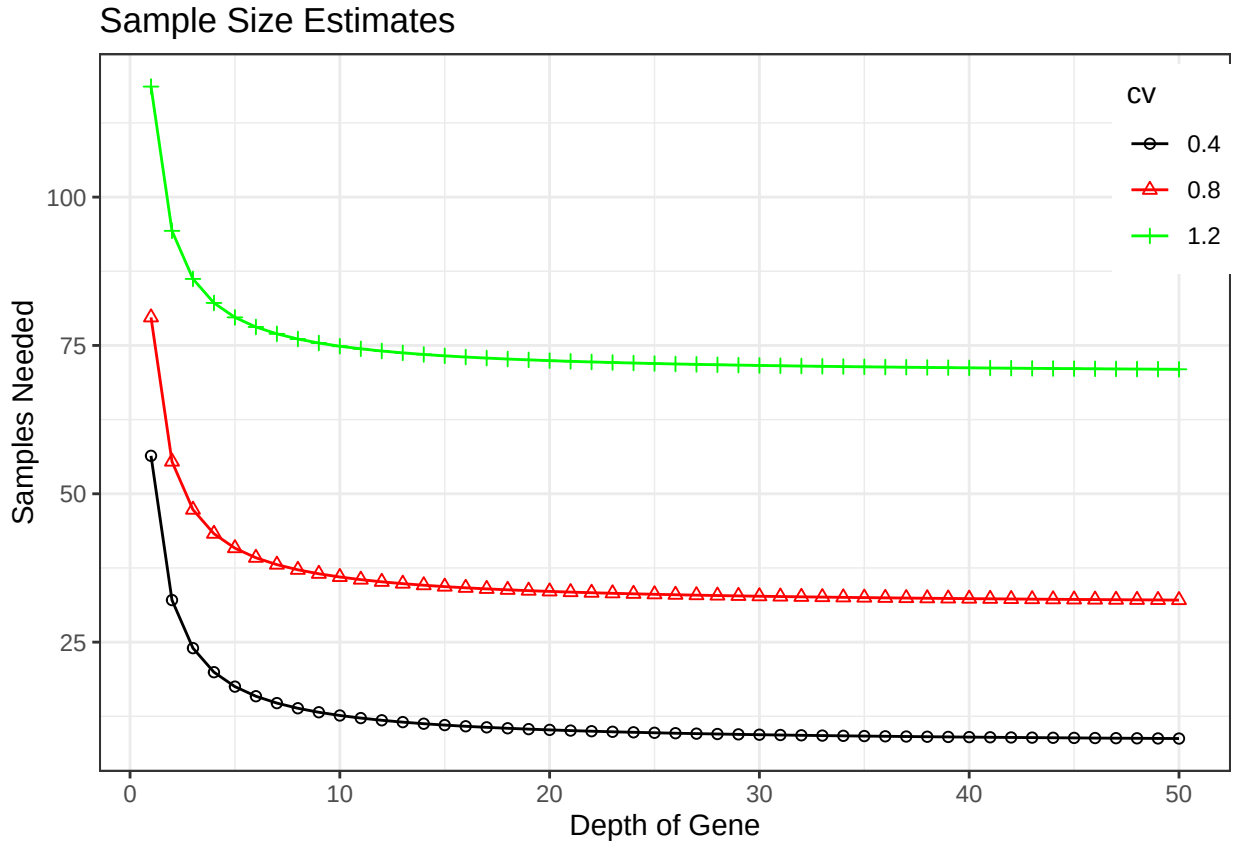
Homework 5 - BIOS 7649

Dominic Adducci

Question 1 - Next Generation Sequencing: Sample Size Estimates

Part A

Using **rnapower()**, recreate Figure 3 from the journal club paper using: Hart et al., (2013) "Calculating sample size estimates for RNA sequencing data." What does this figure show?



The above figure shows the sample size needed as the depth of gene increases from 1 to 50, and cv (coefficient of variation) varies between 0.4, 0.8, and 1.2.

Part B

For the Montgomery data, create a row in Table 1 in the Hart et al. paper. Note that genes that have zero counts in all 10 samples were already excluded, so "% mapped", will be 100%. How does the Montgomery data compare to the other data sets in Table 1?

Table 1: Montgomery Data: Counts per Gene per Millions Reads Mapped

Sample	n	Avg Reads	% Mapped	<0.01	0.01-0.1	0.1-1	1-10	10-100	100-1000	>1000
Montgomery Data	10	4.25	100	0	21.02	22.92	18.5	26.97	10.32	0.27

Table 1 shows the recreation of the Table 1 in Hart et al. for the Montgomery data set. Because genes with 0 counts in all samples were excluded the % mapped is 100% in contrast to the Table 1 in Hart et al., which had a range of mapping from 18-78%. Compared to the Hart et al. data set a higher percentage of genes were detected in the counts per million ranges of 0.01-0.1 and 0.1-1. This may suggest that even with the genes with zero counts across all samples excluded the percent mapped for the Montgomery data set was greater than for the Hart et al. data sets.

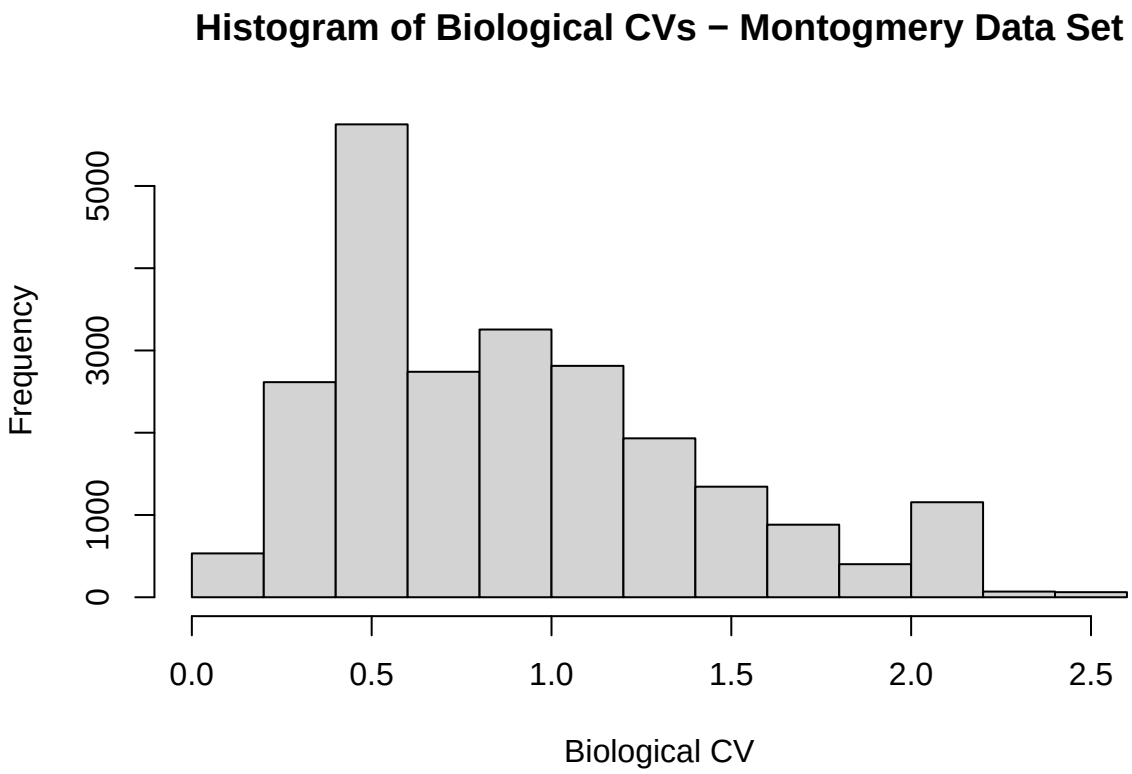
Part C

Calculate the biological coefficient of variations (CV) from the Montgomery human lymphoblastoid data (use the function **estimateTagwiseDisp()** in edgeR). Plot the histogram and empirical cdf (use **ecdf()** function) and report the median and 90% percentile. How do the CVs compare with the examples in the Hart et al., paper in Figure 2?

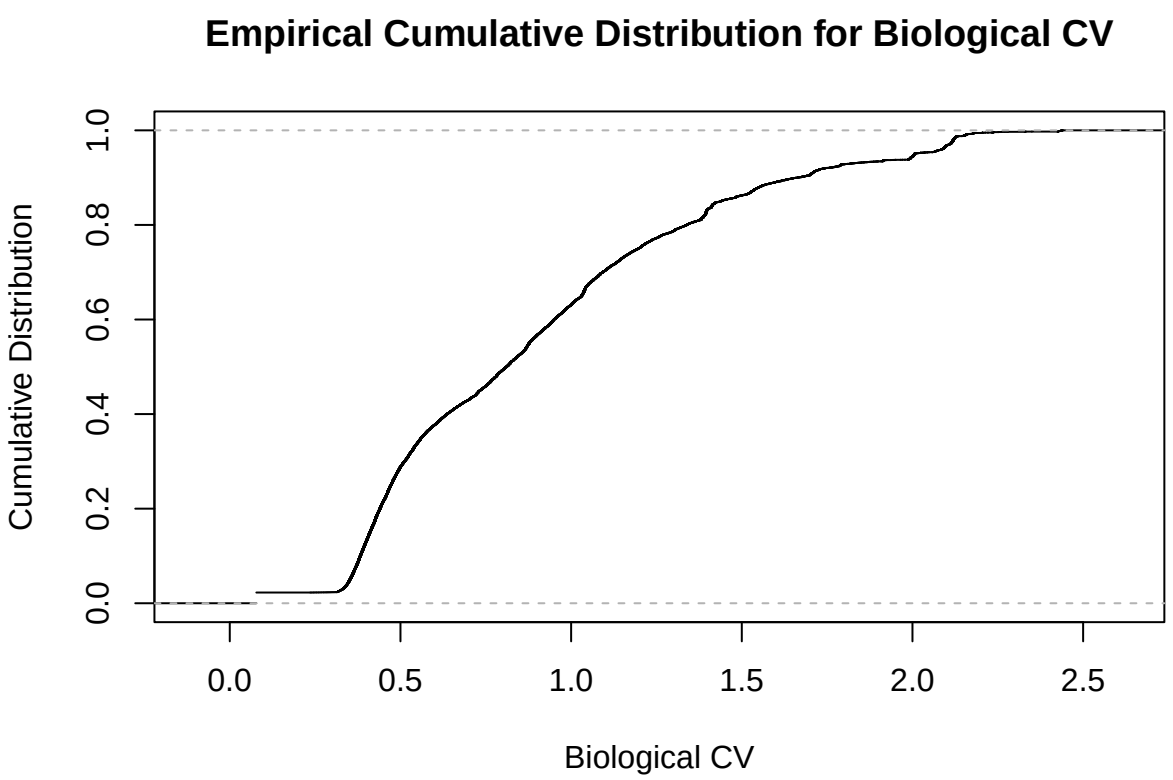
The square root of the negative binomial dispersion is known as the biological coefficient of variation (BCV).

$$BCV = \sqrt{\phi_g} \; ; \; where \; \phi_g = NB \; dispersion$$

Calculating this first involved estimating the common dispersion and then the tagwise dispersion. After this the tagwise dispersion can be extracted and plotted using both a histogram and a cumulative distribution plot.



The above histogram shows the frequency of different biological CV values for the Montgomery data set. There is a noticeable peak around 0.5, as well as some right skew.

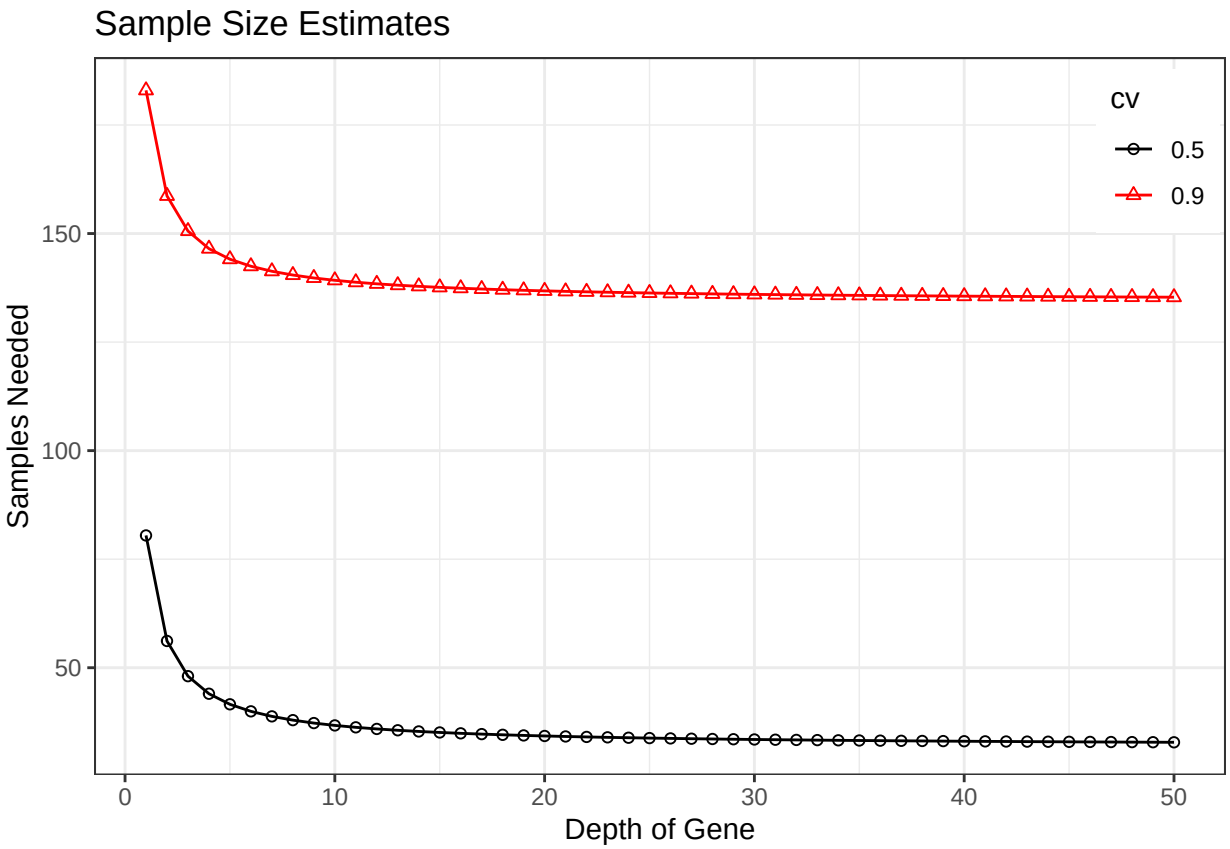


The above plot shows the cumulative distribution of biological CV from the Montgomery data set. Compared to the Hart et al. data sets the Montgomery data set has a cumulative distribution which is more spread out, as the highest biological CV value goes past 2.0, compared to the Hart et al. data sets which reached a cumulative distribution of 1 around a biological CV of 1.

- median percentile: 0.8094617
- 90th percentile: 1.6624852

Part D

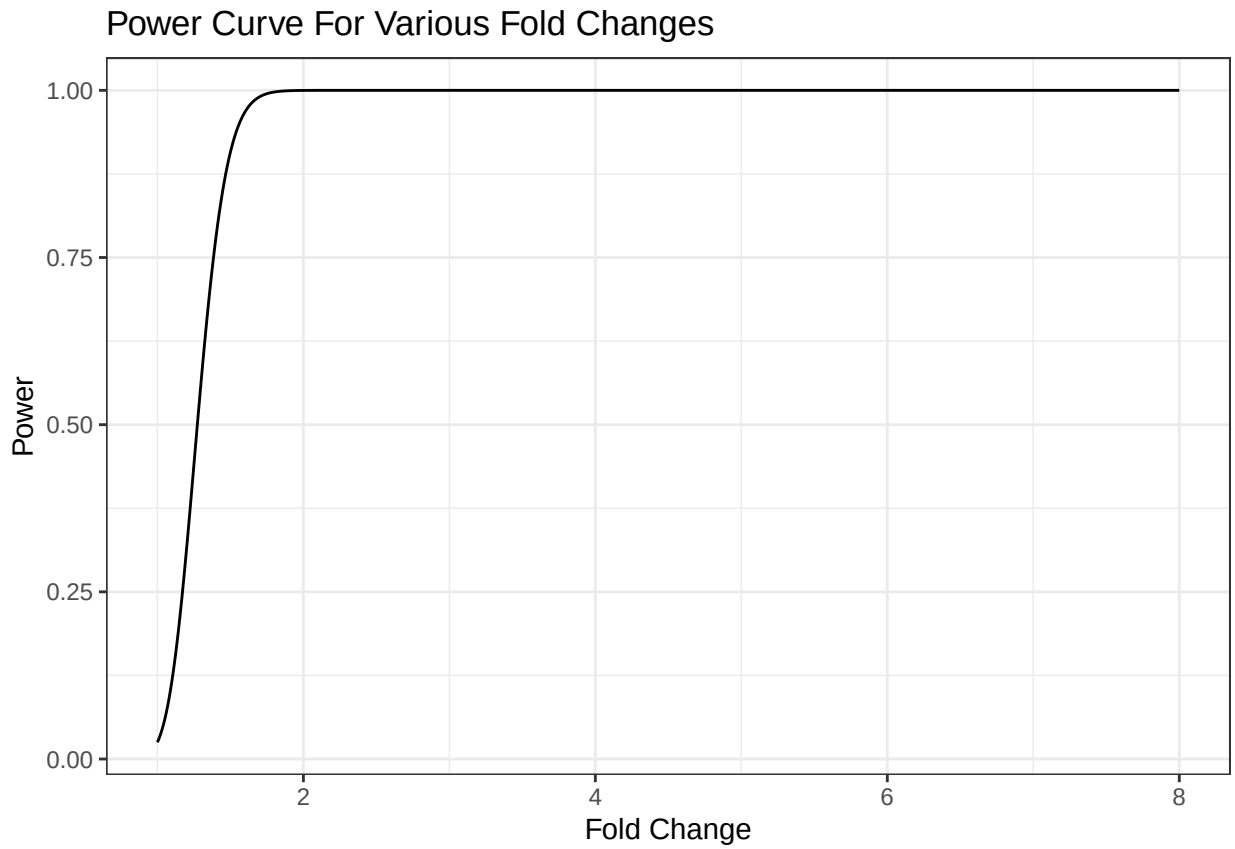
Using **rnapower()**, recreate Figure 3 from Hart et al. again but with two curves using the median and the 90% percentile CV across genes for the Montgomery data. What sample size do you recommend?



Using a CV value of the median to calculate sample size will only ensure that half of the genes are below that CV value. Using the 90th percentile will ensure that 90% of genes are at or below that CV value. Because the sample sizes stabilize relatively quickly as depth of gene changes along the x-axis a sample size of around 135 should be used.

Part E

Using **rnapower()**, recreate the curve (not the histogram) in the top Figure 4 from the Hart et al., paper. If you cannot recreate the figure, please explain any differences.



The above plot shows the relationship between power and fold change.This plot looks relatively close to the plot provided in Hart et al. However, a depth of 20 was assumed for this plot, which was not mentioned in the paper. Additionally, the text of the paper says an $\alpha = 0.001$ was used, while the figure caption says an $\alpha = 0.05$ was used. For the above plot a depth of 20, a sample size of 20, a CV of 0.32, and an alpha of 0.05 were set.

Question 2 - Next Generation Sequencing: Pre-Processing

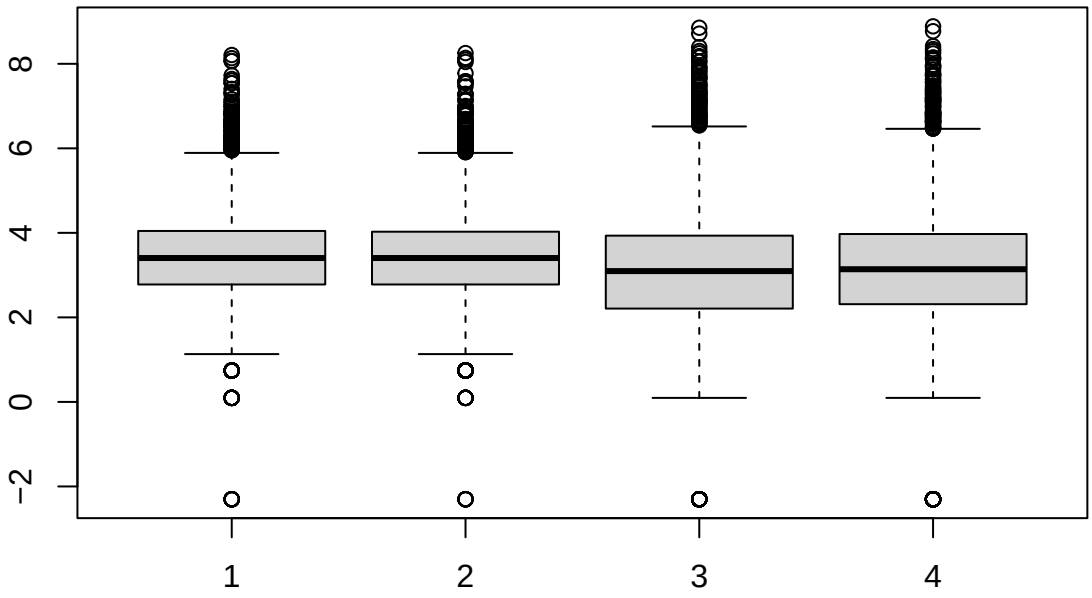
Part A

Within **geneLevelData**, how many genes have all zeros as counts? How many have at least one sample with a zero?

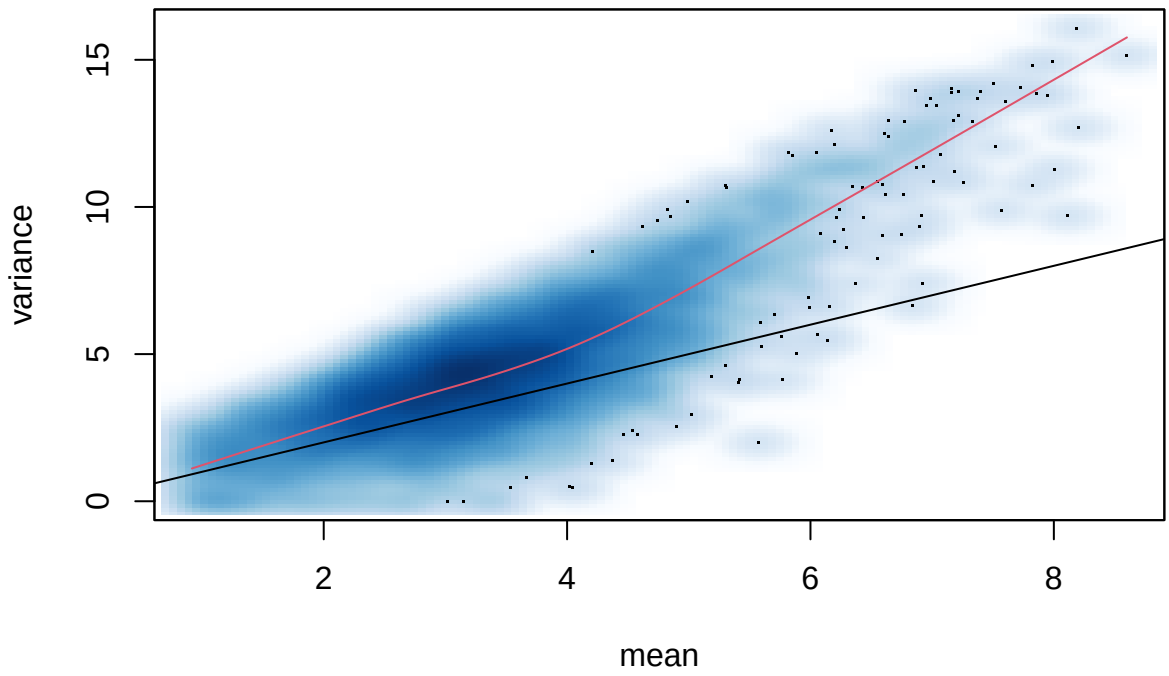
- There are 557 genes that have all zero as counts.
- There are 1043 genes that have at least one zero count between the four different samples.

Part B

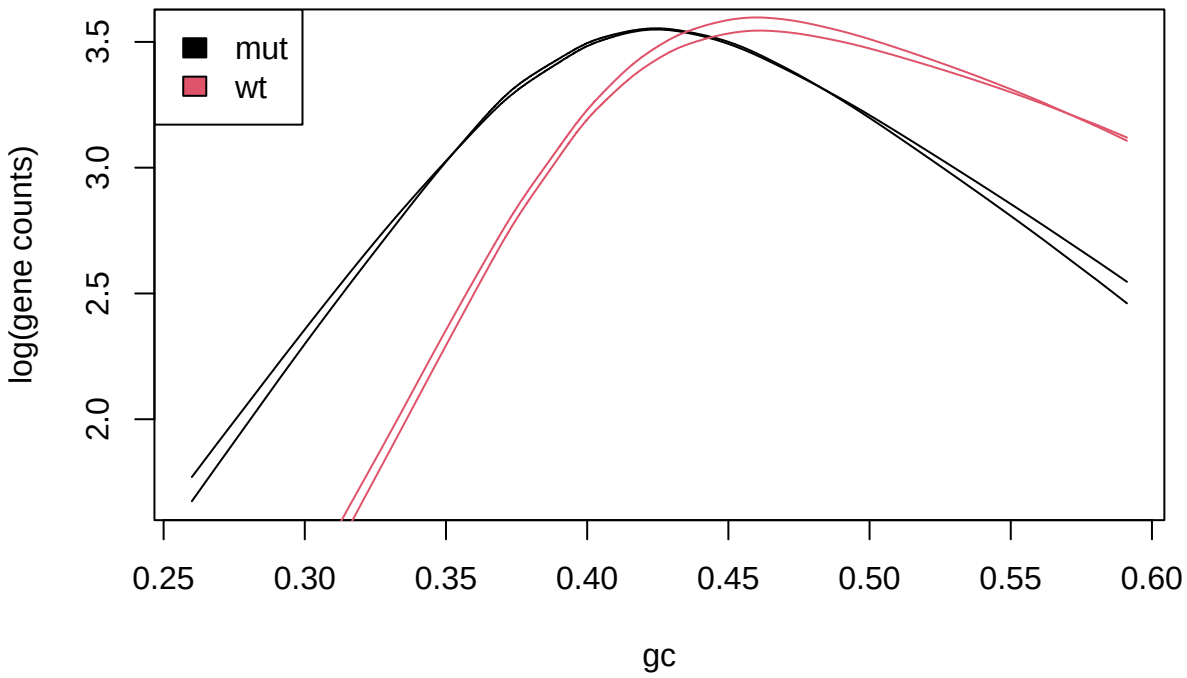
For the following plots, use the log scale.



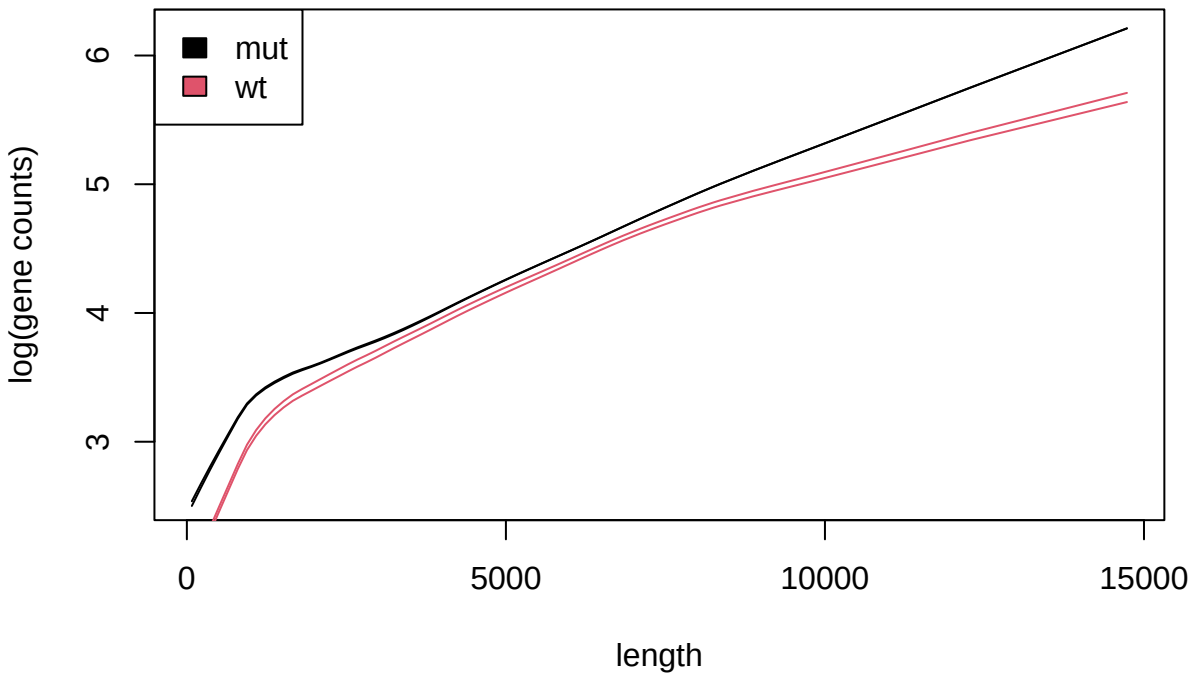
The above figure shows boxplots of the sample counts. Because a **SeqExpressionSet** was created the output from the boxplot should be on the log scale already. The medians are not the same for all samples, with 1 and 2 (mutant) are higher than 3 and 4 (wild type).



The above plot shows the variance-mean relationship for the counts data. Both sample types have a similar trend for lower means, but once mean count gets past 4 the variance of the wild type samples increases rapidly compared to the mutant samples.



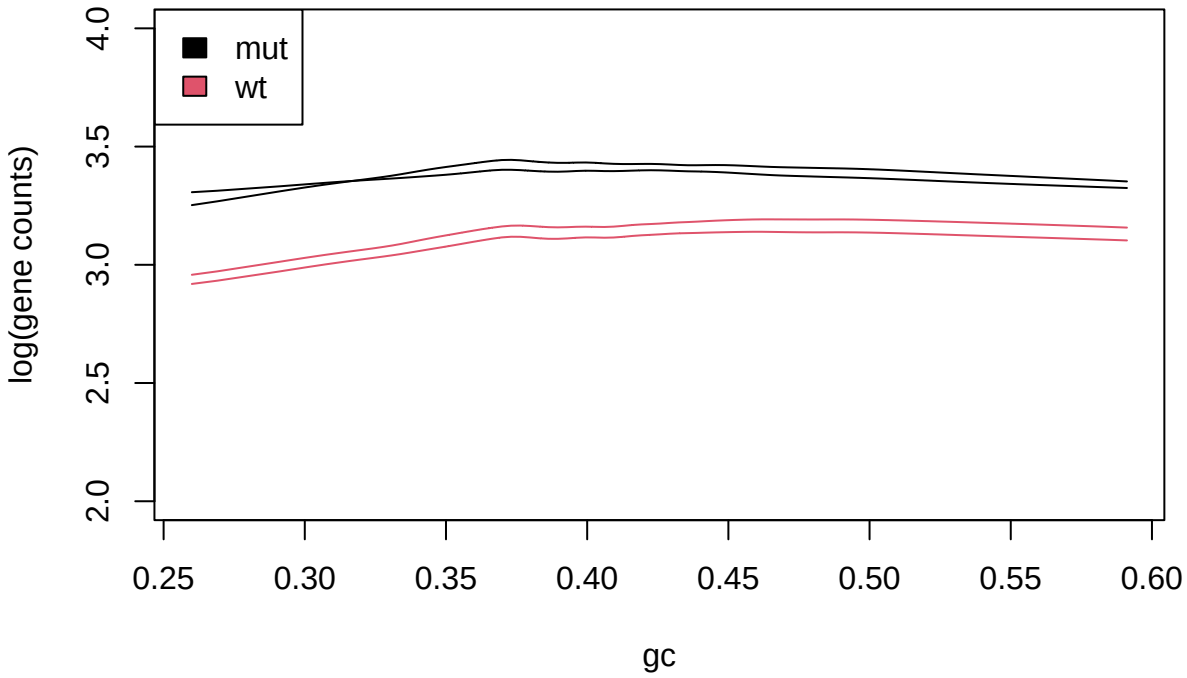
The above plot illustrates the count versus the GC content. As GC content increases both the mutant and wild type samples increase in count. Initially the mutant samples have a higher count with lower GC content, but this changes after both samples peak, where the wild type has higher gene count for the highest GC contents.



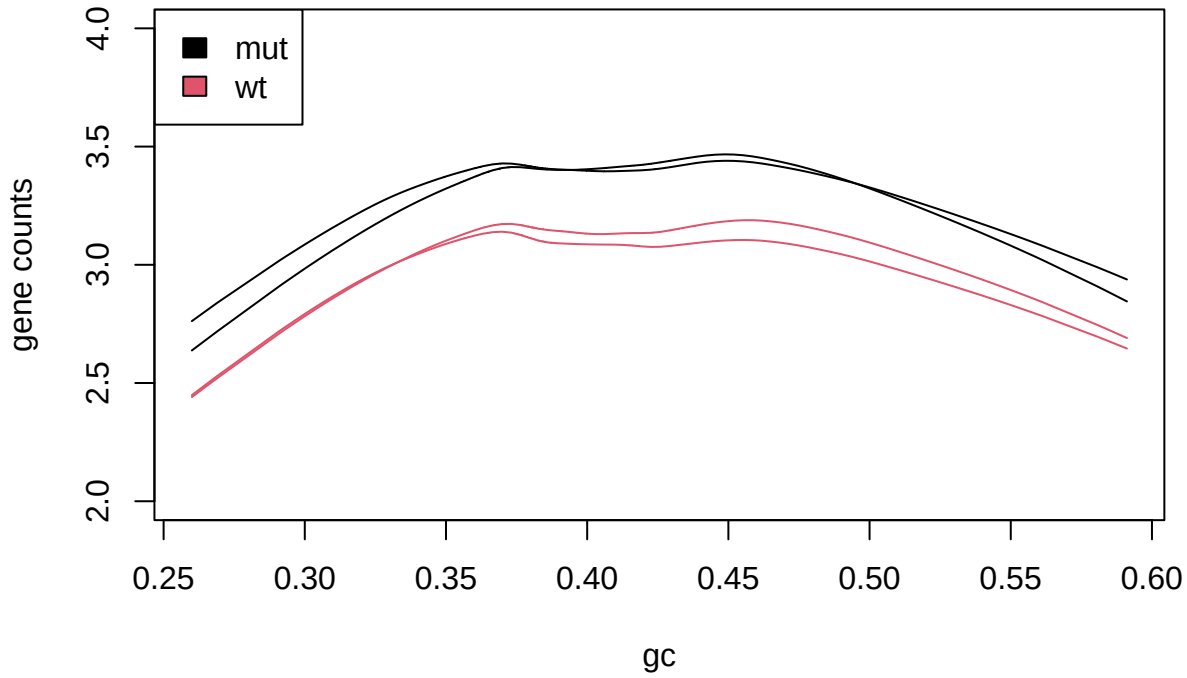
The above plot illustrates the count versus the length (in base pairs). The mutant samples always have higher gene counts for all lengths, with differences increasing as length gets smaller and larger. The gene counts are relativly similar for length around the middle portion of the plot before diverging.

Part C

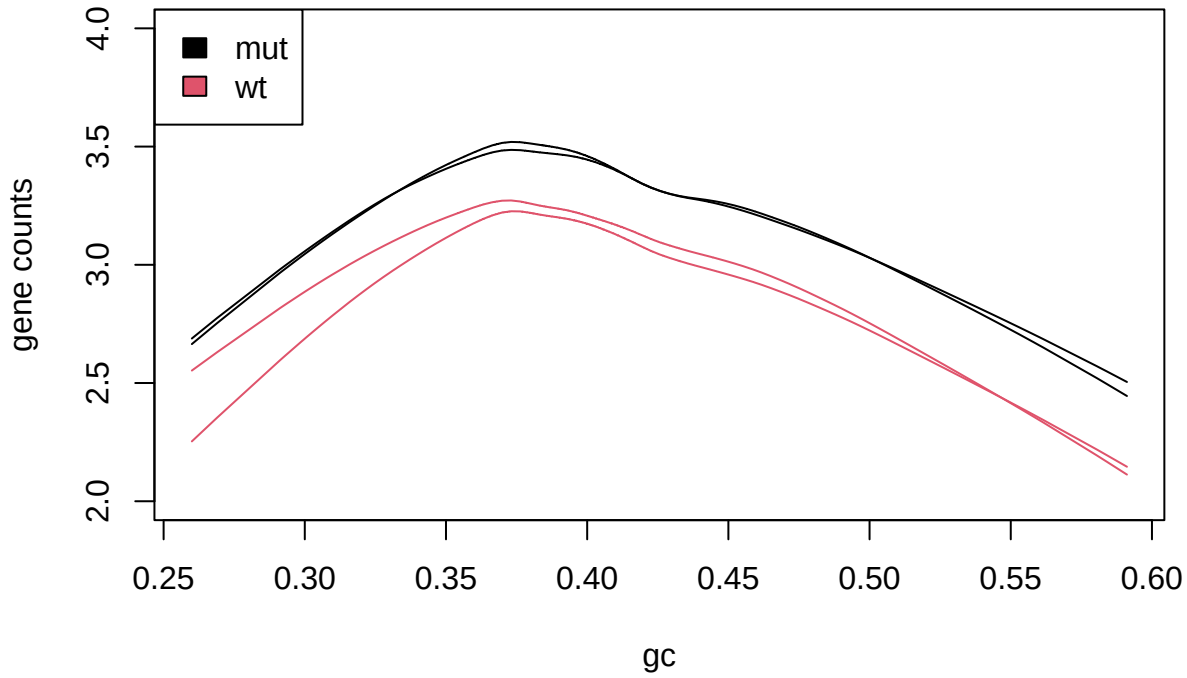
Apply **withinLaneNormalization()** to normalize by GC content. See help file and try different methods for normalization with the "which" option. Describe the different methods. Use **biasPlot()** before and after normalization. How do the methods compare? Save the normalized results in a new object to use in the next part.



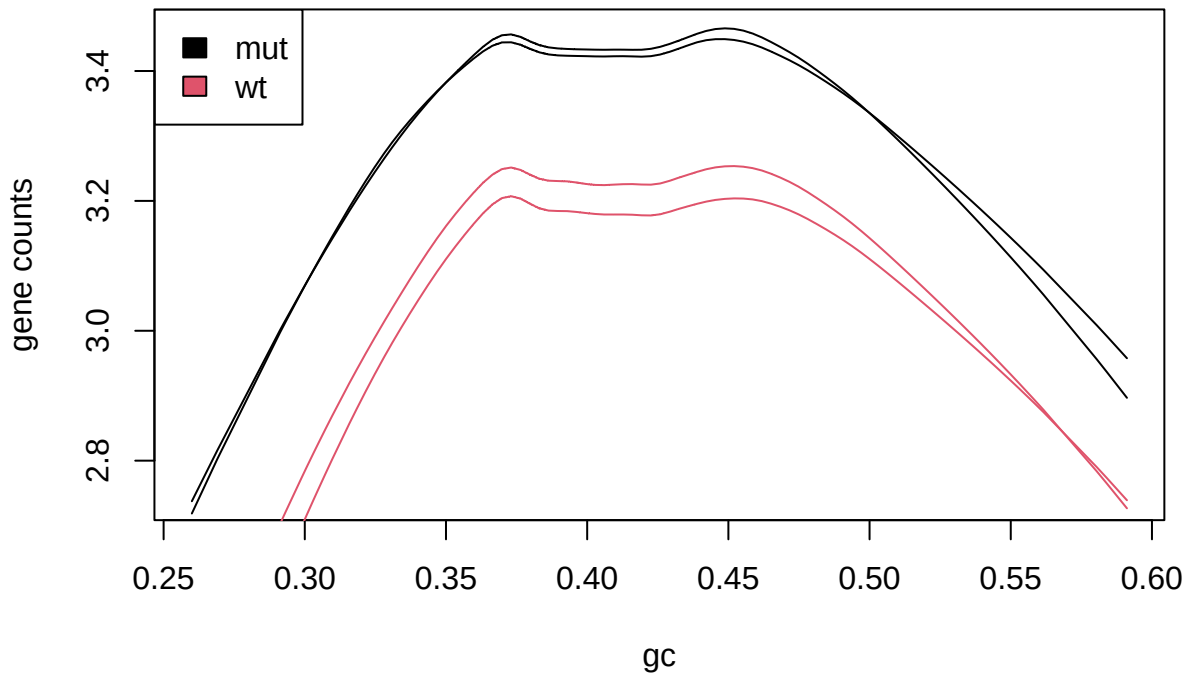
The above plot show the results after normalizing using the within lane loess method. The difference in gene counts as GC increases between the mutant and wild type samples is more consistent throughout the GC range. The loess method works by regressing the counts and subtracting the loess fit from the counts, removing the dependence.



The above plot show the results after normalizing using the within lane median method.The difference in gene counts as the GC increases is similar to the loess method in that there is more consistency between the mutant and wild type samples compared to the plot showing pre-normalization results. However, the median method has a larger spread between samples compared to the loess method. The median method works by stratifying genes and then scaling the data so that each bin has the same median.



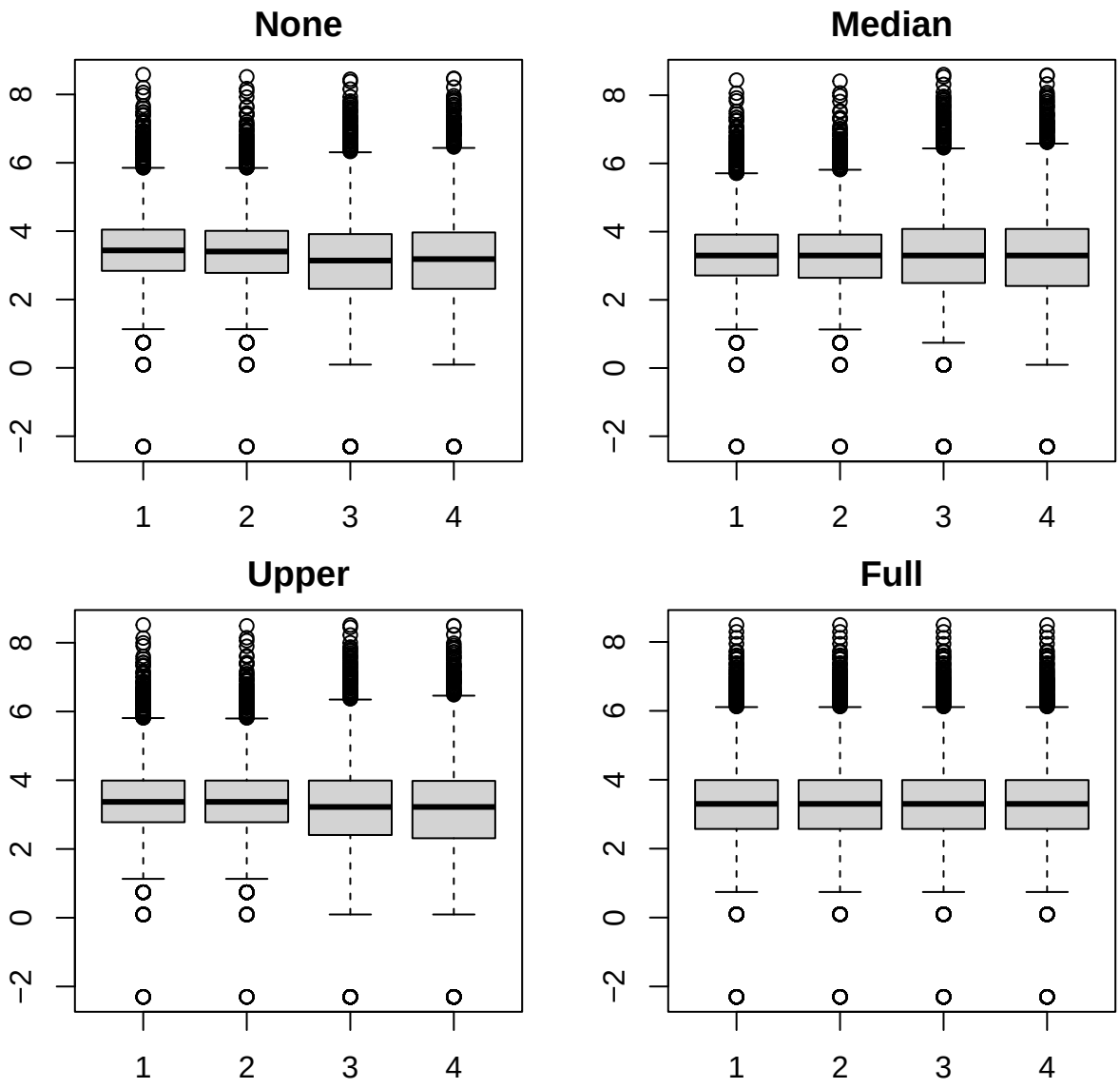
The above plot show the results after normalizing using the within lane upper method. As with the previous two methods there is more consistency between samples across the GC range compared to the unnormalized data. However, the left hand side of this plot shows a significant difference between the wild type samples compared to the other methods. The upper method works by scaling the data so that the same upper quartiles fit in each bin.



The above plot show the results after normalizing using the within lane full method. This method also shows more consistency between the mutant and wild type samples across the GC range. In the middle portion of the plot there is a noticeable difference in counts between the two wild type samples. The full method works by using a non linear full quartile normalization, forcing the distribution to be the same.

Part D

Using the within-lane normalized data from the previous part, apply **betweenLaneNormalization()** to normalize across samples. Try different methods for normalization with the "which" option. Describe the different methods. Use **boxplot()** before and after normalization. How do the methods compare?



The above boxplots show the results of using different between lane normalization methods on the within lane loess normalized data. The loess was used as it seemed to be the most consistent across GC range. Compared to the unnormalized (in terms of between lane normalization) plot, the median shifts the data so that all boxplots have the same median (as the name implies). The full method also shifts the data so that every sample has the same median, as well as forcing the distribution of each sample to be the same. The upper method appears to do very little, if anything, compared to the unnormalized plot.

CODE

```
library(tidyverse)
library(RNASeqPower)
library(edgeR)
library(cqn)
library(kableExtra)
library(yeastRNASeq)
library(EDASeq)

pardefault <- par()


### START QUESTION 1 CODE ###

# Loading data into R environment.
data("montgomery.subset")

data("uCovar")


## START QUESTION 1 PART A CODE ##

# Calculating sample size estimates using rnapower() function.
figure3_data <- data.frame(rnapower(depth = seq(1,50,by = 1),
                                    cv = c(.4,0.8,1.2),effect = 2,
                                    alpha = 0.01, power = 0.8))

# Formatting the results, This includes turning data into long format,
# changing values for cv, and then converting depth and cv from character
# to numeric.
colnames(figure3_data) <- c("cv0.4","cv0.8","cv1.2")

figure3_data <- rownames_to_column(figure3_data,"depth")

figure3_data_long <- pivot_longer(figure3_data,
                                  cols = starts_with("cv"),
                                  names_to = "cv",
                                  values_to = "sample_size")

figure3_data_long$cv <- replace(figure3_data_long$cv,
                                figure3_data_long$cv == "cv0.4",0.4)

figure3_data_long$cv <- replace(figure3_data_long$cv,
                                figure3_data_long$cv == "cv0.8",0.8)

figure3_data_long$cv <- replace(figure3_data_long$cv,
                                figure3_data_long$cv == "cv1.2",1.2)

figure3_data_long <- figure3_data_long %>%
  mutate(depth = as.numeric(depth),
         cv = factor(cv))
```

```r
# Plotting samples size estimates.
figure3_plot <- ggplot(figure3_data_long, aes(x = depth,
                                              y = sample_size,
                                              color = cv, shape = cv)) +
  geom_point() +
  geom_line() +
  scale_color_manual(values = c("black","red","green")) +
  scale_shape_manual(values = c(1,2,3)) +
  labs(title = "Sample Size Estimates",
       x = "Depth of Gene", y = "Samples Needed",
       legend = "CV") +
  theme_bw() +
  theme(legend.position = c(0.95,0.84))

figure3_plot

## FINISH QUESTION 1 PART A CODE ##


## START QUESTION 1 PART B CODE ##

# Making a DGEL list.
mgd_dgel <- DGEList(counts = montgomery.subset)

# Calculating counts per million.
cpm_results <- cpm(montgomery.subset)

# Average count per million for each gene.
cpm_average <- rowMeans(cpm_results)

# Initializing the ranges.
cpg_cpm_lower <- c(0,0.01,0.1,1,10,100,1000)
cpg_cpm_upper <- c(0.01,0.1,1,10,100,1000,Inf)

# Storing genes counts in a vector.
gene_counts <- vector("numeric", 7)

# Using a for loop to calculate the distribution of counts for each range.
for(i in 1:7){
  gene_counts[i] <- sum(cpm_average >= cpg_cpm_lower[i] &
                                cpm_average < cpg_cpm_upper[i]) /
    length(cpm_average) * 100
}


library_size <- sum(mgd_dgel$samples$lib.size)

gene_average <- rowSums(mgd_dgel$counts) / ncol(mgd_dgel$counts)

read_average <- mean(gene_average)/library_size * 1e6

table1_row <- data.frame(matrix(c(read_average,100,gene_counts),nrow = 1))
```

```r
table1_row <- cbind("Montgomery Data",10,table1_row)

colnames(table1_row) <- c("Sample","n","Avg Reads","% Mapped","<0.01",
                          "0.01-0.1","0.1-1","1-10","10-100","100-1000",">1000")

table1_tbl <- table1_row %>%
  kbl(caption = "Montgomery Data: Counts per Gene per Millions Reads Mapped",
      booktabs = TRUE, digits = 2) %>%
  kable_styling(latex_options = "HOLD_position")

table1_tbl

## FINISH QUESTION 1 PART B CODE ##


## START QUESTION 1 PART C CODE ##

# Calculating the biological coefficient of variation for the Montgomery data
# set. The estimateTagwiseDisp() function from edgeR is used.

# First calculating common dispersion.
cd_md <- estimateCommonDisp(mgd_dgel)

# Calculating the tagwise dispersion.
twd_md <- estimateTagwiseDisp(cd_md)

# Creating a histogram for the squareroot of the tagwise dispersion values.
twd_values <- sqrt(twd_md$tagwise.dispersion)

# Creating empirical CDF.
twd_ecdf <- ecdf(twd_values)

# Calculating median and 90th percentile cv across genes.
twd_median <- quantile(twd_ecdf,0.5)
twd_nintieth <- quantile(twd_ecdf,0.9)


hist(twd_values, main = "Histogram of Biological CVs - Montogmery Data Set",
     xlab = "Biological CV")


plot(ecdf(twd_values),
     main = "Empirical Cumulative Distribution for Biological CV",
     ylab = "Cumulative Distribution",xlab = "Biological CV")


## START QUESTION 1 PART D CODE ##

# Making a plot to show sample size using rnapower() function.
twd_rnapower <- data.frame(rnapower(depth = seq(1,50,by = 1),
                                    cv = c(twd_median,twd_nintieth),effect = 2,
                                    alpha = 0.01, power = 0.8))
```

```r
# Formatting the results. This includes turning data into long format,
# changing value for cv, and then converting depth and cv from character
# to numeric.
colnames(twd_rnapower) <- c("cv0.5","cv0.9")

twd_rnapower <- rownames_to_column(twd_rnapower,"depth")

twd_rnapower_long <- pivot_longer(twd_rnapower,
                                  cols = starts_with("cv"),
                                  names_to = "cv",
                                  values_to = "sample_size")

twd_rnapower_long$cv <- replace(twd_rnapower_long$cv,
                                twd_rnapower_long$cv == "cv0.5",0.5)

twd_rnapower_long$cv <- replace(twd_rnapower_long$cv,
                                twd_rnapower_long$cv == "cv0.9",0.9)

twd_rnapower_long <- twd_rnapower_long %>%
  mutate(depth = as.numeric(depth),
         cv = factor(cv))

twd_ss_plot <- ggplot(twd_rnapower_long, aes(x = depth,
                                             y = sample_size,
                                             color = cv, shape = cv)) +
  geom_point() +
  geom_line() +
  scale_color_manual(values = c("black","red")) +
  scale_shape_manual(values = c(1,2)) +
  labs(title = "Sample Size Estimates",
       x = "Depth of Gene",y = "Samples Needed",
       legend = "CV") +
  theme_bw() +
  theme(legend.position = c(0.94,0.87))

twd_ss_plot

## FINISH QUESTION 1 PART D CODE ##


# Calculating power for various fold changes.
power_calc <- data.frame(rnapower(depth = 20,n = 20,cv = 0.32, alpha = 0.05,
                                  effect = seq(1,8,by = 0.001)))

# Formatting the results of the power calculation.
colnames(power_calc) <- "Power"

power_calc <- rownames_to_column(power_calc,"Fold Change")

power_calc <- power_calc %>%
  mutate(`Fold Change` = as.numeric(`Fold Change`))

# Making a plot of the power curve.
```

```r
power_curve <- ggplot(power_calc, aes(x = `Fold Change`,
                                      y = Power)) +
  geom_line() +
  labs(title = "Power Curve For Various Fold Changes",
       x = "Fold Change",y = "Power") +
  theme_bw()

power_curve


### START QUESTION 2 CODE ###

# Loading in yeast data.
data(geneLevelData)
data(yeastGC)
data(yeastLength)


## START QUESTION 2 PART A CODE ##

# Finding number of genes with all zero counts.
genes_all_zero <- geneLevelData %>%
  dplyr::filter(mut_1 == 0 & mut_2 == 0 & wt_1 == 0 & wt_2 == 0)

# Finding number of genes with at least one zero count.
gene_one_zero <- geneLevelData %>%
  dplyr::filter(mut_1 == 0 | mut_2 == 0 | wt_1 == 0 | wt_2 == 0)

# Cleaning data so that only genes with a total count of 10 between samples
# are included.
geneLevelDataFilter <- geneLevelData %>%
  mutate(row_sums = rowSums(across(where(is.numeric)))) %>%
  filter(row_sums >= 10) %>%
  select(-row_sums)

# Creating a SeqExpressionSet
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 a SeqExpressionSet, which contains counts, labels for the samples and GC content/length
counts = newSeqExpressionSet(counts = exprs,
                             phenoData = AnnotatedDataFrame(data.frame(conditions = factor(c("mut","mut
                                                                       row.names = colnames(exprs))),
                             featureData = data.frame(gc = yeastGC[sub],
                                                      length = yeastLength[sub]))

## FINISH QUESTION 2 PART A CODE ##


## START QUESTION 2 PART B CODE ##
```

```r
# Making a boxplot for the counts.
EDASeq::boxplot(counts)

# Making a mean variance plot.
EDASeq::meanVarPlot(counts, log = TRUE)


# Assessing bias by GC content using a bias plot.
EDASeq::biasPlot(counts,"gc",log = T,ylab = "log(gene counts)")


# Assessing bias by length using a bias plot
EDASeq::biasPlot(counts,"length",log = T, ylab = "log(gene counts)")

## FINISH QUESTION 2 PART B CODE ##


## START QUESTION 2 PART C CODE ##

# Normalizing GC content using different methods.

# Using loess method
loess_within <- withinLaneNormalization(counts,"gc",which = "loess")

EDASeq::biasPlot(loess_within,"gc",log = T,ylim = c(2,4),ylab = "log(gene counts)")



# Using median method.
median_within <- withinLaneNormalization(counts,"gc",which = "median")

EDASeq::biasPlot(median_within,"gc",log = T, ylim = c(2,4))


# Using upper method.
upper_within <- withinLaneNormalization(counts,"gc",which = "upper")

EDASeq::biasPlot(upper_within,"gc",log = T, ylim = c(2,4))


# Using full method.
full_within <- withinLaneNormalization(counts,"gc",which = "full")

EDASeq::biasPlot(full_within,"gc",log = T)

## FINISH QUESTION 2 PART C CODE ##


## START QUESTION 2 PART D CODE ##

# Using different between methods on the within loess method.
loess_between_median <- betweenLaneNormalization(loess_within,which = "median")
loess_between_upper <- betweenLaneNormalization(loess_within,which = "upper")
```

```r
loess_between_full <- betweenLaneNormalization(loess_within,which = "full")

par(mfrow = c(1,2),mar = c(2,2,2,2))

EDASeq::boxplot(loess_within, main = "None")
EDASeq::boxplot(loess_between_median, main = "Median")
EDASeq::boxplot(loess_between_upper, main = "Upper")
EDASeq::boxplot(loess_between_full, main = "Full")

## FINISH QUESTION 2 PART D CODE ##

### FINISH QUESTION 2 CODE ###
```