BIOS 7659 Homework 5

Tim Vigers

26 October 2020

# 1. Next Generation Sequencing (NGS): Sample Size Estimates

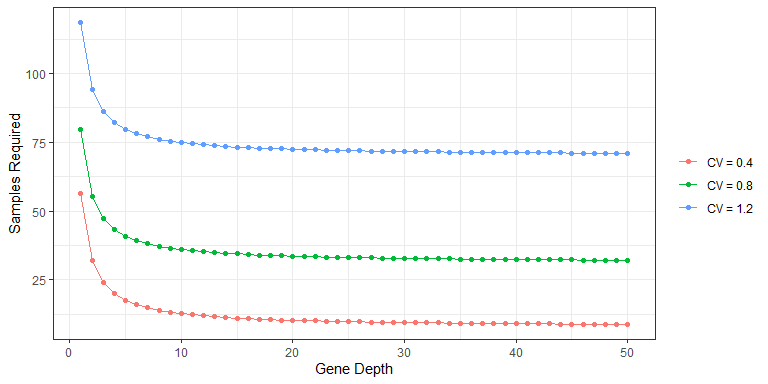
First, load the datasets into R:

data(montgomery.subset)  
data(uCovar)

## a) Recreate Figure 3 from the journal club paper

Figure 3 from Hart et al. shows the sample size needed to detect a two-fold difference in expression with 80% power and for three different CV’s and a range of sequencing depths from 1 to 50. The paper claims that their plot was generated using , but the following figure does not match the paper’s version exactly. The paper also mentions a type error rate of 5%, which suggests , but I wasn’t able to match the paper’s figure using this value for either.

# Parameters  
depths = seq(1,50)  
effect = 2  
alpha = 0.01  
power = 0.8  
# Calculate n  
cv0.4 = rnapower(depths,cv = 0.4,effect = effect,alpha = alpha,power = power)  
cv0.8 = rnapower(depths,cv = 0.8,effect = effect,alpha = alpha,power = power)  
cv1.2 = rnapower(depths,cv = 1.2,effect = effect,alpha = alpha,power = power)  
# Plotting data  
fig3\_data = as.data.frame(cbind(depths,cv0.4,cv0.8,cv1.2))  
fig3\_data = fig3\_data %>% pivot\_longer(cv0.4:cv1.2)  
# Plot  
ggplot(fig3\_data,aes(x=depths,y=value,color=name)) +   
 geom\_point() + geom\_line() +  
 theme\_bw() + xlab("Gene Depth") + ylab("Samples Required") +   
 scale\_color\_discrete(name="",labels=c("CV = 0.4","CV = 0.8",  
 "CV = 1.2"))



This figure shows that as the gene depth increases, the required sample size decreases. After a depth of approximately 10, this effect tends to level off and the gains from increasing gene depth are minimal. Also, a higher coefficient of variation (CV) across samples requires a larger sample size to achieve adequate power, as you would expect.

## b) Create a row for the Mongomery data in Table 1

The more I read the description of this table and try to recreate it, the more confusing it becomes. I think that we need to sum the raw counts for each gene (row-wise), and then divide that sum by the total number of reads in the whole experiment (in millions). Then, we put these counts per million mapped into the bins specified in the Hart paper and then convert counts to percentages.

# Convert  
total\_reads = sum(unlist(montgomery.subset))/1e06  
gene\_totals = rowSums(montgomery.subset)  
avg = round(mean(gene\_totals))  
gene\_totals = gene\_totals / total\_reads  
# Count the number in each bin  
gene\_totals = cut(gene\_totals,c(0,0.01,0.1,1,10,100,1000,Inf),right = F,  
 labels = c("< 0.01","0.01–.1","0.1–1","1–10","10–100",  
 "100–1000","> 1000"))  
# As percentages  
gene\_percs = round(table(gene\_totals)/length(gene\_totals) \* 100,1)  
# Print  
line = c(ID="m",Sample="Montgomery",  
 `Sample Type`="lymphoblastoid cell lines",  
 n=10,`Read Type`="PE",`Avg Reads` = avg,`% mapped`=100,  
 gene\_percs)  
kable(t(line))

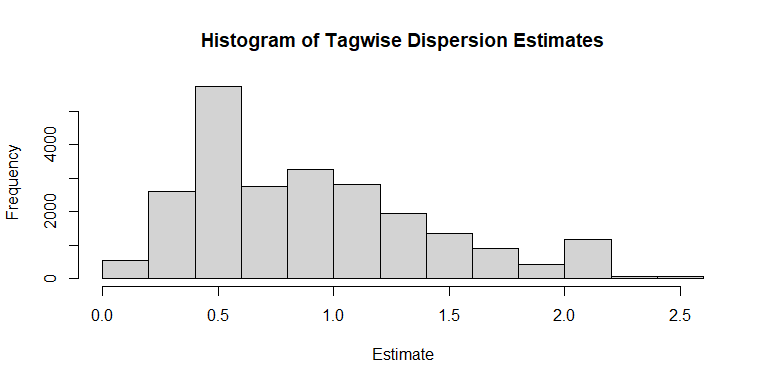
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Sample | Sample Type | n | Read Type | Avg Reads | % mapped | < 0.01 | 0.01–.1 | 0.1–1 | 1–10 | 10–100 | 100–1000 | > 1000 |
| m | Montgomery | lymphoblastoid cell lines | 10 | PE | 1478 | 100 | 0 | 23 | 21.3 | 18.3 | 26.7 | 10.4 | 0.3 |

This dataset has much a much higher average read count than any in the Hart paper, although this could be due to the different pre-filtering steps (i.e. the Montgomery data subset excluded genes with all 0 counts). This is also likely part of the reason that the distribution across the bins is slightly different for the Montgomery data compared to Hart’s datasets. For example, the Montgomery dataset has 0% in the < 0.01 bin. Also, the brain datasets have more genes in the 10 - 100 bin, whereas the Montgomery data is more evenly spread across the bins and has more genes in the 0.01 - 0.1 range than the Hart datasets.

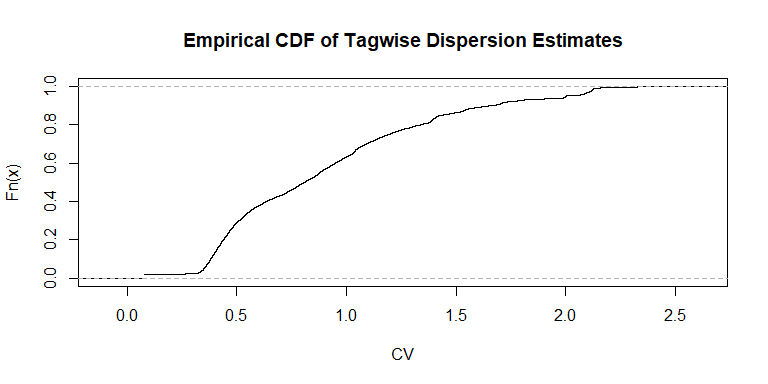
## c) Calculate the biological coefficient of variations (CV)

First, estimate the common dispersion using estimateCommonDisp(), then use this estimate for the dispersion parameter in estimateTagwiseDisp(). According to the edgeR documentation, biological CV “is the square root of the dispersion parameter under the negative binomial model.” Plot a histogram and empirical CDF of these CV estimates, and print the quantiles:

# Get common dispersion estimate  
c = estimateCommonDisp(montgomery.subset)  
# Estimate biological CV  
e = sqrt(estimateTagwiseDisp(montgomery.subset,dispersion = c))  
# Histogram  
hist(e,main="Histogram of Tagwise Dispersion Estimates",  
 xlab = "Estimate")



plot(ecdf(e),main="Empirical CDF of Tagwise Dispersion Estimates",  
 xlab="CV")



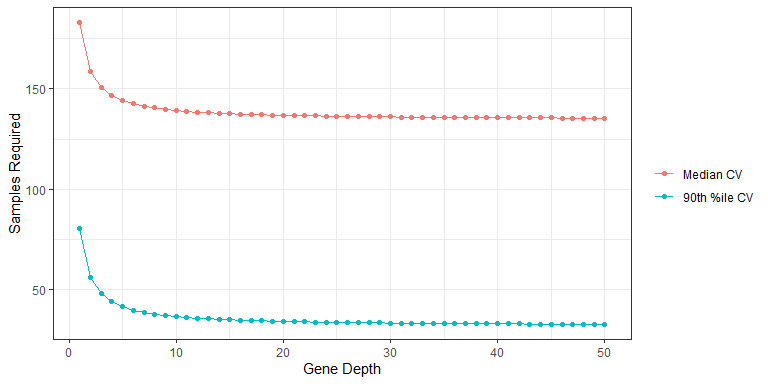
quantile(e,seq(0.1,1,0.1))

## 10% 20% 30% 40% 50% 60% 70% 80%   
## 0.3811800 0.4386644 0.5109126 0.6351171 0.8094617 0.9490281 1.0915243 1.3373902   
## 90% 100%   
## 1.6624852 2.4394217

The median CV using this method is 0.8094617 and the 90th percentile is 1.6624852. These estimates are slightly higher than the Hart paper human samples, which ranged “from 0.32 to 0.74 with a median of 0.43.”

## d) Recreate Figure 3 from Hart et al. again

# Calculate n  
cv\_median = rnapower(depths,cv = median(e),effect = effect,alpha = alpha,  
 power = power)  
cv\_90 = rnapower(depths,cv = quantile(e,0.9),effect = effect,  
 alpha = alpha,power = power)  
# Plotting data  
fig3\_data\_2 = as.data.frame(cbind(depths,cv\_median,cv\_90))  
fig3\_data\_2 = fig3\_data\_2 %>% pivot\_longer(cv\_median:cv\_90)  
# Plot  
ggplot(fig3\_data\_2,aes(x=depths,y=value,color=name)) +   
 geom\_point() + geom\_line() +  
 theme\_bw() + xlab("Gene Depth") + ylab("Samples Required") +   
 scale\_color\_discrete(name="",labels=c("Median CV","90th %ile CV"))

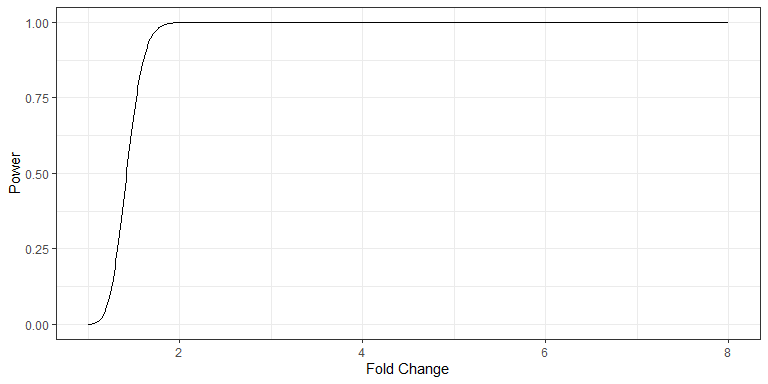


Based on the plot above and assuming depth of at least 10, I would recommend a sample size of approximately 100. This also assumes that the median biological CV estimated here is reasonable, but if there’s good reason to expect that the biological CV will be closer to the 90th percentile estimate, then it’s a possible that a sample size of 25 would be acceptable.

## e) Recreate the curve in the top of Figure 4

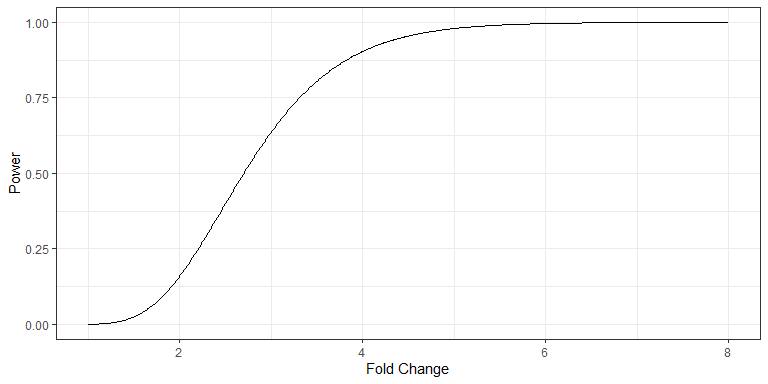
The curve we are trying to recreate is “the power curve for n = 20 per group, coverage of 100, (60th percentile of observed) and ”:

# Set parameters  
n = 20  
depth = 100  
cv = 0.32  
alpha = 0.001  
effects = seq(1,8,0.01)  
# Plot power as a function of fold change  
p = rnapower(depth=depth,n=n,cv=cv,alpha=alpha,effect=effects)  
plot\_data = as.data.frame(cbind(effects,p))  
ggplot(plot\_data,aes(x=effects,y=p)) + geom\_line() +  
 theme\_bw() + ylab("Power") + xlab("Fold Change")



The power curve in the Hart et al. paper tops out at just over 0.6. I was not able to recreate this, but the plot above looks much more like what we would expect for a power curve. Below is the same plot using the 60th percentile of biological CV from the Montgomery data as :

# Plot power as a function of fold change  
p = rnapower(depth=depth,n=n,cv=quantile(e,0.6),alpha=alpha,  
 effect=effects)  
plot\_data = as.data.frame(cbind(effects,p))  
ggplot(plot\_data,aes(x=effects,y=p)) + geom\_line() +  
 theme\_bw() + ylab("Power") + xlab("Fold Change")



# 2. Next Generation Sequencing: Pre-Processing

First load the example data:

data(geneLevelData)  
data(yeastGC)  
data(yeastLength)

## a)

all\_zero = length(which(rowSums(geneLevelData)==0))  
one\_zero = length(which(rowSums(geneLevelData==0)>0))

Within geneLevelData, 557 genes have all 0 counts and 1043 have at least one sample with a 0.

Now, create a filtered dataset (geneLevelDataFilter) with only genes containing 10 counts summed across all samples:

geneLevelDataFilter = geneLevelData[which(rowSums(geneLevelData)>=10),]

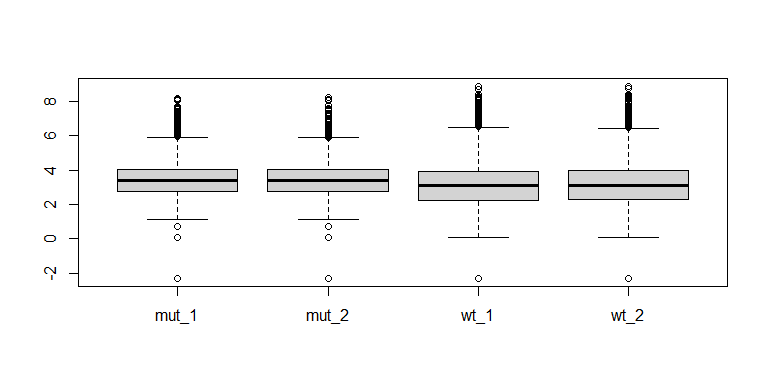
Create a SeqExpressionSet object for the EDASeq functions:

exprs = as.matrix(geneLevelDataFilter) # matrix of counts  
sub = intersect(rownames(geneLevelDataFilter), names(yeastGC))  
exprs = exprs[sub,] #only examine genes with annotated GC content/length  
row.names(exprs) = NULL #remove row and column names  
colnames(exprs) = NULL  
#Create SeqExpressionSet, which contains counts, labels for the  
#samples and GC content/length  
feature <- data.frame(gc=yeastGC[sub],length=yeastLength[sub])  
counts <-   
 newSeqExpressionSet(counts=as.matrix(geneLevelData[sub,]),  
 featureData=feature,  
 phenoData=data.frame(  
 conditions=factor(c(rep("mut",2),rep("wt",2))),  
 row.names=colnames(geneLevelData)))

## b)

### Plot the counts by sample:

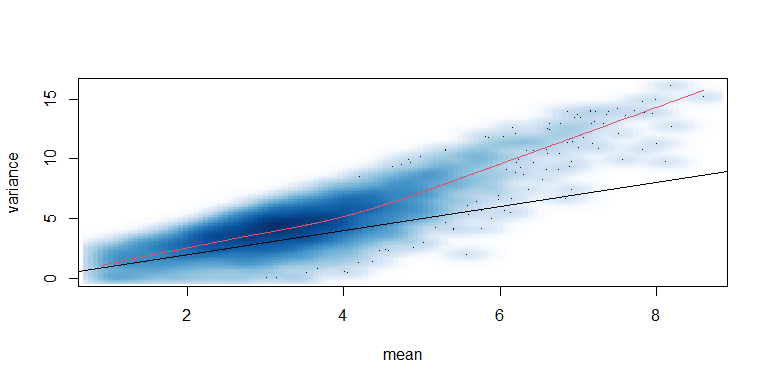
EDASeq::boxplot(counts)



The two wildtype samples (3 and 4 above) do appear to be slightly different from the other two samples. Their distributions look slightly wider, and the medians appear to be slightly different as well, which indicates a need for normalization.

### Plot the mean by variance plot:

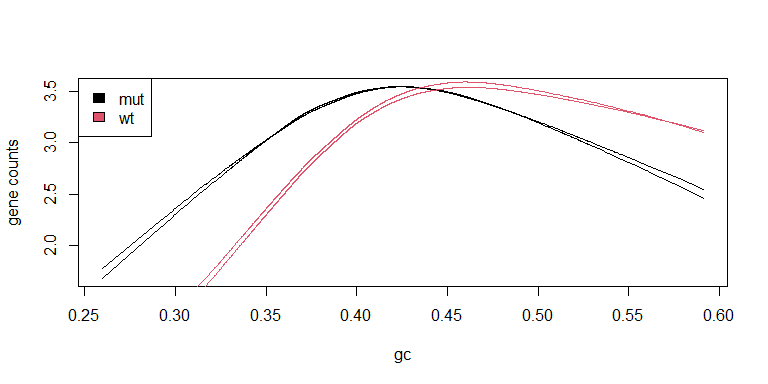
meanVarPlot(counts,log=T)



Based on this plot it appears that as the mean increases so does the variance. The red lowess curve indicates that this relationship is particularly strong for a mean above 4 (on the log scale).

### Assess any biases by GC content

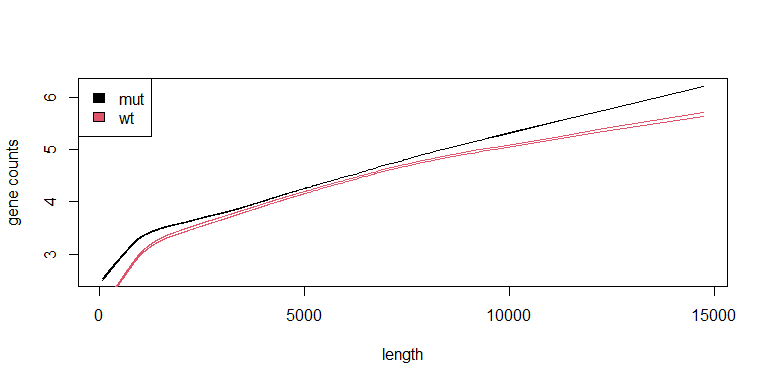
biasPlot(counts,"gc",log=TRUE)



Based on the above plot, it appears that there is GC bias in this dataset because the wildtype samples generally appear to have a higher proportion of GC than the mutants.

### Assess any biases by length

biasPlot(counts,"length",log=TRUE)



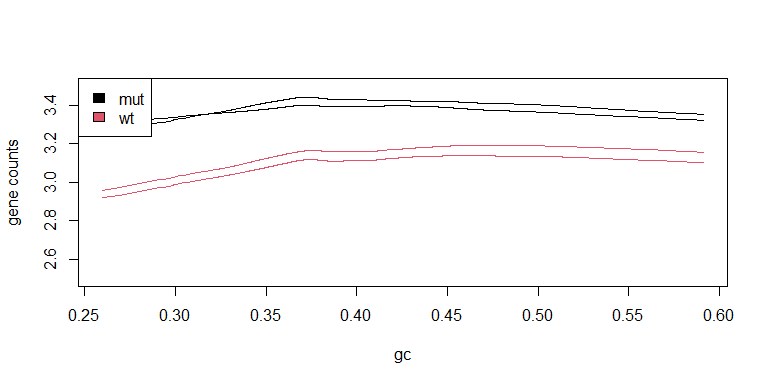
Overall there doesn’t appear to be much bias by length. As gene length goes above 10,000, the wildtype samples tend to have fewer counts than the mutants but it does not appear to be particularly drastic in this dataset.

## c) Apply withinLaneNormalization() to normalize by GC content

The withinLaneNormalization() function has four options for normalization: loess, median, full, and upper. The loess approach regresses the counts on the GC proportion and then subtracts the loess line from the counts. The median, upper, and full methods stratify genes into a given number of bins (10 being the default) based on GC content. Then the median approaches scales the data so each bin has the same median, the upper does the same but for the upper quartile instead of the median, and the full method “forces the distribution of each stratum to be the same using a non linear full quantile normalization.”

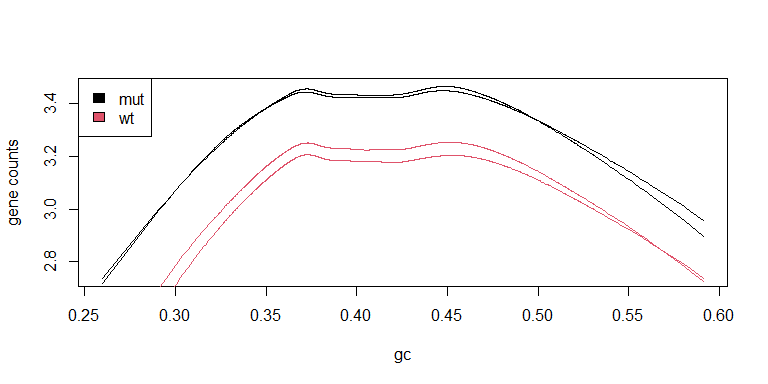
### Loess method

loess = withinLaneNormalization(counts,"gc",which = "loess")  
biasPlot(loess,"gc",log=TRUE,ylim=c(2.5,3.5))



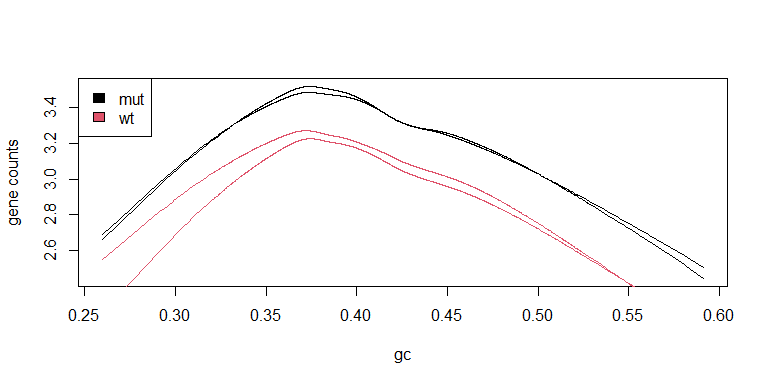
### Full method

full = withinLaneNormalization(counts,"gc",which = "full")  
biasPlot(full,"gc",log=TRUE)



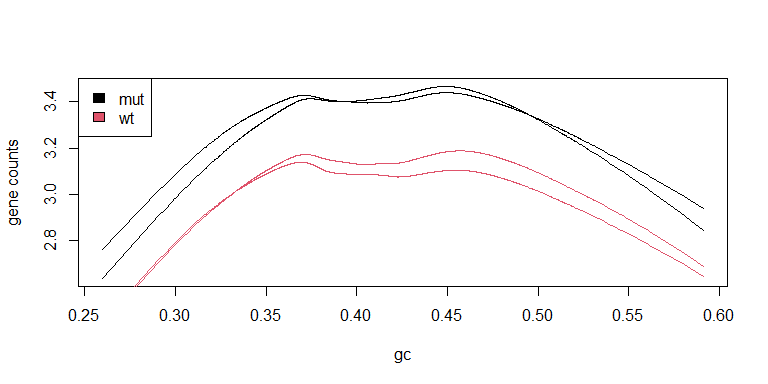
### Upper method

upper = withinLaneNormalization(counts,"gc",which = "upper")  
biasPlot(upper,"gc",log=TRUE)



### Median method

median = withinLaneNormalization(counts,"gc",which = "median")  
biasPlot(median,"gc",log=TRUE)



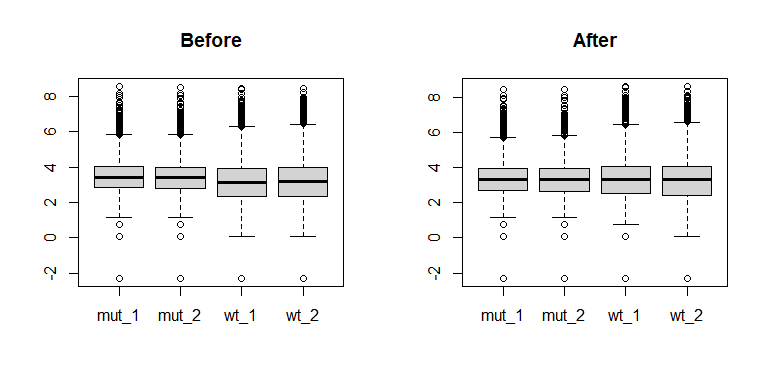
Of these methods, the loess approach appears to be the best as the gene counts are more even across the GC proportions (i.e. the lines are flatter). The other methods align the peaks of the wildtype and mutant curves, but the gene counts still depend heavily on GC proportion for both conditions.

## d) Apply betweenLaneNormalization() to normalize across samples

This method is similar to the withinLaneNormalization() function, but does not include a loess method. Instead of normalizing bins within a lane, however, these methods ensure that the lanes are similar. So, the median approach forces the median of all the lanes to the be the same, the upper approach does the same but using the upper quartile rather than the median, and the full method uses the full quantile method. First, plot before and after normalization for each between-lane method, using the data normalized within-lane using the loess method:

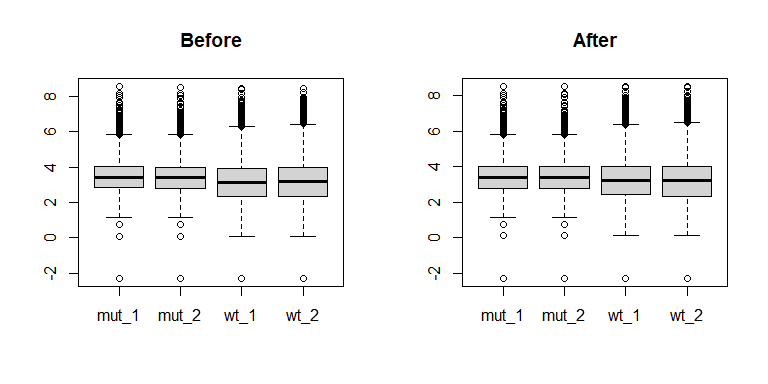
### Median

par(mfrow=c(1,2))  
# Loess  
loess\_median = betweenLaneNormalization(loess,"median")  
boxplot(loess,main = "Before")  
boxplot(loess\_median,main = "After")



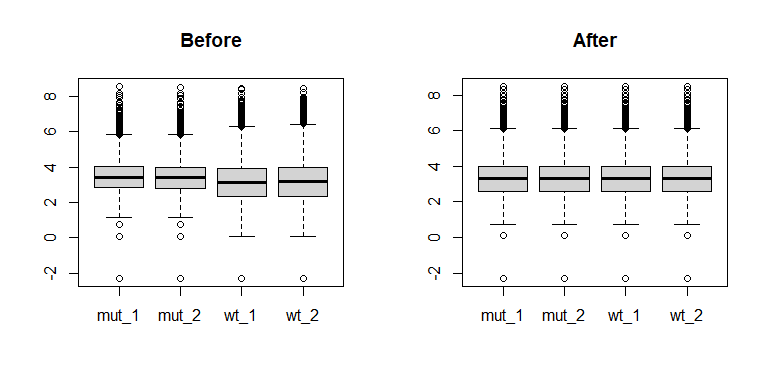
### Upper

par(mfrow=c(1,2))  
# Loess  
loess\_upper = betweenLaneNormalization(loess,"upper")  
boxplot(loess,main = "Before")  
boxplot(loess\_upper,main = "After")



### Full

par(mfrow=c(1,2))  
# Loess  
loess\_full = betweenLaneNormalization(loess,"full")  
boxplot(loess,main = "Before")  
boxplot(loess\_full,main = "After")



Between the median, upper, and full methods, the full approach appears to work the best as it aligns the whole distribution of each sample, rather than just the median.