\_tagwise = topTags(et) %>% as.data.frame(.) %>%   
 rownames\_to\_column(.,var = "Gene")  
autofit(flextable(top\_tagwise))

| Gene | logFC | logCPM | PValue | FDR |
| --- | --- | --- | --- | --- |
| ENSG00000253701 | 6.218664 | 3.734188 | 1.600582e-07 | 0.002500109 |
| ENSG00000211892 | -6.115517 | 2.693831 | 1.922166e-06 | 0.014598105 |
| ENSG00000134184 | -6.491103 | 1.066652 | 3.295763e-06 | 0.014598105 |
| ENSG00000148411 | -2.392071 | 5.108220 | 3.738311e-06 | 0.014598105 |
| ENSG00000239223 | 3.267199 | 3.205279 | 1.076691e-05 | 0.033635814 |
| ENSG00000211642 | -11.007935 | 6.496593 | 2.720879e-05 | 0.070833559 |
| ENSG00000189337 | -4.037749 | 4.080578 | 7.910264e-05 | 0.176511884 |
| ENSG00000211660 | -10.806472 | 6.306155 | 9.288806e-05 | 0.181363929 |
| ENSG00000180611 | 2.036941 | 5.225731 | 1.132895e-04 | 0.183409674 |
| ENSG00000101210 | -4.465062 | 4.915172 | 1.174198e-04 | 0.183409674 |

There are only 3 overlapping genes in the top 10 table for the two methods. Also, the top genes as determined by the common dispersion estimate approach appear to be driven more by fold change than those that are most significant using the tagwise method, because the table is essentially in decreasing order of logFC (with a couple of minor exceptions). This is because the common dispersion factor fails to correctly account for dispersion, which means that the fold change drives the significance test. The incorrect variability estimate also results in much lower p values than when using tagwise dispersion, which suggests that common dispersion probably results in more false positives.

## g) Extract the raw counts

### For the top 10 genes based on the common dispersion

top\_common\_counts = filtered\_dge$counts[top\_common$Gene,] %>%  
 as.data.frame(.) %>% rownames\_to\_column(var = "Gene")  
set\_table\_properties(flextable(top\_common\_counts),  
 width = .5, layout = "autofit")

| Gene | NA06985 | NA06994 | NA07037 | NA10847 | NA11920 | NA11918 | NA11931 | NA12003 | NA12006 | NA12287 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ENSG00000211642 | 0 | 1831 | 0 | 155 | 1 | 0 | 0 | 0 | 0 | 1 |
| ENSG00000211660 | 0 | 0 | 0 | 11 | 2994 | 0 | 0 | 0 | 0 | 1 |
| ENSG00000211890 | 0 | 2 | 0 | 3216 | 45407 | 53 | 9 | 29 | 86 | 15 |
| ENSG00000211937 | 0 | 0 | 0 | 630 | 1727 | 0 | 0 | 0 | 23 | 0 |
| ENSG00000211638 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 82 | 1889 | 1 |
| ENSG00000243063 | 0 | 0 | 0 | 1 | 2356 | 9 | 0 | 0 | 0 | 0 |
| ENSG00000211651 | 0 | 0 | 0 | 1 | 1 | 1 | 682 | 0 | 0 | 2 |
| ENSG00000238649 | 1 | 0 | 2 | 2 | 0 | 294 | 0 | 0 | 0 | 1 |
| ENSG00000239951 | 0 | 0 | 0 | 7 | 3 | 0 | 0 | 212 | 850 | 0 |
| ENSG00000253701 | 0 | 0 | 0 | 4 | 0 | 77 | 8 | 134 | 33 | 138 |

### For the top 10 genes based on tagwise dispersion

top\_tagwise\_counts = filtered\_dge$counts[top\_tagwise$Gene,] %>%  
 as.data.frame(.) %>% rownames\_to\_column(var = "Gene")  
set\_table\_properties(flextable(top\_tagwise\_counts),  
 width = .5, layout = "autofit")

| Gene | NA06985 | NA06994 | NA07037 | NA10847 | NA11920 | NA11918 | NA11931 | NA12003 | NA12006 | NA12287 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ENSG00000253701 | 0 | 0 | 0 | 4 | 0 | 77 | 8 | 134 | 33 | 138 |
| ENSG00000211892 | 0 | 19 | 84 | 14 | 55 | 0 | 0 | 0 | 2 | 1 |
| ENSG00000134184 | 13 | 9 | 3 | 12 | 5 | 0 | 0 | 0 | 0 | 0 |
| ENSG00000148411 | 232 | 96 | 260 | 102 | 75 | 31 | 28 | 23 | 100 | 33 |
| ENSG00000239223 | 7 | 4 | 4 | 5 | 1 | 59 | 52 | 42 | 114 | 15 |
| ENSG00000211642 | 0 | 1831 | 0 | 155 | 1 | 0 | 0 | 0 | 0 | 1 |
| ENSG00000189337 | 248 | 108 | 23 | 9 | 4 | 2 | 18 | 11 | 3 | 4 |
| ENSG00000211660 | 0 | 0 | 0 | 11 | 2994 | 0 | 0 | 0 | 0 | 1 |
| ENSG00000180611 | 6 | 16 | 76 | 48 | 60 | 88 | 350 | 131 | 277 | 381 |
| ENSG00000101210 | 22 | 11 | 21 | 46 | 933 | 1 | 29 | 5 | 0 | 20 |

The genes that are significant when using the common dispersion tend to have low counts across most samples with a few samples with high expression levels. On the other hand, the top genes based on tagwise dispersion tend to have counts more evenly spread across the samples.

### Get Ensembl information for the top genes

#### Based on common dispersion

ensembl = useEnsembl(biomart="ensembl", dataset="hsapiens\_gene\_ensembl")  
en = getBM(attributes=c('ensembl\_gene\_id','description'),   
 filters ='ensembl\_gene\_id', values = top\_common\_counts$Gene,   
 mart = ensembl)  
set\_table\_properties(flextable(en),  
 width = .75, layout = "autofit")

| ensembl\_gene\_id | description |
| --- | --- |
| ENSG00000211638 | immunoglobulin lambda variable 8-61 [Source:HGNC Symbol;Acc:HGNC:5931] |
| ENSG00000211642 | immunoglobulin lambda variable 10-54 [Source:HGNC Symbol;Acc:HGNC:5884] |
| ENSG00000211651 | immunoglobulin lambda variable 1-44 [Source:HGNC Symbol;Acc:HGNC:5879] |
| ENSG00000211660 | immunoglobulin lambda variable 2-23 [Source:HGNC Symbol;Acc:HGNC:5890] |
| ENSG00000211890 | immunoglobulin heavy constant alpha 2 (A2m marker) [Source:HGNC Symbol;Acc:HGNC:5479] |
| ENSG00000211937 | immunoglobulin heavy variable 2-5 [Source:HGNC Symbol;Acc:HGNC:5576] |
| ENSG00000238649 | small nucleolar RNA, C/D box 42A [Source:HGNC Symbol;Acc:HGNC:10180] |
| ENSG00000239951 | immunoglobulin kappa variable 3-20 [Source:HGNC Symbol;Acc:HGNC:5817] |
| ENSG00000243063 | immunoglobulin kappa variable 3-7 (non-functional) [Source:HGNC Symbol;Acc:HGNC:5821] |

#### Based on tagwise dispersion

en = getBM(attributes=c('ensembl\_gene\_id','description'),   
 filters ='ensembl\_gene\_id', values = top\_tagwise\_counts$Gene,   
 mart = ensembl)  
set\_table\_properties(flextable(en),  
 width = .75, layout = "autofit")

| ensembl\_gene\_id | description |
| --- | --- |
| ENSG00000101210 | eukaryotic translation elongation factor 1 alpha 2 [Source:HGNC Symbol;Acc:HGNC:3192] |
| ENSG00000134184 | glutathione S-transferase mu 1 [Source:HGNC Symbol;Acc:HGNC:4632] |
| ENSG00000148411 | NACC family member 2 [Source:HGNC Symbol;Acc:HGNC:23846] |
| ENSG00000180611 | Mab-21 domain containing 2 [Source:HGNC Symbol;Acc:HGNC:30438] |
| ENSG00000189337 | kazrin, periplakin interacting protein [Source:HGNC Symbol;Acc:HGNC:29173] |
| ENSG00000211642 | immunoglobulin lambda variable 10-54 [Source:HGNC Symbol;Acc:HGNC:5884] |
| ENSG00000211660 | immunoglobulin lambda variable 2-23 [Source:HGNC Symbol;Acc:HGNC:5890] |
| ENSG00000211892 | immunoglobulin heavy constant gamma 4 (G4m marker) [Source:HGNC Symbol;Acc:HGNC:5528] |
| ENSG00000239223 | ribosomal protein L34 pseudogene 31 [Source:HGNC Symbol;Acc:HGNC:36899] |

Almost all of the top ten genes based on common dispersion code for some sort of immunoglobulin. A few of the top tagwise genes are also immunoglobulins, but there appears to be a greater variety of function among the top tagwise genes.

# 2. Remove Unwanted Variation.

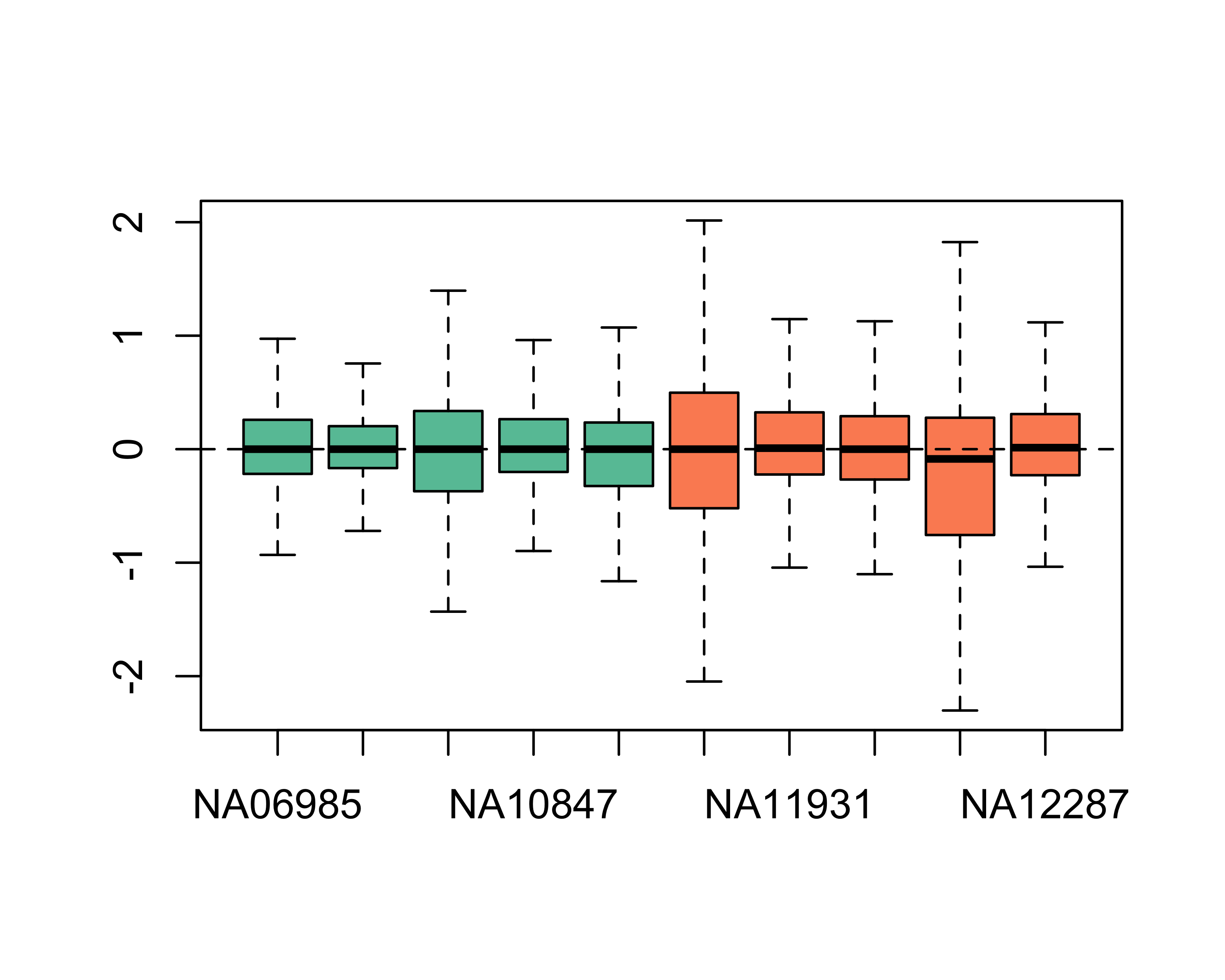
## a) Create a design matrix

# Fit the GLM   
group = rep(c(1,2),each=5)  
design = model.matrix(~group)  
filtered\_dge = DGEList(filtered,group = group)  
filtered\_dge = calcNormFactors(filtered\_dge,"upperquartile")  
filtered\_dge = estimateDisp(filtered\_dge,design = design)  
glm\_fit\_a = glmFit(filtered\_dge,  
 dispersion = filtered\_dge$tagwise.dispersion)  
glm\_lrt\_a = glmLRT(glm\_fit\_a)$table  
glm\_lrt\_a$PValue\_FDR = p.adjust(glm\_lrt\_a$PValue,"fdr")

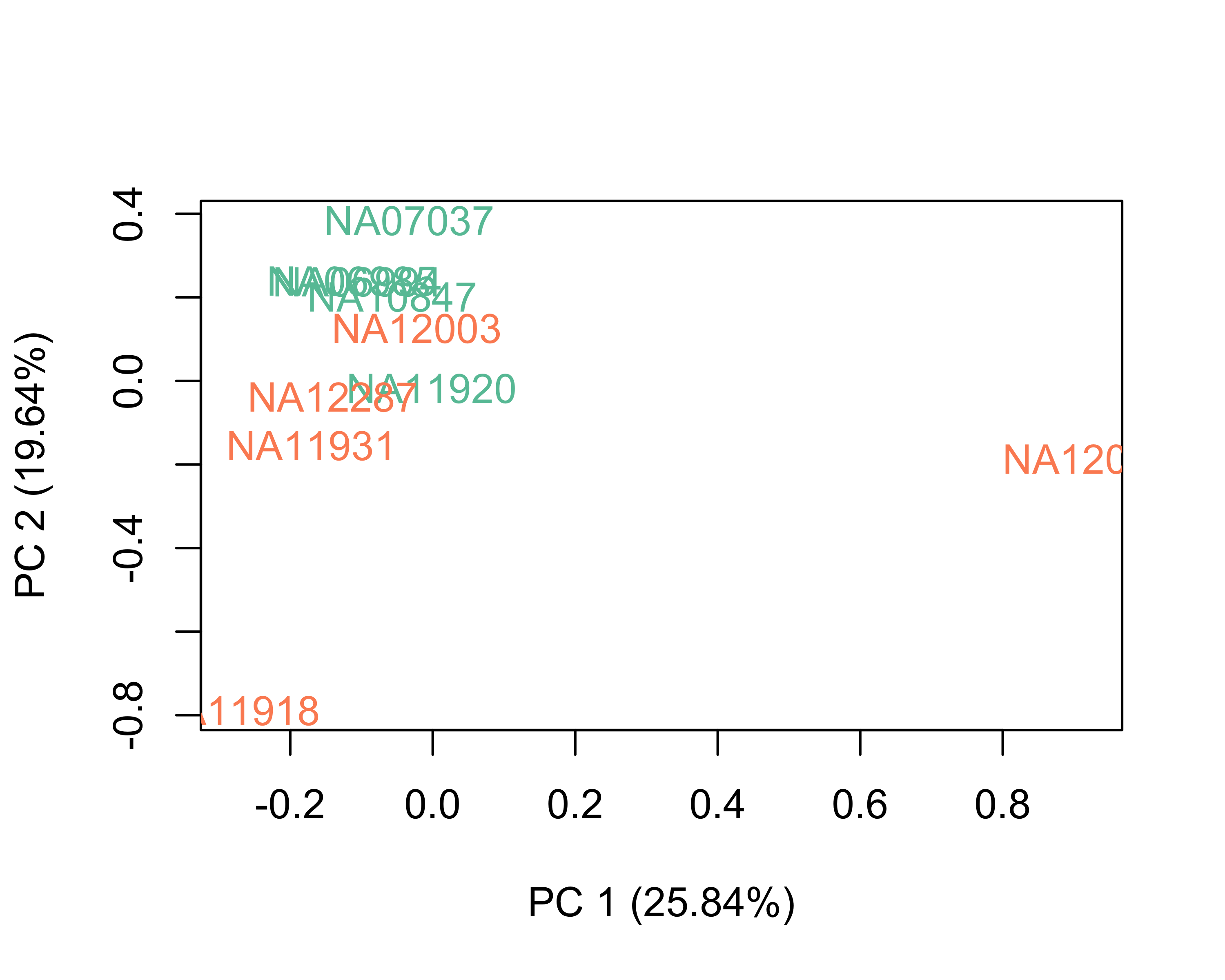
Using the likelihood ratio test, there are 6 genes with FDR-adjusted p values < 0.05.

## b) Fit edgeR models that adjust for unwanted variation

colors = brewer.pal(3, "Set2")  
set = newSeqExpressionSet(  
 as.matrix(filtered),  
 phenoData = data.frame(group,row.names=colnames(filtered)))  
set = betweenLaneNormalization(set, which="upper")  
plotRLE(set, outline=FALSE,col=colors[group])



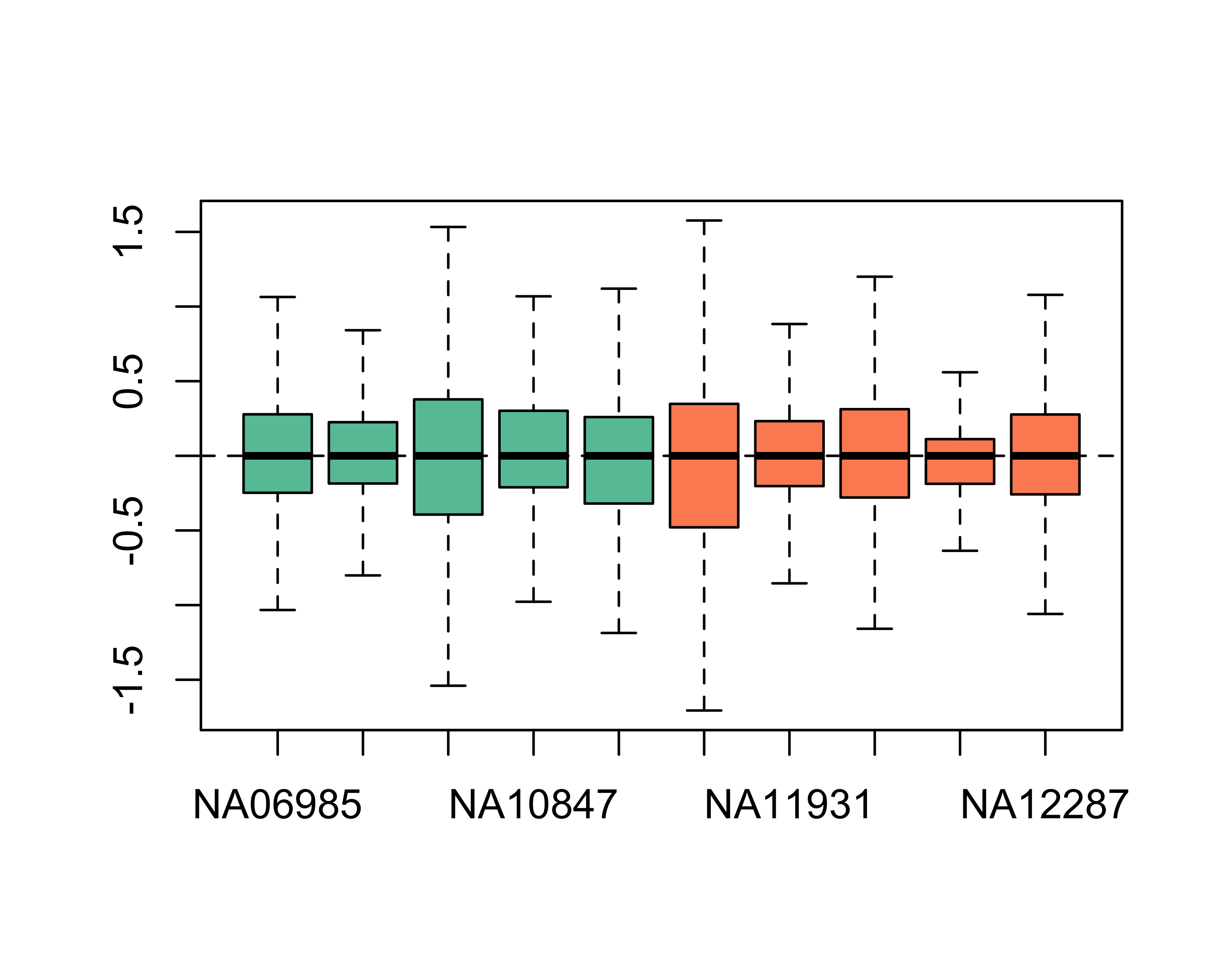
plotPCA(set,col=colors[group])



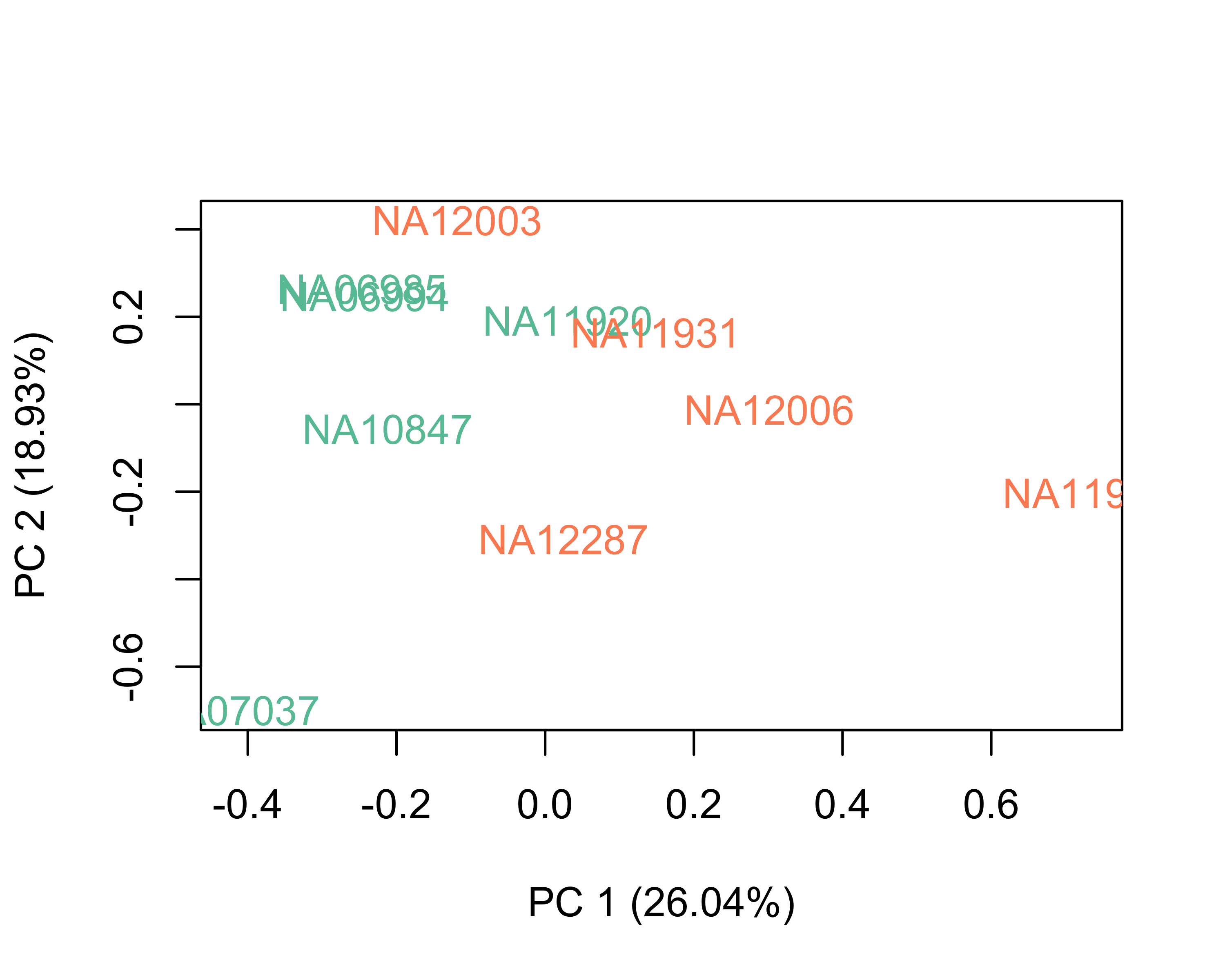
Two of the samples in group 2 (NA11918 and NA12006) appear to be very different from the rest of the samples. In the boxplot these samples have a much wider range (more variability) than the others. Also, in the PCA plot there is one cluster with samples from both groups, but the two outliers from group 2 are clearly different from everything else. The first PC appears to be driven by the difference between NA12006 and the other samples, whereas PC2 seems to be driven by NA11918. Between-lane normalization alone does not appear to be sufficient for these data.

## c) Perform RUVg using negative empirical control genes

# take the 10,000 genes with the largest likelihood ratio test   
# p-values from part a)  
neg\_controls =   
 rownames(head(glm\_lrt\_a[order(glm\_lrt\_a$PValue,  
 decreasing = T),],10000))  
set1 = RUVg(set,neg\_controls,k=1)  
plotRLE(set1, outline=FALSE,col=colors[group])



plotPCA(set1,col=colors[group])



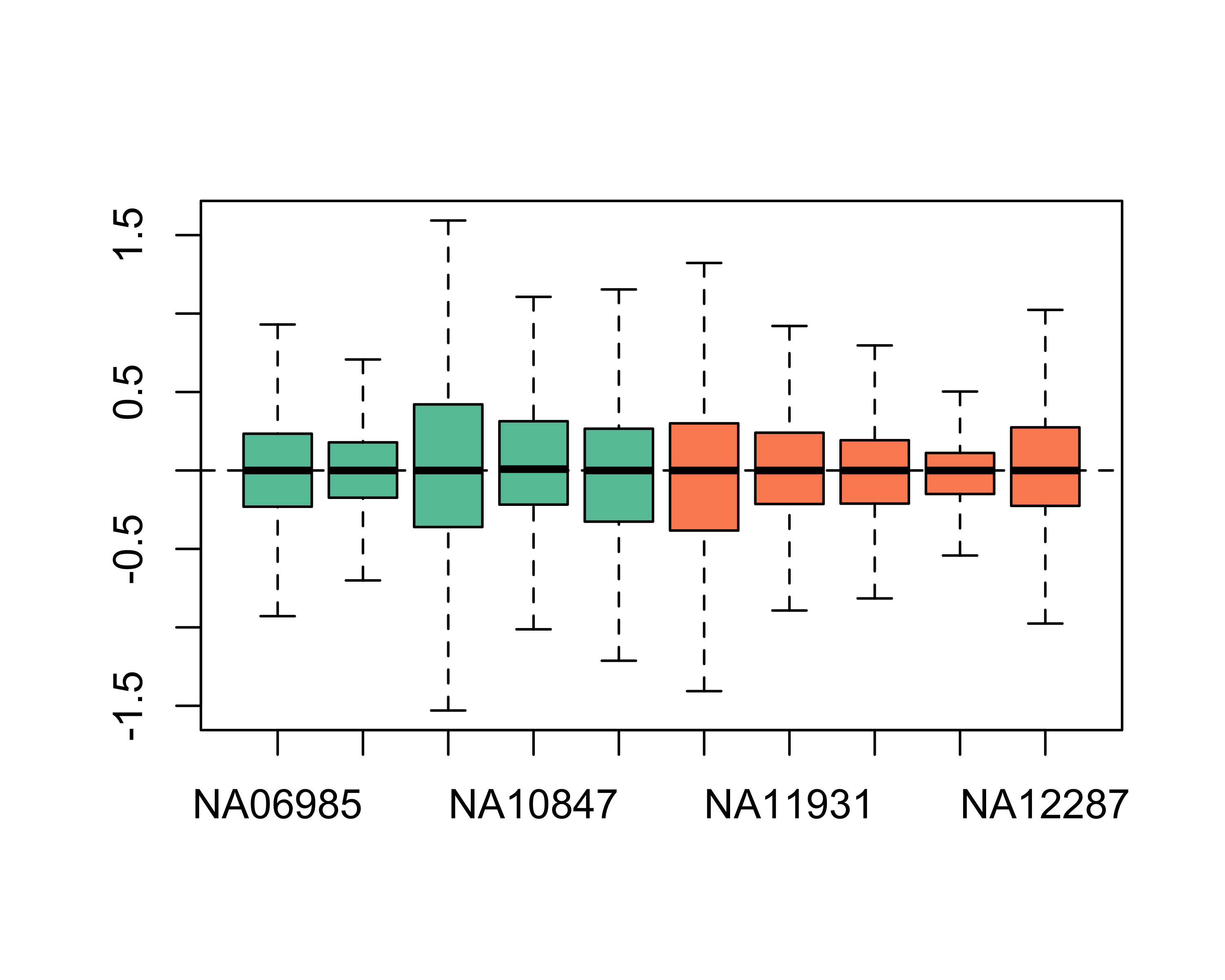
These plots are definitely an improvement on the previous ones. There still appear to be two samples that are different from the others (NA07037 and NA11918), but in both the boxplot and PCA plot all of the samples are closer to one another than before RUVg.

design = model.matrix(~group + W\_1, data=pData(set1))  
y = DGEList(counts=counts(set1),group = group)  
y = calcNormFactors(y, method="upperquartile")  
y = estimateDisp(y, design)  
glm\_fit\_c = glmFit(y, design,dispersion = y$tagwise.dispersion)  
glm\_lrt\_c = glmLRT(glm\_fit\_c, coef=2)$table  
glm\_lrt\_c$PValue\_FDR = p.adjust(glm\_lrt\_c$PValue,"fdr")

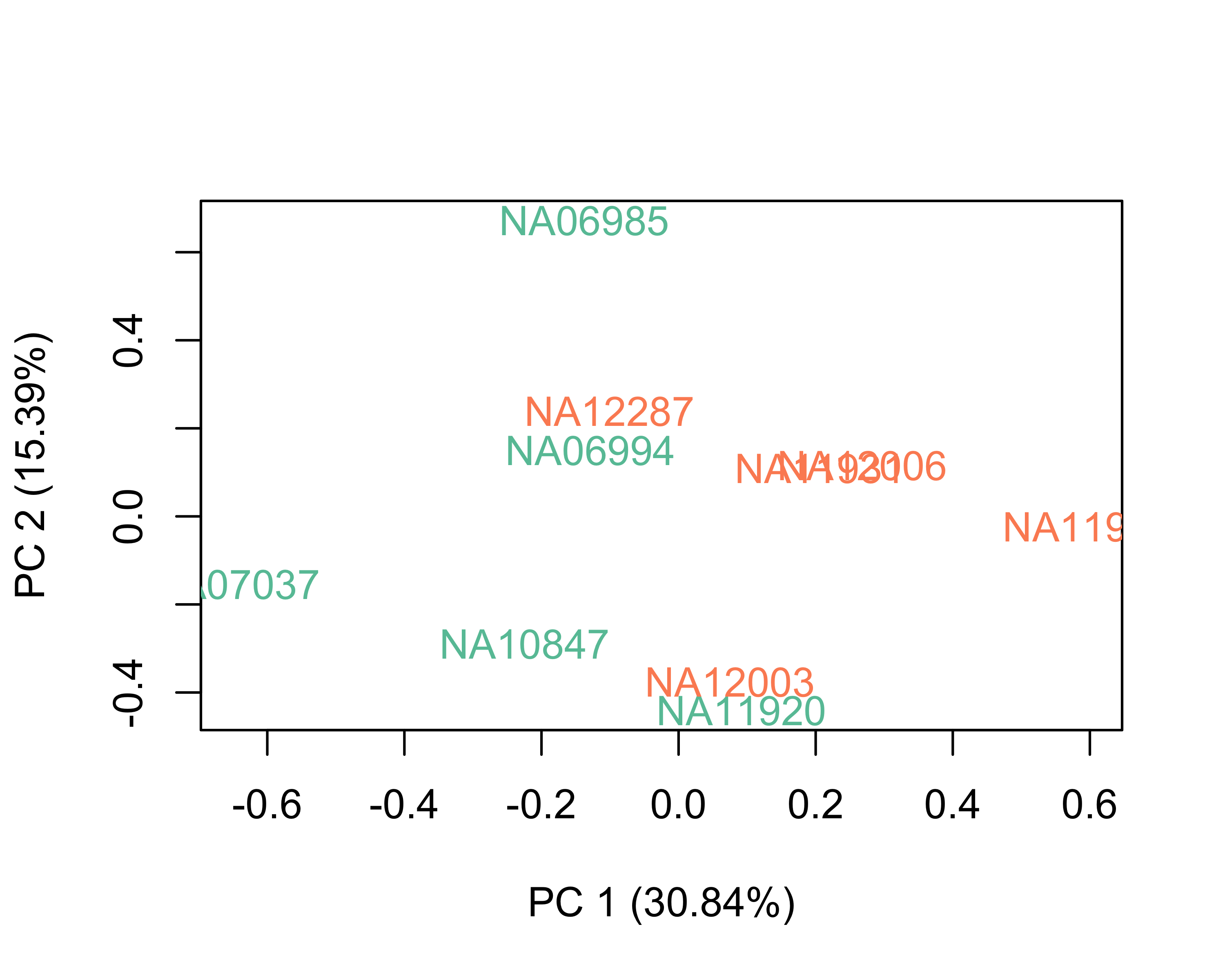
After controlling for unwanted variation, there are 27 genes with FDR adjusted p values < 0.05.

## d) Repeat part c) using k=2

set2 = RUVg(set,neg\_controls,k=2)  
plotRLE(set2, outline=FALSE,col=colors[group])



plotPCA(set2,col=colors[group])



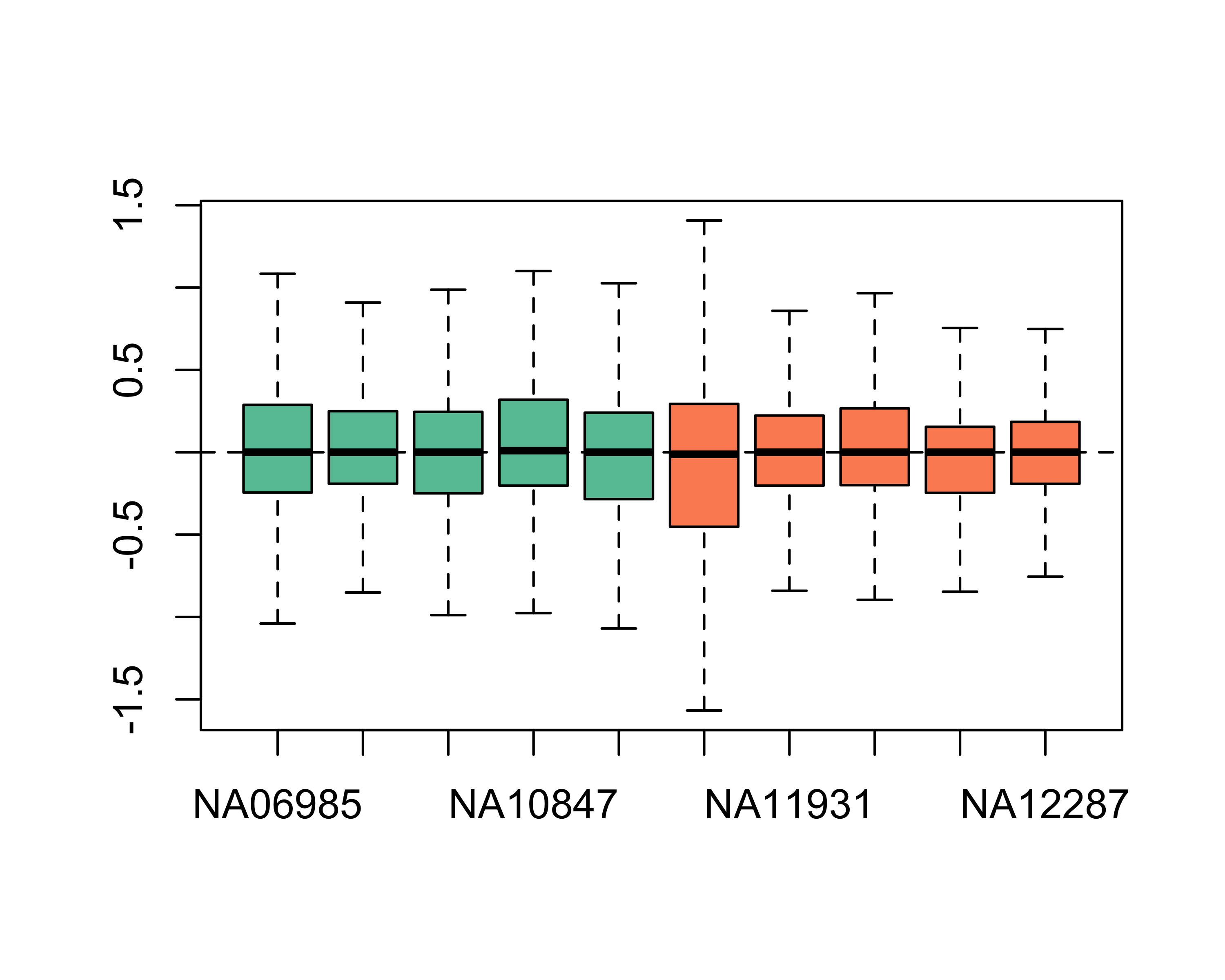
Increasing the number of factors of unwanted variation appears to improve these plots somewhat, although now sample NA07037 looks like a bit of an outlier (particularly in the boxplot). Overall though, the distribution of the samples in the boxplot appears to be more similar than in previous steps, and separation in the PCA plot doesn’t appear to be driven as much by single samples.

design = model.matrix(~group + W\_1 + W\_2, data=pData(set2))  
y = DGEList(counts=counts(set2),group = group)  
y = calcNormFactors(y, method="upperquartile")  
y = estimateDisp(y, design)  
glm\_fit\_d = glmFit(y, design,dispersion = y$tagwise.dispersion)  
glm\_lrt\_d = glmLRT(glm\_fit\_d, coef=2)$table  
glm\_lrt\_d$PValue\_FDR = p.adjust(glm\_lrt\_d$PValue,"fdr")

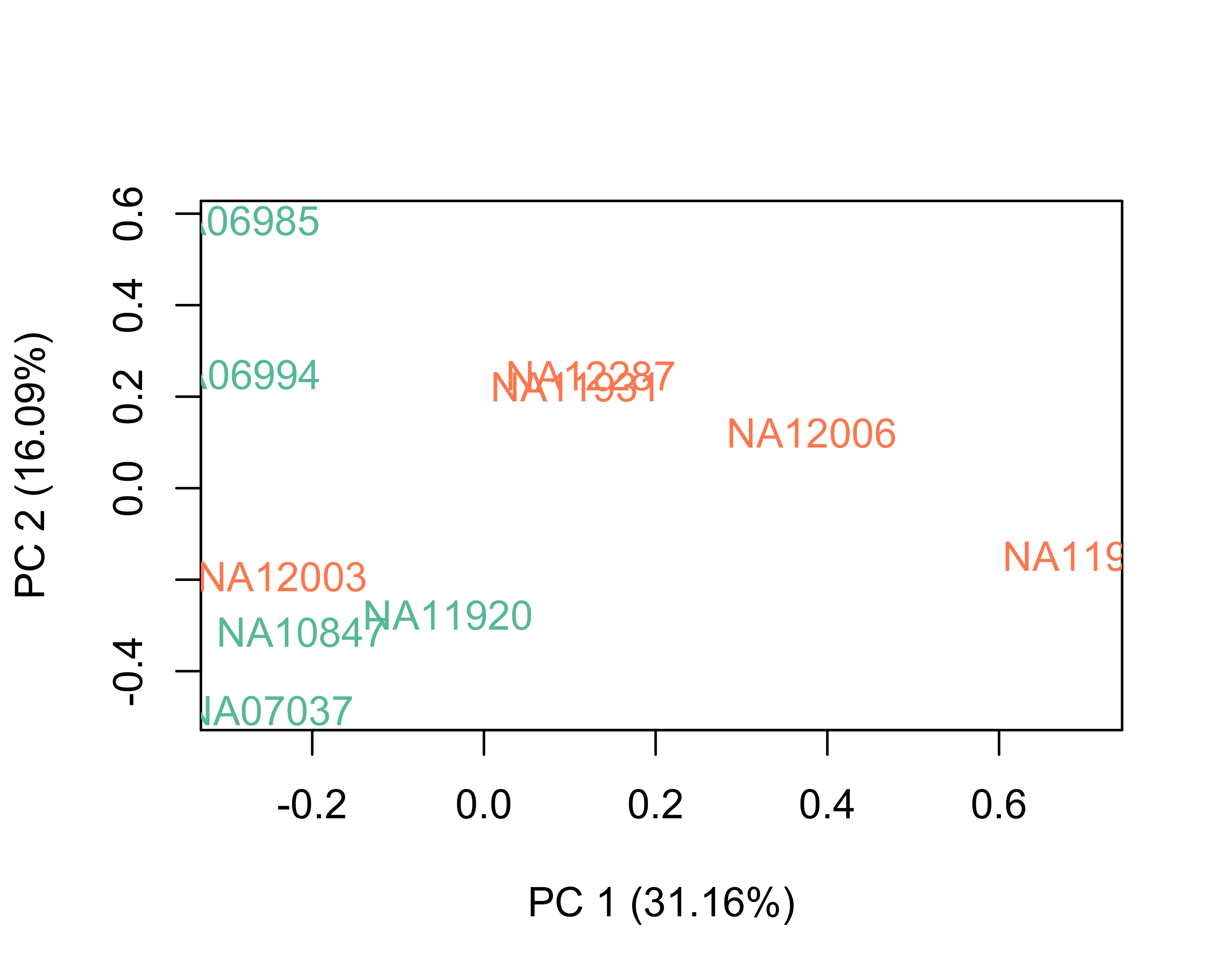
After controlling for two factors of unwanted variation, there are 44 genes with FDR adjusted p values < 0.05.

## e) Repeat part (d) using the RUVr method with k=2

# GLM residuals  
res = residuals(glm\_fit\_a, type="deviance")  
# RUVr  
set3 = RUVr(set,neg\_controls,k=2,res)  
plotRLE(set3,outline=FALSE,col=colors[group])



plotPCA(set3,col=colors[group])



Overall these plots look similar to using RUVg with k = 2, although it appears that NA11918 is still different from the other samples. NA11918 still seems to driving PC1, although this PCA plot is still a significant improvement over the original plot without any RUV methods applied.

design = model.matrix(~group + W\_1 + W\_2, data=pData(set2))  
y = DGEList(counts=counts(set2),group = group)  
y = calcNormFactors(y, method="upperquartile")  
y = estimateDisp(y, design)  
glm\_fit\_d = glmFit(y, design,dispersion = y$tagwise.dispersion)  
glm\_lrt\_d = glmLRT(glm\_fit\_d, coef=2)$table  
glm\_lrt\_d$PValue\_FDR = p.adjust(glm\_lrt\_d$PValue,"fdr")  
  
  
design = model.matrix(~group + W\_1 + W\_2, data=pData(set3))  
y = DGEList(counts=counts(set3),group = group)  
y = calcNormFactors(y, method="upperquartile")  
y = estimateDisp(y, design)  
glm\_fit\_e = glmFit(y, design, dispersion = y$tagwise.dispersion)  
glm\_lrt\_e = glmLRT(glm\_fit\_e, coef=2)$table  
glm\_lrt\_e$PValue\_FDR = p.adjust(glm\_lrt\_e$PValue,"fdr")

After controlling for two factors of unwanted variation using RUVr, there are 72 genes with FDR adjusted p values < 0.05.

## f) Concerns

Based on the diagnostic plots alone, there doesn’t appear to be anything wrong with using RUV on this dataset. However, one of the important assumptions of RUVg is that there are “control” genes that are not differentially expressed. Here we chose 10,000 genes with large p values from a standard likelihood ratio test, which seems like a somewhat ad-hoc approach. It would be better if we could use prior knowledge or spike-ins to determine control genes.

The assumptions for RUVr are less strict, so I would be more inclined to move forward with that method. RUVr can use information from all the genes for normalization (although above I decided to use the same negative control genes as for RUVg), but assumes that the unwanted factors are unrelated to variables of interest. This seems like a reasonable assumption for this dataset, although it’s difficult to say for sure.

The major concern I have with RUV in general is that it’s easy to accidentally remove wanted variation. This isn’t unique to RUV, and I think it can be avoided by carefully checking the diagnostic plots and keeping the k parameter low, but it’s a concern nonetheless.

# 3. Method Comparisons

Load package and data:

library(DESeq2)  
load(url("http://bowtie-bio.sourceforge.net/recount/ExpressionSets/bottomly\_eset.RData"))  
bottomly.count.table = exprs(bottomly.eset)

## a) Create a new data frame with genes that have at least 10 counts

# Filter  
filtered = bottomly.count.table[rowSums(bottomly.count.table)>=10,]  
# DESeq2 object  
pheno = factor(gsub("/","\_",phenoData(bottomly.eset)$strain))  
filtered\_dseq = DESeqDataSetFromMatrix(filtered, DataFrame(pheno),~pheno)  
# edgeR object  
filtered\_dge = DGEList(filtered,group = pheno)

There are 11870 genes with at least 10 counts across all samples.

## b) Calculate the DESeq2 size factors

filtered\_dseq = estimateSizeFactors(filtered\_dseq)  
filtered\_dge = calcNormFactors(filtered\_dge,method = "TMM")  
# Compare  
sizes = data.frame(sizeFactors(filtered\_dseq))  
sizes = cbind(sizes,filtered\_dge$samples$norm.factors) %>%  
 rownames\_to\_column()   
colnames(sizes) = c("Sample","DESeq Size Factor","TMM Norm Factor")  
autofit(flextable(sizes))

| Sample | DESeq Size Factor | TMM Norm Factor |
| --- | --- | --- |
| SRX033480 | 0.6439291 | 0.9858019 |
| SRX033488 | 1.3453539 | 0.9789546 |
| SRX033481 | 0.5784839 | 1.0094511 |
| SRX033489 | 1.4295303 | 0.9965089 |
| SRX033482 | 0.6355123 | 0.9880781 |
| SRX033490 | 1.5239501 | 0.9840943 |
| SRX033483 | 0.7933382 | 0.9942930 |
| SRX033476 | 1.1271894 | 0.9971411 |
| SRX033478 | 1.0772279 | 1.0023364 |
| SRX033479 | 0.8984474 | 0.9963740 |
| SRX033472 | 0.8886335 | 1.0193662 |
| SRX033473 | 1.0255149 | 0.9924446 |
| SRX033474 | 0.7987292 | 0.9975642 |
| SRX033475 | 0.7795619 | 1.0124273 |
| SRX033491 | 1.6161933 | 1.0133338 |
| SRX033484 | 0.9881892 | 1.0370408 |
| SRX033492 | 1.5720164 | 1.0070477 |
| SRX033485 | 0.7557824 | 1.0135210 |
| SRX033493 | 1.5922159 | 0.9960492 |
| SRX033486 | 0.8264069 | 0.9831050 |
| SRX033494 | 1.4715139 | 0.9969959 |

Size factors as calculated by DESeq2 are the median of the ratios of each sample over a psuedosample (the same as the RLE method in edgeR). The pseudosample is the geometric mean for each gene across all samples. The formula for the size factor of sample (with indexing gene) is:

The TMM approach estimates relative RNA expression using the trimmed mean of the M values, where the M value of gene g in samples k and k’ is:

And indicates the observed count.

The general idea of size factors is to make samples which may have been sequenced at different depths more comparable. For this dataset, the size factors estimated by TMM tend to be close to 1, whereas those estimated by DESeq2 have a much larger range and are more variable.

## c) Calculate the DESeq2 dispersions

Histograms of log dispersion:

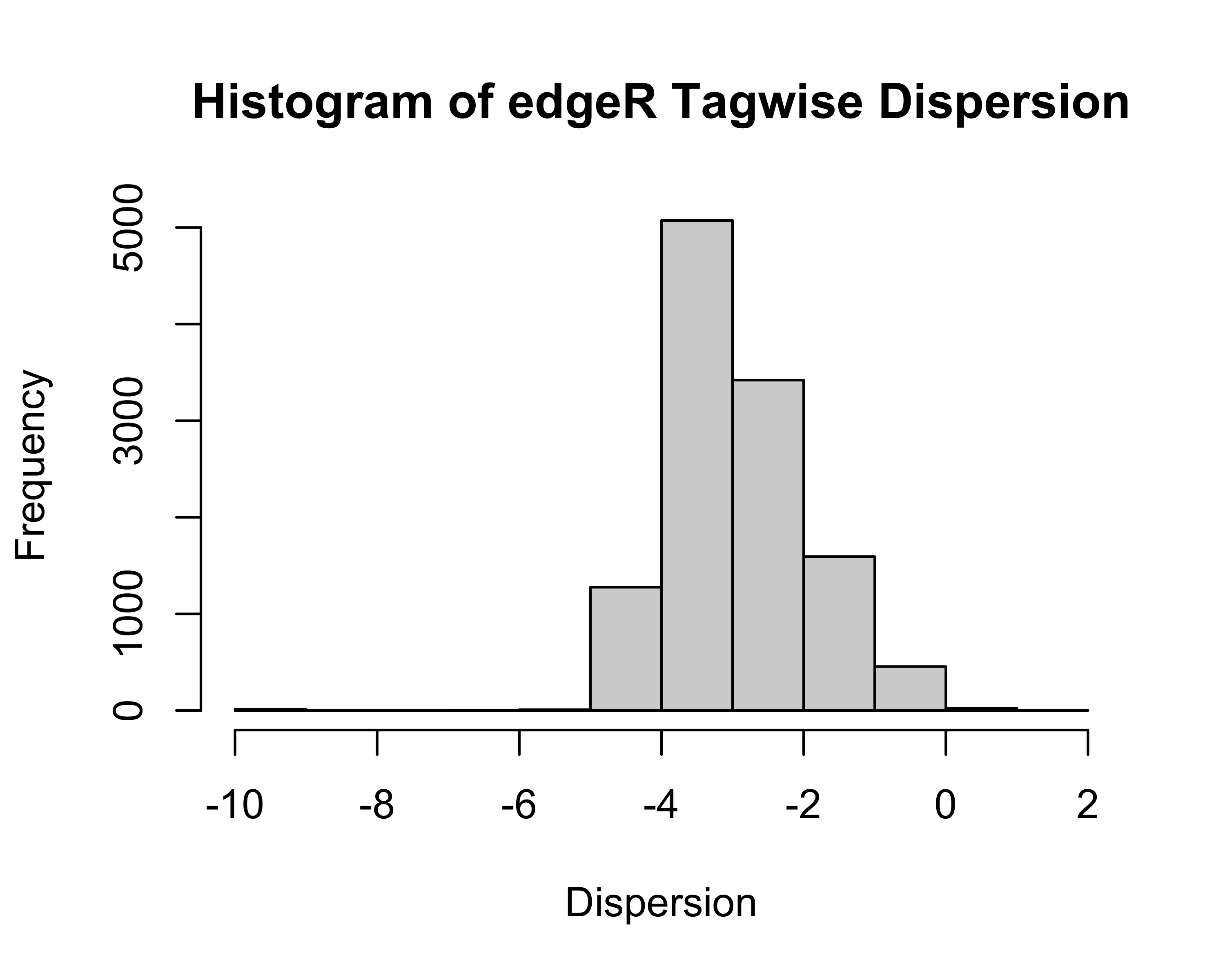
filtered\_dseq = estimateDispersions(filtered\_dseq,fitType = "local")  
hist(log(dispersions(filtered\_dseq)),xlab = "Size Factor",  
 main = "Histogram of DESeq2 Size Factors")



filtered\_dge = estimateDisp(filtered\_dge)

## Design matrix not provided. Switch to the classic mode.

hist(log(filtered\_dge$tagwise.dispersion),xlab = "Dispersion",  
 main = "Histogram of edgeR Tagwise Dispersion")



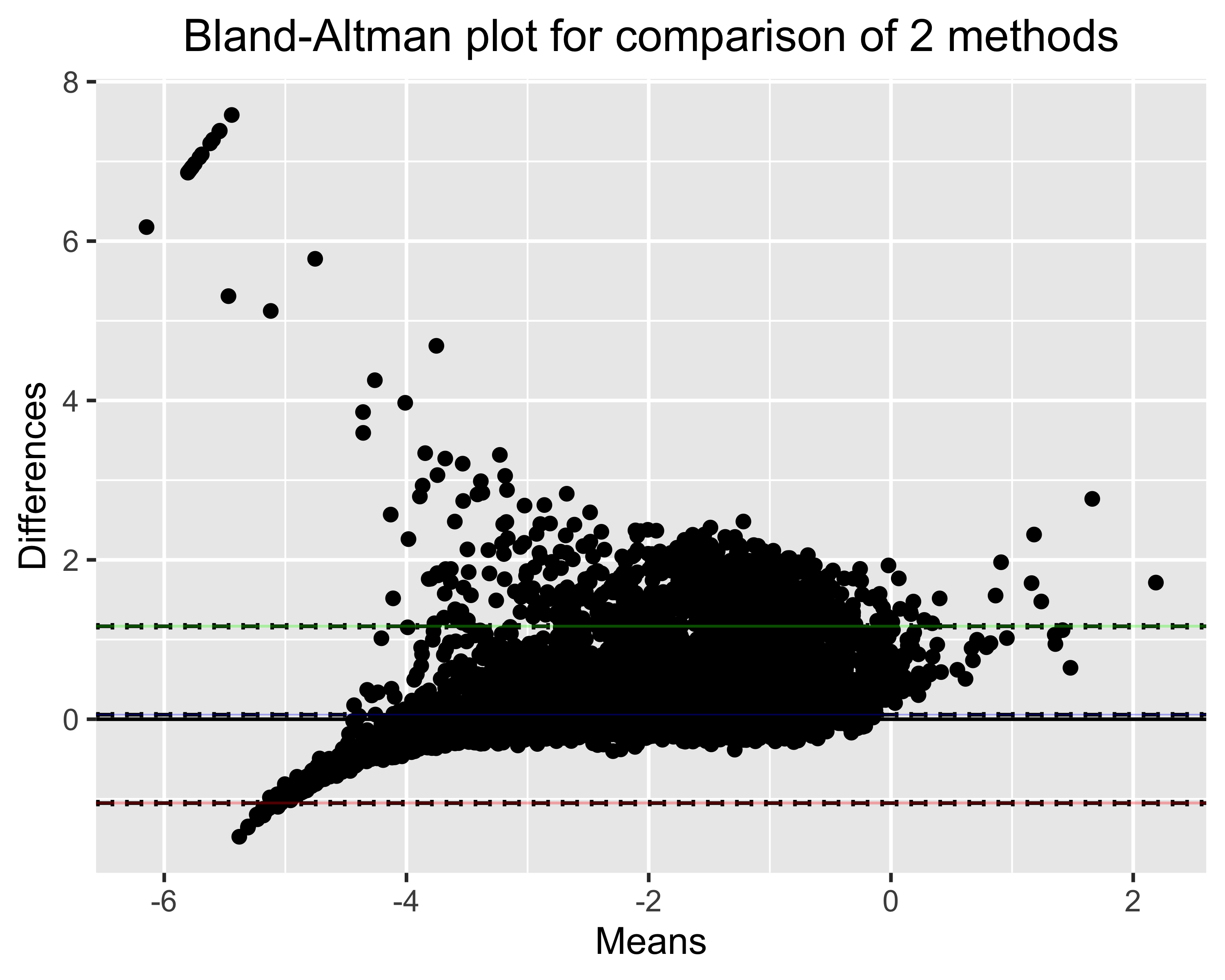
I found it much easier to look at the histograms of log-transformed dispersion. Based on these plots it appears that the mean dispersion estimate is similar between the two packages. However, there the DESeq estimates are distributed across a wider of range than the edgeR values.

Bland-Altman plot:

blandr.draw(log(dispersions(filtered\_dseq)),  
 log(filtered\_dge$tagwise.dispersion))

## Warning: Use of `plot.data$x.axis` is discouraged. Use `x.axis` instead.

## Warning: Use of `plot.data$y.axis` is discouraged. Use `y.axis` instead.



The Bland-Altman (BA) plot suggests that there is not good agreement between the two methods. Here the green and red lines represent 1.96 SD of the differences, so we would want at least 95% of the points to lie in this range in order to be confident in the methods’ agreement. The disagreement is particularly striking when the mean log dispersion is low, where some of the DESeq estimates are much larger than edgeR (the cluster in the top left of the plot), and some are a little bit smaller (the tail towards the bottom left). When the mean is in the -4 to 0 range there’s less of a linear trend in the plot, but many of the points are above the green line, which indicates that DESeq’s estimates are generally higher than edgeR’s.

## d) Test for differences between the two strains

filtered\_dseq = nbinomWaldTest(filtered\_dseq)  
res = as.data.frame(results(filtered\_dseq)) %>% rownames\_to\_column(var = "Gene")  
dseq\_sig = res$Gene[which(res$padj < 0.05)]  
  
design = model.matrix(~pheno)  
glm\_fit = glmFit(filtered\_dge,design,  
 dispersion = filtered\_dge$tagwise.dispersion)  
glm\_lrt = as.data.frame(glmLRT(glm\_fit)$table)  
glm\_lrt$PValue\_BH = p.adjust(glm\_lrt$PValue,"BH")  
glm\_lrt = glm\_lrt %>% rownames\_to\_column(var = "Gene")  
edger\_sig = glm\_lrt$Gene[which(glm\_lrt$PValue\_BH < 0.05)]

The DESeq2 method finds 1404 genes with p values < 0.05 after Benjamini-Hochberg (BH) correction, and edgeR finds 1259, with an overlap of 1207 genes. Now look at the results for a few genes that were significant in the DESEq results but not significant in edgeR:

diff = setdiff(dseq\_sig,edger\_sig)  
dseq\_diff = intersect(diff,dseq\_sig)  
head(res[res$Gene %in% dseq\_diff,]) %>% flextable(.) %>%   
 set\_caption(.,"DESeq Results") %>%   
 set\_table\_properties(width = .75, layout = "autofit")

DESeq Results

| Gene | baseMean | log2FoldChange | lfcSE | stat | pvalue | padj |
| --- | --- | --- | --- | --- | --- | --- |
| ENSMUSG00000000131 | 1534.076286 | 0.1758871 | 0.04823412 | 3.646528 | 0.0002658075 | 0.004429877 |
| ENSMUSG00000001802 | 9.968885 | 0.8499347 | 0.29258972 | 2.904869 | 0.0036740732 | 0.035383756 |
| ENSMUSG00000002190 | 116.831113 | -0.3333402 | 0.10911258 | -3.055012 | 0.0022505173 | 0.024477212 |
| ENSMUSG00000004032 | 1038.555866 | -0.3081941 | 0.10646393 | -2.894821 | 0.0037937428 | 0.036070792 |
| ENSMUSG00000005125 | 1315.024564 | -0.2693855 | 0.09382087 | -2.871275 | 0.0040882023 | 0.037933096 |
| ENSMUSG00000006095 | 224.998463 | -0.3056265 | 0.10773393 | -2.836864 | 0.0045558976 | 0.040800212 |

head(glm\_lrt[glm\_lrt$Gene %in% dseq\_diff,]) %>% flextable(.) %>%   
 set\_caption(.,"edgeR Results") %>% set\_table\_properties(layout = "autofit")

edgeR Results

| Gene | logFC | logCPM | LR | PValue | PValue\_BH |
| --- | --- | --- | --- | --- | --- |
| ENSMUSG00000000131 | 0.1859815 | 8.379640 | 6.943910 | 0.008410506 | 0.06899288 |
| ENSMUSG00000001802 | 0.8553064 | 1.364926 | 7.585023 | 0.005885522 | 0.05341067 |
| ENSMUSG00000002190 | -0.3253090 | 4.679250 | 7.702393 | 0.005514765 | 0.05112415 |
| ENSMUSG00000004032 | -0.2990005 | 7.818430 | 7.110246 | 0.007664460 | 0.06429480 |
| ENSMUSG00000005125 | -0.2612030 | 8.157872 | 6.593210 | 0.010236843 | 0.07804196 |
| ENSMUSG00000006095 | -0.2974941 | 5.625524 | 6.547396 | 0.010503777 | 0.07962527 |

The log fold change results are really close between the two packages, which suggests that the dispersion estimates are driving the differences we see in the significance testing. Based on this it’s difficult to say which package is better. edgeR may be slightly more conservative, so it’s likely a better option if you’re concerned about type 1 error. However, DESeq produces more significant hits, which can be good for exploratory research which aims to discover candidate genes for future studies. These differences are fairly small though, so I think the two approaches are pretty comparable overall, despite the concerning BA plot. Also, to me neither package has an advantage in terms of usability (both are pretty good but with documentation quirks), so I think I would pick my workflow based on the question at hand.