12-2018

Statistical Methods to Account for Gene-Level Covariates in Normalization of High-Dimensional Read-Count Data

Lauren Holt Lenz
Utah State University

STATISTICAL METHODS TO ACCOUNT FOR GENE-LEVEL COVARIATES IN NORMALIZATION OF HIGH-DIMENSIONAL READ-COUNT DATA

by

Lauren Holt Lenz

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTERS OF SCIENCE

in

Statistics

Approved:

John Stevens, Ph.D. Major Professor	Christopher Corcoran, Ph.D. Committee Member
Adele Cutler, Ph.D. Committee Member	Laurens H. Smith, Ph.D. Interim Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

Statistical Methods to Account for Gene-Level Covariates in Normalization of High-Dimensional Read-Count Data

by

Lauren Holt Lenz, Master of Science

Utah State University, 2018

Major Professor: Dr. John R. Stevens
Department: Mathematics and Statistics

Normalization of RNA-Seq read-count data is a necessary pre-processing step in order to account for differences in read-count values due to non-expression related variables. It is common in recent RNA-Seq normalization methods to also account for gene-level covariates, namely gene length in base pairs and GC-content. Here a colorectal cancer RNA-Seq read-count data set comprised of 30,220 genes and 378 samples is examined. Two of the normalization methods that account for gene length and GC-content, CQN and EDASeq, are extended to account for protein coding status as a third gene-level covariate. The binary nature of protein coding status results in unique computation issues. The results of using the normalized read counts from CQN, EDASeq, and four new normalization methods are used for differential expression analysis via the nonparametric Wilcoxon Rank-Sum Test as well as the `lme4` pipeline that produces per-gene models based on a negative binomial distribution. The resulting differential expression results are compared for two genes of interest in colorectal cancer, APC and CTNNB1, both of the WNT signaling pathway.

(105 pages)

PUBLIC ABSTRACT

Statistical Methods to Account for Gene-Level Covariates in Normalization of High-Dimensional Read-Count Data

Lauren Holt Lenz

The goal of genetic-based cancer research is often to identify which genes behave differently in cancerous and healthy tissue. This difference in behavior, referred to as differential expression, may lead researchers to more targeted preventative care and treatment. One way to measure the expression of genes is though a process called RNA-Seq, that takes physical tissue samples and maps gene products and fragments in the sample back to the gene that created it, resulting in a large read-count matrix with genes in the rows and a column for each sample. The read-counts for tumor and normal samples are then compared in a process called differential expression analysis. However, normalization of these read-counts is a necessary pre-processing step, in order to account for differences in the read-count values due to non-expression related variables. It is common in recent RNA-Seq normalization methods to also account for gene-level covariates, namely gene length in base pairs and GC-content, the proportion of bases in the gene that are Guanine and Cytosine.

Here a colorectal cancer RNA-Seq read-count data set comprised of 30,220 genes and 378 samples is examined. Two of the normalization methods that account for gene length and GC-content, CQN and EDASeq, are extended to account for protein coding status as a third gene-level covariate. The binary nature of protein coding status results in unique computation issues. The results of using the normalized read counts from CQN, EDASeq, and four new normalization methods are used for differential expression analysis via the nonparametric Wilcoxon Rank-

Sum Test as well as the `lme4` pipeline that produces per-gene models based on a negative binomial distribution. The resulting differential expression results are compared for two genes of interest in colorectal cancer, APC and CTNNB1, both of the WNT signaling pathway.

ACKNOWLEDGMENTS

This research was made possible by funding from NIH grant award number 1R01CA16383-01A1 and from the Utah State University Department of Mathematics and Statistics, as well as computing resources from the Center for High Performance Computing at the University of Utah. This project could not have been completed with out the guidance and support of Dr. John Stevens.

CONTENTS

Page

ABSTRACT ii

PUBLIC ABSTRACT iii

ACKNOWLEDGMENTS v

LIST OF TABLES viii

LIST OF FIGURES x

CHAPTER

1. INTRODUCTION 1
   - Motivating Example 1
   - Motivating Data Set 2
   - Covariate Data Set 3
     - Covariate Sources 4
     - Imputation of Covariate Values 6
   - Differential Expression Analysis Methods 12

2. EXISTING NORMALIZATION METHODS 15
   - CQN 15
     - Background 15
     - Application to Motivating Data Set 16
   - EDASeq 20
     - Background 20
     - Application to Motivating Data Set 20

3. NEW NORMALIZATION METHODS 25
   - Normalize Read-Counts Separately Based on Binary Covariate 25
     - CQN 26
     - EDASeq 29
     - Compare Differential Expression Analysis Results 33
   - Edit Existing Methods to Account for a Binary Covariate 37
     - CQN 38
     - EDASeq 41
     - Compare Differential Expression Analysis Results 50

4. COMPARISON OF RESULTS 55
   - CQN Normalization and Variations 55
   - EDASeq Normalization and Variations 57
   - Genes of Interest 61
     - APC 61
     - CTNNB1 62

5. CONCLUSION, RECOMMENDATIONS, AND FUTURE WORK . . 64

REFERENCES 66
APPENDICES 70

LIST OF TABLES

Table		Page
0.1	Distribution of mean read-count per gene	3
0.2	Summary of Covariates Pulled from Ensembl GRCh37	6
0.3	Wilcoxon Rank Sum Test results using CQN normalized read-counts	19
0.4	`lme4` results using CQN offsets	20
0.5	Wilcoxon Rank Sum Test results using EDASeq normalized read-counts	24
0.6	`lme4` results using EDASeq offsets	24
0.7	Wilcoxon Rank Sum Test results using CQN split on protein coding status	28
0.8	`lme4` results using CQN split on protein coding status	29
0.9	Wilcoxon Rank Sum Test results using EDASeq split on protein coding status normalized read-counts	32
0.10	`lme4` results using EDASeq split on protein coding status offsets	33
0.11	Wilcoxon Rank Sum Test results using CQN with jittered protein coding status normalized read-counts	41
0.12	`lme4` results using CQN with jittered protein coding status offsets	41
0.13	Wilcoxon Rank Sum Test results using EDASeq normalizing for GC-content then protein coding status	47
0.14	Wilcoxon Rank Sum Test results using EDASeq normalizing for protein coding status then GC-content	48
0.15	Comparing the calls of differential expression from the Wilcoxon Rank Sum test on all genes when using EDASeq offsets on three covariates	48
0.16	Comparing the calls of differential expression from the Wilcoxon Rank Sum test on filtered genes when using EDASeq offsets on three covariates	48

0.17 `lme4` results using EDASeq normalizing for GC-content then protein coding status 49

0.18 `lme4` results using EDASeq normalizing for protein coding status then GC-content 49

0.19 Comparing the calls of differential expression from the `lme4` pipeline on all genes when using EDASeq offsets on three covariates 50

0.20 Comparing the calls of differential expression from the `lme4` pipeline on filtered genes when using EDASeq offsets on three covariates . . . 50

LIST OF FIGURES

Figure		Page
0.1	Distribution of mean read-count per-gene	3
0.2	Distributions of Covariates Pulled from Ensembl GRCh37	5
0.3	Example of imputation robustness test - true vs. imputed values	7
0.4	Example of imputation and linear adjustment robustness test - GC-content	8
0.5	Example of imputation and linear adjustment robustness test - gene length	9
0.6	Comparison of imputed and adjusted covariate values - GC-content	10
0.7	Comparison of imputed and adjusted covariate values - gene length	11
0.8	Comparison of imputed and adjusted covariate values - log(gene length)	12
0.9	Distribution of CQN offsets when normalizing for GC-content	16
0.10	Systematic effects for CQN with all genes and GC-content	17
0.11	Systematic effects for CQN with filtered genes and GC-content	17
0.12	Distribution of EDASeq offsets when normalizing for GC-content	21
0.13	Read-count mean variance plot for over-dispersion	21
0.14	Comparison of lowess regression on raw read-counts and normalized read-counts - All Genes	22
0.15	Comparison of lowess regression on raw read-counts and normalized read-counts - Filtered Genes	23
0.16	Distribution of CQN offsets when split on protein coding status prior to normalizing for GC-content	26
0.17	Systematic effects for CQN with all genes split on protein coding status	27

0.18 Systematic effects for CQN with filtered genes split on protein coding status . . . 28

0.19 Distribution of EDASeq offsets when split on protein coding status prior to normalizing for GC-content . . . 30

0.20 Comparison of lowess regression on raw read-counts and normalized read-counts - All Genes Split on Protein Coding Status . . . 31

0.21 Comparison of lowess regression on raw read-counts and normalized read-counts - Filtered Genes Split on Protein Coding Status . . . 32

0.22 Comparison of results using CQN with GC-content and CQN split on protein coding status . . . 34

0.23 Comparison of results using EDASeq with GC-content and EDASeq split on protein coding status . . . 35

0.24 Comparing Mean Squared Differences from `lme4` log_2 fold change values . . . 37

0.25 Distribution of CQN offsets when normalizing with jittered protein coding status . . . 39

0.26 Systematic effects for CQN with all genes and jittered protein coding status . . . 40

0.27 Systematic effects for CQN with filtered genes and jittered protein coding status . . . 40

0.28 Distribution of EDASeq offsets when normalizing for GC-content then protein coding status . . . 42

0.29 Distribution of EDASeq offsets when normalizing for protein coding status then GC-content . . . 42

0.30 Comparison of lowess regression on all genes raw read-counts and read-counts normalized with GC-content then protein coding status . . . 44

0.31 Comparison of lowess regression on all genes raw read-counts and read-counts normalized with protein coding status then GC-content . . . 45

0.32 Comparison of lowess regression on filtered genes' raw read-counts and read-counts normalized with GC-content then protein coding status . . . 46

0.33 Comparison of lowess regression on filtered genes raw read-counts and read-counts normalized with protein coding status then GC-content 47

0.34 Comparing the log_2 fold changes from the `lme4` pipeline when using EDASeq offsets on three covariates 49

0.35 Comparison of results using CQN with GC-content and CQN with jittered protein coding status 51

0.36 Comparison of results using EDASeq with GC-content and EDASeq with GC-content, then protein coding status 53

0.37 Comparison of results using EDASeq with GC-content and EDASeq with protein coding status, then GC-content 54

0.38 Comparison of `lme4` results using CQN variations 56

0.39 Comparison of Wilcoxon Rank Sum Test results using CQN variations 57

0.40 Comparison of `lme4` results using EDASeq variations 59

0.41 Comparison of Wilcoxon Rank Sum Test results using EDASeq variations 60

0.42 Comparison of All Test Results for APC 62

0.43 Comparison of All Test Results for CTNNB1 63

0.44 Comparison of log(gene length) and GC-content values for all genes . 65

0.45 Comparison of log(gene length) and GC-content values for filtered genes 65

CHAPTER 1

INTRODUCTION

RNA-Seq technology takes mRNA from a biological sample and maps the mRNA present to a reference genome ("Illumina: RNA Sequencing Data Analysis Solutions" San Deigo, California, Wang, Gerstein, and Snyder (2009)). The resulting data consists of a count of mRNA fragments ("reads") whose sequences map to each gene in each sample. These read-counts represent measurements of the expression levels of the genome's genes in each biological sample. Traditionally, researchers are interested in identifying differentially expressed genes whose expression levels change systematically between groups, such as tumor vs. normal in a cancer study.

When using read-count data to identify differentially expressed genes, normalization is a necessary pre-processing step. Normalization accounts for differences that cause some read-counts to be higher than others due to non-expression related variables such as differences in the size of biological samples, the length of each gene, and machine calibrations. Classical normalization approaches deal with reads per kilo-million, or RPKM, which normalizes by taking into account the total number of reads each sample has across all genes, commonly referred to as the sequencing depth.

Motivating Example

RNA-Seq was run using the Illumina pipeline ("Illumina: RNA Sequencing Data Analysis Solutions" San Deigo, California) on 378 samples from 209 individuals. These 378 samples consist of 187 normal (non-tumor) samples and 191 tumor samples, of which there are 169 pairs and 40 unpaired samples. The RNA-Seq pro-

cess resulted in a read-count for each sample at each of the 30,220 genes, of which 17,462 (57.78%) are protein coding and 12,758 (42.22%) are non-protein coding (Slattery et al. 2017).

The original team examining this read-count data set at the University of Utah used the `DESeq2` pipeline (Love, Huber, and Anders 2014) and the Wilcoxon Rank Sum test (Hothorn et al. 2008) for differential expression analysis between tumor and normal samples; however, some genes did not show the expression differences expected. In colorectal cancer the WNT signaling pathway is studied extensively and fairly well understood (Suzuki et al. 2004, Nagase and Nakamura (1993), Segditsas and Tomlinson (2006)). The WNT signaling pathway gene APC is believed to be down-regulated in colorectal cancer (Nagase and Nakamura 1993), and the gene CTNNB1 is expected to be up-regulated in colorectal cancer (Suzuki et al. 2004). The Wilcoxon Rank Sum test run with standard RPKM normalization on this data set showed that APC was not significantly differentially expressed. This "flip" in expression caused the original researchers to question the validity of the other differential expression analysis results and raises this project's research question: Does accounting for gene-level covariates in pre-processing normalization of high-dimensional RNA-Seq read-count data improve the accuracy of differential expression analysis?

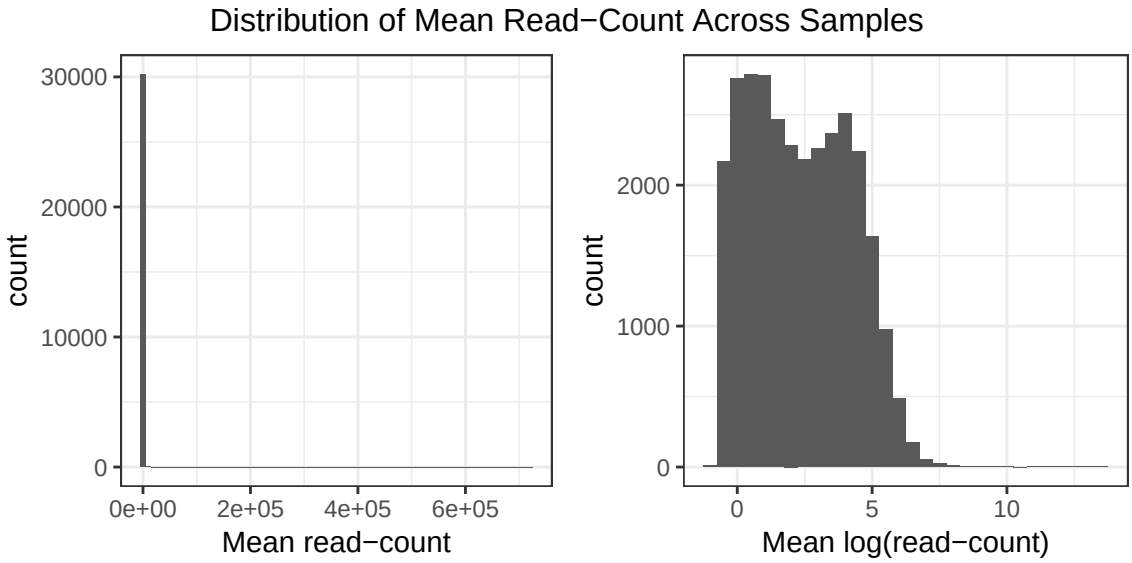
Motivating Data Set

The read-counts in this data set have a very large spread. Figure 0.1 shows the mean read-count across all samples for each gene, on the true and log scales. The majority of genes have a mean read count across samples of less than 1,000, and only 77 of the 30,220 genes have mean read counts higher than 1,500. Table 0.1 shows summary statistics for the mean read-count across samples for each gene. Notice the huge difference in the mean and median of the genes' mean read-counts.

Table 0.1: Distribution of mean read-count per gene

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum
0.5	2.0	9.2	193.4	47.8	719969.7

Figure 0.1: Distribution of mean read-count per-gene



A filtered data set was also produced by filtering out the "non-expressed" genes. A mean read-count threshold of 10 was set at the suggestion of Risso et al. (2011), and all genes with a mean-read count across all samples of less then 10 were labeled as "non-expressed" and were removed from the data set, leaving 14,715 genes, 12,618 of which are protein coding (85.75%). All covariate calculations explained below were performed on all genes, with this filter applied before normalization.

Covariate Data Set

Gene-level covariates previously used in read-count normalization are gene length in bases, and GC-content, the proportion of bases in a gene sequence that are Guanine and Cytosine as opposed to Adenine and Thymine (Hansen, Irizarry, and Wu 2012). Because RNA-Seq maps gene fragments and gene products back to

the gene that produced them, longer genes will have higher read-counts and expression levels, leading to more significance in differential expression analysis (Risso et al. 2011). When designing a new normalization method, Risso et al. (2011) stated "GC-rich and GC-poor fragments tend to be underrepresented in RNA-Seq" (Risso et al. 2011), and it has been shown that bias related to GC-content leads to false positives in downstream differential expression analysis (Hansen, Irizarry, and Wu 2012).

Ideally, researchers would be aware of normalization methods that account for gene-level covariates while designing their experiment, allowing the sequences used in the initial RNA-Seq process to be used to calculate both GC-content and gene length. However, in cases like this motivating example where the sequences used in the RNA-Seq process that provided the read-counts are unavailable, the researcher attempting to use a method that requires per-gene covariate information must calculate covariates from suitable gene sequences that are as close to what were used in RNA-Seq as possible.

Covariate Sources

Several options for pulling gene sequences are available for free online, including the University of California Santa Cruz Genome Browser (Kent et al. 2002), and the Ensembl Database website (Zerbino et al. 2017). The company that produced the materials used in the original RNA-Seq process may also provide reference sequences online.

Gene sequences used for this data set's RNA-Seq process were pulled from the GRCh37 build of the human genome data base via the UCSC website, and aligned using novoalign v2.08.01. (Slattery et al. 2017). Since those sequences were not available for my use, I attempted to pull the sequences from the data base. The UCSC Gene Browser allows a user to pull transcript sequences corresponding to an

Ensembl gene ID, but not entire gene sequences. Genes are typically made up of several transcripts, and some may overlap or leave gaps in the gene sequence. Because I am not confident in my ability to piece together a gene sequence from transcripts, I instead pulled gene sequences from the Ensembl database, build GRCh37 (Zerbino et al. 2017). While these sequences are not the exact ones used in the original RNA-Seq process, I expect that they are reasonably close, as they are from the same data base build that was accessed by the original researchers (Slattery et al. 2017). Graphical displays of the distributions of gene lengths and GC-content values are shown in Figure 0.2, with summaries of the distributions shown in Table 0.2.

Figure 0.2: Distributions of Covariates Pulled from Ensembl GRCh37

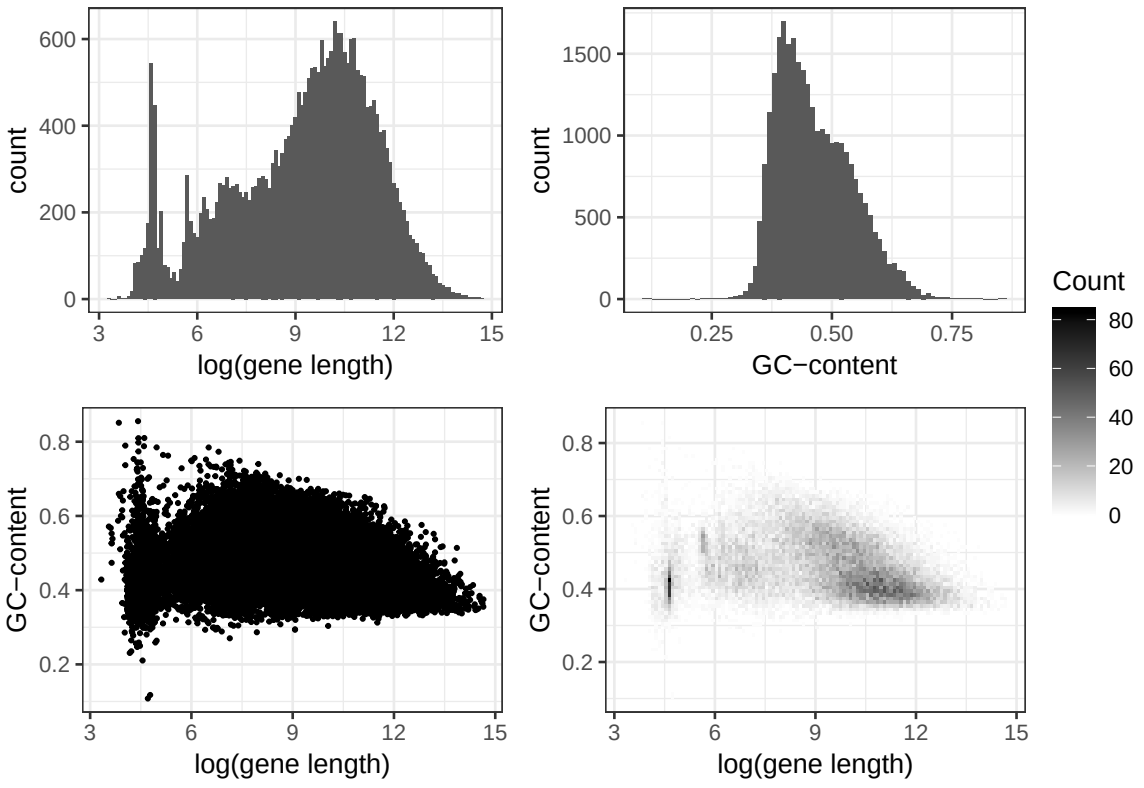


Table 0.2: Summary of Covariates Pulled from Ensembl GRCh37

Gene Length		**GC-Content**	
Minimum	28	**Minimum**	0.1081
1st Quartile	2,241	**1st Quartile**	0.4023
Median	14,964	**Median**	0.4500
Mean	51,276	**Mean**	0.4640
3rd Quartile	50,671	**3rd Quartile**	0.5179
Maximum	2,304,638	**Maximum**	0.8554

Even when pulling from the same genome database build as the original researchers, 1,885 of the 30,220 (6.24%) Ensembl gene IDs were not in the database, resulting in missing gene lengths and GC-content values for those genes. Of the 1,885 genes not in the database, 1,852 are non-protein coding genes (98.25%), and 1,365 (72.41%) have mean read-count values less than 10, and would be classified as non-expressed.

Since the read-count data set had no missing expression values for these 1,885 genes, it would be unwise to remove them from the data set and not include them in differential expression analysis. However, in order to include these genes in methods that consider gene-level covariates, the missing gene length and GC-content values must be imputed.

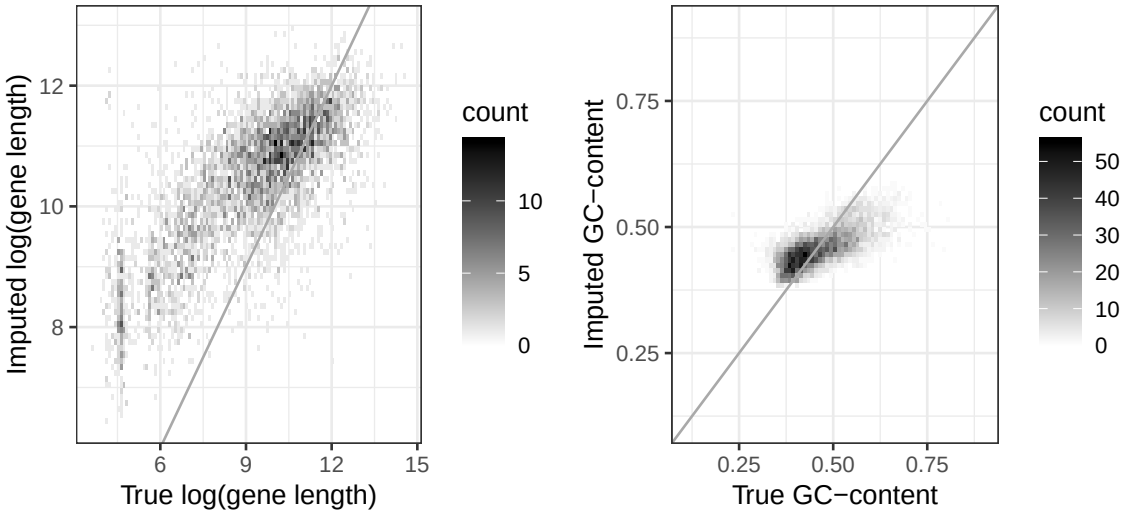
Imputation of Covariate Values

The package `missForest` (Stekhoven 2013) was used to impute missing gene length and GC-content values. Prior to imputing the 1,885 missing values, the accuracy of `missForest` imputation needed to be established. To do so, a random sample of 5,000 genes was taken from the 28,335 known genes, their covariates were removed from the data set, and then the 5,000 "missing" values were imputed based on the gene length and GC-content values for the 23,335 remaining known genes, and read count values for all 28,335 genes.

The process described above was repeated for five replicates, in order to estab-

lish that the accuracy of `missForest` was not a result of the 5,000 genes randomly selected. Imputed and true gene length and GC-content values were plotted for a visual measure of the accuracy and robustness of `missForest` imputation. All five replicates showed very similar patterns, Figure 0.3 shows one of these robustness test replicates.

Figure 0.3: Example of imputation robustness test - true vs. imputed values



Note in Figure 0.3, the log-scale gene length plots show some bi-modality in the true gene length values that is not seen in the imputed values. Also note how both the imputed gene length values and the imputed GC-content values appear to follow a fairly linear trend, but the slopes of those linear trends are not one. For this reason I decided to apply a linear model adjustment to each covariate to improve imputation accuracy.

To calculate and test linear models, I took each of the five replicates mentioned above and randomly split them into two groups, training and testing. Each of the five training sets were use to create a linear model where the imputed value depends on the true value, and then the corresponding test set were used to visualize the accuracy of the linear models. Figure 0.4 shows true vs. imputed GC-

content and true vs. imputed-then-adjusted GC-content. The application of the linear model adjustment results in adjusted values that are much closer to the true values on average.

Next the linear models themselves were examined, and because all five models are very similar, the mean of the intercept and the GC-content coefficient were used to create a single linear model adjustment to apply to the imputed GC-content values of the 1,885 missing genes.

The same process was followed for a linear model adjustment for gene lengths on the log scale. As seen in Figure 0.5, the linear model adjustment is less beneficial for log-scale gene lengths than for GC-content, but it is an improvement, so the linear model adjustment was applied to the 1,885 imputed values.

Figure 0.4: Example of imputation and linear adjustment robustness test - GC-content

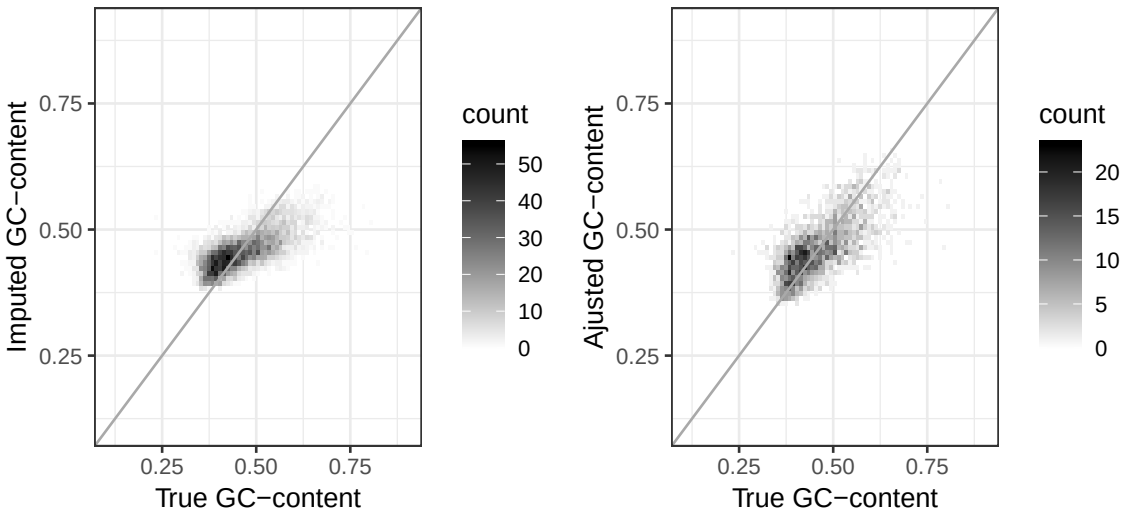
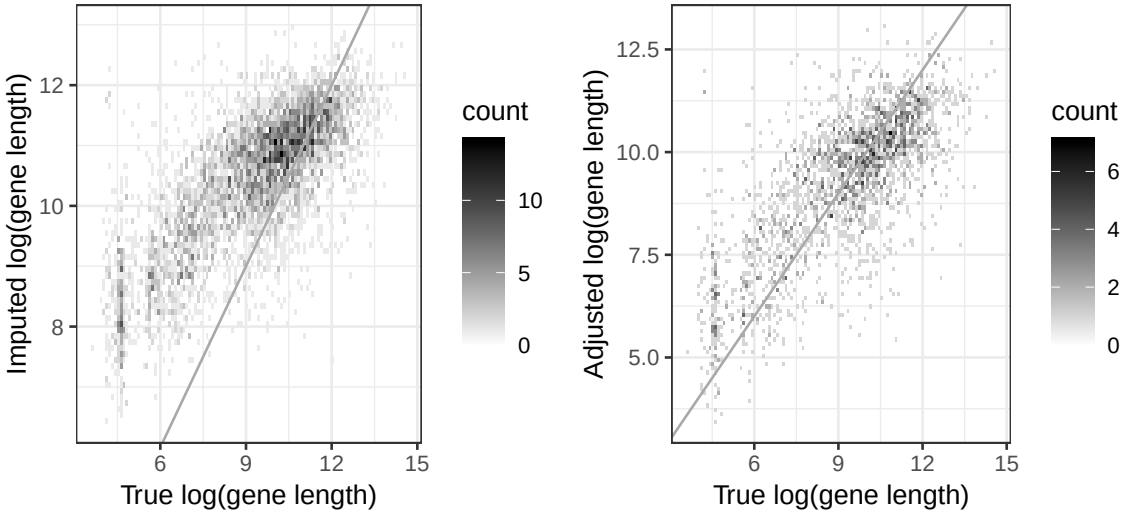


Figure 0.5: Example of imputation and linear adjustment robustness test - gene length



After establishing the validity and robustness of `missForest` imputation and a linear model was constructed to increase accuracy, the missing gene length and GC-content values were imputed using the covariates from the 28,335 known genes and the read counts for all 30,220 genes. The linear model adjustments for gene length and GC-content were applied to the 1,885 imputed values, resulting in the covariate data set used throughout the rest of this thesis. Distributions of known, imputed, and adjusted covariates are seen in Figures 0.6, 0.7, and 0.8.

Figure 0.6: Comparison of imputed and adjusted covariate values - GC-content

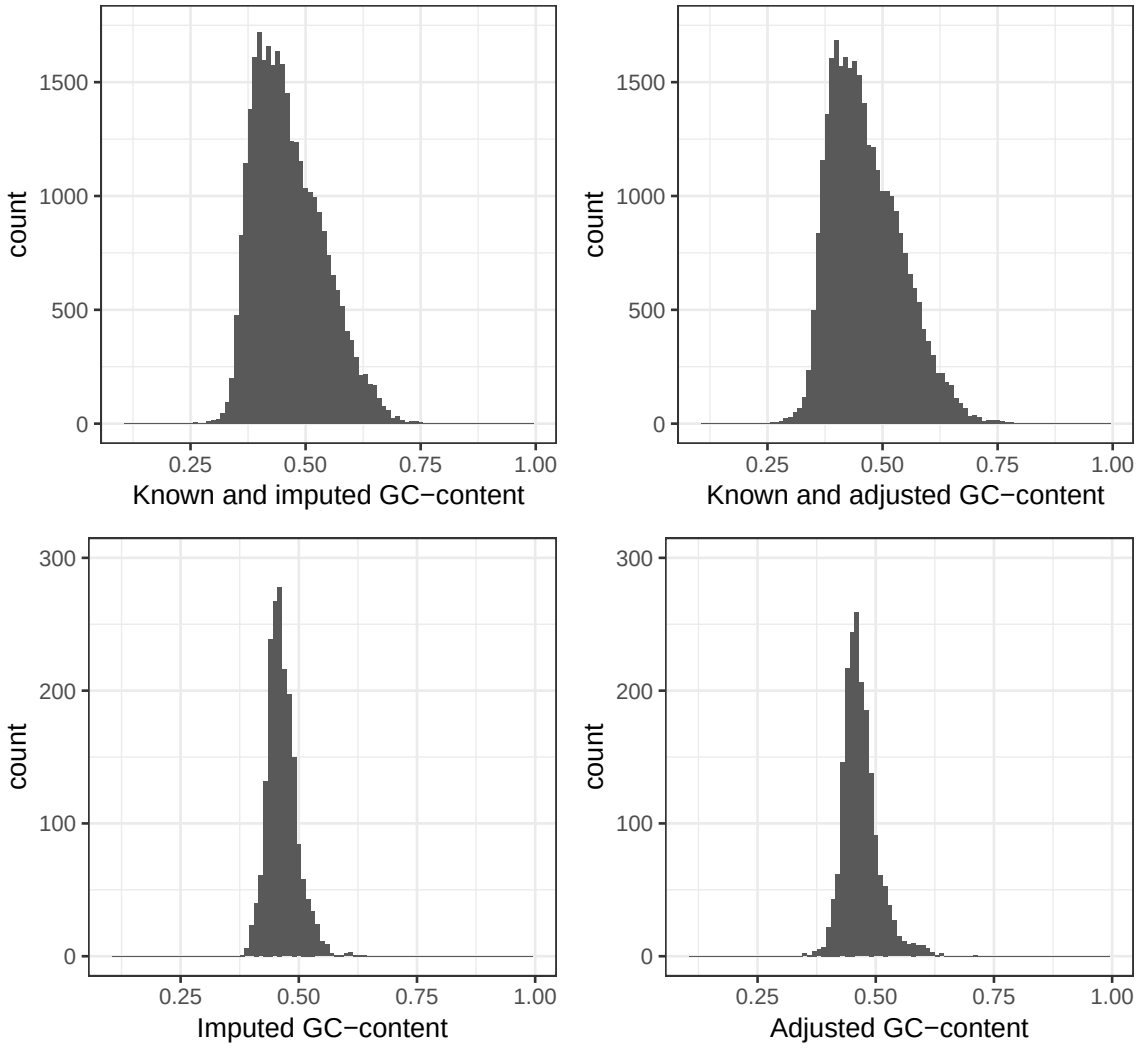


Figure 0.7: Comparison of imputed and adjusted covariate values - gene length

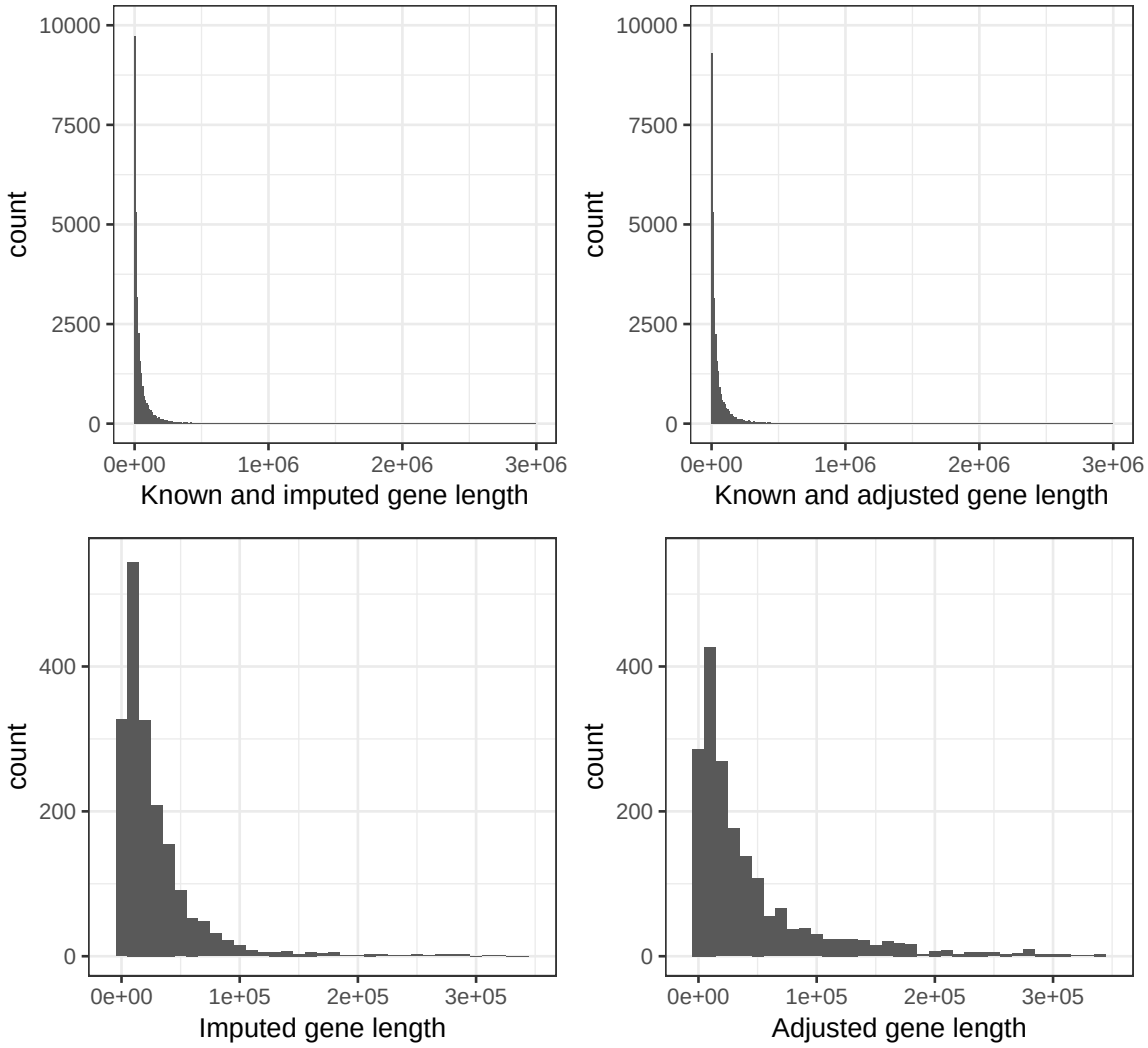
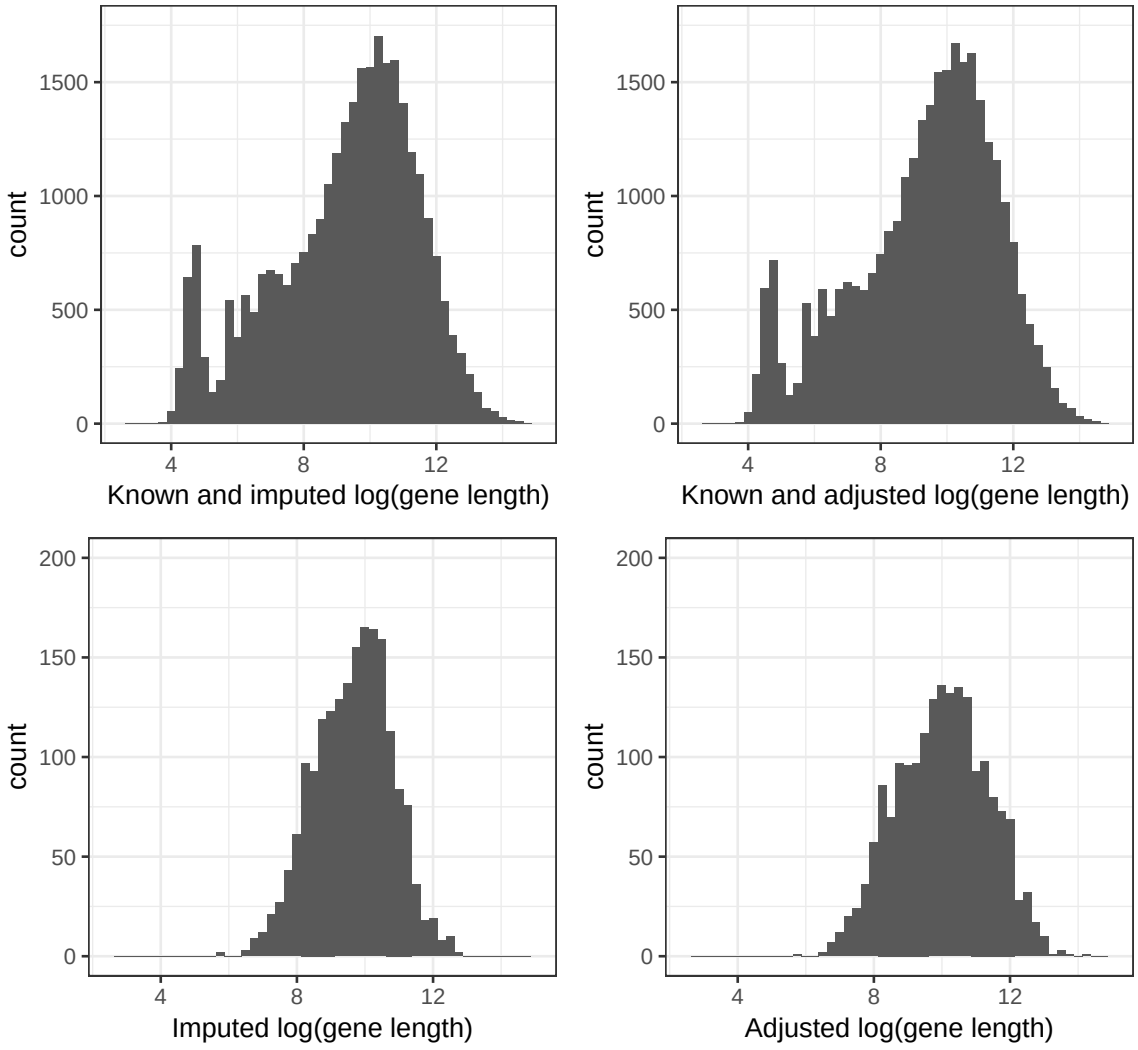


Figure 0.8: Comparison of imputed and adjusted covariate values - log(gene length)



Differential Expression Analysis Methods

Several methods for determining the differential expression of genes are available, including methods designed especially for RNA-Seq data. The University of Utah researchers used the `DESeq2` pipeline (Love, Huber, and Anders 2014) and the Wilcoxon Rank Sum test, but this project will focus on the use of the `lme4` pipeline and the Wilcoxon Rank Sum test from the R package `coin` (Hothorn et al. 2008). The `lme4` pipeline (Bates et al. 2015) is most appropriate for this data set because it accounts for both the sample type (tumor or normal) fixed effect, as well as the

sample ID random effect, when producing per-gene models. Other, computationally faster, differential expression analysis methods including `DESeq2` (Love, Huber, and Anders 2014) and `edgeR` (Robinson, McCarthy, and Smyth 2010) account for fixed effects but have no way to account for the sample ID random effect.

Let N_i be the total number of mRNA fragments in sample i, p_i be the probability that a fragment maps to a given gene in sample i, and R_i be the number of fragments in sample i that map back to the given gene. Then the `lme4` pipeline allows $R_i \sim NegativeBinomial(\mu_i, \sigma_i^2)$ where $\mu_i = N_i p_i$ so $\log\left(E[R_i]\right) = \log N_i + \log p_i$. The default "offset" ($\log N_i$) in this model can be replaced by a normalizing offset value, and the link function ($\log p_i$) is set equal to a linear model such as $\log p_i = \mu +$ (effect of sample i's treatment group) + (effect of sample i's subject).

All of the normalization methods discussed in Chapters 2 and 3 return both offsets and normalized read-counts, which can then be passed to differential expression analysis methods. There is some debate on whether offsets or normalized read-counts are better to use overall (Risso et al. 2011), so in this project I followed the published vignettes for the normalization methods. The Wilcoxon Rank Sum test takes normalized-read counts, but the `lme4` pipeline takes raw read-counts and the offset values.

The documentation for `lme4` describes the offset as a way to "specify an *a priori* known component to be included in the linear predictor during [the] fitting [of per-gene models]", which essentially means that the offset values are added into the linear mixed-effect model to adjust the inputted raw read-counts for normalization (Bates et al. 2015). Mathematically, this is seen as follows, let Y represent the response (gene expression values), n is the dimension of the response vector, $\mathbf{W}$ is a diagonal matrix of known prior weights, β is a p-dimensional coefficient vector, $\mathbf{X}$ is an $n \times p$ model matrix, o is a vector of known prior offset terms, and the parameters of the model are the coefficients β and the scale parameter σ. Then

$Y \sim N(\mathbf{X}\beta + o, \sigma^2\mathbf{W}^{-1})$ (Bates et al. 2015).

CHAPTER 2
EXISTING NORMALIZATION METHODS

In this thesis project I will be using two normalization methods that use gene-level covariates: Conditional Quantile Normalization (CQN) (Hansen, Irizarry, and Wu 2012) and Exploratory Data Analysis and Normalization for RNA-Seq (EDASeq) (Risso et al. 2011). Both methods are published as R packages by Bioconductor (Huber et al. 2015) and use gene length and GC-content as gene-level covariates. The results of these normalization methods include normalized read-counts that are used in the non-parametric Wilcoxon Rank Sum Test (Hothorn et al. 2008), and offsets that are used in the `lme4` pipeline to produce per-gene models (Bates et al. 2015).

CQN

Background

Hansen, Irizarry, and Wu (2012) found that GC-content has a strong sample-specific effect on RNA-Seq read-counts that can lead to false positives in differential expression analysis. This motivated the new conditional quantile normalization algorithm that removes systematic bias introduced by covariates, as well as global distortions caused by differences in sequencing depths. As described by Risso et al. (2011),

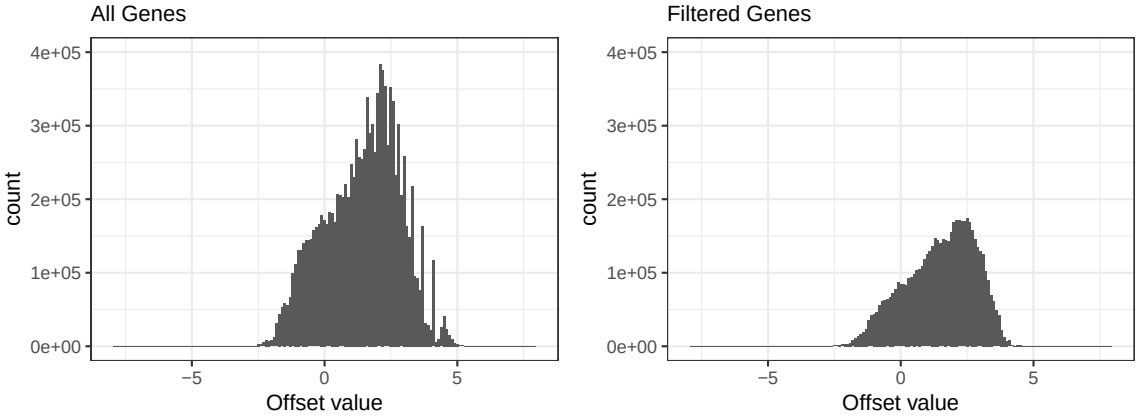
> "[The CQN] procedure, which combines both within and between lane [or sample] normalization and is based on a Poisson model for read counts. Lane-specific systematic biases, such as GC-content and length effects, are incorporated as smooth functions using natural cubic splines

and estimated using robust quantile regression. In order to account for distributional differences between lanes, a full-quantile normalization procedure is adopted, in the spirit of that considered in Bullard et al. (2010). The main advantage of this approach is that it is lane-specific, i.e., it works independently in each lane, aiming at removing the bias rather than equalizing it across lanes. Modeling simultaneously GC-content and length (and in principle other sources of bias) leads to a flexible normalization method." (Risso et al. 2011)

Application to Motivating Data Set

Following the published vignette for the `cqn` package (Hansen, Irizarry, and Wu 2012), CQN was run on all genes, as well as the filtered subset. The resulting object of class "cqn" contains offsets that can be used to calculate normalized read-count values. The distribution of those offsets is visualized in Figure 0.9. The full and filtered data sets produce offsets with the same general distribution, a fairly unimodal, slightly left-skewed, distribution.

Figure 0.9: Distribution of CQN offsets when normalizing for GC-content



The object of class "cqn" can be passed to the function *cqnplot* which "plots [the] systematic effects" (Hansen, Irizarry, and Wu 2012). The system-

atic effects are (1) whatever covariate passed to *cqn* and (2) the gene lengths passed to *cqn*. The lines on the resulting plots show estimated beta-spline dependence of counts on the systematic effect, with the systematic effect on the x-axis (with the knots used to fit the beta splines shown as additional ticks), and the QR fit on the y-axis (Love 2016). The cqnplots for the full data set are seen in Figure 0.10, and the cqnplots for the filtered data set are in Figure 0.11.

Figure 0.10: Systematic effects for CQN with all genes and GC-content

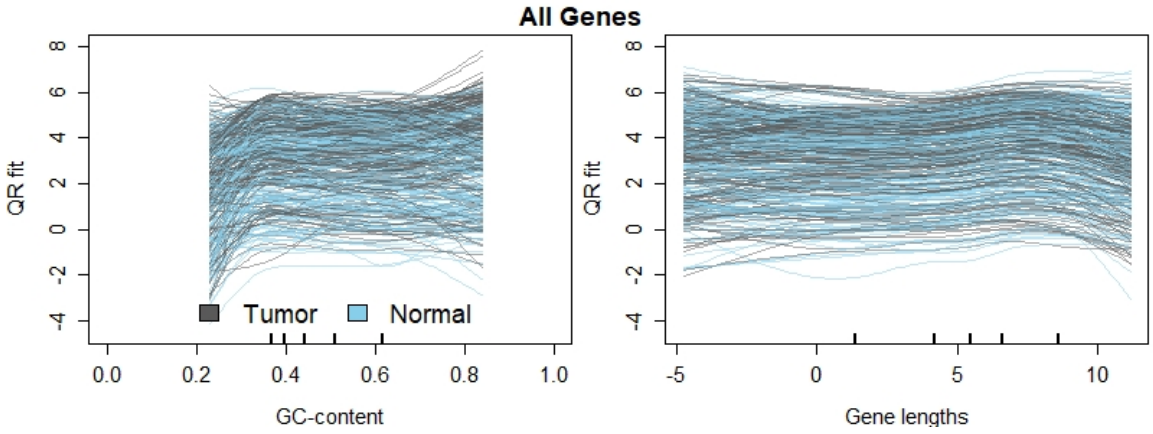
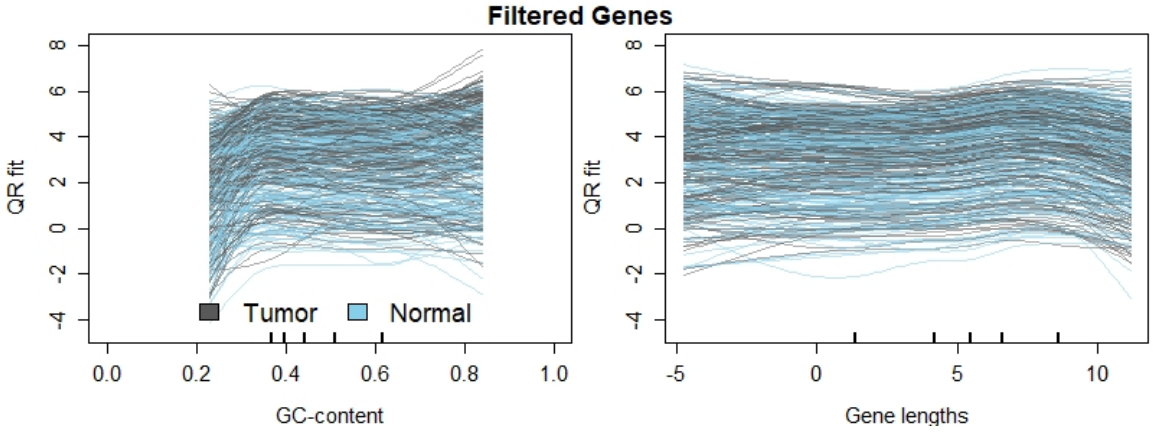


Figure 0.11: Systematic effects for CQN with filtered genes and GC-content



Although Figures 0.10 and 0.11 look to be identical, there are very slight variations in the splines due to the filtering of the read-counts matrix. CQN normalization does not require filtering out "non-expressed" genes (where EDASeq highly

recommends it), likely because the non-expressed genes have very little impact on the overall normalization of the read-counts, which is supported by the similarity of Figures 0.10 and 0.11. The spread of the splines in both the GC-content and gene lengths plots suggest that there is quite a bit of variability between samples in the estimated splines (Love 2016).

According to Love, a typical cqn plot for GC-content would have an "upside-down U" shape, indicating that the low and high GC-content fragments are under-represented (Love 2016). Figures 0.10 and 0.11 appear to have no distinct upside-down U shape in the GC-content splines, suggesting that the genes with more extreme GC-content are not as underrepresented as we would have expected. There is, however, an overall positive slope on the far left-hand side of the GC-content plot, representing small GC-content values, so there may be evidence that genes with low GC-content are actually underrepresented. Love also describes the typical pattern for the gene lengths splines to show more counts for longer genes (Love 2016). In the gene lengths plots in Figures 0.10 and 0.11, however, there is no overall positive slope, suggesting that there may not be more counts for longer genes in our data set.

Wilcoxon Rank Sum Test

The normalized read counts (on the log_2 scale) from CQN were used in the Wilcoxon Rank Sum Test function *wilcox_test* from the package `coin` (Hothorn et al. 2008). The log_2 scale was used so that the resulting object would contain an "estimate" value that estimates the log_2 fold change comparing tumor samples to normal samples (i.e. a negative estimate would mean that gene is down-regulated in tumor samples, with significance depending on the FDR corrected p-value).

A combination of the resulting log_2 fold change values and the FDR corrected p-values (using FDR threshold of 0.05) were used to assign significance to each

gene. Results are summarized in Table 0.3.

Table 0.3: Wilcoxon Rank Sum Test results using CQN normalized read-counts

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	0 (0%)	7,738 (25.61%)	12,522 (41.44%)	9,960 (32.96%)
Filtered Genes: 14,715	0 (0%)	5,179 (35.20%)	3,559 (24.19%)	5,977 (40.62%)

Filtering the genes prior to normalization has a large effect on the proportion of genes identified as significant. Filtering out 15,505 "non-expressed" genes removed nearly 10,000 genes from the not-significantly-differentially-expressed category, resulting in a higher percentage of the remaining 14,715 genes being identified as significantly differentially expressed.

`lme4` Models

The `lme4` R package (Bates et al. 2015) available from CRAN produces per-gene models that account for fixed and random effects, while assuming read-counts follow a negative binomial distribution. The offsets from CQN normalization were passed into the `lme4` pipeline following the `lme4` vignette. `lme4` is fairly computationally intensive, so models were run using the University of Utah Center for High Performance Computing (Barton 2016). The output from the *glmer.nb* function in the `lme4` pipeline contains fixed-effect estimates, which are equivalent to log_2 fold change values, and raw p-values, from which FDR corrected p-values were calculated. As in the Wilcoxon Rank Sum Test, the sign of the log_2 fold change and the FDR corrected p-value were used to determine the significance and dysregulation (up or down) of each gene. Results are summarized in Table 0.4.

Table 0.4: `lme4` results using CQN offsets

lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	310 (1.03%)	7, 602 (25.16%)	10,540 (34.88%)	11,768 (38.94%)
Filtered Genes: 14,715	38 (0.265)	4,903 (33.32%)	3,245 (22.05%)	6,529 (44.37%)

Comparing the results from `lme4` (Table 0.4) and the Wilcoxon Rank Sum test (Table 0.3), `lme4` tends to identify more significantly up-regulated genes, in both the full and filtered data sets. This is likely because `lme4` accounts for variability due to a subject random effect, making the test for the tumor fixed effect more powerful due to there being less unexplained variability.

EDASeq

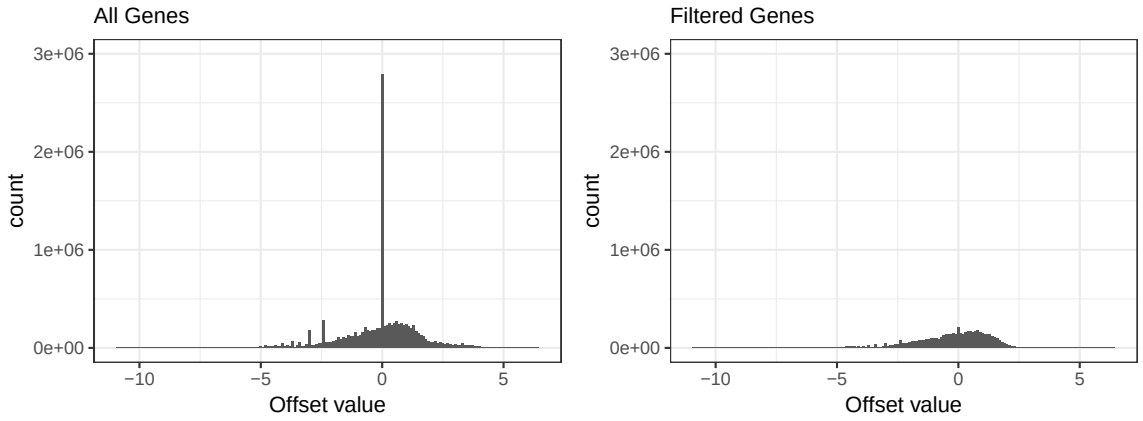
Background

In their original 2011 *BMC Bioinformatics* paper, Risso et al. (2011) break down sources of bias in RNA-Seq read counts into two main categories: within-lane gene-specific effects, including GC-content and gene length, and between-lane distributional differences, including sequencing depth. Risso et al. (2011) also states that normalizing by "scaling counts by gene length is not sufficient for removing [within-lane] bias", which is similar to what is done in the classic RPKM normalization method. Therefore by normalizing within-lane and then between-lane, EDASeq claims to lead to more accurate gene expression levels, "making statistical inference of differential expression less prone to false discoveries" (Risso et al. 2011).

Application to Motivating Data Set

Following the published vignette for the `EDASeq` package (Risso et al. 2011), EDASeq normalization was run on all genes, as well as the filtered subset. The distribution of the resulting offsets are shown in Figure 0.12.

Figure 0.12: Distribution of EDASeq offsets when normalizing for GC-content



The distribution of offsets produced by EDASeq normalization for the full data set has a very large spike at zero that is not seen in the filtered data set. This suggests that the "non-expressed" genes filtered out of the data set were likely "non-expressed" across both tumor and normal samples.

In the EDASeq package there is more of a focus on exploratory analysis than in the CQN package, including some useful diagnostic plots. The most useful diagnostic plot for this data set is the over-dispersion plot, seen in Figure 0.13.

Figure 0.13: Read-count mean variance plot for over-dispersion

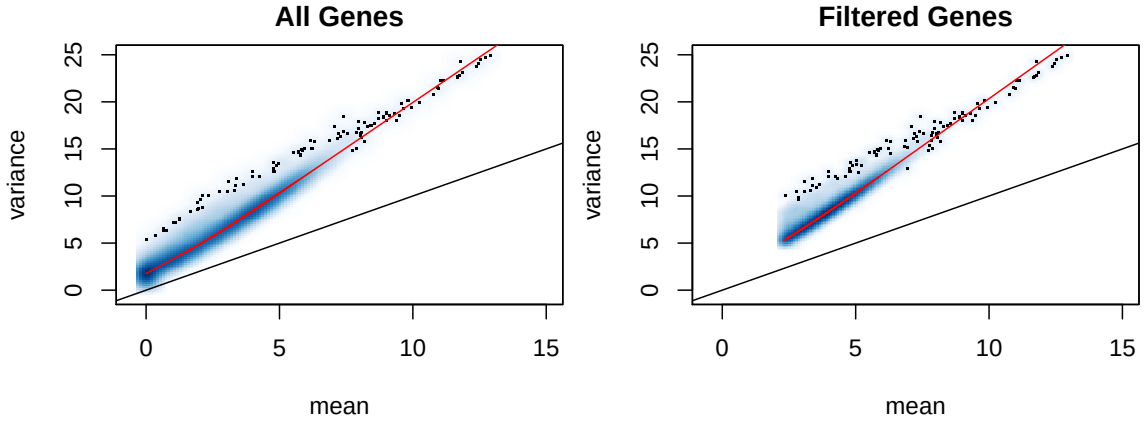
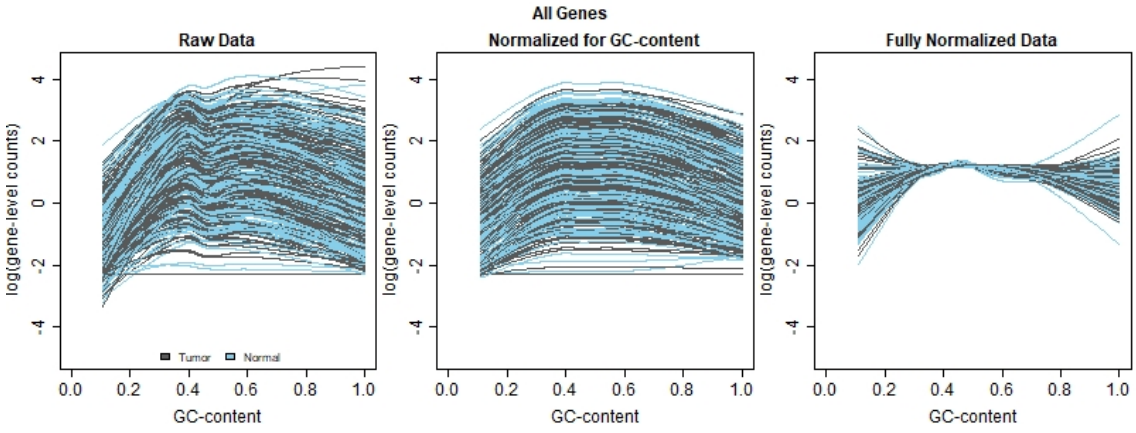


Figure 0.13 shows the mean and variance of read-counts within each sample. If the mean and variance were equal (following the black line of equality), then

a Poisson distribution would be appropriate for the data set. The red line shows a lowess fit for the mean-variance relationship, that in both the filtered and unfiltered data sets are significantly different than the black line of equality. This suggests that a Poisson distribution is inappropriate, and that a negative binomial distribution for the read-counts should be used. This is another reason why the `lme4` pipeline for differential expression analysis is most appropriate, as it allows the read-counts to follow a negative binomial distribution (Bates et al. 2015).

EDASeq also produces bias plots on normalized data. These bias plots (Figures 0.14 and 0.15) show lowess regression curves, one for each sample, with GC-content on the x-axis and log(gene-level counts) on the y-axis.

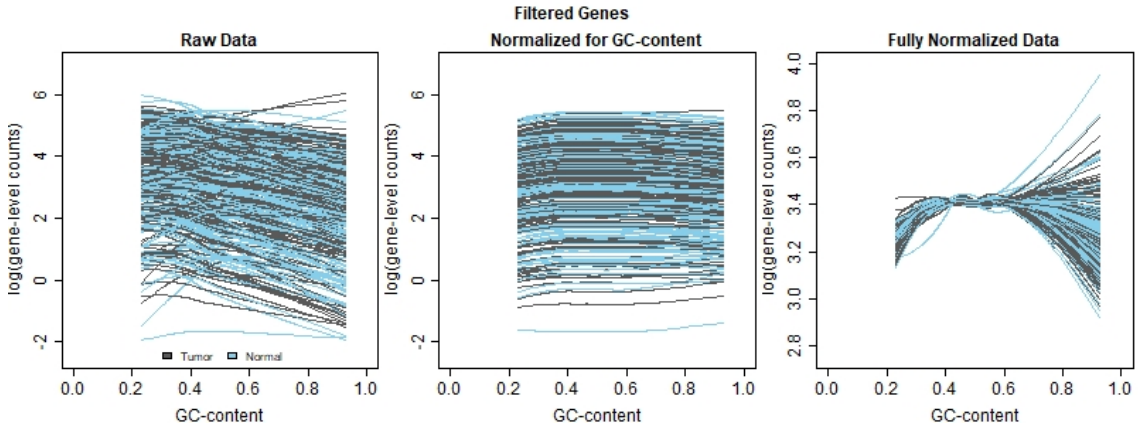
Figure 0.14: Comparison of lowess regression on raw read-counts and normalized read-counts - All Genes



The raw data plotted in Figure 0.14 shows quite a bit of spread, with more of the overall upside-down U shape anticipated in the typical cqn plot (Love 2016). The data normalized for GC-content smooths out some of the variation in the mid-level GC-content, and the dramatic change comes from accounting for sequencing depth, the between lane normalization. The fully normalized data plot is very compressed toward the center of the GC-content distribution, which matches the interpretation that genes with extreme GC-content values may be underrepresented in

read-counts, and genes with moderate GC-content values may be over-represented.

Figure 0.15: Comparison of lowess regression on raw read-counts and normalized read-counts - Filtered Genes



Notice the difference in scale between the first two and third plots of Figure 0.15, as well as the difference from the all genes plot (Figure 0.14) to the filtered genes plot (Figure 0.15). There is much less of an upside down U shape in the filtered raw data than in the complete raw data. Likely this means that many of the genes with extreme GC-content values and lower read-counts were filtered out. As in the all genes plot, normalizing for GC-content appears to remove some of the variation seen in the raw data. The fully normalized data plot has a very narrow range, even on the far high range of GC-content with the most vertical spread, which means EDASeq on filtered data has removed much of the variation in the distribution of read-counts across samples.

Wilcoxon Rank Sum Test

The Wilcoxon Rank Sum tests were run in the same way for EDASeq as for CQN (described above). Results of the Wilcoxon Rank Sum test run on EDASeq normalized read-counts are summarized in Table 0.5, with FDR controlled at 0.05.

Table 0.5: Wilcoxon Rank Sum Test results using EDASeq normalized read-counts

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	1 (0.003 %)	9,707 (32.12%)	11,447 (37.89%)	9,065 (30.00%)
Filtered Genes: 14,715	2 (0.014%)	5,352 (36.37%)	3,447 (23.43%)	5,914 (40.19 %)

It is interesting to note that the CQN normalized read-counts produced no NA values (failed tests), but here there was one failed test in the full data set, and two failed tests in the filtered data set (compare Tables 0.3 and 0.5). These failed tests were accompanied by the warning "The conditional covariance matrix has zero diagonal elements".

`lme4` Models

The `lme4` pipeline was handled the same way for EDASeq as for CQN (described above). Results of the per-gene models are summarized in Table 0.6.

Table 0.6: `lme4` results using EDASeq offsets

lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	544 (1.80%)	9,310 (30.81%)	11,033 (36.51%)	9,333 (30.88%)
Filtered Genes: 14,715	437 (2.97%)	5,111 (34.73%)	3,404 (23.13%)	5,736 (39.16%)

The percentage of failed models is slightly higher when using EDASeq offsets than when using CQN offsets for all variations of both normalization methods. Another point of interest is that filtering out non-expressed genes is explicitly recommended in the EDASeq vignette, but not in the CQN vignette, so here focus should be placed on the results of the filtered data set (Risso et al. 2011).

CHAPTER 3
NEW NORMALIZATION METHODS

Both CQN and EDASeq normalization methods only account for gene length and GC-content. While the original Hansen, Irizarry, and Wu (2012) paper introducing CQN states that the CQN model "permits the inclusion of other [covariates]: for example, mappability or more elaborate models of sequence effects", no published examples of using CQN with any covariates other than gene length and GC-content could be found. And, for both CQN and EDASeq, trying to run the pipeline as-is with gene length and protein coding status causes fatal computational errors to occur. For this reason I began looking for a solution to account for protein coding status, a binary covariate, as well as gene length and GC-content in normalization.

Normalize Read-Counts Separately Based on Binary Covariate

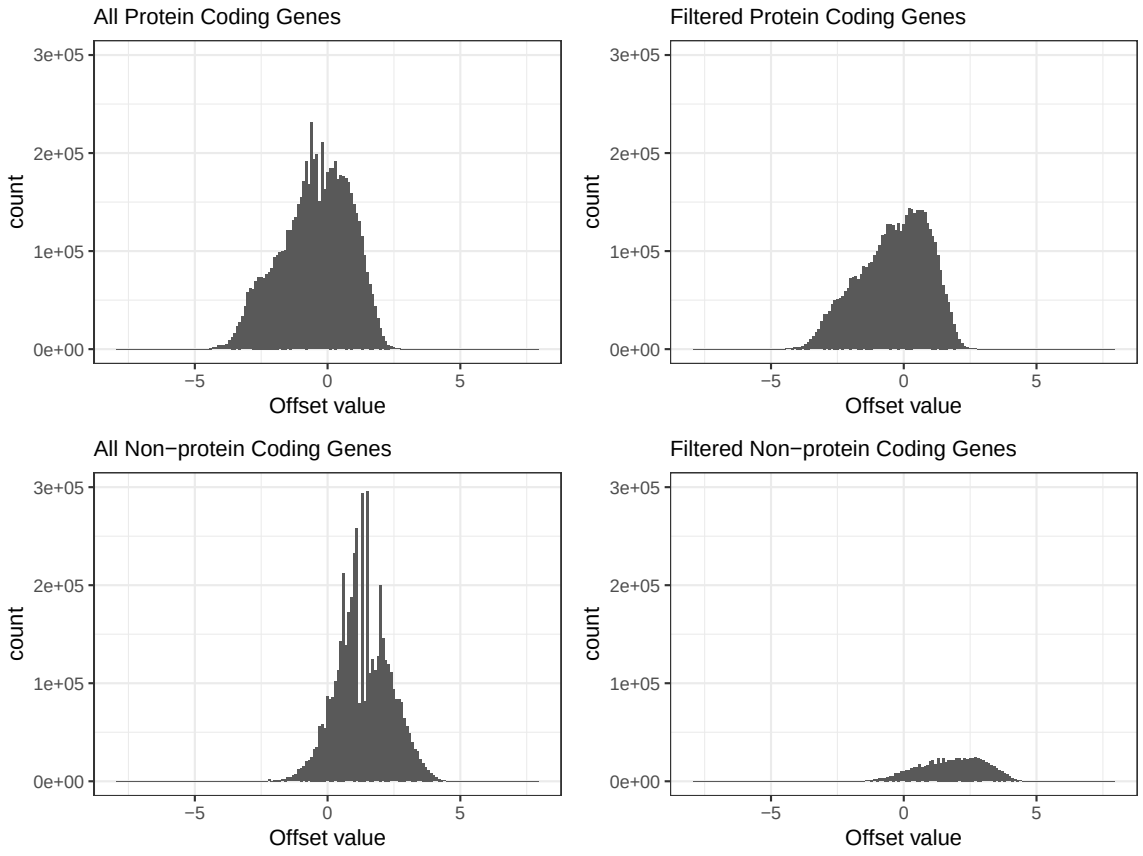
When the original issue of APC's unexpected expression was found by the University of Utah researchers (recall "Motivating Example" section in Chapter 1), it was suggested that perhaps only the protein coding genes should be examined for differential expression. This would mean that the non-protein coding genes being included in the analysis changed the distribution of read-counts enough to alter the results of the test. So, if the distributions of read-counts for protein and non-protein coding genes are so different, perhaps they should be normalized separately. This would allow for the protein coding genes' read-counts to not be affected by the read-counts for the non-protein coding genes, without throwing out the data for the non-protein coding genes.

CQN

Application to Motivating Data Set

The distribution of offset values when running CQN normalization on protein coding and non-protein coding genes separately is shown in Figure 0.16.

Figure 0.16: Distribution of CQN offsets when split on protein coding status prior to normalizing for GC-content



All four data sets in Figure 0.16 appear to have fairly normally distributed offsets, with the non-protein coding genes' offsets being centered further to the right. This difference in the distribution of offsets suggests that the distribution of read-counts for protein and non-protein coding genes are different.

The cqnplots for the full data set and the filtered data set are shown in

Figures 0.17 and 0.18, respectively. The difference in cqnplots for protein coding and non-protein coding genes is apparent for both GC-content and gene lengths. This significant difference in the distribution of the beta-splines suggests that there is some fundamental difference between protein and non-protein coding genes.

Figure 0.17: Systematic effects for CQN with all genes split on protein coding status

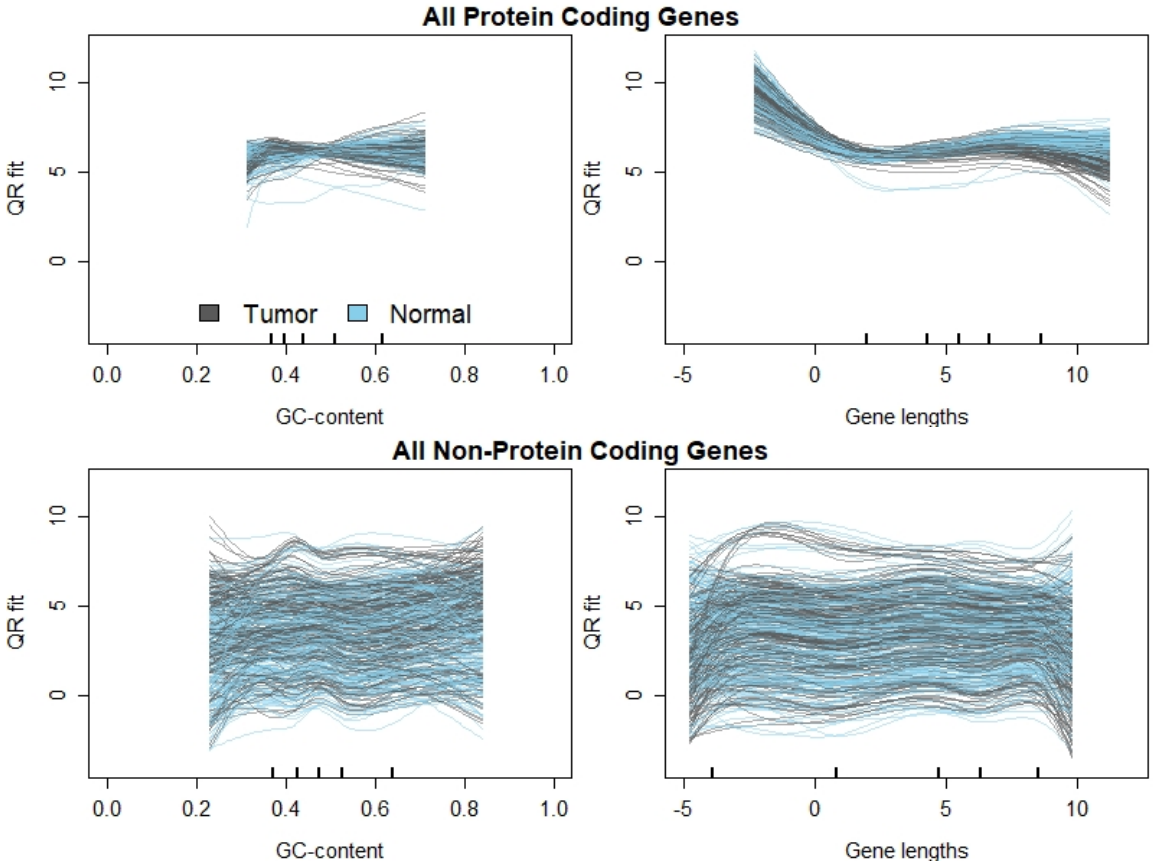
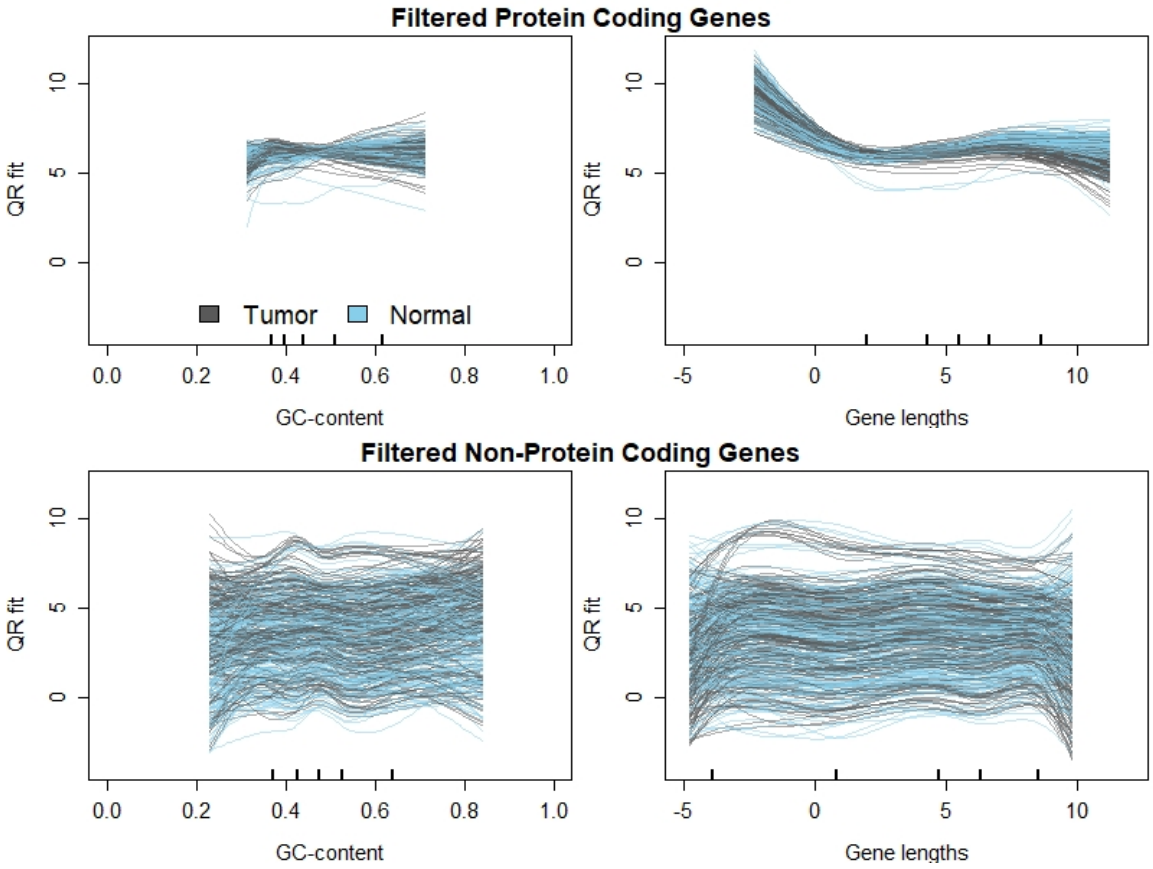


Figure 0.18: Systematic effects for CQN with filtered genes split on protein coding status



Differential Expression Analysis Results

Summaries of the results of the Wilcoxon Rank Sum test and the `lme4` pipelines are shown below in Tables 0.7 and 0.8.

Table 0.7: Wilcoxon Rank Sum Test results using CQN split on protein coding status

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	0 (0%)	9,135 (30.23%)	11,726 (38.80%)	9,359 (30.97%)
Filtered Genes: 14,715	0 (0%)	5,116 (34.77%)	3,544 (24.08%)	6,055 (41.15%)

Table 0.8: `lme4` results using CQN split on protein coding status

lme4 Test Results	**Failed**	**Down Regulated**	**Not Significant**	**Up Regulated**
All Genes: 30,220	104 (0.34%)	8,353 (27.64%)	9,486 (31.39%)	12,277 (40.63%)
Filtered Genes: 14,715	92 (0.63%)	4,888 (33.22%)	3,063 (20.82%)	6,672 (45.34%)

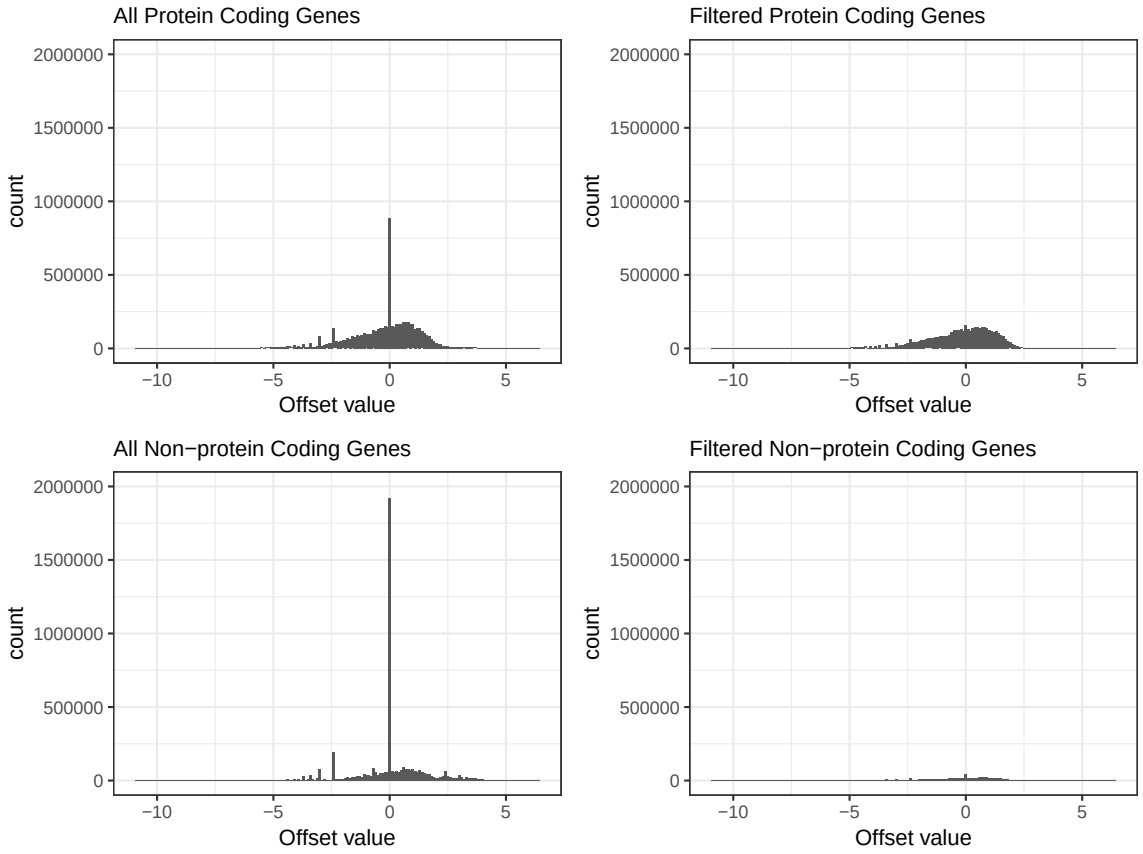
The Wilcoxon Rank Sum test and the `lme4` per-gene models result in fairly similar percentages of genes being categorized as down regulated, but with many more genes being identified as significantly up-regulated by the `lme4` pipeline. This may be evidence that the Wilcoxon Rank Sum test is more conservative or less powerful than the `lme4` pipeline.

EDASeq

Application to Motivating Data Set

The distribution of offset values when running EDASeq normalization on protein coding and non-protein coding genes separately is shown in Figure 0.19. The distribution of offsets for both the protein coding and non-protein coding genes show the spike at zero seen in the "out-of-the-box" implementation of EDASeq (compare to Figure 0.12). However, the spike is much larger in the non-protein coding genes. There are also smaller spikes seen at fairly regular intervals between -2.5 and -5 in the full data set, and these too are reduced in the offsets for the filtered data set.

Figure 0.19: Distribution of EDASeq offsets when split on protein coding status prior to normalizing for GC-content



The bias plots for the full data set are seen in Figure 0.20, and the bias plots for the filtered data set are in Figure 0.21. The lowess curves are very different between the protein coding and non-protein coding genes for all three data sets, but especially for the fully normalized data. Notice in Figure 0.20, the significant difference in scale for the fully normalized protein coding genes. While it appears to be normalizing between lanes that compresses the data, the protein coding genes are much more compressed than the non-protein coding genes.

Very similar patterns are seen in the full (Figure 0.21) and filtered data set plots (Figure 0.20). The major difference is that both fully normalized plots are much more compressed than the raw data or normalized for GC-content plots. The fact that both fully normalized data sets are so compressed is evidence to why

EDASeq so strongly suggests filtering the data set prior to normalization, because the removal of the "non-expressed" genes does affect the normalization results.

Figure 0.20: Comparison of lowess regression on raw read-counts and normalized read-counts - All Genes Split on Protein Coding Status

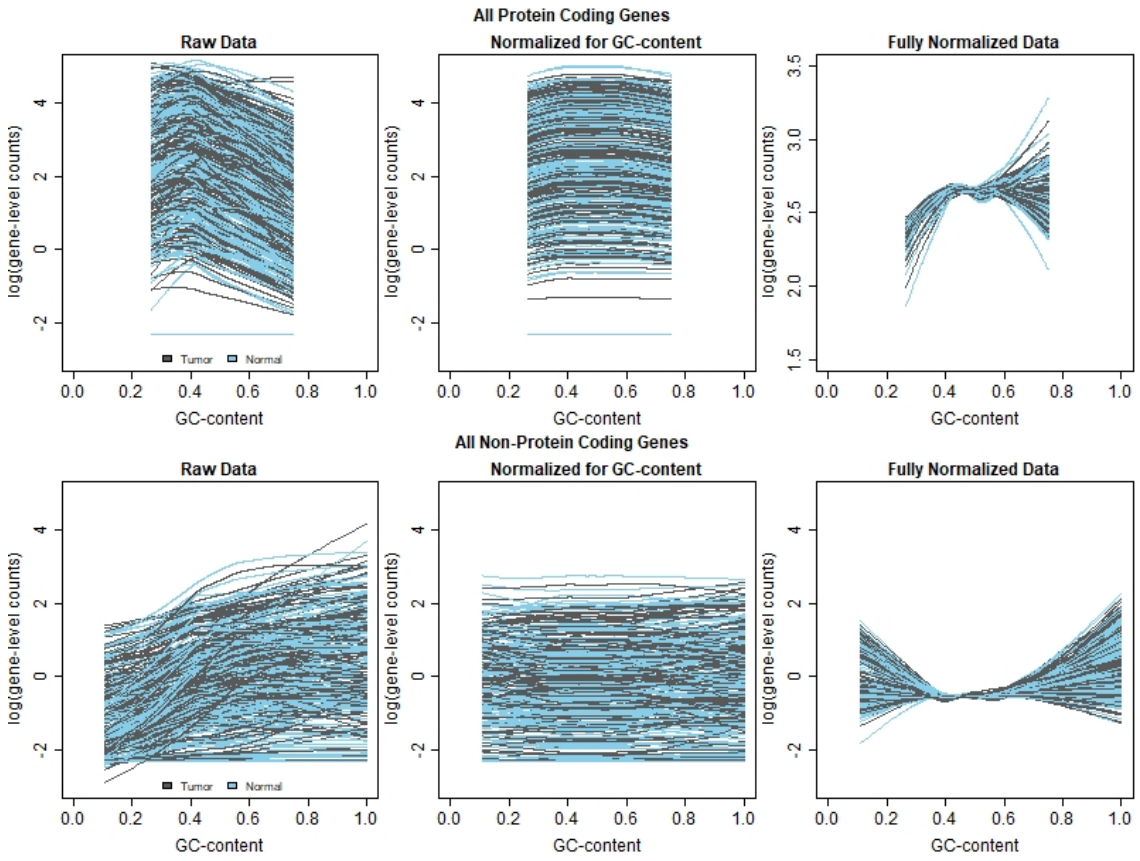
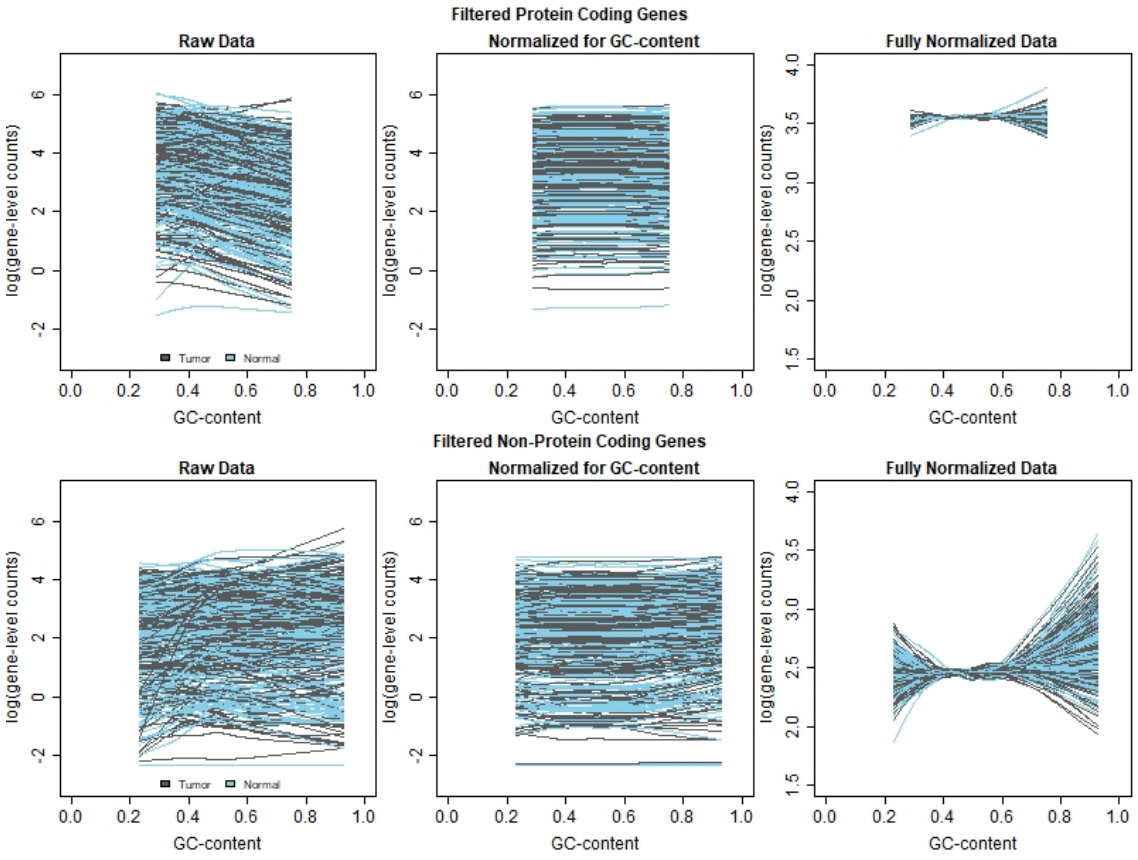


Figure 0.21: Comparison of lowess regression on raw read-counts and normalized read-counts - Filtered Genes Split on Protein Coding Status



Differential Expression Analysis Results

Summaries of the results of the Wilcoxon Rank Sum test and the `lme4` pipelines are shown below in Tables 0.9 and 0.10.

Table 0.9: Wilcoxon Rank Sum Test results using EDASeq split on protein coding status normalized read-counts

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	2 (0.006%)	9,600 (31.77%)	11,155 (36.92%)	9,463 (31.31%)
Filtered Genes: 14,715	1 (0.007%)	5,307 (36.07%)	3,461 (23.52%)	5,946 (40.41%)

Table 0.10: `lme4` results using EDASeq split on protein coding status offsets

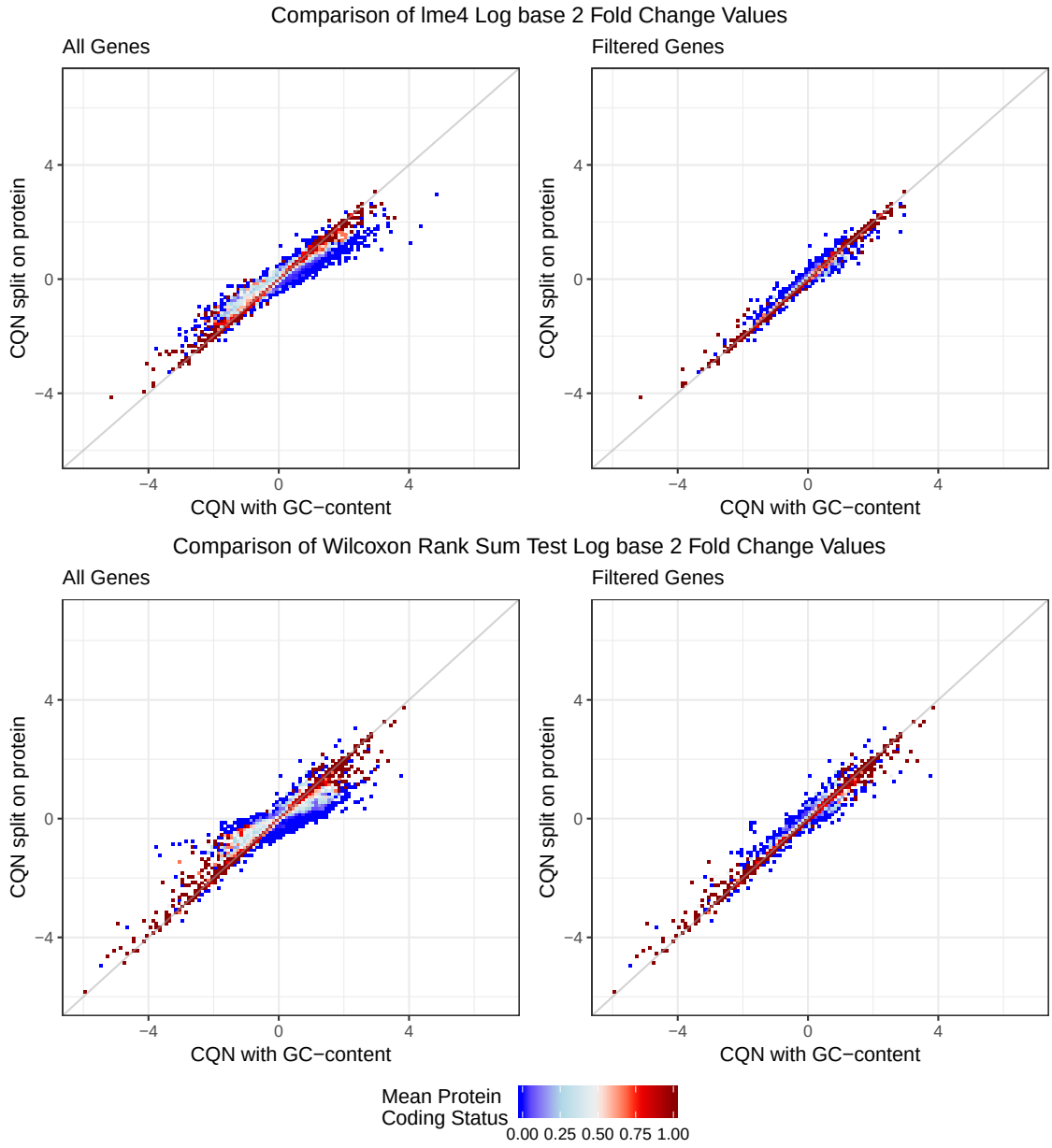
lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	648 (2.14%)	9,410 (31.14%)	10,625 (35.16%)	9,537 (31.56%)
Filtered Genes: 14,715	378 (2.57%)	5,122 (34.81%)	3,378 (22.86%)	5,837 (39.67%)

As when using EDASeq "out-of-the-box" (see Tables 0.5 and 0.6), very few Wilcoxon Rank Sum tests failed and a small percent of `lme4` models failed when normalizing protein and non-protein coding genes separately. Filtering also seems to have an effect on the results of both methods. Higher proportions of genes are identified as significant in the filtered data set than in the non-filtered data set, suggesting that many of the genes that were filtered out of the data set, the "non-expressed" genes, were identified in the full data set as not significantly differentially expressed.

Compare Differential Expression Analysis Results

In order to compare the results of splitting on protein coding status to the out-of-the-box methods, the log_2 fold change values were plotted. The results from using CQN methods (both Wilcoxon Rank Sum and `lme4`) are compared in Figure 0.22, and the results from using EDASeq methods are shown in Figure 0.23.

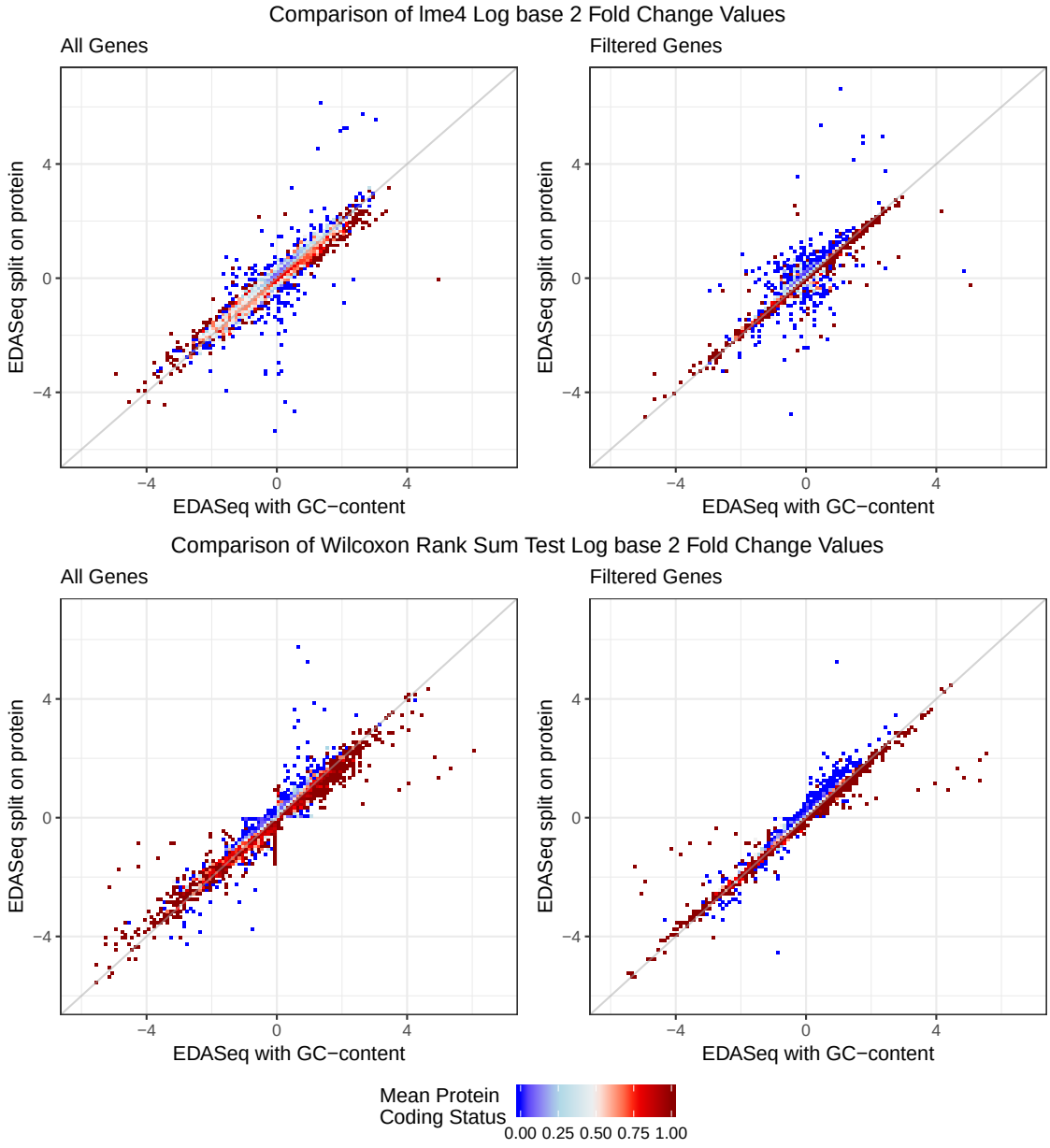
Figure 0.22: Comparison of results using CQN with GC-content and CQN split on protein coding status



Notice in Figure 0.22 how in both the `lme4` and the Wilcoxon Rank Sum plots, the protein coding genes (shown in red) appear to be clustered around the line of equality, and many of the non-protein coding genes (shown in blue) tend to be farther from the line of equality. This suggests that CQN treats the non-protein coding genes differently than the protein coding genes when

separated prior to normalization. This is evidence that the distribution of read-counts actually does differ between protein and non-protein coding genes, supporting the inclusion of protein coding status as a normalization covariate.

Figure 0.23: Comparison of results using EDASeq with GC-content and EDASeq split on protein coding status



The pattern of separation for non-protein coding genes as seen in Figure 0.22 is not as clear in the Figure 0.23 subplots for the full data set as it in the subplots for the filtered data set. This suggests that EDASeq does not treat the non-protein coding genes as differently as CQN when dealing with the full data set, but that the filtered data set is treating the protein and non-protein coding genes differently. It is important to note here that the filtered data set of 14,715 genes is comprised of 12,618 protein coding genes (85.75%) and 2,097 non-protein coding genes (14.25%). This unbalanced split may be a reason for the separation seen in the filtered data subplots.

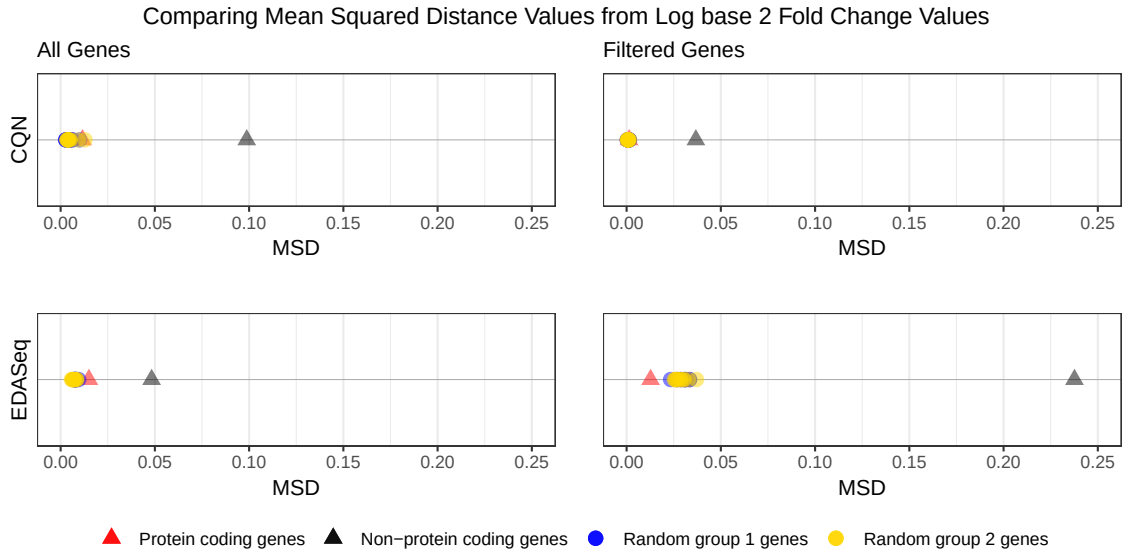
Quantifying differences due to protein-coding status

How different are the protein coding and non-protein coding genes? In order to quantify the difference between normalizing with GC-content and splitting on protein coding status prior to normalization, a method of measuring this difference was created. The Mean Squared Difference (MSD) is defined as the mean across all genes x of (log_2 fold change value for gene x when split on protein coding status - log_2 fold change value for gene x when run with GC-content)^2.

In order to determine if the difference in the distribution of log_2 fold change values is truly due to the distribution of read counts from protein coding and non-protein coding genes being different, or if it is a result of splitting the data, a random split of the data was applied. Random groups 1 and 2 were generated (with sizes comparable to numbers of protein-and non-protein coding genes), the resulting two read-count matrices were normalized, and the offsets were used in the `lme4` pipeline. This was repeated a total of 11 times. Ideally thousands of random splits would be used in order to compare the observed $MSD_{protein}$ and $MSD_{nonprotein}$ to the distribution of MSD_{random} splits, but because each batch of 30,220 genes requires about 25 computing hours to complete the `lme4` pipeline, 11 ran-

dom splits will be used here. The resulting MSD values are plotted in Figure 0.24.

Figure 0.24: Comparing Mean Squared Differences from `lme4` log_2 fold change values



In both the full and filtered data set and with both CQN and EDASeq normalization, the randomly split data produces offsets and then log_2 fold change values from `lme4` that are very similar to each other and to the protein coding genes. This suggests that simply splitting the data into two read-count matrices is not what produces the different distributions seen in Figures 0.22 and 0.23. The clear separation of the non-protein coding genes from the protein coding genes and randomly split MSD values suggests that there is a difference in the distribution of read-counts between protein and non-protein coding genes, and that splitting on protein coding status allows for that difference to be taken into account.

Edit Existing Methods to Account for a Binary Covariate

After normalizing the protein coding and non-protein coding genes separately, I next explored editing the CQN and EDASeq functions to take a binary covariate without producing a fatal computational error. Ideally, the edited normalization

method would take all three gene-level covariates (gene length, GC-content, and protein coding status), as it has been shown that normalizing for GC-content produces more accurate normalized read counts (Hansen, Irizarry, and Wu 2012), but that has not been shown for protein coding status. Therefore, replacing GC-content with protein coding status is not as justified as accounting for a third covariate if possible.

CQN

As described previously, CQN uses natural cubic splines to incorporate gene-level covariates in its quantile normalization. This means that extending CQN normalization to take a third gene-level covariate would increase the dimensionality of the splines to spline surfaces. Due to the complexity of that dimensionality change, CQN was not edited to take a third covariate. Instead, I opted to replace GC-content with protein coding status.

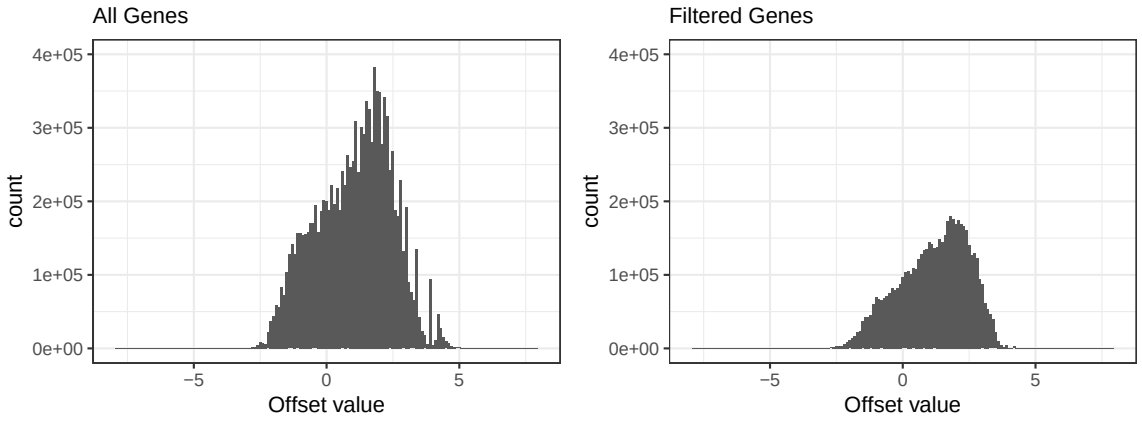
However, as mentioned previously in this chapter, the binary nature of protein coding status causes CQN to encounter fatal computational errors. This is due to the knots that support the splines (generated from specific quantiles) having only two unique values (0 and 1). Therefore, for protein coding status to be included in CQN, the binary covariate needed to be transformed into a pseudo-continuous variable, which was done via "jittering."

This "jittering" of the binary covariate was done by randomly sampling with replacement 30,220 epsilon values between 0 and 1×10^{-5} in uniform steps of 1×10^{-10}, and then adding those epsilon values to the 0/1 protein coding status variable. The epsilon values were generated under the same seed each time for reproducibility. The epsilon limit values used here are fairly arbitrary, but are as small as possible to limit the amount of added noise without crashing RStudio 1.1.456 running on R version 3.5.0 (R Core Team 2018).

Application to Motivating Data Set

The distribution of the offsets generated by running CQN with gene length and jittered protein coding status are shown in Figure 0.25.

Figure 0.25: Distribution of CQN offsets when normalizing with jittered protein coding status



The same overall distribution of offsets is seen in these offsets from running CQN on jittered protein coding status as was seen in the offsets from running CQN on GC-content (Figure 0.9) and in the offsets for the protein coding genes when running CQN split on protein coding status (Figure 0.16).

Figures 0.26 and 0.27 show cqnplots run on jittered protein coding status and on gene lengths.

Figure 0.26: Systematic effects for CQN with all genes and jittered protein coding status

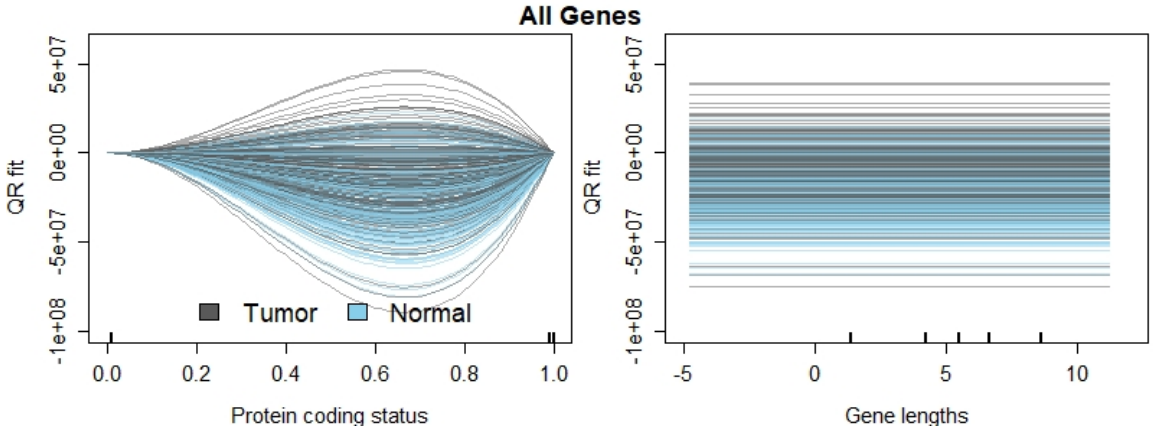
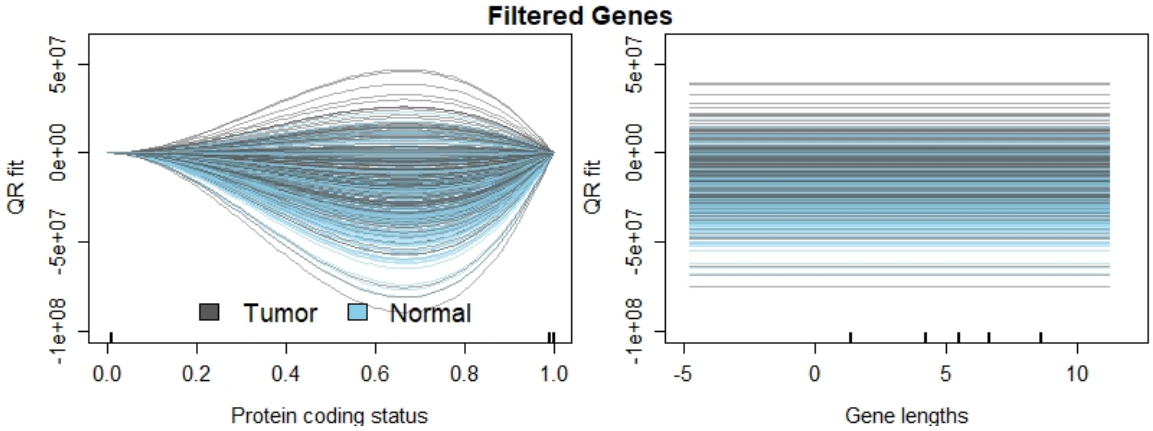


Figure 0.27: Systematic effects for CQN with filtered genes and jittered protein coding status



As seen in the previous cqnplots (Figures 0.10, 0.11, 0.17, 0.18), Figures 0.26 and 0.27 do not appear to differ, and indeed only differ very slightly. Notice how massive the scale is for these plots, and how all of the variation is removed from the beta-splines calculated on the gene length values. The shape of the beta-splines calculated on protein coding status is also very different than what we saw for GC-content. This is due to the difference in the distribution of the knots, or quantiles, of the variable. The tick marks seen above the x-axis representing these knots are all clustered right near 0 and right near 1, where the jittered protein coding status values exist.

Differential Expression Analysis Results

Summaries of the results of the Wilcoxon Rank Sum test and the `lme4` pipelines are shown below in Tables 0.11 and 0.12.

Table 0.11: Wilcoxon Rank Sum Test results using CQN with jittered protein coding status normalized read-counts

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	0 (0%)	6,977 (23.09%)	11,886 (39.33%)	11,357 (37.58%)
Filtered Genes: 14,715	0 (0%)	5,026 (34.16%)	3,621 (24.61%)	6,068 (41.24%)

Table 0.12: `lme4` results using CQN with jittered protein coding status offsets

lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	274 (0.91%)	7,204 (23.84%)	10,190 (33.72%)	12,552 (41.54%)
Filtered Genes: 14,715	54 (0.37%)	4,752 (32.29%)	3,279 (22.28%)	6,630 (45.06%)

As seen when splitting on protein coding status (Tables 0.7 and 0.8), filtering the data prior to normalization to remove non-expressed genes appears to also remove a substantial portion of non-significantly differentially expressed genes as identified by both the Wilcoxon Rank Sum test and the `lme4` per-gene models.

EDASeq

EDASeq uses lowess regression, not natural cubic splines, in its normalization process. The pipeline is also set up to normalize within lane then between lanes by running two functions sequentially. This means that I was able to create an edited version of the package's *withinLaneNormalization* function to run with a binary covariate, and introduce a third normalization step to normalize for the third covariate.

The sequential nature of this edited EDASeq pipeline produced two separate normalized read count data sets, one normalized for GC-content then protein cod-

ing status and gene length, and a second normalized for protein coding status then GC-content and gene length.

Application to Motivating Data Set

The offsets from normalizing for GC-content then protein coding status and gene length (Figure 0.28), were slightly different than the offsets from normalizing for protein coding status then GC-content and gene length (Figure 0.29).

Figure 0.28: Distribution of EDASeq offsets when normalizing for GC-content then protein coding status

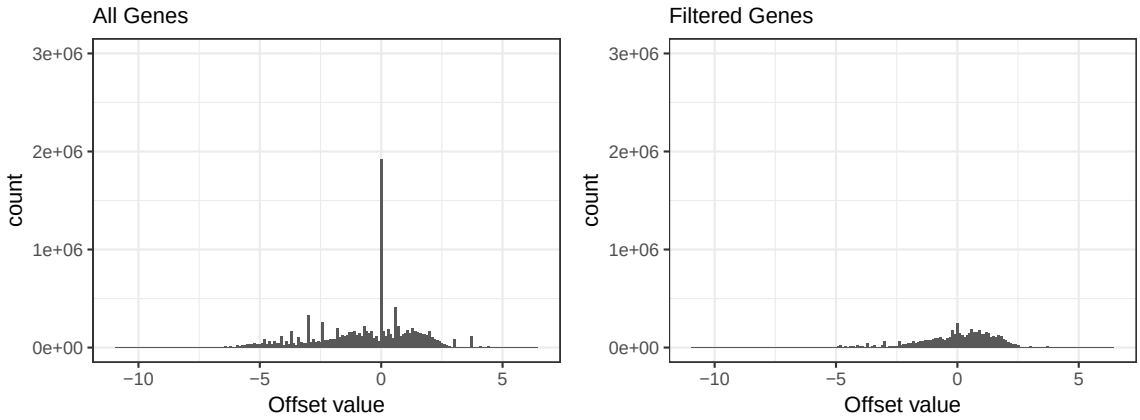
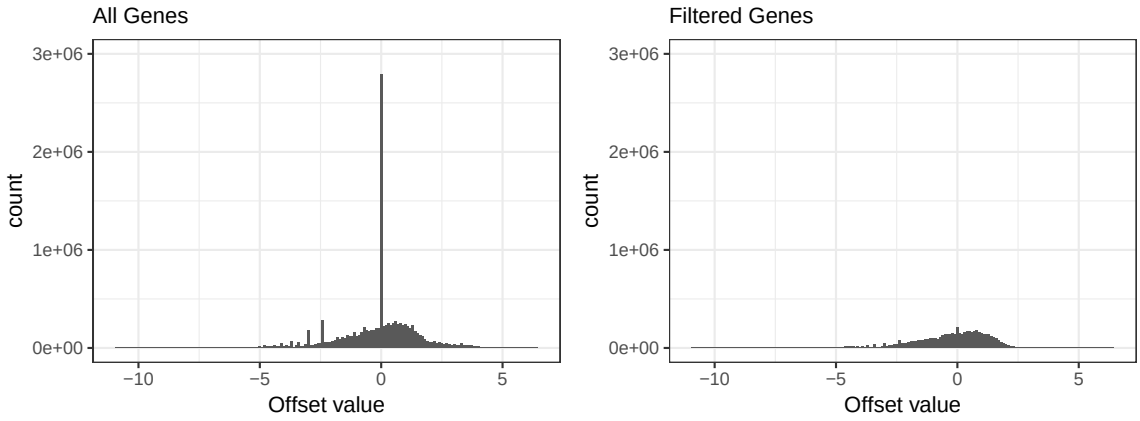


Figure 0.29: Distribution of EDASeq offsets when normalizing for protein coding status then GC-content



While the distribution of offset values for the filtered data sets are very similar, the offsets for the full data set appear to be somewhat different. Normalizing for protein coding status and then GC-content (Figure 0.29) results in a distribution much more similar to those seen in both EDASeq variations previously examined (Figures 0.12 and 0.19). Normalizing for GC-content and then protein coding status (Figure 0.28) increases the spread of the offsets, accentuating the spikes described in the plots from splitting the read-count data on protein coding status prior to EDASeq normalization (Figure 0.19).

The bias plots for normalizing the full data set (Figures 0.30 and 0.31) are remarkably similar, with only the plot order changed. The biggest difference between Figures 0.30 and 0.31 is in the subplot based on the fully normalized data. Normalizing for protein coding status then GC-content appears to produce more compressed end results (Figure 0.30).

Figure 0.30: Comparison of lowess regression on all genes raw read-counts and read-counts normalized with GC-content then protein coding status

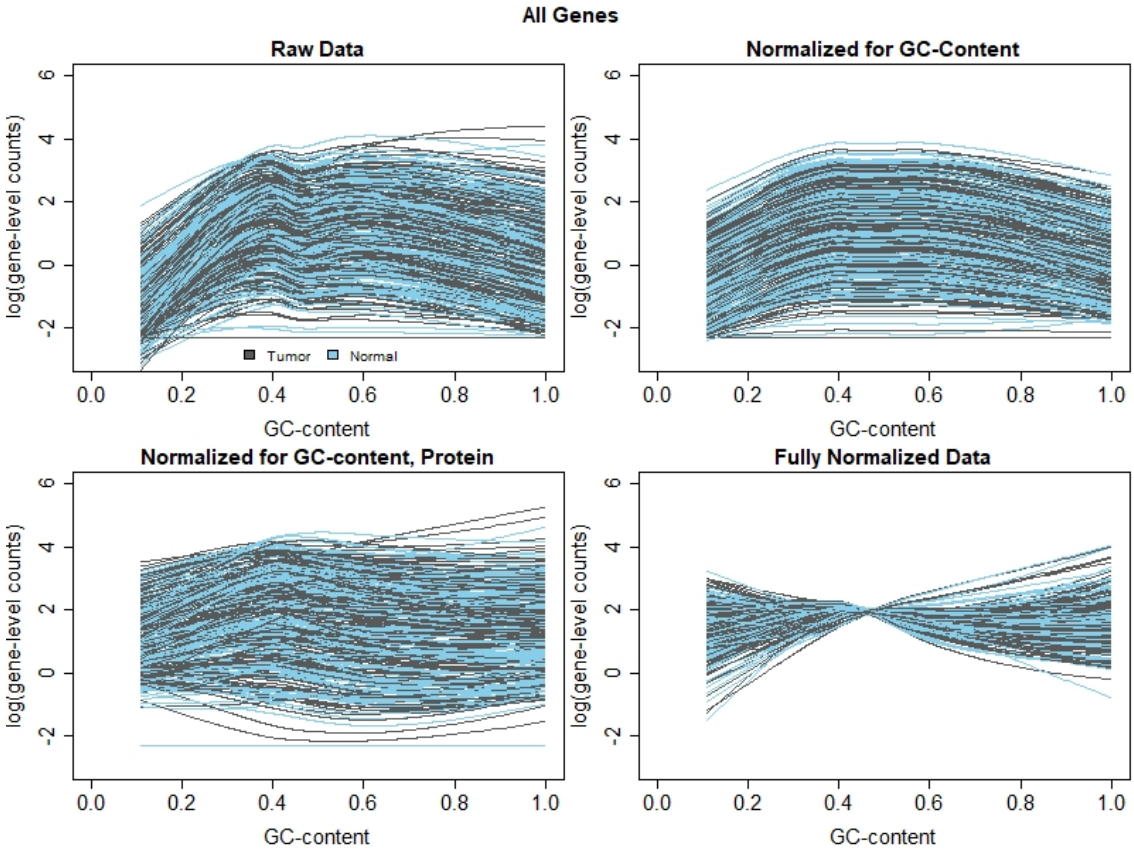
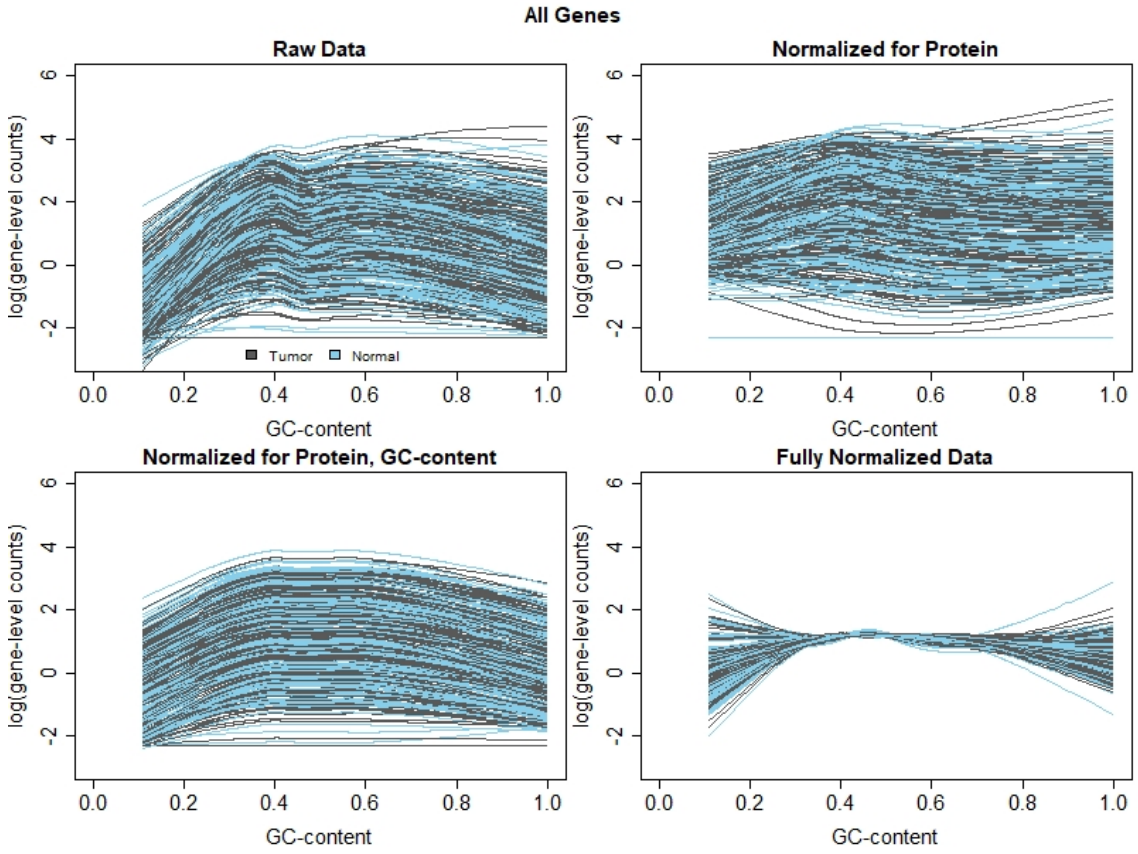


Figure 0.31: Comparison of lowess regression on all genes raw read-counts and read-counts normalized with protein coding status then GC-content



The bias plots are also included for the filtered data set, Figures 0.32 and 0.33. The pattern seen in the plots created on the full data set (Figures 0.30 and 0.31) is also seen in the plots created on the filtered data set (Figures 0.32 and 0.33). It is interesting, however, that the lowess curves after normalizing for protein coding status look very similar to the lowess curves calculated on the raw data, even when the data was previously normalized for GC-content.

Figure 0.32: Comparison of lowess regression on filtered genes' raw read-counts and read-counts normalized with GC-content then protein coding status

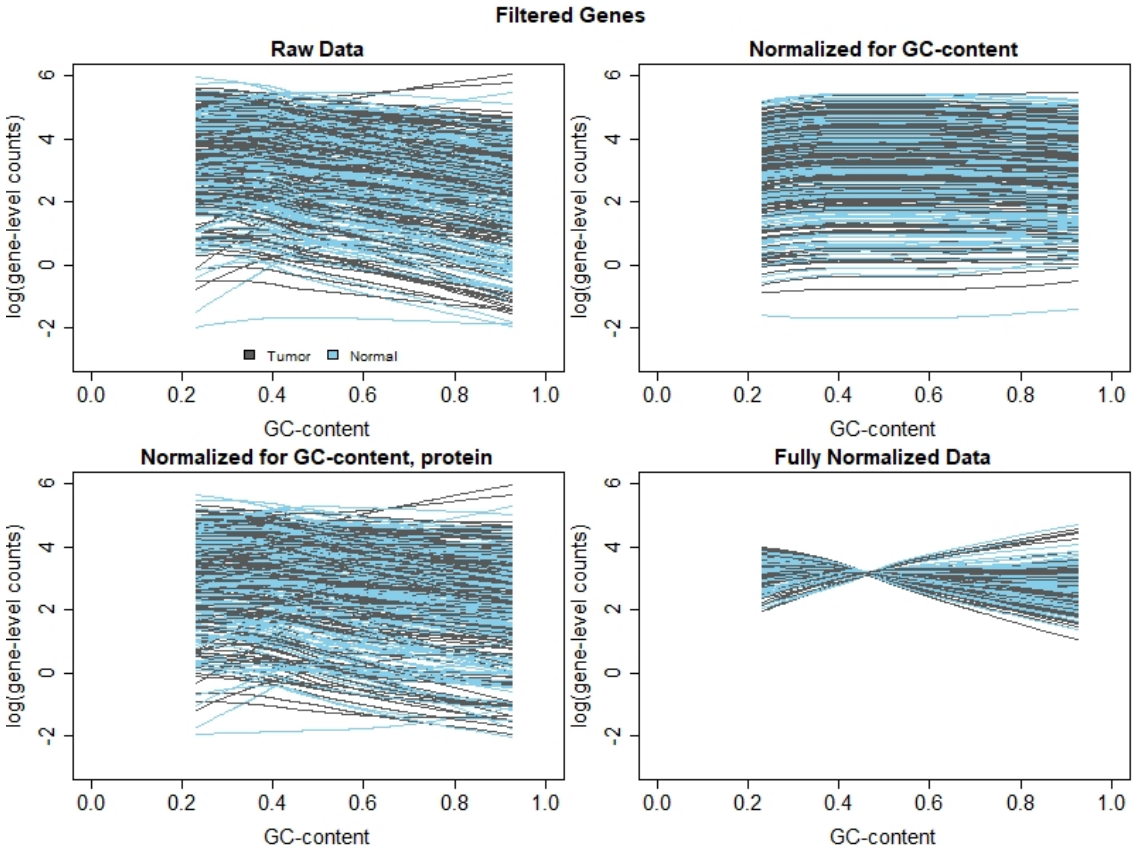
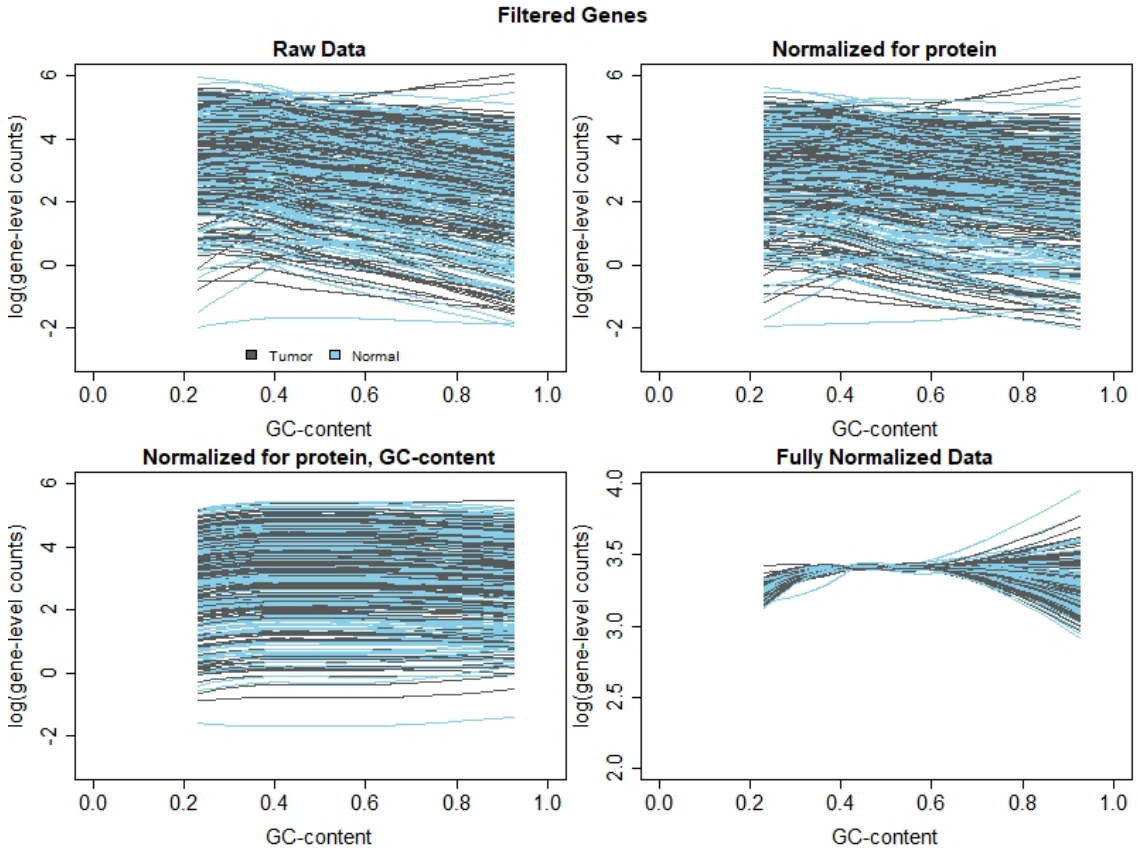


Figure 0.33: Comparison of lowess regression on filtered genes raw read-counts and read-counts normalized with protein coding status then GC-content



Differential Expression Analysis Results

The normalized read counts were very similar for each of the two orderings of covariates, and results from the Wilcoxon Rank Sum test were identical, as summarized in Tables 0.13 and 0.14.

Table 0.13: Wilcoxon Rank Sum Test results using EDASeq normalizing for GC-content then protein coding status

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	0 (0%)	9,932 (32.87%)	10,895 (36.05%)	9,393 (31.08%)
Filtered Genes: 14,715	0 (0%)	5,255 (35.71%)	3,566 (24.23%)	5,894 (40.01%)

Table 0.14: Wilcoxon Rank Sum Test results using EDASeq normalizing for protein coding status then GC-content

Wilcoxon Rank Sum Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	0 (0%)	9,932 (32.87%)	10,895 (36.05%)	9,393 (31.08%)
Filtered Genes: 14,715	0 (0%)	5,255 (35.71%)	3,566 (24.23%)	5,894 (40.05%)

Notice that for both the full and filtered data sets, the same number of genes are identified as significantly down-regulated, not significantly differentially expressed, and significantly up-regulated. In fact, there is perfect agreement between the differential expression calls between the two orderings of the covariates, as seen in Tables 0.15 and 0.16.

Table 0.15: Comparing the calls of differential expression from the Wilcoxon Rank Sum test on all genes when using EDASeq offsets on three covariates

		Protein, GC-content		
		Down Regulated	**Not Significant**	**Up Regulated**
GC-content, Protein	**Down Regulated**	9,932	0	0
	Not Significant	0	10,895	0
	Up Regulated	0	0	9,393

Table 0.16: Comparing the calls of differential expression from the Wilcoxon Rank Sum test on filtered genes when using EDASeq offsets on three covariates

		Protein, GC-content		
		Down Regulated	**Not Significant**	**Up Regulated**
GC-content, Protein	**Down Regulated**	5,255	0	0
	Not Significant	0	3,566	0
	Up Regulated	0	0	5,894

Whatever slight variations there might be in log_2 fold change values, the categorization of significantly up- or down-regulated and not significantly differentially expressed for each gene is identical.

The results of the `lme4` pipeline are also very similar, as summarized in Tables 0.17 and 0.18, and Figure 0.34.

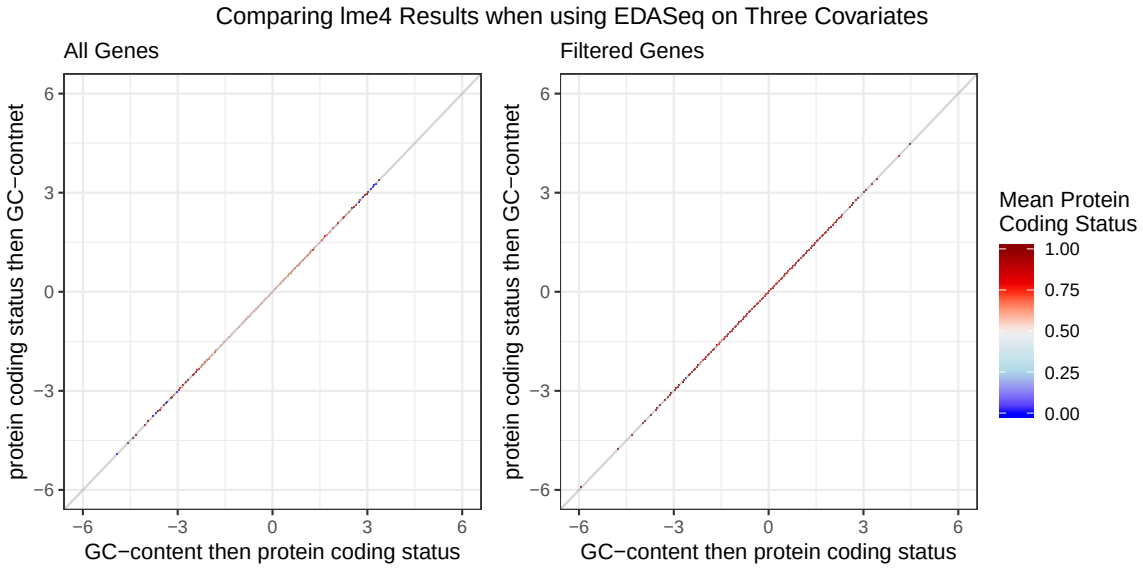
Table 0.17: `lme4` results using EDASeq normalizing for GC-content then protein coding status

lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	366 (1.21%)	9,022 (29.85%)	11,558 (38.25%)	9,274 (30.69%)
Filtered Genes: 14,715	936 (6.36%)	4,801 (32.63%)	3,458 (23.50%)	5,520 (37.51%)

Table 0.18: `lme4` results using EDASeq normalizing for protein coding status then GC-content

lme4 Test Results	Failed	Down Regulated	Not Significant	Up Regulated
All Genes: 30,220	367 (1.21%)	9,022 (29.85%)	11,559 (38.25%)	9,272 (30.68%)
Filtered Genes: 14,715	936 (6.36%)	4,804 (32.65%)	3,456 (23.49%)	5,519 (37.51%)

Figure 0.34: Comparing the log_2 fold changes from the `lme4` pipeline when using EDASeq offsets on three covariates



There is very little spread around the line of equality in Figure 0.34, and overall the results are incredibly similar. In the full data set, 5 genes' significance switches between the two orders of covariates (Table 0.19), and in the filtered data

set only 3 genes switch (Table 0.20).

Table 0.19: Comparing the calls of differential expression from the `lme4` pipeline on all genes when using EDASeq offsets on three covariates

		Protein, GC-content		
		Down Regulated	**Not Significant**	**Up Regulated**
GC-content, Protein	**Down Regulated**	9,020	1	0
	Not Significant	2	11,556	0
	Up Regulated	0	2	9,272

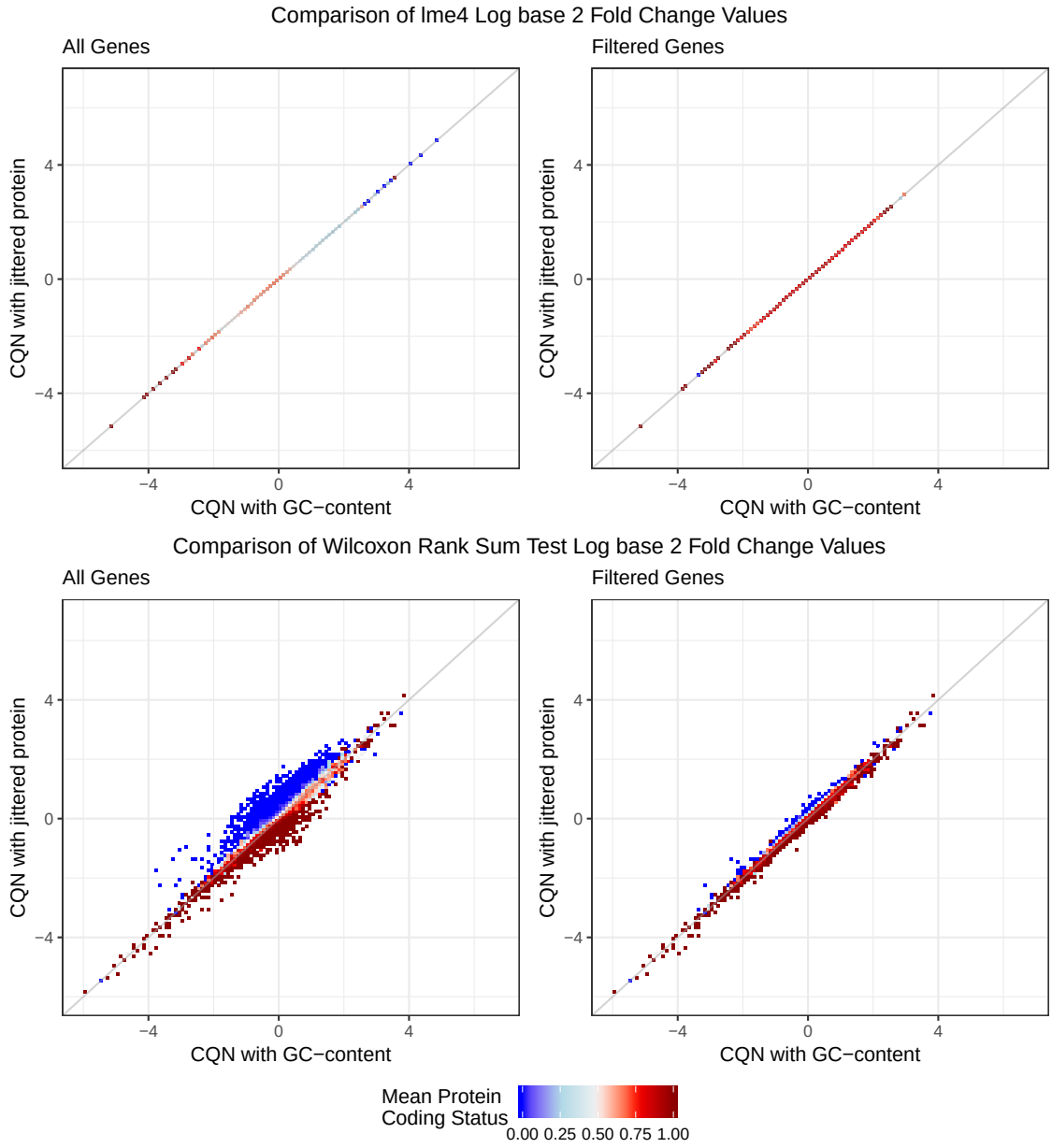
Table 0.20: Comparing the calls of differential expression from the `lme4` pipeline on filtered genes when using EDASeq offsets on three covariates

		Protein, GC-content		
		Down Regulated	**Not Significant**	**Up Regulated**
GC-content, Protein	**Down Regulated**	4,801	0	0
	Not Significant	2	3,455	0
	Up Regulated	0	1	5,517

Compare Differential Expression Analysis Results

Comparing the log_2 fold change values from CQN normalization with jittered protein coding status to CQN with GC-content is very surprising (see Figure 0.35).

Figure 0.35: Comparison of results using CQN with GC-content and CQN with jittered protein coding status



First examining the log_2 fold change values from `lme4`, the points are all very close to the line of equality. This was fairly surprising, since I expected the results from the two methods to be very different. The only pattern that is very visible in the `lme4` plots is in the All Genes plot: it appears that the protein coding genes tend to have lower log_2 fold change values than the non-protein coding genes. Be-

cause so many of the non-protein coding genes are filtered out, the pattern is not seen in the Filtered Genes subplot.

The log_2 fold change values from the Wilcoxon Rank Sum tests have much more variation, and have a more interesting pattern. Non-protein coding genes have higher log_2 fold change values and protein coding genes have lower log_2 fold change values when run on normalized read-counts from CQN with jittered protein coding status than on the normalized read-counts from CQN on GC-content. This very clean separation is interesting, and is seen in the filtered data set even though most of the non-protein coding genes were filtered out prior to normalization.

A different story is seen in the plots comparing the log_2 fold change values from the `lme4` pipeline and the Wilcoxon Rank Sum test run on EDASeq with GC-content and protein coding status (and vice versa) to the out-of-the-box method (Figures 0.36 and 0.37).

Figures 0.36 and 0.37 are very similar, as expected. Looking at the `lme4` results, again the protein coding genes appear to have very similar results from both methods, while the non-protein coding genes appear to be more variable. The filtered genes show better separation than the full data set.

The Wilcoxon rank sum test log_2 fold change values are messier than those from the `lme4` pipeline, but still show a majority of protein coding genes clustered around the line of equality.

Figure 0.36: Comparison of results using EDASeq with GC-content and EDASeq with GC-content, then protein coding status

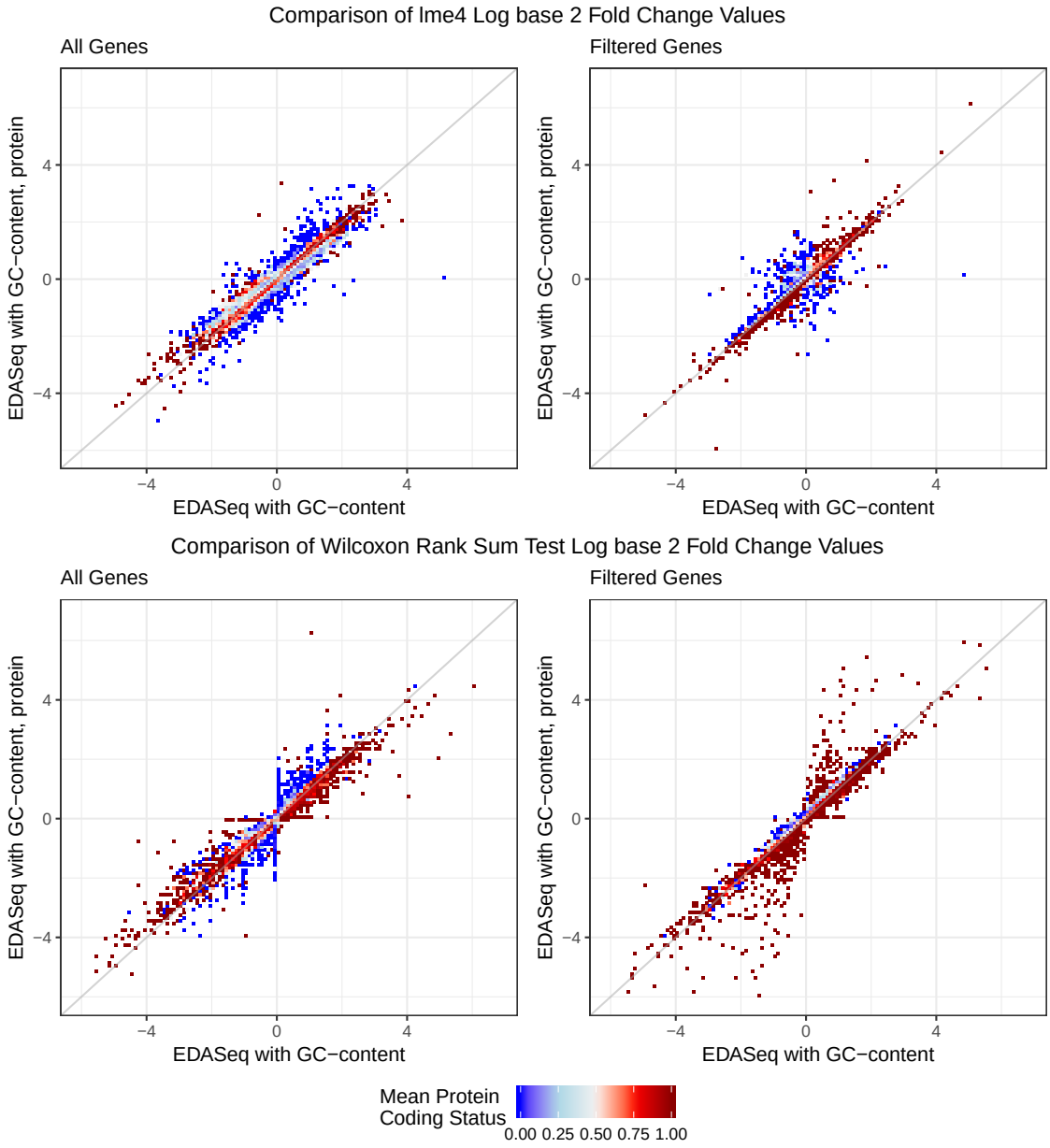
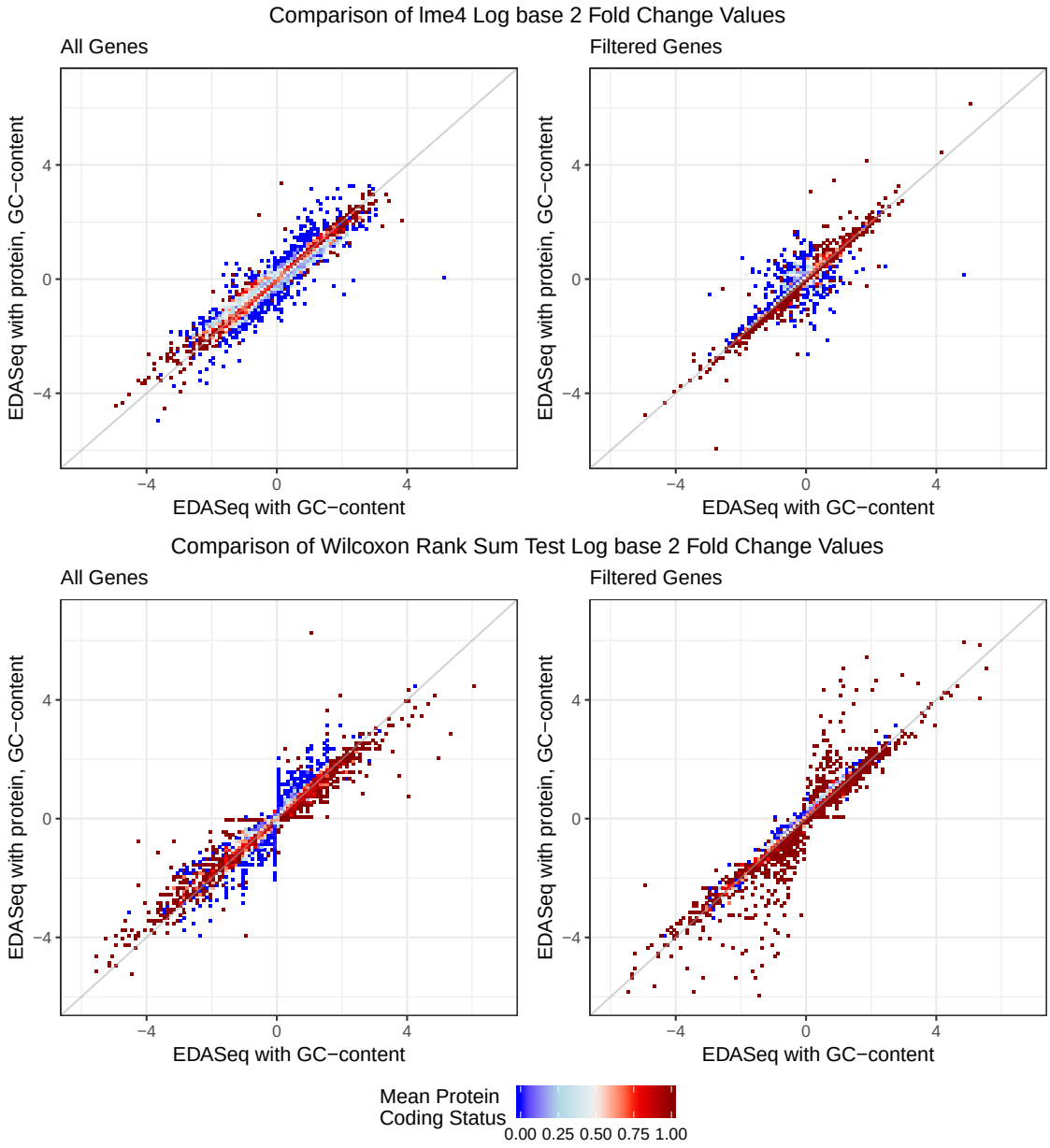


Figure 0.37: Comparison of results using EDASeq with GC-content and EDASeq with protein coding status, then GC-content



CHAPTER 4
COMPARISON OF RESULTS

CQN Normalization and Variations

A comparison of the log_2 fold changes from the `lme4` per-gene models when run using offsets from all CQN variations (with GC-content, split on protein coding status, with jittered protein coding status) is shown in Figure 0.38. The `lme4` log_2 fold change values are most similar when using the offsets from CQN with jittered protein coding status and CQN with GC-content. I would have expected the most agreement to be with CQN with jittered protein coding status and CQN split on protein coding status. The very strong agreement seen here would suggest that protein coding status may be able to replace GC-content as a gene-level covariate.

Comparing the `lme4` log_2 fold change values (Figure 0.38) from CQN split on protein coding status to the results from both CQN with GC-content and CQN with jittered protein coding status, it is the protein coding genes that are clustered around the line of equality and the non-protein coding genes that vary. As mentioned before, this suggests that the non-protein coding genes are treated differently depending on when and how protein coding status is accounted for.

A comparison of the log_2 fold changes from the Wilcoxon Rank Sum test when run using normalized read-counts from all CQN variations (with GC-content, split on protein coding status, with jittered protein coding status) is shown in Figure 0.39. The Wilcoxon Rank Sum test log_2 fold change values are also most similar when using the normalized read counts from CQN with jittered protein coding status and CQN with GC-content, although only in the filtered data set. Jittering protein coding status appears to produce higher log_2 fold change values

for non-protein coding genes and lower log_2 fold change values for protein coding genes than either CQN with GC-content or CQN split on protein coding status.

Figure 0.38: Comparison of `lme4` results using CQN variations

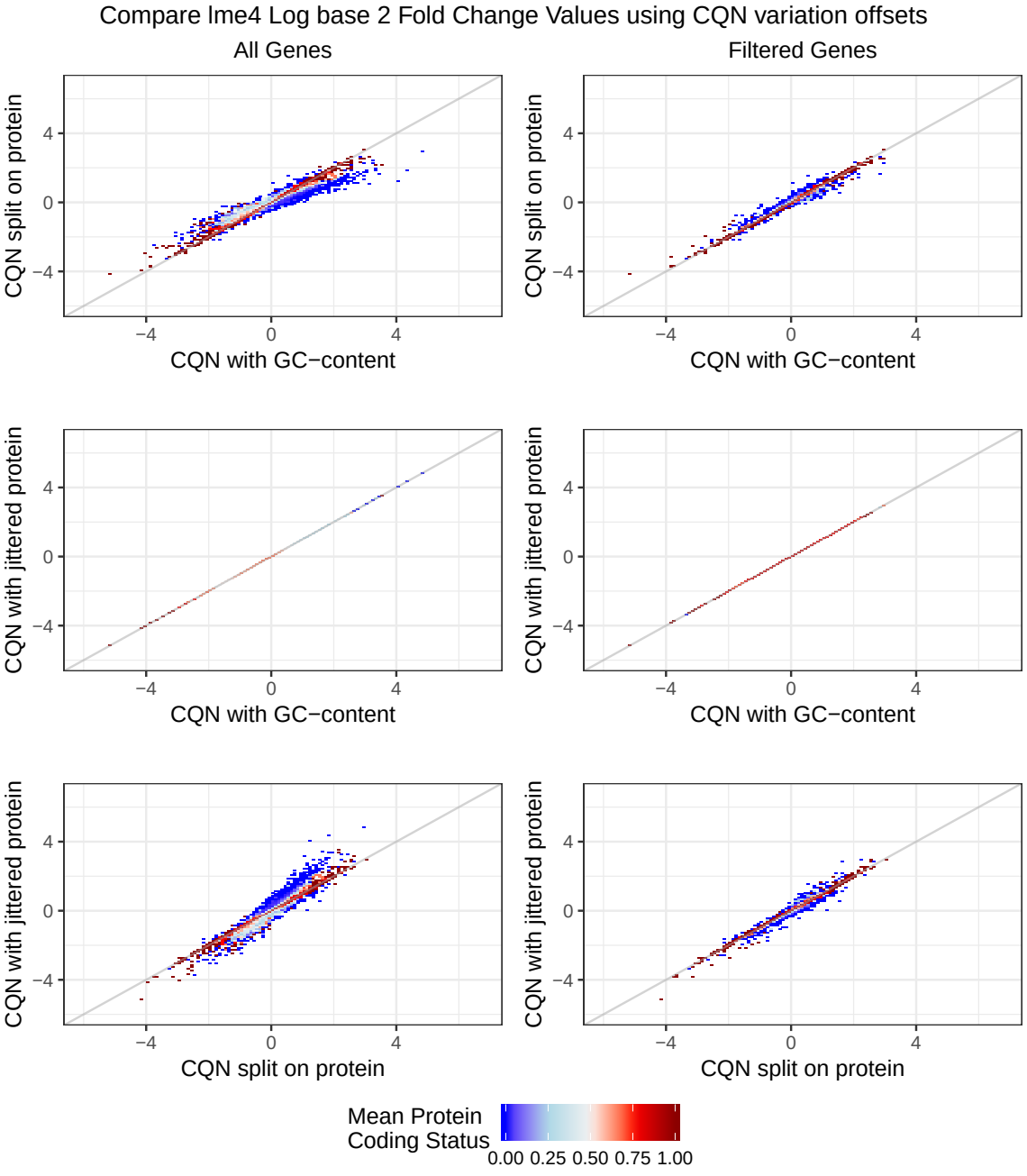
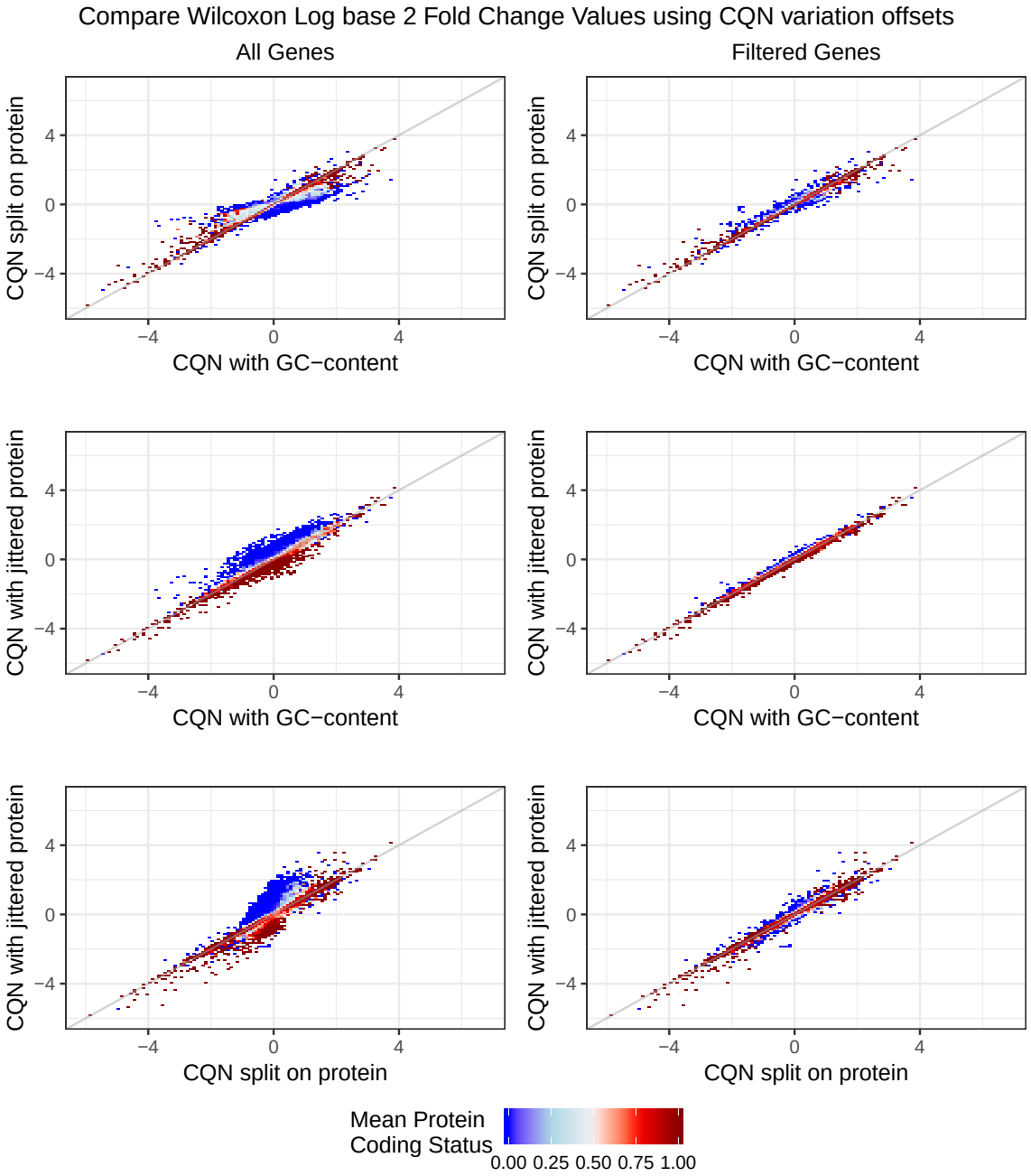


Figure 0.39: Comparison of Wilcoxon Rank Sum Test results using CQN variations



EDASeq Normalization and Variations

Since the results of EDASeq are not hugely changed by the normalization order of GC-content and protein coding status, the results in this section correspond to running EDASeq normalizing for GC-content, then protein coding status, then

gene length. Figure 0.40 compares the log_2 fold change values from the `lme4` per-gene models run using the offsets from three variations of EDASeq normalization (with GC-content, split on protein coding status, with GC-content then protein coding status).

The log_2 fold change values when using EDASeq offsets are more different between the full data set and the filtered data set than they are for CQN normalization. This is likely due to EDASeq strongly recommending filtering, while CQN does not.

As in the comparison of the `lme4` log_2 fold change values from CQN variations (Figure 0.39), here we again see protein coding genes clustered around the line of equality and the non-protein coding genes spread out away from the line of equality.

A comparison of the log_2 fold changes from the Wilcoxon Rank Sum test when run using normalized read-counts from all EDASeq normalization variations (with GC-content, split on protein coding status, with GC-content then protein coding status) is shown in Figure 0.41.

Figure 0.41 is the most unique of the plots in this section. The normalization methods that produce the most similar plots are EDASeq with GC-content and EDASeq split on protein coding status when dealing with the filtered data set. The other plots show quite a bit of spread around the line of equality (compare to Figure 0.40), indicating that EDASeq normalization is more variable than CQN (compare to Figure 0.39). It is also important to note that while there are quite a few points spread away from the line of equality, no genes appear to be so far off the line of equality that one method would identify it as significantly down-regulated and another would identify it as significantly up-regulated, or vice versa. A stricter significance cut off than what was used here ($\alpha = 0.05$) would reduce the number of genes with disagreeable significant/not-significant labels.

Figure 0.40: Comparison of `lme4` results using EDASeq variations

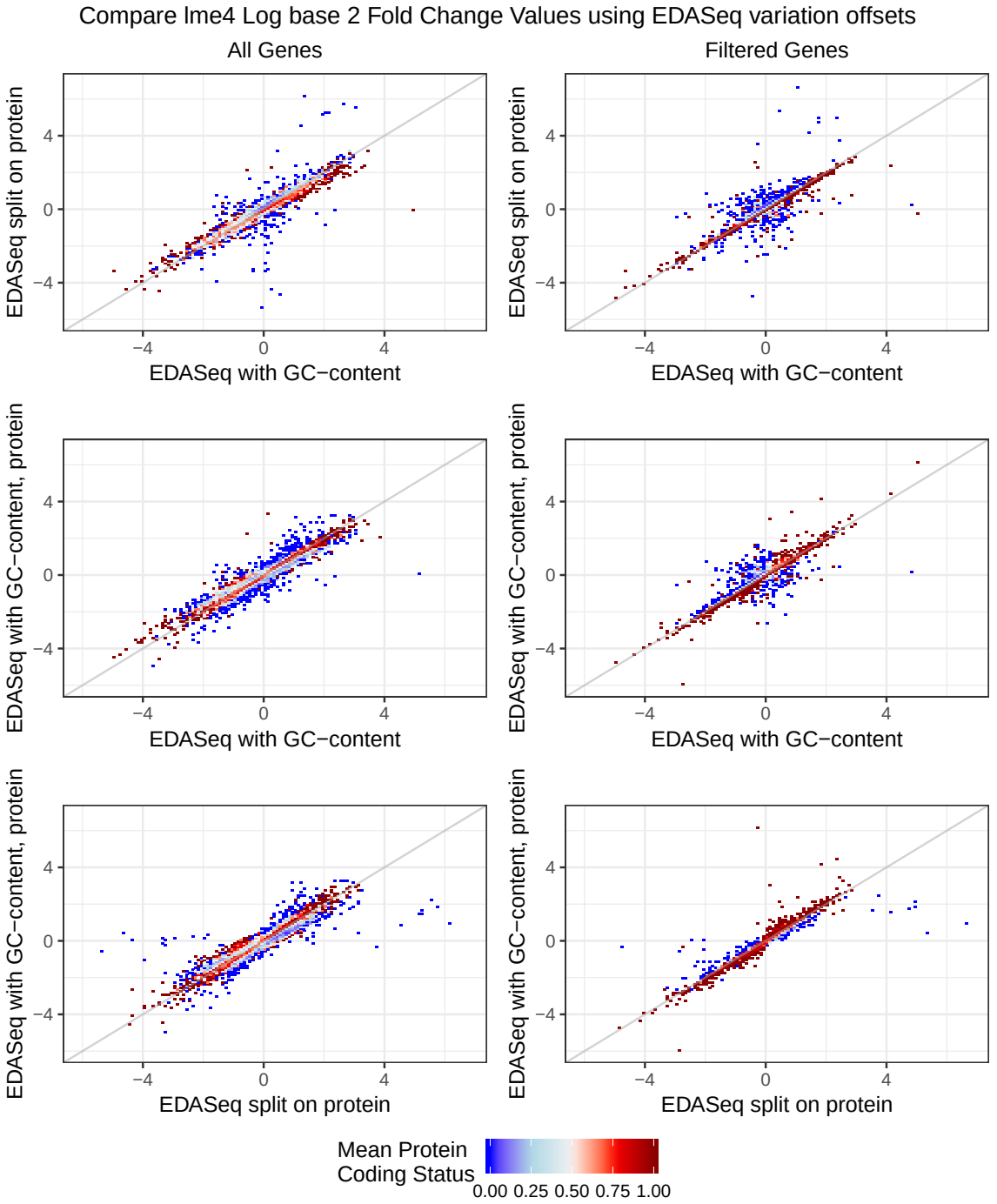
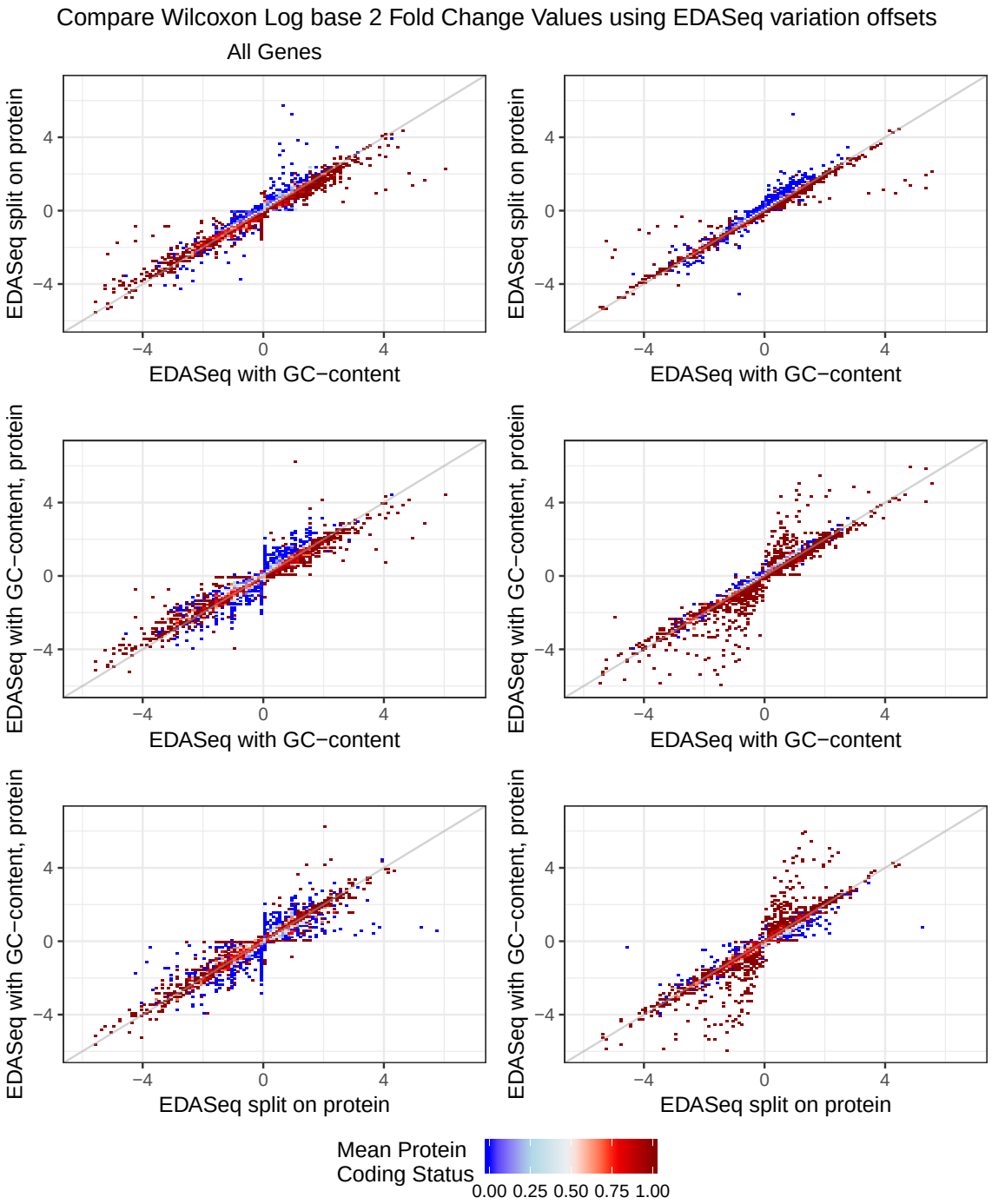


Figure 0.41: Comparison of Wilcoxon Rank Sum Test results using EDASeq variations



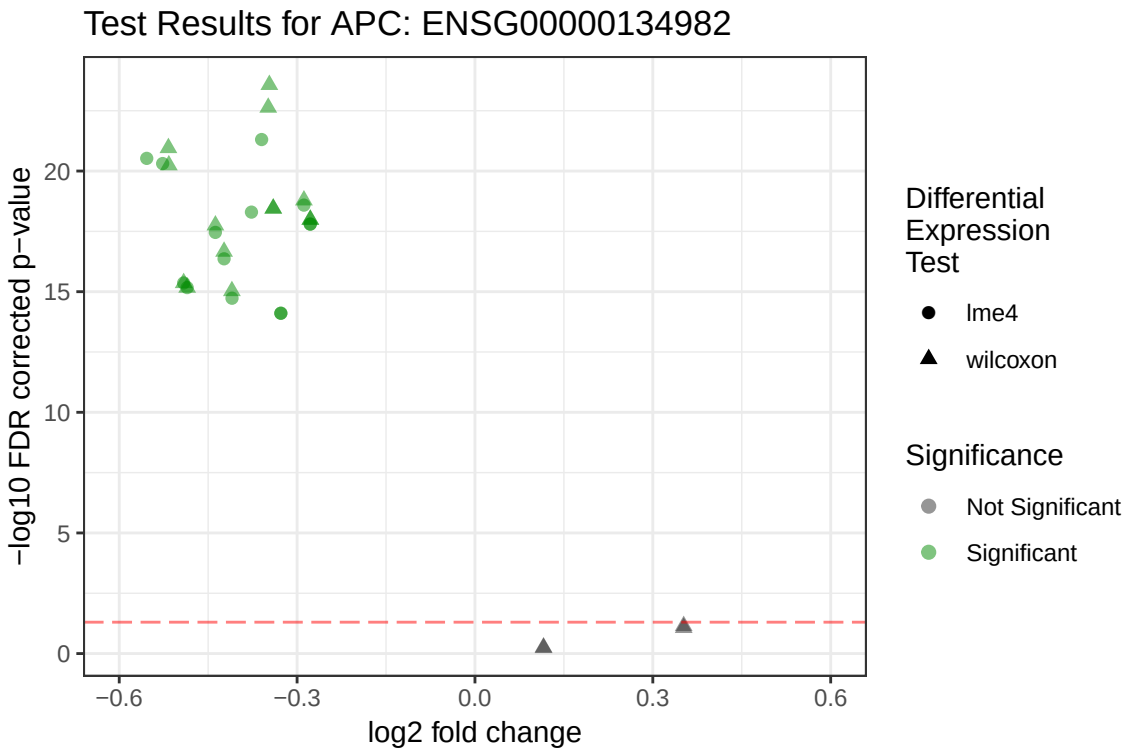
Genes of Interest

As discussed in the "Motivating Example" section of chapter 1, the WNT signaling pathway gene APC is believed to be down-regulated in colorectal cancer (Nagase and Nakamura 1993), and the gene CTNNB1 is expected to be up-regulated in colorectal cancer (Suzuki et al. 2004).

APC

The Ensembl gene ID (used here) for APC is ENSG00000134982 (Zerbino et al. 2017). A comparison of the results for APC from every method previously discussed is listed in Appendix A1, and shown graphically in Figure 0.42. The only tests that do not identify APC as significantly down-regulated in tumor samples are the Wilcoxon Rank Sum tests run on the raw log_2 read-count data or on RPKM normalized data, which was the approach used in the initial analysis. This is evidence that normalizing with respect to gene-level covariates improves down-stream differential expression analysis.

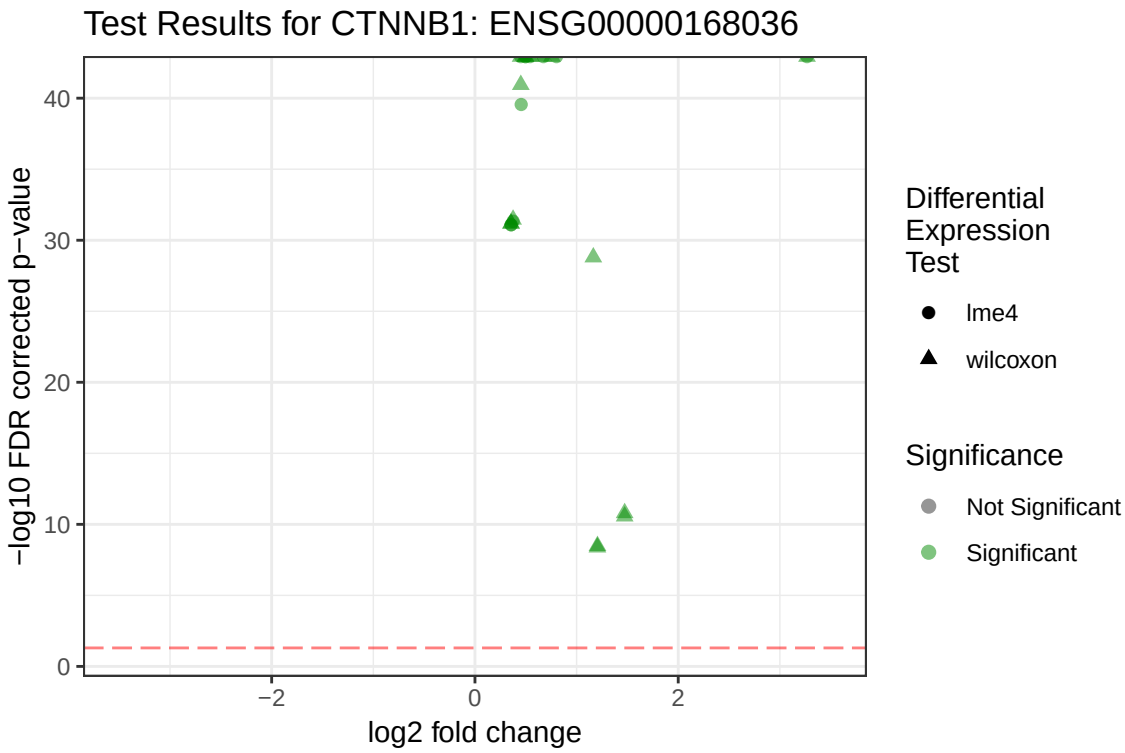
Figure 0.42: Comparison of All Test Results for APC



CTNNB1

The Ensembl gene ID (used here) for CTNNB1 is ENSG00000168036 (Zerbino et al. 2017). A comparison of the results for CTNNB1 from every method previously discussed is listed in Appendix A2, and shown graphically in Figure 0.43. Of the 32 combinations of normalization and differential expression analysis methods, all but five identified CTNNB1 as significantly up-regulated. The five combinations that did not identify CTNNB1 as significantly up-regulated were all failed `lme4` per-gene models using EDASeq variations: `lme4` with the full data set EDASeq offset, and `lme4` with the full and filtered data set EDASeq on both orderings of three covariates (see Appendix A2). Also note that many of the Wilcoxon Rank Sum tests produced FDR corrected p-values of 0, which are represented by the points at the top margin of Figure 0.43.

Figure 0.43: Comparison of All Test Results for CTNNB1



CHAPTER 5
CONCLUSION, RECOMMENDATIONS, AND FUTURE WORK

In conclusion, normalization is a critical part of differential expression analysis for high-dimensional read-count data, and accounting for gene-level covariates reduces bias and false discoveries. It is best to know during the experimental design stage that these gene-level covariates will be used, so that the same values used in RNA-Seq can be used in normalization.

For other researchers wanting to account for gene-level covariates, I would suggest splitting RNA-Seq read-counts on protein coding status prior to using CQN normalization with GC-content and gene length. This allows all three covariates to be accounted for without having to extend splines to spline surfaces. I recommend CQN over EDASeq normalization because it removes the need for filtering, produces better-behaved offsets, and tends to produce fewer failed per-gene `lme4` models.

Some interesting ancillary findings include a weak negative correlation between gene length and GC-content. For the full data set the correlation coefficient is -0.254, and for the filtered data set the correlation coefficient is -0.2595. This weak correlation can be seen in Figures 0.44 and 0.45. Notice how the majority of the bimodality seen in Figure 0.44 is removed in Figure 0.45. It would be interesting to see which genes are in that cluster of short genes with fairly low GC-content. Also notice the fairly clean separation of the protein and non-protein coding genes in the colored density plots. Protein coding genes appear to be much longer than non-protein coding genes, on average.

Possible extensions of this project could include the extending of additional normalization methods to include binary covariates, applying the

new normalization methods discussed here to additional high-dimensional read-count data sets, or working closely with a geneticist to establish a biological basis for the inclusion of protein coding status as a normalization covariate.

Figure 0.44: Comparison of log(gene length) and GC-content values for all genes

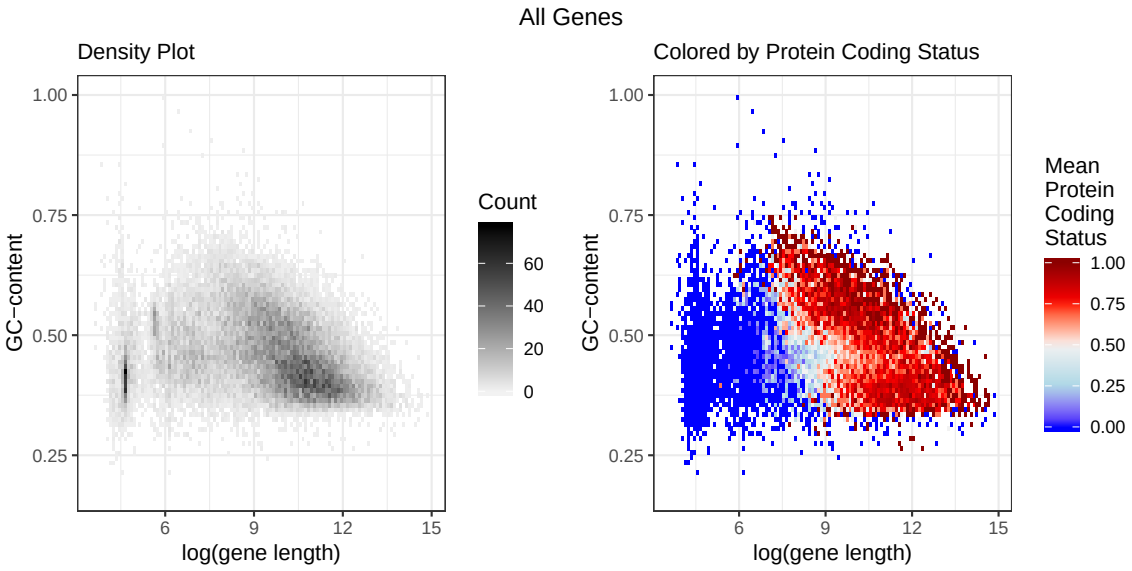
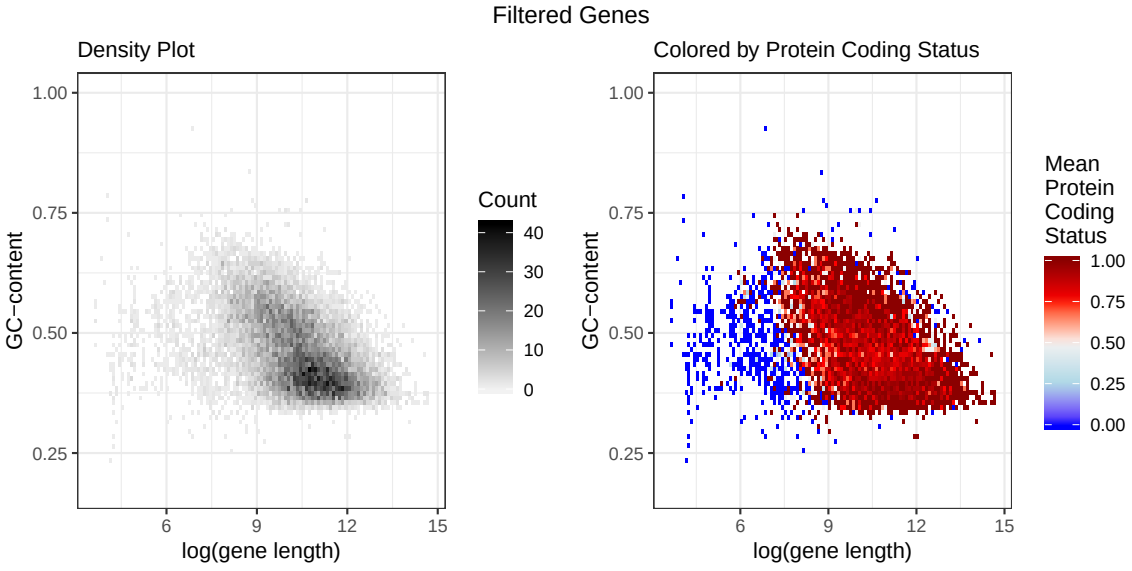


Figure 0.45: Comparison of log(gene length) and GC-content values for filtered genes



Note that R code for this thesis is provided in Appendix B.

REFERENCES

Auguie, Baptiste. 2017. *GridExtra: Miscellaneous Functions for "Grid" Graphics*. https://CRAN.R-project.org/package=gridExtra.

Barton, Stephen W. 2016. "Tutorial for Using the Center for High Performance Computing at the University of Utah and an Example Using Random Forest." PhD thesis, Utah State University. https://digitalcommons.usu.edu/gradreports/873.

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. doi:10.18637/jss.v067.i01.

Bengtsson, H. 2004. "aroma - An R Object-Oriented Microarray Analysis Environment." Preprint in Mathematical Sciences 18. Mathematical Statistics, Centre for Mathematical Sciences, Lund University, Sweden. http://www1.maths.lth.se/bioinformatics/publications/.

Bullard, James H, Elizabeth Purdom, Kasper D Hansen, and Sandrine Dudoit. 2010. "Evaluation of Statistical Methods for Normalization and Differential Expression in mRNA-Seq Experiments." *BMC Bioinformatics* 11 (1): 94. doi:10.1186/1471-2105-11-94.

Corporation, Microsoft, and Steve Weston. 2017. *DoParallel: Foreach Parallel Adaptor for the 'Parallel' Package*. https://CRAN.R-project.org/package=doParallel.

Hansen, K.D., R. A. Irizarry, and Z. Wu. 2012. "Removing technical variability in RNA-seq data using conditional quantile normalization." *Biostatistics* 13 (2):

204–16.

Hothorn, Torsten, Kurt Hornik, Mark A. van de Wiel, and Achim Zeileis. 2008. "Implementing a Class of Permutation Tests: The coin Package." *Journal of Statistical Software* 28 (8): 1–23. http://www.jstatsoft.org/v28/i08/.

Huber, W., V. J. Carey, R. Gentleman, S. Anders, M. Carlson, B. S. Carvalho, H. C. Bravo, et al. 2015. "Orchestrating High-Throughput Genomic Analysis with Bioconductor." *Nature Methods* 12 (2): 115–21. http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html.

"Illumina: RNA Sequencing Data Analysis Solutions." San Deigo, California. Illumina. https://www.illumina.com/informatics/sequencing-data-analysis/rna.html.

Kassambara, Alboukadel. 2018. *Ggpubr: 'Ggplot2' Based Publication Ready Plots*. https://CRAN.R-project.org/package=ggpubr.

Kent, W. J., C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and A. D. Haussler. 2002. "The Human Genome Browser at Ucsc." *Genome Research* 12 (6): 996–1006. doi:10.1101/gr.229102.

Love, Michael. 2016. "Batch Effects and Gc Content." *Bio Data Science*. GitHub. https://biodatascience.github.io/compbio/dist/batch.html.

Love, Michael, Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for Rna-Seq Data with Deseq2." *Genome Biology* 15 (12): 550. doi:10.1186/s13059-014-0550-8.

Nagase, Hiroki, and Yusuke Nakamura. 1993. "Mutations of the Apc Adenomatous Polyposis Coli) Gene." *Human Mutation* 2 (6): 425–34. doi:10.1002/humu.1380020602.

R Core Team. 2018. *R: A Language and Environment for Statistical Com-*

puting. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Risso, Davide, Katja Schwartz, Gavin Sherlock, and Sandrine Dudoit. 2011. "GC-Content Normalization for RNA-Seq Data." *BMC Bioinformatics* 12 (1): 480.

Robinson, Mark D, Davis J McCarthy, and Gordon K Smyth. 2010. "EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40.

Segditsas, S, and I Tomlinson. 2006. "Colorectal Cancer and Genetic Alterations in the Wnt Pathway." *Oncogene* 25 (57): 7531–7. doi:10.1038/sj.onc.1210059.

Shotwell, Matt. 2014. *Sas7bdat: SAS Database Reader (Experimental)*. https://CRAN.R-project.org/package=sas7bdat.

Slattery, Martha L., Jennifer S. Herrick, Lila E. Mullany, Wade S. Samowitz, John R. Sevens, Lori Sakoda, and Roger K. Wolff. 2017. "The Co-Regulatory Networks of Tumor Suppressor Genes, Oncogenes, and miRNAs in Colorectal Cancer." *Genes, Chromosomes and Cancer* 56 (11): 769–87. doi:10.1002/gcc.22481.

Stekhoven, Daniel J. 2013. *MissForest: Nonparametric Missing Value Imputation Using Random Forest*.

Stekhoven, Daniel J., and Peter Buehlmann. 2012. "MissForest - Non-Parametric Missing Value Imputation for Mixed-Type Data." *Bioinformatics* 28 (1). Oxford Univ Press: 112–18.

Suzuki, Hiromu, D Neil Watkins, Kam-Wing Jair, Kornel E Schuebel, Sanford D Markowitz, Wei Dong Chen, Theresa P Pretlow, et al. 2004. "Epigenetic Inactivation of Sfrp Genes Allows Constitutive Wnt Signaling in Colorectal Can-

cer." *Nature Genetics* 36 (4): 417–22. doi:10.1038/ng1330.

"University of Virginia Library Research Data Services Sciences." n.d. *Research Data Services Sciences*. University of Virginia. https://data.library.virginia.edu/the-wilcoxon-rank-sum-test/.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with S*. Fourth. New York: Springer. http://www.stats.ox.ac.uk/pub/MASS4.

Wang, Zhong, Mark Gerstein, and Michael Snyder. 2009. "RNA-Seq: A Revolutionary Tool for Transcriptomics." *Nature Reviews Genetics* 10 (1): 57–63. doi:10.1038/nrg2484.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

Wickham, Hadley, and Yue Hue. n.d. *Bigvis: Tools for Visualisation of Big Data Sets*.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2018. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, Jim Hester, and Romain Francois. 2017. *Readr: Read Rectangular Text Data*. https://CRAN.R-project.org/package=readr.

Wilke, Claus O. 2018. *Cowplot: Streamlined Plot Theme and Plot Annotations for 'Ggplot2'*. https://CRAN.R-project.org/package=cowplot.

Zerbino, Daniel R, Premanand Achuthan, Wasiu Akanni, M Ridwan Amode, Daniel Barrell, Jyothish Bhai, Konstantinos Billis, et al. 2017. "Ensembl 2018." *Nucleic Acids Research* 46 (D1). doi:10.1093/nar/gkx1098.

APPENDICES

APPENDIX A1

ENSG00000134982	**log2 Fold Change**	**FDR Corrected P-Value**	**Sig.**
Wilcoxon on log2 raw read-counts	0.3520705	0.08586423	0
Wilcoxon on log2 filtered raw read-counts	0.3520705	0.07122071	0
Wilcoxon on log2 RPKM values	0.1157507	0.5826143	0
Wilcoxon on log2 filtered RPKM values	0.1155894	0.5482018	0
Wilcoxon on CQN normalized read-counts	-0.4231423	4.348134e-17	-1
Wilcoxon on filtered CQN normalized read-counts	-0.4231514	2.17874e-17	-1
Wilcoxon on CQN split on protein normalized read-counts	-0.4378743	3.49284e-18	-1
Wilcoxon on filtered CQN split on protein normalized read-counts	-0.4377934	1.773966e-18	-1
Wilcoxon on CQN jittered protein normalized read-counts	-0.4097811	1.869163e-15	-1
Wilcoxon on filtered CQN jittered protein normalized read-counts	-0.4099624	9.36919e-16	-1
Wilcoxon on EDASeq normalized read-counts	-0.553612	2.987339e-21	-1
Wilcoxon on filtered EDASeq normalized read-counts	-0.5163342	5.721432e-21	-1
Wilcoxon on EDASeq split on protein normalized read-counts	-0.5267559	4.905526e-21	-1
Wilcoxon on filtered EDASeq split on protein normalized read-counts	-0.5170426	1.105086e-21	-1
Wilcoxon on EDASeq GC then protein normalized read-counts	-0.4854175	6.763833e-16	-1
Wilcoxon on filtered EDASeq GC then protein normalized read-counts	-0.4913302	4.355035e-16	-1
Wilcoxon on EDASeq protein then GC normalized read-counts	-0.4854175	6.763833e-16	-1
Wilcoxon on filtered EDASeq protein then GC normalized read-counts	-0.4913302	4.355035e-16	-1
lme4 with CQN offset	-0.277786336895461	1.5811139764114e-18	-1
lme4 with filtered CQN offset	-0.277691661305648	1.03053005912279e-18	-1
lme4 with CQN split on protein offset	-0.288489282643725	2.58763537217111e-19	-1
lme4 with filtered CQN split on protein offset	-0.288437527826866	1.63175811218859e-19	-1
lme4 with CQN jittered protein offset	-0.273451482858003	4.53282841024337e-17	-1
lme4 with filtered CQN jittered protein offset	-0.273563142806014	2.86939288532285e-17	-1

lme4 with EDASeq offset	-0.376966561991754	5.04332424137294e-19	-1
lme4 with filtered EDASeq offset	-0.348675567359156	2.33827933029916e-23	-1
lme4 with EDASeq split on protein offset	-0.359842083280969	4.95496111770556e-22	-1
lme4 with filtered EDASeq split on protein offset	-0.346796838422889	2.66474603646788e-24	-1
lme4 with EDASeq GC then protein offset	-0.327412739384651	7.8349871079098e-15	-1
lme4 with filtered EDASeq GC then protein offset	-0.340213110358308	3.55867800546103e-19	-1
lme4 with EDASeq protein then GC offset	-0.327412739384651	7.83649963605301e-15	-1
lme4 with filtered EDASeq protein then GC offset	-0.340213066393474	3.55623420622907e-19	-1

APPENDIX A2

ENSG00000168036	**log2 Fold Change**	**FDR Corrected P-Value**	**Sig.**
Wilcoxon on log2 raw read-counts	1.47278	2.725906e-11	1
Wilcoxon on log2 filtered raw read-counts	11.47278	1.589349e-11	1
Wilcoxon on log2 RPKM values	1.204727	4.156204e-09	1
Wilcoxon on log2 filtered RPKM values	1.20543070136837	3.29264611282794e-09	1
Wilcoxon on CQN normalized read-counts	0.449429	0	1
Wilcoxon on filtered CQN normalized read-counts	0.4490535	0	1
Wilcoxon on CQN split on protein normalized read-counts	0.4997162	0	1
Wilcoxon on filtered CQN split on protein normalized read-counts	0.4997402	0	1
Wilcoxon on CQN jittered protein normalized read-counts	0.495035	0	1
Wilcoxon on filtered CQN jittered protein normalized read-counts	0.4944717	0	1
Wilcoxon on EDASeq normalized read-counts	0.8019379	0	1
Wilcoxon on filtered EDASeq normalized read-counts	0.7568954	0	1
Wilcoxon on EDASeq split on protein normalized read-counts	0.5425325	0	1
Wilcoxon on filtered EDASeq split on protein normalized read-counts	0.5500703	0	1
Wilcoxon on EDASeq GC then protein normalized read-counts	0.673922	0	1
Wilcoxon on filtered EDASeq GC then protein normalized read-counts	3.266908	0	1
Wilcoxon on EDASeq protein then GC normalized read-counts	0.673922	0	1
Wilcoxon on filtered EDASeq protein then GC normalized read-counts	3.266908	0	1
lme4 with CQN offset	0.355755761922771	8.17254126631267e-32	1
lme4 with filtered CQN offset	0.355045799948538	6.60640991209145e-32	1
lme4 with CQN split on protein offset	0.375518556747636	4.59499427070807e-32	1
lme4 with filtered CQN split on protein offset	0.375008150812017	3.47693198650836e-32	1
lme4 with CQN jittered protein offset	0.39921807499461	3.681596565832e-40	1
lme4 with filtered CQN jittered protein offset	0.399090546432636	3.09166850727921e-40	1
lme4 with EDASeq offset	NA	NA	NA

lme4 with filtered EDASeq offset	1.16447509833456	1.57862927372398e-29	1
lme4 with EDASeq split on protein offset	0.454865039071717	2.79012809443175e-40	1
lme4 with filtered EDASeq split on protein offset	0.451191478641428	1.14372351316666e-41	1
lme4 with EDASeq GC then protein offset	NA	NA	NA
lme4 with filtered EDASeq GC then protein offset	NA	NA	NA
lme4 with EDASeq protein then GC offset	NA	NA	NA
lme4 with filtered EDASeq protein then GC offset	NA	NA	NA

APPENDIX B

Load Libraries

```
# For reading in data
library(readr)
library(sas7bdat)
# For manipulating data
library(dplyr)
# For imputation
library(missForest)
# For normalization calculations
library(cqn)
library(EDASeq)
# For differentail expression analysis
library(lme4)
library(MASS)
library(edgeR)
library(DESeq2)
# For graphics
library(ggplot2)
library(bigvis)
library(gridExtra)
library(ggpubr)
library(cowplot)
# For parallel
library(doParallel)
```

Convert SAS data file to .csv

```
counts <- read.sas7bdat("C:/Users/Lauren/Documents/Research/Data Sets/count.sas7bdat")
#write.csv(counts, file = "C:/Users/Lauren/Documents/Research/Data Sets/count.csv")

counts <- read.csv("C:/Users/Lauren/Documents/Research/Data Sets/count.csv")
```

Covariate Data Set

Data pulled from Ensembl Database Build GRCh37 (Zerbino et al. 2017)

```
geneLevel <- as.data.frame(read.table(file =
                                        "C:/Users/Lauren/Documents/Research/Data Sets/sequences from USCS 37/geneLevelInfoEnsembl37.txt",
                                      sep = ",", header = TRUE))

geneLevel2 <- mutate(geneLevel, geneLevel$Gene.end..bp. - geneLevel$Gene.start..bp. + 1)
geneLevel2 <- mutate(geneLevel2, geneLevel2$Gene...GC.content * 0.01)
geneLevel2 <- geneLevel2[, c(1, 2, 7, 8)]
colnames(geneLevel2) <- c("geneName", "geneID", "geneLength", "GCcont")
head(geneLevel2)

p1 <- ggplot() +
  geom_histogram(aes(x = geneLevel2$geneLength), binwidth = 1000) +
  theme_bw() + labs(x = "Gene length")
p2 <- ggplot() +
  geom_histogram(aes(x = log(geneLevel2$geneLength)), binwidth = 0.1) +
  theme_bw() + labs(x = "log(gene length)")
p3 <- ggplot() +
  geom_histogram(aes(x = geneLevel2$GCcont), binwidth = 0.01) +
  theme_bw() + labs(x = "GC-content")
p4 <- ggplot() +
  geom_point(aes(x = log(geneLevel2$geneLength), y = geneLevel2$GCcont), cex = 0.5) +
  theme_bw() + labs(x = "log(gene length)", y = "GC-content")
p5 <- autoplot(condense(x = bin(log(geneLevel2$geneLength), width = 0.1),
                        y = bin(geneLevel2$GCcont, width = 0.01))) +
  theme_bw() + labs(x = "log(gene length)", y = "GC-content")
grid.arrange(p2, p3, p4, p5, ncol = 2,
             top = "Distribuitons of Covariates Pulled from Ensembl GRCh37")
```

Imputation code

On 5000 of the 28335 known genes

```
knownECovCounts <- merge(geneLevel2, countsOG, by.x = "geneID", by.y = "EnsemblId", all = FALSE)
dim(knownECovCounts)
knownECovCounts$Protein <- ifelse(knownECovCounts$Protein == "Yes", 1, 0)
rownames(knownECovCounts) <- knownECovCounts$geneID
knownECovCounts <- knownECovCounts[, -c(1, 2, 5)]
knownECovCounts[1:5, 1:8]
#write.csv(knownECovCounts, file ="C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/knownEnsCovCounts.csv")

# Pull Genes to Impute
set.seed(206)
```

```
pulled1 <- sample(nrow(knownECovCounts), 5000, replace = FALSE)
# Save True Values
known1 <- knownECovCounts[pulled1, ]
#write.csv(known1, "~/Research/Summer Week 04 Restart/known1.csv")
# Remove Pulled Genes
impute1 <- as.matrix(knownECovCounts)
impute1[pulled1, 1:2] <- NA
impute1[1:10, 1:5]

forest1 <- missForest(xmis = impute1, verbose = TRUE, parallelize = "no")
forest1$OOBerror # NRMSE 0.6404718
imputedValues1 <- forest1$ximp
write.csv(imputedValues1, file = "~/Research/Summer Week 04 Restart/forest1.csv")

r1.1 <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  geom_point(aes(x = knownECovCounts[pulled1, "geneLength"],
                 y = imp1[pulled1, "geneLength"]), cex = 0.5) +
  labs(x = "Known gene length", y = "Imputed gene length") + theme_bw()
r1.2 <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  geom_point(aes(x = log(knownECovCounts[pulled1, "geneLength"]),
                 y = log(imp1[pulled1, "geneLength"])), cex = 0.5) +
  labs(x = "log(known gene length)", y = "log(imputed gene length)") + theme_bw()
GC1 <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  geom_point(aes(x = knownECovCounts[pulled1, "GCcont"],
                 y = imp1[pulled1, "GCcont"]), cex = 0.5) +
  labs(x = "Known GC-content", y = "Imputed GC-content") +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + theme_bw()
grid.arrange(r1.1, r1.2, GC1, ncol = 1)
```

Robustness Test

Run the same code on 4 additional random samples of 5,000

```
pullSetUp <- function(countsCov, seed, size) {
  set.seed(seed)
  pulled <- sample(nrow(countsCov), size, replace = FALSE)
  known <- countsCov[pulled, 1:2]
  toImpute <- as.matrix(countsCov)
  toImpute[pulled, 1:2] <- NA
  return(list(known, toImpute))
}

try2 <- pullSetUp(knownECovCounts, 352, 5000)
forest2 <- missForest(xmis = try2[[2]], verbose = TRUE, parallelize = "no")
oob2 <- forest2$OOBerror # NRMSE 0.02766506
write.csv(forest2$ximp[, 1:2], "impValues2.csv")
write.csv(try2[[1]], "known2.csv")

try3 <- pullSetUp(knownECovCounts, 353, 5000)
forest3 <- missForest(xmis = try3[[2]], verbose = TRUE, parallelize = "no")
oob3 <- forest3$OOBerror # NRMSE 0.02766506
write.csv(forest3$ximp[, 1:3], "impValues3.csv")
write.csv(try3[[1]], "known3.csv")

try4 <- pullSetUp(knownECovCounts, 354, 5000)
forest4 <- missForest(xmis = try4[[2]], verbose = TRUE, parallelize = "no")
oob4 <- forest4$OOBerror # NRMSE 0.02910902
write.csv(forest4$ximp[, 1:3], "impValues4.csv")
write.csv(try4[[1]], "known4.csv")

try5 <- pullSetUp(knownECovCounts, 355, 5000)
forest5 <- missForest(xmis = try5[[2]], verbose = TRUE, parallelize = "no")
oob5 <- forest5$OOBerror # NRMSE 0.02825448
write.csv(forest5$ximp[, 1:3], "impValues5.csv")
write.csv(try5[[1]], "known5.csv")

# known1 and imp1 read in above
known2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/known2.csv")
imp2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/impValues2.csv")
known3 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/known3.csv")
imp3 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/impValues3.csv")
known4 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/known4.csv")
imp4 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/impValues4.csv")
known5 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/known5.csv")
imp5 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 04 Restart/impValues5.csv")

plot1  <- merge(known1, imp1, by = "X", all = FALSE)
plot2  <- merge(known2, imp2, by = "X", all = FALSE)
plot3  <- merge(known3, imp3, by = "X", all = FALSE)
plot4  <- merge(known4, imp4, by = "X", all = FALSE)
plot5  <- merge(known5, imp5, by = "X", all = FALSE)

GLbase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "Known gene length", y = "Imputed gene length")
logGLbase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
```

```
  theme_bw() + labs(x = "log(known gene length)", y = "log(imputed gene length)")
r1.3 <- autoplot(condense(x = bin(plot1$geneLength.x, width = 10000),
                  y = bin(plot1$geneLength.y, width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "Known gene length", y = "Imputed gene length")
r1.4 <- autoplot(condense(x = bin(log(plot1$geneLength.x), width = 0.1),
                  y = bin(log(plot1$geneLength.y), width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "log(true gene length)", y = "log(imputed gene length)")
r2.1 <- GLbase + geom_point(aes(x = plot2$geneLength.x, y = plot2$geneLength.y), cex = 0.5)
r2.2 <- logGLbase + geom_point(aes(x = log(plot2$geneLength.x), y = log(plot2$geneLength.y)), cex = 0.5)
r2.3 <- autoplot(condense(x = bin(plot2$geneLength.x, width = 10000),
                  y = bin(plot2$geneLength.y, width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "Known gene length", y = "Imputed gene length")
r2.4 <- autoplot(condense(x = bin(log(plot2$geneLength.x), width = 0.1),
                  y = bin(log(plot2$geneLength.y), width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "log(true gene length)", y = "log(imputed gene length)")
r3.1 <- GLbase + geom_point(aes(x = plot3$geneLength.x, y = plot3$geneLength.y), cex = 0.5)
r3.2 <- logGLbase + geom_point(aes(x = log(plot3$geneLength.x), y = log(plot3$geneLength.y)), cex = 0.5)
r3.3 <- autoplot(condense(x = bin(plot3$geneLength.x, width = 10000), y = bin(plot3$geneLength.y, width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "True gene length", y = "Imputed gene length")
r3.4 <- autoplot(condense(x = bin(log(plot3$geneLength.x), width = 0.1), y = bin(log(plot3$geneLength.y), width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "log(true gene length)", y = "log(imputed gene length)")
r4.1 <- GLbase + geom_point(aes(x = plot4$geneLength.x, y = plot4$geneLength.y), cex = 0.5)
r4.2 <- logGLbase + geom_point(aes(x = log(plot4$geneLength.x), y = log(plot4$geneLength.y)), cex = 0.5)
r4.3 <- autoplot(condense(x = bin(plot4$geneLength.x, width = 10000), y = bin(plot4$geneLength.y, width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "True gene length", y = "Imputed gene length")
r4.4 <- autoplot(condense(x = bin(log(plot4$geneLength.x), width = 0.1), y = bin(log(plot4$geneLength.y), width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "log(true gene length)", y = "log(imputed gene length)")
r5.1 <- GLbase + geom_point(aes(x = plot5$geneLength.x, y = plot5$geneLength.y), cex = 0.5)
r5.2 <- logGLbase + geom_point(aes(x = log(plot5$geneLength.x), y = log(plot5$geneLength.y)), cex = 0.5)
r5.3 <- autoplot(condense(x = bin(plot5$geneLength.x, width = 10000), y = bin(plot5$geneLength.y, width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "True gene length", y = "Imputed gene length")
r5.4 <- autoplot(condense(x = bin(log(plot5$geneLength.x), width = 0.1), y = bin(log(plot5$geneLength.y), width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "log(true gene length)", y = "log(imputed gene length)")
grid.arrange(r1.1, r1.3, r2.1, r2.3, r3.1, r3.3, r4.1, r4.3, r5.1, r4.3, ncol = 2)
grid.arrange(r1.2, r1.4, r2.2, r2.4, r3.2, r3.4, r4.2, r4.4, r5.2, r4.4, ncol = 2)
GCbase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
r1.3 <- autoplot(condense(x = bin(plot1$GCcont.x, width = 0.01), y = bin(plot1$GCcont.y, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
r2.1 <- GCbase + geom_point(aes(x = plot2$GCcont.x, y = plot2$GCcont.y), cex = 0.5)
r2.3 <- autoplot(condense(x = bin(plot2$GCcont.x, width = 0.01), y = bin(plot2$GCcont.y, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
r3.1 <- GCbase + geom_point(aes(x = plot3$GCcont.x, y = plot3$GCcont.y), cex = 0.5)
r3.3 <- autoplot(condense(x = bin(plot3$GCcont.x, width = 0.01), y = bin(plot3$GCcont.y, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
r4.1 <- GCbase + geom_point(aes(x = plot4$GCcont.x, y = plot4$GCcont.y), cex = 0.5)
r4.3 <- autoplot(condense(x = bin(plot4$GCcont.x, width = 0.01), y = bin(plot4$GCcont.y, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
r5.1 <- GCbase + geom_point(aes(x = plot5$GCcont.x, y = plot5$GCcont.y), cex = 0.5)
r5.3 <- autoplot(condense(x = bin(plot5$GCcont.x, width = 0.01), y = bin(plot5$GCcont.y, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Imputed GC-content")
grid.arrange(GC1, r1.3, r2.1, r2.3, r3.1, r3.3, r4.1, r4.3, r5.1, r5.3, ncol = 2)
```

Linear Model Adjustments on Imputed GC-Content Values

```
set.seed(321)
half1 <- sample(nrow(known1), nrow(known1)/2, replace = FALSE)
test1 <- plot1[half1, ]; train1 <- plot1[-half1, ]
set.seed(322)
half2 <- sample(nrow(known2), nrow(known2)/2, replace = FALSE)
test2 <- plot2[half2, ]; train2 <- plot2[-half2, ]
set.seed(323)
half3 <- sample(nrow(known3), nrow(known3)/2, replace = FALSE)
test3 <- plot3[half3, ]; train3 <- plot3[-half3, ]
set.seed(324)
half4 <- sample(nrow(known4), nrow(known4)/2, replace = FALSE)
test4 <- plot4[half4, ]; train4 <- plot4[-half4, ]
set.seed(325)
half5 <- sample(nrow(known5), nrow(known5)/2, replace = FALSE)
test5 <- plot5[half5, ]; train5 <- plot5[-half5, ]

lm1 <- lm(formula = GCcont.x ~ GCcont.y, data = train1)
lm2 <- lm(formula = GCcont.x ~ GCcont.y, data = train2)
lm3 <- lm(formula = GCcont.x ~ GCcont.y, data = train3)
```

```
lm4 <- lm(formula = GCcont.x ~ GCcont.y, data = train4)
lm5 <- lm(formula = GCcont.x ~ GCcont.y, data = train5)

lm1.1 <- predict(lm1, newdata = test1)
lm2.1 <- predict(lm2, newdata = test2)
lm3.1 <- predict(lm3, newdata = test3)
lm4.1 <- predict(lm4, newdata = test4)
lm5.1 <- predict(lm5, newdata = test5)

adjBase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + xlim(0.11, 0.9) + ylim(0.11, 0.9) +
  labs(x = "True GC-content", y = "Ajusted GC-content")
adj1  <- adjBase + geom_point(aes(x = test1$GCcont.x, y = lm1.1), cex = 0.5)
adj2  <- adjBase + geom_point(aes(x = test2$GCcont.x, y = lm2.1), cex = 0.5)
adj3  <- adjBase + geom_point(aes(x = test3$GCcont.x, y = lm3.1), cex = 0.5)
adj4  <- adjBase + geom_point(aes(x = test4$GCcont.x, y = lm4.1), cex = 0.5)
adj5  <- adjBase + geom_point(aes(x = test5$GCcont.x, y = lm5.1), cex = 0.5)
bv1 <- autoplot(condense(x = bin(test1$GCcont.x, width = 0.01), y = bin(lm1.1, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Ajusted GC-content")
bv2 <- autoplot(condense(x = bin(test2$GCcont.x, width = 0.01), y = bin(lm2.1, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Ajusted GC-content")
bv3 <- autoplot(condense(x = bin(test3$GCcont.x, width = 0.01), y = bin(lm3.1, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Ajusted GC-content")
bv4 <- autoplot(condense(x = bin(test4$GCcont.x, width = 0.01), y = bin(lm4.1, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Ajusted GC-content")
bv5 <- autoplot(condense(x = bin(test5$GCcont.x, width = 0.01), y = bin(lm5.1, width = 0.01))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  xlim(0.11, 0.9) + ylim(0.11, 0.9) + labs(x = "True GC-content", y = "Ajusted GC-content")
grid.arrange(adj1, bv1, adj2, bv2, adj3, bv3, adj4, bv4, adj5, bv5, ncol = 2)

lm1; lm2; lm3; lm4; lm5

interceptGC <- mean(lm1$coefficients[1], lm2$coefficients[1], lm3$coefficients[1],
                    lm4$coefficients[1], lm5$coefficients[1])
slopeGC <- mean(lm1$coefficients[2], lm2$coefficients[2], lm3$coefficients[2],
                lm4$coefficients[2], lm5$coefficients[2])
interceptGC; slopeGC
```

Linear Model Adjustments on Imputed Gene Length Values (log scale)

```
set.seed(1131)
half1 <- sample(nrow(known1), nrow(known1)/2, replace = FALSE)
test1 <- plot1[half1, ]; train1 <- plot1[-half1, ]
set.seed(1132)
half2 <- sample(nrow(known2), nrow(known2)/2, replace = FALSE)
test2 <- plot2[half2, ]; train2 <- plot2[-half2, ]
set.seed(1133)
half3 <- sample(nrow(known3), nrow(known3)/2, replace = FALSE)
test3 <- plot3[half3, ]; train3 <- plot3[-half3, ]
set.seed(1134)
half4 <- sample(nrow(known4), nrow(known4)/2, replace = FALSE)
test4 <- plot4[half4, ]; train4 <- plot4[-half4, ]
set.seed(1135)
half5 <- sample(nrow(known5), nrow(known5)/2, replace = FALSE)
test5 <- plot5[half5, ]; train5 <- plot5[-half5, ]

lm1 <- lm(formula = log(geneLength.x) ~ log(geneLength.y), data = train1)
lm2 <- lm(formula = log(geneLength.x) ~ log(geneLength.y), data = train2)
lm3 <- lm(formula = log(geneLength.x) ~ log(geneLength.y), data = train3)
lm4 <- lm(formula = log(geneLength.x) ~ log(geneLength.y), data = train4)
lm5 <- lm(formula = log(geneLength.x) ~ log(geneLength.y), data = train5)

lm1.1 <- predict(lm1, newdata = test1)
lm2.1 <- predict(lm2, newdata = test2)
lm3.1 <- predict(lm3, newdata = test3)
lm4.1 <- predict(lm4, newdata = test4)
lm5.1 <- predict(lm5, newdata = test5)

adjBase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "True gene length", y = "Ajusted gene length")
adj1 <- adjBase + geom_point(aes(x = test1$geneLength.x, y = exp(lm1.1)), cex = 0.5)
adj2 <- adjBase + geom_point(aes(x = test2$geneLength.x, y = exp(lm2.1)), cex = 0.5)
adj3 <- adjBase + geom_point(aes(x = test3$geneLength.x, y = exp(lm3.1)), cex = 0.5)
adj4 <- adjBase + geom_point(aes(x = test4$geneLength.x, y = exp(lm4.1)), cex = 0.5)
adj5<- adjBase + geom_point(aes(x = test5$geneLength.x, y = exp(lm5.1)), cex = 0.5)
bv1 <- autoplot(condense(x = bin(test1$geneLength.x, width = 10000), y = bin(exp(lm1.1), width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bv2 <- autoplot(condense(x = bin(test2$geneLength.x, width = 10000), y = bin(exp(lm2.1), width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bv3 <- autoplot(condense(x = bin(test3$geneLength.x, width = 10000), y = bin(exp(lm3.1), width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
```

```
bv4 <- autoplot(condense(x = bin(test4$geneLength.x, width = 10000), y = bin(exp(lm4.1), width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bv5 <- autoplot(condense(x = bin(test5$geneLength.x, width = 10000), y = bin(exp(lm5.1), width = 10000))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")

adjLogBase <- ggplot() +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") +
  theme_bw() + labs(x = "True log(gene length)", y = "Ajusted log(gene length)")
adjL1 <- adjLogBase + geom_point(aes(x = log(test1$geneLength.x), y = lm1.1), cex = 0.5)
adjL2 <- adjLogBase + geom_point(aes(x = log(test2$geneLength.x), y = lm2.1), cex = 0.5)
adjL3 <- adjLogBase + geom_point(aes(x = log(test3$geneLength.x), y = lm3.1), cex = 0.5)
adjL4 <- adjLogBase + geom_point(aes(x = log(test4$geneLength.x), y = lm4.1), cex = 0.5)
adjL5 <- adjLogBase + geom_point(aes(x = log(test5$geneLength.x), y = lm5.1), cex = 0.5)
bvL1 <- autoplot(condense(x = bin(log(test1$geneLength.x), width = 0.1), y = bin(lm1.1, width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bvL2 <- autoplot(condense(x = bin(log(test2$geneLength.x), width = 0.1), y = bin(lm2.1, width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bvL3 <- autoplot(condense(x = bin(log(test3$geneLength.x), width = 0.1), y = bin(lm3.1, width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bvL4 <- autoplot(condense(x = bin(log(test4$geneLength.x), width = 0.1), y = bin(lm4.1, width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
bvL5 <- autoplot(condense(x = bin(log(test5$geneLength.x), width = 0.1), y = bin(lm5.1, width = 0.1))) +
  geom_abline(intercept = 0, slope = 1, col = "darkgray") + theme_bw() +
  labs(x = "True log(gene length)ent", y = "Adjusted log(gene length)")
grid.arrange(adjL1, bvL1, adjL2, bvL2, adjL3, bvL3, adjL4, bvL4, adjL5, bvL5, ncol = 2)
grid.arrange(adj1, bv1, adj2, bv2, adj3, bv3, adj4, bv4, adj5, bv5, ncol = 2)

lm1; lm2; lm3; lm4; lm5

interceptGL <- exp(mean(lm1$coefficients[1], lm2$coefficients[1], lm3$coefficients[1],
                  lm4$coefficients[1], lm5$coefficients[1]))
slopeGL <- exp(mean(lm1$coefficients[2], lm2$coefficients[2], lm3$coefficients[2],
              lm4$coefficients[2], lm5$coefficients[2]))
interceptGL; slopeGL
```

Apply *missForest* Imputation to 1885 Missing values

```
# Set Up
toImp <- geneLevel2
rownames(toImp) <- geneLevel2$geneID
toImp <- toImp[, c("geneLength", "GCcont")]
toImp <- merge(toImp, countsOG, by.x = "row.names", by.y = "EnsemblId", all = TRUE)
rownames(toImp) <- toImp$Row.names
toImp$Protein <- ifelse(toImp$Protein == "Yes", 1, 0)
toImp <- as.matrix(toImp[, -c(1, 4)])

# Imputation
forest <- missForest(xmis = toImp, verbose = TRUE, parallelize = "no")
forest$OOBerror
imputedEValues <- forest$ximp

# Linear Adjustments of GC-Content and Gene Length
missings <- as.numeric(rownames(missings))
# Separate what values were Imputed
toAdj <- imputedEValues[missings, ]
dim(toAdj) # should be 1885 rows
head(toAdj)

# Apply Adjustment
adjGC <- interceptGC + (slopeGC * toAdj$GCcont)
adjGL <- interceptGL + (slopeGL * toAdj$geneLength)
adjCov <- imputedEValues
adjCov[missings, "GCcont"] <- adjGC
adjCov[missings, "geneLength"] <- adjGL

# Compare
summary(imputedEValues[missings, "GCcont"])
summary(adjCov$GCcont)
summary(imputedEValues[missings, "geneLength"])
summary(adjCov$geneLength)
```

New Distributions

```
GC0 <- ggplot() +
  geom_histogram(aes(x = imputedEValues$GCcont), binwidth = 0.01) +
  theme_bw() + labs(x = "Known and imputed GC-content") + xlim(0.1, 1) + ylim(0, 1750)
GC1 <- ggplot() +
  geom_histogram(aes(x = imputedEValues[missings, "GCcont"]), binwidth = 0.01) +
  theme_bw() + labs(x = "Imputed GC-content") + xlim(0.1, 1) + ylim(0, 155)
GC2 <- ggplot() +
  geom_histogram(aes(x = adjGC), binwidth = 0.01) +
  theme_bw() + labs(x = "Adjusted GC-content") + xlim(0.1, 1) + ylim(0, 155)
GC3 <- ggplot() +
  geom_histogram(aes(x = adjCov$GCcont), binwidth = 0.01) +
```

```
  theme_bw() + labs(x = "Known and adjusted GC-content")  + xlim(0.1, 1) + ylim(0, 1750)
grid.arrange(GC0, GC3, GC1, GC2, ncol = 2)
GL0 <- ggplot() +
  geom_histogram(aes(x = imputedEValues$geneLength), binwidth = 10000) +
  theme_bw() + labs(x = "Known and imputed gene length") +
  xlim(-10000, 3e+06) + ylim(0, 10000)
GL1 <- ggplot() +
  geom_histogram(aes(x = imputedEValues[missings, "geneLength"]), binwidth = 10000) +
  theme_bw() + labs(x = "Imputed gene length") + xlim(-10000, 3e+06) + ylim(0, 1000)
GL2 <- ggplot() +
  geom_histogram(aes(x = adjGL), binwidth = 10000) +
  theme_bw() + labs(x = "Adjusted gene length") + xlim(-10000, 3e+06) + ylim(0, 1000)
GL3 <- ggplot() +
  geom_histogram(aes(x = adjCov$geneLength), binwidth = 10000) +
  theme_bw() + labs(x = "Known and adjusted gene length") +
  xlim(-10000, 3e+06) + ylim(0, 10000)
grid.arrange(GL0, GL3, GL1, GL2, ncol = 2)
lGL0 <- ggplot() +
  geom_histogram(aes(x = log(imputedEValues$geneLength)), binwidth = 0.25) +
  theme_bw() + labs(x = "log(known and imputed gene length)") + xlim(3.25, 15.25) + ylim(0, 1750)
lGL1 <- ggplot() +
  geom_histogram(aes(x = log(imputedEValues[missings, "geneLength"])), binwidth = 0.25) +
  theme_bw() + labs(x = "log(imputed gene length)") + xlim(3.25, 15.25) + ylim(0, 100)
lGL2 <- ggplot() +
  geom_histogram(aes(x = log(adjGL)), binwidth = 0.25) +
  theme_bw() + labs(x = "log(adjusted gene length)") + xlim(3.25, 15.25) + ylim(0, 100)
lGL3 <- ggplot() +
  geom_histogram(aes(x = log(adjCov$geneLength)), binwidth = 0.25) +
  theme_bw() + labs(x = "log(known and adjusted gene length)") + xlim(3.25, 15.25) + ylim(0, 1750)
grid.arrange(lGL0, lGL3, lGL1, lGL2, ncol = 2)
```

RPKM Values

$$RPKMv2 = \frac{log(ReadCount)}{\frac{GeneLength}{1000} * \frac{log(Numberofmappedreadsinthesample)}{1000000}}$$

Here the *rpkm* function for `edgeR` was used.

```
counts1 <- counts + 1
counts2 <- log2(counts1)
rpkmLog2 <- rpkm(counts1, gene.length = ensCov$geneLength,
                 normalized.lib.sizes = TRUE, log = TRUE)
rpkmLog2Fil <- rpkm(counts1[filter, ], gene.length = ensCov[filter, "geneLength"],
                 normalized.lib.sizes = TRUE, log = TRUE)
```

CQN

With GC-Content

Referred to as "cqnOG" throughout.

```
cqnAll <- cqn(counts = counts,
    x = ensCov$GCcont,
    lengths = ensCov$geneLength,
    lengthMethod = "smooth",
    sizeFactors = NULL, verbose = FALSE)
filter <- apply(counts, 1 , function(x) mean(x) > 10)
cqnFil <- cqn(counts = counts[filter, ],
    x = ensCov[filter, "GCcont"],
    lengths = ensCov[filter, "geneLength"],
    lengthMethod = "smooth",
    sizeFactors = NULL, verbose = FALSE)
```

Split on Protein Coding Status

```
# All Genes
covCounts <- merge(ensCov, counts, by = "row.names")
rownames(covCounts) <- covCounts$Row.names
covCounts <- covCounts[, -1]
covCounts1 <- covCounts[which(covCounts$Protein == 1), ]
covCounts0 <- covCounts[which(covCounts$Protein != 1), ]
# Filtered Genes
filterCC <- apply(covCounts[, 4:ncol(covCounts)], 1 , function(x) mean(x) > 10)
covCountsF <- covCounts[filterCC, ]
covCounts1F <- covCountsF[which(covCountsF$Protein == 1), ]
covCounts0F <- covCountsF[which(covCountsF$Protein != 1), ]

cqnAllPro <- cqn(counts = covCounts1[, 4:ncol(covCounts1)],
              x = covCounts1$GCcont,
              lengths = covCounts1$geneLength,
              lengthMethod = "smooth",
              sizeFactors = NULL, verbose = FALSE)
cqnAllNon <- cqn(counts = covCounts0[, 4:ncol(covCounts0)],
              x = covCounts0$GCcont,
              lengths = covCounts0$geneLength,
              lengthMethod = "smooth",
```

```
              sizeFactors = NULL, verbose = FALSE)

cqnFilPro <- cqn(counts = covCounts1F[, 4:ncol(covCounts1F)],
              x = covCounts1F$GCcont,
              lengths = covCounts1F$geneLength,
              lengthMethod = "smooth",
              sizeFactors = NULL, verbose = FALSE)
cqnFilNon <- cqn(counts = covCounts0F[, 4:ncol(covCounts0F)],
              x = covCounts0F$GCcont,
              lengths = covCounts0F$geneLength,
              lengthMethod = "smooth",
              sizeFactors = NULL, verbose = FALSE)

cqnAllProCounts <- cqnAllPro$y + cqnAllPro$offset
cqnAllNonCounts <- cqnAllNon$y + cqnAllNon$offset
cqnAllSplitCounts <- rbind(cqnAllProCounts, cqnAllNonCounts)
cqnAllSplitCounts <- cqnAllSplitCounts[order(rownames(cqnAllSplitCounts)), ]
write.csv(cqnAllSplitCounts, file = "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/cqnAllSplitNormCounts.csv")

cqnFilProCounts <- cqnFilPro$y + cqnFilPro$offset
cqnFilNonCounts <- cqnFilNon$y + cqnFilNon$offset
cqnFilSplitCounts <- rbind(cqnFilProCounts, cqnFilNonCounts)
cqnFilSplitCounts <- cqnFilSplitCounts[order(rownames(cqnFilSplitCounts)), ]
write.csv(cqnFilSplitCounts, file = "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/cqnFilSplitNormCounts.csv")
```

With Jittered Protein Coding Status

Test Effect of "Jittering" on 10,000 genes

```
set.seed(1245)
pull10 <- sample(30220, size = 10000, replace = FALSE)
# Counts and Covariates
counts10 <- counts[pull10, ]
ensCov10 <- ensCov[pull10, ]
genes <- read.csv("C:/Users/Lauren/Documents/Research/CHPC Materials/genes.csv")
genes10 <- genes[pull10, ]
# Effects
groups <- vector("numeric", length = ncol(counts))
for (i in 1:length(groups)){
  if (grepl("^T", colnames(counts[i]))){
    groups[i] <- 1 # tumor samples are group 1
  } else {
    groups[i] <- 0 # normal samples are group 0
  }
}
sampleID <- vector("numeric", length = ncol(counts))
for(i in 1:length(sampleID)){
  sampleID[i] <- as.numeric(gsub("^[[:upper:]]", "", colnames(counts[i])))
}
sampleID <- as.factor(sampleID)
effects10 <- as.data.frame(cbind(groups, sampleID, t(counts10)))
# Offsets
offsetCqn1 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 11 Edit/offsetCqnTest1.csv")
offsetCqn2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 11 Edit/offsetCqnTest2.csv")
offsetCqn3 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 11 Edit/offsetCqnTest3.csv")
offsetCqn4 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 11 Edit/offsetCqnTest4.csv")
# Save Data
write.csv(counts10, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/counts10.csv")
write.csv(ensCov10, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/ensCov10.csv")
write.csv(effects10, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/effects10.csv")
write.csv(genes10, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/genes10.csv")
write.csv(offsetCqn1[pull10, ], file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/offset10Cqn1.csv")
write.csv(offsetCqn2[pull10, ], file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/offset10Cqn2.csv")
write.csv(offsetCqn3[pull10, ], file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/offset10Cqn3.csv")
write.csv(offsetCqn4[pull10, ], file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/offset10Cqn4.csv")

#lme4 results
cqn1 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/cqn1Results.csv")
cqn2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/cqn2Results.csv")
cqn3 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/cqn3Results.csv")
cqn4 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/pull10/cqn4Results.csv")
# Compare Fixed Effect Estimates (synonymous with logFC values)
comp12 <- autoplot(condense(x = bin(cqn1$fixedEst, width = 0.05),
                  y = bin(cqn2$fixedEst, width = 0.05))) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-2)", y = "Epsilon = (0, 1e-3)") +
  scale_fill_gradient("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
comp13 <- autoplot(condense(x = bin(cqn1$fixedEst, width = 0.05),
                  y = bin(cqn3$fixedEst, width = 0.05))) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-2)", y = "Epsilon = (0, 1e-4)") +
  scale_fill_gradient2("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
comp14 <- autoplot(condense(x = bin(cqn1$fixedEst, width = 0.05),
                  y = bin(cqn4$fixedEst, width = 0.05))) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-2)", y = "Epsilon = (0, 1e-5)") +
  scale_fill_gradient2("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
comp23 <- autoplot(condense(x = bin(cqn2$fixedEst, width = 0.05),
                  y = bin(cqn3$fixedEst, width = 0.05))) +
```

```
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-3)", y = "Epsilon = (0, 1e-4)") +
  scale_fill_gradient2("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
comp24 <- autoplot(condense(x = bin(cqn2$fixedEst, width = 0.05),
                   y = bin(cqn4$fixedEst, width = 0.05))) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-3)", y = "Epsilon = (0, 1e-5)") +
  scale_fill_gradient2("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
comp34 <- autoplot(condense(x = bin(cqn3$fixedEst, width = 0.05),
                   y = bin(cqn4$fixedEst, width = 0.05))) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5), size = 0.1) +
  theme_bw() + labs(x = "Epsilon = (0, 1e-4)", y = "Epsilon = (0, 1e-5)") +
  scale_fill_gradient2("Count", breaks = seq(0, 500, 100), low = "gray95", high = "black")
ggarrange(comp12, comp23, comp13, comp24, comp14, comp34, nrow = 3, ncol = 2, common.legend = TRUE, legend = "right")
summary(cqn1[, 3:4])
summary(cqn2[, 3:4])
summary(cqn3[, 3:4])
summary(cqn4[, 3:4])
```

Apply Jitter and Run CQN

```
# Jitter and Run CQN
set.seed(922)
epsilon <- base::sample(seq(0, 1e-5, 1e-10), size = 30220, replace = TRUE)
proJ <- protein + epsilon

cqnJ <- cqn(counts = counts,
                    x = proJ,
                    lengths = ensCov$geneLength,
                    lengthMethod = "smooth")

#write.csv(cqnJ$glm.offset, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJoffset.csv")
cqnJCounts <- cqnJ$y + cqnJ$offset
#write.csv(cqnJCounts, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJCounts.csv")
filter <- apply(counts, 1 , function(x) mean(x) > 10)
cqnJFil <- cqn(counts = counts[filter, ],
                    x = proJ[filter],
                    lengths = ensCov[filter, "geneLength"],
                    lengthMethod = "smooth")

#write.csv(cqnJFil$glm.offset, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJFiloffset.csv")
cqnJFilCounts <- cqnJFil$y + cqnJFil$offset
write.csv(cqnJFilCounts, file = "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJFilCounts.csv")

# lme4 materials
groups <- vector("numeric", length = ncol(counts))
for (i in 1:length(groups)){
  if (grepl("^T", colnames(counts[i]))){
    groups[i] <- 1 # tumor samples are group 1
  } else {
    groups[i] <- 0 # normal samples are group 0
  }
}
sampleID <- vector("numeric", length = ncol(counts))
for(i in 1:length(sampleID)){
  sampleID[i] <- as.numeric(gsub("^[[:upper:]]", "", colnames(counts[i])))
}
sampleID <- as.factor(sampleID)
effects <- cbind(groups, sampleID, t(counts))
effectsFil <- cbind(groups, sampleID, t(counts[filter, ]))
write.csv(effects, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/effects.csv")
write.csv(effectsFil, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/effectsFil.csv")

# lme4 results
cqnJ <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJ/cqnJResultsraw.csv")[, -1]
cqnJFil <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJ/cqnJFilResultsraw.csv")[, -1]
cqnJ2 <- mutate(cqnJ, p.adjust(cqnJ$pval, method = "fdr"))
cqnJFil2 <- mutate(cqnJFil, p.adjust(cqnJFil$pval, method = "fdr"))
colnames(cqnJ2)[4] <- "FDR"
colnames(cqnJFil2)[4] <- "FDR"
#write.csv(cqnJ2, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJ/cqnJResults.csv")
#write.csv(cqnJFil2, "C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/cqnJ/cqnJFilResults.csv")

summary(cqnJ[, c("fixedEst", "pval", "FDR")])
summary(cqnJFil[, c("fixedEst", "pval", "FDR")])
```

EDASeq

With GC-Content

```
gc <- ensCov$GCcont
names(gc) <- rownames(ensCov)
length <- ensCov$geneLength
names(length) <- rownames(ensCov)
common  <- intersect(names(gc), rownames(counts))
feature <- data.frame(gc = gc, length = length)
dataAll <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                           featureData = feature[common, ],
                           phenoData = data.frame(
                             conditions = ifelse(groups == 1, "tumor", "normal"),
```

```
                            row.names = colnames(counts)))

dataWithin  <- withinLaneNormalization(dataAll, "gc", which = "full", offset = TRUE)
dataNormAll   <- betweenLaneNormalization(dataWithin, which = "full", offset = TRUE)
# Save Normalized Counts
write.csv(counts(dataNormAll), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsAll.csv")

gc <- ensCov[filter, "GCcont"]
names(gc) <- rownames(ensCov[filter, ])
length <- ensCov[filter, "geneLength"]
names(length) <- rownames(ensCov[filter, ])
common  <- intersect(names(gc), rownames(counts[filter, ]))
feature <- data.frame(gc = gc, length = length)
dataFil <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                            featureData = feature[common, ],
                            phenoData = data.frame(
                              conditions = ifelse(groups == 1, "tumor", "normal"),
                              row.names = colnames(counts)))

dataWithin  <- withinLaneNormalization(dataFil, "gc", which = "full", offset = TRUE)
dataNormFil   <- betweenLaneNormalization(dataWithin, which = "full", offset = TRUE)
# Save Normalized Counts
write.csv(normCounts(dataNormFil), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsFil.csv")
```

Split on Protein Coding Status

```
# All Genes
covCounts <- merge(ensCov, counts, by = "row.names")
rownames(covCounts) <- covCounts$Row.names
covCounts <- covCounts[, -1]
covCounts1 <- covCounts[which(covCounts$Protein == 1), ]
covCounts0 <- covCounts[which(covCounts$Protein != 1), ]

allProGC <- covCounts1$GCcont
names(allProGC) <- rownames(covCounts1)
allProL <- covCounts1$geneLength
names(allProL) <- rownames(covCounts1)
allNonGC <- covCounts0$GCcont
names(allNonGC) <- rownames(covCounts0)
allNonL <- covCounts0$geneLength
names(allNonL) <- rownames(covCounts0)

# Filtered Genes
filterCC <- apply(covCounts[, 4:ncol(covCounts)], 1 , function(x) mean(x) > 10)
covCountsF <- covCounts[filterCC, ]
covCounts1F <- covCountsF[which(covCountsF$Protein == 1), ]
covCounts0F <- covCountsF[which(covCountsF$Protein != 1), ]

# All Protein Coding Genes
common <- intersect(names(allProGC), rownames(covCounts1)) # no filter
length(common)
feature <- data.frame(gc = allProGC, length = allProL)
data <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                            featureData = feature[common, ],
                            phenoData = data.frame(
                              conditions = ifelse(grepl("^T", colnames(covCounts1[, 4:ncol(covCounts1)])), "tumor", "normal"),
                              row.names = colnames(covCounts1[, 4:ncol(covCounts1)])))
dataWithin <- withinLaneNormalization(data, "gc", which = "full")
dataNormAllP <- betweenLaneNormalization(dataWithin, which = "full")
# All Non-Protein Coding Genes
common <- intersect(names(allNonGC), rownames(covCounts0)) # no filter
length(common)
feature <- data.frame(gc = allNonGC, length = allNonL)
data <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                            featureData = feature[common, ],
                            phenoData = data.frame(
                              conditions = ifelse(grepl("^T", colnames(covCounts0[, 4:ncol(covCounts0)])), "tumor", "normal"),
                              row.names = colnames(covCounts0[, 4:ncol(covCounts0)])))
dataWithin <- withinLaneNormalization(data, "gc", which = "full")
dataNormAllN <- betweenLaneNormalization(dataWithin, which = "full")
edaAllSplitCounts <- as.data.frame(rbind(normCounts(dataNormAllP), normCounts(dataNormAllN)))
edaAllSplitCounts <- edaAllSplitCounts[order(rownames(edaAllSplitCounts)), ]
write.csv(edaAllSplitCounts, file = "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaAllSplitNormCounts.csv")

# Filtered Protein Coding Genes
## Set Up
FilProGC <- covCounts1F$GCcont
names(FilProGC) <- rownames(covCounts1F)
FilProL <- covCounts1F$geneLength
names(FilProL) <- rownames(covCounts1F)
## Build Data Structure
common <- intersect(names(FilProGC), rownames(covCounts1F)) # no filter
length(common)

feature <- data.frame(gc = FilProGC, length = FilProL)
data <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                            featureData = feature[common, ],
                            phenoData = data.frame(
                              conditions = ifelse(grepl("^T", colnames(covCounts1F[, 4:ncol(covCounts1F)])), "tumor", "normal"),
                              row.names = colnames(covCounts1F[, 4:ncol(covCounts1F)])))
## Normalization
```

```
dataWithin <- withinLaneNormalization(data, "gc", which = "full")
dataNormFilP <- betweenLaneNormalization(dataWithin, which = "full")
normCountsFilP <- normCounts(dataNormFilP)
# Filtered Non-Protein Coding Genes
## Set Up
FilNonGC <- covCountsOF$GCcont
names(FilNonGC) <- rownames(covCountsOF)
FilNonL <- covCountsOF$geneLength
names(FilNonL) <- rownames(covCountsOF)
## Build Data Structure
common <- intersect(names(FilNonGC), rownames(covCountsOF)) # no filter
length(common)
feature <- data.frame(gc = FilNonGC, length = FilNonL)
data <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                            featureData = feature[common, ],
                            phenoData = data.frame(
                              conditions = ifelse(grepl("^T", colnames(covCountsOF[, 4:ncol(covCountsOF)])), "tumor", "normal"),
                              row.names = colnames(covCountsOF[, 4:ncol(covCountsOF)])))
## Normalization
dataWithin <- withinLaneNormalization(data, "gc", which = "full")
dataNormFilN <- betweenLaneNormalization(dataWithin, which = "full")
normCountsFilN <- normCounts(dataNormFilN)
normCountsFil <- rbind(normCountsFilP, normCountsFilN)
normCountsFil <- normCountsFil[order(rownames(normCountsFil)), ]
write.csv(normCountsFil, file = "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaFilSplitCounts.csv")
```

With Protein Coding Status

Edited Functions

```
# Edited Functions
library(aroma.light)

.gcQuant2 <- function(counts, gc, num.bins = 10, which = c("full", "median", "upper")) {
  which <- match.arg(which)
  bins <- cut(gc, breaks = c(-quantile(gc, probs = 0.5), quantile(gc, probs = c(0, 1))))
  bins[is.na(bins)] <- levels(bins)[1]
  names(bins) <- names(gc)
  f <- function(y) {
    if(is.null(names(y))) {
      names(y) <- 1:length(y)
    }
    tmp <- tapply(y, bins, function(x) x)
    switch(which,
           full = {y.norm <- normalizeQuantileRank(tmp)},
           median = {y.norm <- lapply(tmp,function(x) exp(log(x) - median(log(x)) + median(log(unlist(tmp))))))},
           upper = {y.norm <- lapply(tmp,function(x) exp(log(x) - quantile(log(x), probs=.75) + quantile(log(unlist(tmp)), prob = 0.75)))}
           )
    names <- unlist(sapply(y.norm, names))
    y.norm <- unlist(y.norm)
    names(y.norm) <- names
    y.norm[names(y)]
  }
  apply(counts,2, f)
}

withinLaneNormalization2 <- function(x, y, which = c("loess", "median", "upper", "full"),
                                offset = FALSE, num.bins = 10, round = TRUE)

setMethod(
          f = "withinLaneNormalization2",
          signature = signature(x = "matrix", y = "numeric"),
          definition = function(x, y, which = c("loess", "median", "upper", "full"),
                                offset = FALSE, num.bins = 10, round = TRUE) {
            which <- match.arg(which)
            if(which =="loess") {
              retval <- .gcLoess(x,y)
            } else {
              retval <- .gcQuant2(x, y, num.bins, which)
            }
            if(!offset) {
              if(round) {
                retval <- round(retval)
              }
              return(retval)
            } else {
              ret <- log(retval + 0.1) - log(x + 0.1)
              return(ret)
            }
          }
          )

setMethod(
          f = "withinLaneNormalization2",
          signature = signature(x = "SeqExpressionSet", y = "character"),
          definition = function(x, y, which = c("loess", "median", "upper", "full"),
                                offset = FALSE, num.bins = 10, round = TRUE) {
            if(offset) {
              o <- withinLaneNormalization2(counts(x),
                                            fData(x)[, y],
```

```
                                        which, offset, num.bins, round)
    } else {
      o <- offst(x)
    }

    newSeqExpressionSet(counts = counts(x),
                        normalizedCounts = withinLaneNormalization2(counts(x),
                          fData(x)[, y], which, offset = FALSE, num.bins, round),
                        offset = o,
                          phenoData = phenoData(x), featureData = featureData(x))
  }
  )
```

With GC-Content, then Protein Coding Status

```
# Data Structure
gc <- ensCov$GCcont
names(gc) <- rownames(ensCov)
pro <- ensCov$Protein
names(pro) <- rownames(ensCov)
length <- ensCov$geneLength
names(length) <- rownames(ensCov)

common  <- intersect(names(gc), rownames(counts)) # no filter
feature <- data.frame(gc = gc, pro = pro, length = length)
dataGCP <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                           featureData = feature[common, ],
                           phenoData = data.frame(
                             conditions = ifelse(grepl("^T", colnames(counts)), "tumor", "normal"),
                             row.names = colnames(counts)))

common  <- intersect(names(gc), rownames(counts[filter, ])) # filtered
feature <- data.frame(gc = gc, pro = pro, length = length)
dataGCPf <- newSeqExpressionSet(counts = as.matrix(counts[common, ]),
                           featureData = feature[common, ],
                           phenoData = data.frame(
                             conditions = ifelse(groups == 1, "tumor", "normal"),
                             row.names = colnames(counts)))

# GC-Content Then Protein Coding Status
dataWithinG  <- withinLaneNormalization(dataGCP, "gc", which = "full", offset = TRUE)
dataWithinGP <- withinLaneNormalization2(dataWithinG, "pro", which = "full", offset = TRUE)
dataNormGP   <- betweenLaneNormalization(dataWithinGP, which = "full", offset = TRUE)
## Save Normalized Count Values
write.csv(normCounts(dataNormGP), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsAllGP.csv")
edaAllEditCountsGP <- normCounts(dataNormGP)

dataWithinGf  <- withinLaneNormalization(dataGCPf, "gc", which = "full", offset = TRUE)
dataWithinGPf <- withinLaneNormalization2(dataWithinGf, "pro", which = "full", offset = TRUE)
dataNormGPf   <- betweenLaneNormalization(dataWithinGPf, which = "full", offset = TRUE)
## Save Normalized Count Values
write.csv(normCounts(dataNormGPf), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsFilGP.csv")
edaFilEditCountsGP <- normCounts(dataNormGPf)
```

With Protein Coding Status, then GC-Content

```
# Protein Coding Status Then GC-Content
dataWithinP  <- withinLaneNormalization(dataGCP, "gc", which = "full", offset = TRUE)
dataWithinPG <- withinLaneNormalization2(dataWithinP, "pro", which = "full", offset = TRUE)
dataNormPG   <- betweenLaneNormalization(dataWithinPG, which = "full", offset = TRUE)
## Save Normalized Count Values
write.csv(normCounts(dataNormPG), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsAllPG.csv")
edaAllEditCountsPG <- normCounts(dataNormPG)

dataWithinPf  <- withinLaneNormalization(dataGCPf, "gc", which = "full", offset = TRUE)
dataWithinPGf <- withinLaneNormalization2(dataWithinPf, "pro", which = "full", offset = TRUE)
dataNormPGf   <- betweenLaneNormalization(dataWithinPGf, which = "full", offset = TRUE)
## Save Normalized Count Values
write.csv(normCounts(dataNormPGf), "C:/Users/Lauren/Documents/Research/Summer Week 12 Finish/edaNormCountsFilPG.csv")
edaFilEditCountsPG <- normCounts(dataNormPGf)
```

lme4 Pipeline

Using the CHPC:

Example of a .R File

```
print(date())

# Set up for receiving numeric arguments from command line
.libPaths("/uufs/chpc.utah.edu/common/home/u0776140/R_libs")
library(R.utils)
library(lme4)
args <- as.numeric(cmdArgs())

# Read in needed data
## effects (data structure)
effects <- read.csv("effects.csv")
```

```
rownames(effects) <- effects$X
effects <- effects[, -1]
## offset
offsetCQN <- read.csv("offsetEnsCQN.csv")
rownames(offsetCQN) <- offsetCQN$X
offsetCQN <- as.matrix(offsetCQN[, -1])
## genes
genes <- read.csv("genes.csv")
genes <- as.character(genes[, 2])

# Set up needed elements
glmer <- vector("list", length = (args[2] - args[1] + 1))
pvalmat <- vector("list", length = (args[2] - args[1] + 1))
pval <- vector("numeric", length = (args[2] - args[1] + 1))
fixedEst <- vector("numeric", length = (args[2] - args[1] + 1))

# Loop
for (i in args[1]:args[2]){
  glmer[[i]] <- try(glmer.nb(effects[, i + 2] ~ groups01 + (1|sampleID), data = effects, offset = offsetCQN[i, ]))
  if (class(glmer[[i]]) != "glmerMod") {
    pvalmat[[i]] <- NA
    pval[i] <- NA
    fixedEst[i] <- NA
  } else {
    pvalmat[[i]] <- coef(summary(glmer[[i]]))
    pval[i] <- pvalmat[[i]][2, 4]
    fixedEst[i] <- pvalmat[[i]][2, 1]
  }
}

# Send results to file
write.csv(as.data.frame(cbind(genes[args[1]:args[2]], fixedEst)), file = paste("fixedEst", args[1], "to", args[2], ".csv", sep = ""))
write.csv(as.data.frame(cbind(genes[args[1]:args[2]], pval)),
          file = paste("pval", args[1], "to", args[2], ".csv", sep = ""))
save(list = c("genes", "glmer", "pvalmat", "pval", "fixedEst"), file = paste("cqnModels", args[1], "to", args[2], ".Rdata", sep = ""))

print(date())
```

Example of a .conf file

```
0 Rscript cqnAll.R 1 1000
1 Rscript cqnAll.R 1001 2000
2 Rscript cqnAll.R 2001 3000
3 Rscript cqnAll.R 3001 4000
4 Rscript cqnAll.R 4001 5000
5 Rscript cqnAll.R 5001 6000
6 Rscript cqnAll.R 6001 7000
7 Rscript cqnAll.R 7001 8000
8 Rscript cqnAll.R 8001 9000
9 Rscript cqnAll.R 9001 10000
```

Example of a .slurm file

```
#!/bin/bash
#SBATCH -- job-name = lme4models
#SBATCH -- time = 01:30:00
#SBATCH -- nodes = 1
#SBATCH -- ntasks = 10

#SBATCH -o out.%j
#SBATCH -e err.%j

#SBATCH --mail-type = FAIL, BEGIN, END
#SBATCH --mail-user = Lauren.Holt.Tutor@gmail.com

#SBATCH -- account = usu-em
#SBATCH -- partition = usu-em

module load R
srun --multi-prog cqnAllpt1.conf
```

Wilcoxon Rank Sum Pipeline

This online tutorial https://data.library.virginia.edu/the-wilcoxon-rank-sum-test/ states that the "difference in location returns the median difference between a sample from x and a sample from y", this means that the I want to run wilcox_._test(tumor, normal) ("University of

Virginia Library Research Data Services Sciences," n.d.)

```
counts2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/log2counts.csv")
rownames(counts2) <- counts2$X
counts2 <- counts2[, -1]

filter <- apply(counts, 1 , function(x) mean(x) > 10)

rpkmLog2 <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/log2rpkm.csv")
rownames(rpkmLog2) <- rpkmLog2$X
rpkmLog2 <- rpkmLog2[, -1]
rpkmLog2Fil <- read.csv("C:/Users/Lauren/Documents/Research/Summer Week 14 Finish/log2rpkmFil.csv")
rownames(rpkmLog2Fil) <- rpkmLog2Fil$X
rpkmLog2Fil <- rpkmLog2Fil[, -1]

groups <- vector("numeric", length = ncol(counts))
for (i in 1:length(groups)){
  if (grepl("^T", colnames(counts[i]))){
    groups[i] <- 1 # tumor samples are group 1
  } else {
    groups[i] <- 0 # normal samples are group 0
  }
}
```

Based on Read-Counts

Very similar to Wilcoxon Rank Sum test with RPKM Values (code not included).

```
wilcoxLog2raw <- data.frame(matrix(data = NA, ncol = 3, nrow = 30220))
colnames(wilcoxLog2raw) <- c("gene", "pvalue", "estLog2FC")
for(i in 1:30220){
  result <- wilcox_test(as.numeric(counts2[i, ]) ~ relevel(as.factor(groups), "1"), conf.int = TRUE)
  wilcoxLog2raw[i, 1] <- rownames(counts2)[i]
  wilcoxLog2raw[i, 2] <- pvalue(result)
  ifelse(is.na(pvalue(result)), wilcoxLog2raw[i, 3] <- NA, wilcoxLog2raw[i, 3] <- confint(result)$estimate)
}

wilcoxLog2raw1 <- mutate(wilcoxLog2raw, "FDR" = p.adjust(wilcoxLog2raw$pvalue, method = "fdr"))
wilcoxLog2raw2 <- mutate(wilcoxLog2raw1, "sig" = rep(NA, 30220))

for (i in 1:30220){
  if (is.na(wilcoxLog2raw2[i, "pvalue"])){
    wilcoxLog2raw2[i, "sig"] = NA
  }else {
    if (wilcoxLog2raw2[i, "pvalue"] >= 0.05){
      wilcoxLog2raw2[i, "sig"] = 0
    } else {
      if (wilcoxLog2raw2[i, "estLog2FC"] > 0){
        wilcoxLog2raw2[i, "sig"] = 1
      } else {
        wilcoxLog2raw2[i, "sig"] = -1
      }
    }
  }
}
summary(wilcoxLog2raw2)
write.csv(wilcoxLog2raw2, file = "C:/Users/Lauren/Documents/Research/New Wilcox Tests/rawAllWilcox_test.csv")

table(wilcoxLog2raw2$sig)
```

Example of Wilcoxon Rank Sum test with Normalized Read-Counts

The same process was used for all variations of CQN and EDASeq normalization.

```
# CQN with GC-content
wilcoxLog2cqnOG <- data.frame(matrix(data = NA, ncol = 3, nrow = 14715))
colnames(wilcoxLog2cqnOG) <- c("gene", "pvalue", "estLog2FC")
for(i in 1:14715){
  result <- wilcox_test(as.numeric(cqnOGcounts[i, ]) ~ relevel(as.factor(groups), "1"), conf.int = TRUE)
  wilcoxLog2cqnOG[i, 1] <- rownames(cqnOGcounts)[i]
  wilcoxLog2cqnOG[i, 2] <- pvalue(result)
  wilcoxLog2cqnOG[i, 3] <- confint(result)$estimate
}

wilcoxLog2cqnOG1 <- mutate(wilcoxLog2cqnOG, "FDR" = p.adjust(wilcoxLog2cqnOG$pvalue, method = "fdr"))
wilcoxLog2cqnOG2 <- mutate(wilcoxLog2cqnOG1, "sig" = rep(NA, 14715))

for (i in 1:14715){
  if (wilcoxLog2cqnOG2[i, "pvalue"] >= 0.05){
    wilcoxLog2cqnOG2[i, "sig"] = 0
  } else {
    if (wilcoxLog2cqnOG2[i, "estLog2FC"] > 0){
      wilcoxLog2cqnOG2[i, "sig"] = 1
    } else {
      wilcoxLog2cqnOG2[i, "sig"] = -1
    }
  }
}
summary(wilcoxLog2cqnOG2)
```

```
write.csv(wilcoxLog2cqnOG2, file = "C:/Users/Lauren/Documents/Research/New Wilcox Tests/cqnOGAllWilcox_test.csv")

table(wilcoxLog2cqnOG2$sig)
```

Additional Plots

Additional Plots rendered for this write-up.

CQN Plots

Here for CQN with GC-content; the same code with different variables was used for all variations of CQN.

```
plotColors <- ifelse(grepl("^T", colnames(counts)), "gray35", "skyblue")
par(mfrow = c(1, 2), mar = c(4, 4, 2, 0.5))
cqnplot(x = cqnAll, n = 1, col = alpha(plotColors, 0.5), xlab = "GC-content",
        ylim = c(-4.5, 8), xlim = c(0, 1), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5)
legend("bottom", c("Tumor", "Normal"), fill = c("gray35", "skyblue"), horiz = TRUE, bty = "n", cex = 1.5)
cqnplot(x = cqnAll, n = 2, col = alpha(plotColors, 0.5), xlab = "Gene lengths",
        ylim = c(-4.5, 8), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5)
title("All Genes", outer = TRUE, line = -1.5, cex.main = 1.5)
cqnplot(x = cqnFil, n = 1, col = alpha(plotColors, 0.5), xlab = "GC-content",
        ylim = c(-4.5, 8), xlim = c(0, 1), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5)
legend("bottom", c("Tumor", "Normal"), fill = c("gray35", "skyblue"), horiz = TRUE, bty = "n", cex = 1.5)
cqnplot(x = cqnFil, n = 2, col = alpha(plotColors, 0.5), xlab = "Gene lengths",
        ylim = c(-4.5, 8), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5)
title("Filtered Genes", outer = TRUE, line = -1.5, cex.main = 1.5)
```

EDASeq Plots

Here for EDASeq with GC-content; the same code with different variables was used for all variations of EDASeq.

```
par(mfrow = c(1, 3), mar = c(4, 4, 2, 0.5), oma = c(0, 0, 2, 0))
biasPlot(dataAll, "gc", log = TRUE, col = c("skyblue", "gray35"),
         ylim = c(-5, 4.5), xlim = c(0, 1), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5,
         xlab = "GC-content", ylab = "log(gene-level counts)")
title("Raw Data", cex.main = 1.5)
legend("bottom", c("Tumor", "Normal"), fill = c("gray35", "skyblue"), horiz = TRUE, bty = "n")

biasPlot(dataWithin, "gc", log = TRUE, col = c("skyblue", "gray35"),
         ylim = c(-5, 4.5), xlim = c(0, 1), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5,
         xlab = "GC-content", ylab = "log(gene-level counts)")
title("Normalized for GC-content", cex.main = 1.5)

biasPlot(dataNormAll, "gc", log = TRUE, col = c("skyblue", "gray35"),
         ylim = c(-5, 4.5), xlim = c(0, 1), cex.lab = 1.5, cex.main = 1.5, cex.axis = 1.5,
         xlab = "GC-content", ylab = "log(gene-level counts)")
title("Fully Normalized Data", cex.main = 1.5)
title("All Genes", outer = TRUE, cex.main = 1.5)

# Overdispersion Plot
par(mfrow = c(1, 2), mar = c(4, 4, 2, 0.5))
meanVarPlot(dataAll, log = TRUE, xlim = c(-0.5, 15), ylim = c(-0.5, 25))
title("All Genes")
meanVarPlot(dataFil, log = TRUE, xlim = c(-0.5, 15), ylim = c(-0.5, 25))
title("Filtered Genes")
```

Comparison Plots

Here comparing the results CQN, nearly identical code was used for the variations of EDASeq.

```
# lme4
p1 <- autoplot(condense(x = bin(cqnOG$fixedEst, width = 0.1),
                        y = bin(cqnSplit$fixedEst, width = 0.1),
                        z = ensCov$Protein)) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) + theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(subtitle = "All Genes", x = "CQN with GC-content", y = "CQN split on protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p2 <- autoplot(condense(x = bin(cqnOGFil$fixedEst, width = 0.1),
                        y = bin(cqnSplitFil$fixedEst, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(subtitle = "Filtered Genes", x = "CQN with GC-content", y = "CQN split on protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p3 <- autoplot(condense(x = bin(cqnOG$fixedEst, width = 0.1),
                        y = bin(cqnJ$fixedEst, width = 0.1),
                        z = ensCov$Protein)) +
```

```
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN with GC-content", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p4 <- autoplot(condense(x = bin(cqnOGFil$fixedEst, width = 0.1),
                        y = bin(cqnJFil$fixedEst, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN with GC-content", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p5 <- autoplot(condense(x = bin(cqnSplit$fixedEst, width = 0.1),
                        y = bin(cqnJ$fixedEst, width = 0.1),
                        z = ensCov$Protein)) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN split on protein", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p6 <- autoplot(condense(x = bin(cqnSplitFil$fixedEst, width = 0.1),
                        y = bin(cqnJFil$fixedEst, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN split on protein", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
grid.arrange(ggarrange(p1, p2, p3, p4, p5, p6,
                       ncol = 2, nrow = 3, common.legend = TRUE, legend = "bottom"),
             top = "Compare lme4 Log base 2 Fold Change Values using CQN variation offsets")

# Wilcoxon
p1 <- autoplot(condense(x = bin(cqnOGw$estLog2FC, width = 0.1),
                        y = bin(cqnSplitw$estLog2FC, width = 0.1),
                        z = ensCov$Protein)) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) + theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(subtitle = "All Genes", x = "CQN with GC-content", y = "CQN split on protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p2 <- autoplot(condense(x = bin(cqnOGFilw$estLog2FC, width = 0.1),
                        y = bin(cqnSplitFilw$estLog2FC, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(subtitle = "Filtered Genes", x = "CQN with GC-content", y = "CQN split on protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p3 <- autoplot(condense(x = bin(cqnOGw$estLog2FC, width = 0.1),
                        y = bin(cqnJw$estLog2FC, width = 0.1),
                        z = ensCov$Protein)) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN with GC-content", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p4 <- autoplot(condense(x = bin(cqnOGFilw$estLog2FC, width = 0.1),
                        y = bin(cqnJFilw$estLog2FC, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN with GC-content", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p5 <- autoplot(condense(x = bin(cqnSplitw$estLog2FC, width = 0.1),
                        y = bin(cqnJw$estLog2FC, width = 0.1),
                        z = ensCov$Protein)) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN split on protein", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
p6 <- autoplot(condense(x = bin(cqnSplitFilw$estLog2FC, width = 0.1),
                        y = bin(cqnJFilw$estLog2FC, width = 0.1),
                        z = ensCov[filter, "Protein"])) +
  geom_abline(intercept = 0, slope = 1, col = alpha("darkgray", 0.5)) +
  theme_bw() + xlim(-6, 6.75) + ylim(-6, 6.75) +
  labs(subtitle = " ", x = "CQN split on protein", y = "CQN with jittered protein") +
  scale_fill_gradientn("Mean Protein\nCoding Status\n",
                       colors = c("blue", "lightblue", "gray95", "red", "darkred"), breaks = seq(0, 1, 0.25))
grid.arrange(ggarrange(p1, p2, p3, p4, p5, p6,
                       ncol = 2, nrow = 3, common.legend = TRUE, legend = "bottom"),
             top = "Compare Wilcoxon Log base 2 Fold Change Values using CQN variation offsets")
```

MSD Plots

```
# Code
MSDprotein<- function(filtered, method, variable) {
```

```
  together <- read.csv(paste("C:/Users/Lauren/Documents/Research/Summer week 06 Extend/", tolower(method), filtered, ".csv", sep = ""))
  ensCov <- read.csv("C:/Users/Lauren/Documents/Research/Data Sets/EnsGCGLP.csv")
    sep <- read.csv(paste("C:/Users/Lauren/Documents/Research/Data Sets/", tolower(method), filtered, "Sep.csv", sep = ""))
    sep <- merge(sep, ensCov, by = "X", all = FALSE)
    sep <- sep[order(sep$X), ]
    if (any(together$X != sep$X)){
      stop("genes out of order!")
    } else {
      toSave <- vector("numeric", 3)
      names(toSave) <- c("protein", "nonprotein", "p-n")
      toSave[1] <- mean((sep[which(sep$Protein == 1), as.character(variable)] -
                         together[which(sep$Protein == 1), as.character(variable)])^2, na.rm = TRUE)
      toSave[2] <- mean((sep[which(sep$Protein == 0), as.character(variable)] -
                         together[which(sep$Protein == 0), as.character(variable)])^2, na.rm = TRUE)
      toSave[3] <-  toSave[1] - toSave[2]
      assign(paste("MSDpro", method, filtered, variable, sep = ""), toSave, .GlobalEnv)
    }
}

MSDprotein("All", "CQN", "fixedEst")
MSDprotein("All", "CQN", "fdr")
MSDprotein("Fil", "CQN", "fixedEst")
MSDprotein("Fil", "CQN", "fdr")
MSDprotein("All", "EDA", "fixedEst")
MSDprotein("All", "EDA", "fdr")
MSDprotein("Fil", "EDA", "fixedEst")
MSDprotein("Fil", "EDA", "fdr")

MSDrandom <- function(lower, upper, filtered, method, variable) {
  together <- read.csv(paste("C:/Users/Lauren/Documents/Research/Summer week 06 Extend/",
                             tolower(method), filtered, ".csv", sep = ""))

  toSave <- vector("numeric", length = upper - lower + 1)
  MSDrand1 <- vector("numeric", length = upper - lower + 1)
  MSDrand2 <- vector("numeric", length = upper - lower + 1)
  for (i in lower:upper) {
    rand <- read.csv(paste("C:/Users/Lauren/Documents/Research/Summer Week 10 Edit/random split 2/",
                           method, "/", filtered, "/", method, filtered, i, ".csv", sep = ""))
    rand <- rand[order(rand$gene), ]
    if (any(together$X != rand$gene)){
      stop("genes out of order!")
    } else {
      MSDrand1[as.numeric(i - 299)] <- mean((rand[which(rand$group == 1), as.character(variable)] -
                         together[which(rand$group == 1), as.character(variable)])^2, na.rm = TRUE)
      MSDrand2[as.numeric(i - 299)] <- mean((rand[which(rand$group == 2), as.character(variable)] -
                         together[which(rand$group == 2), as.character(variable)])^2, na.rm = TRUE)
      toSave[as.numeric(i - 299)] <- MSDrand1[as.numeric(i - 299)] - MSDrand2[as.numeric(i - 299)]
    }
  }
  assign(paste("MSDrandG1", method, filtered, variable, sep = ""), MSDrand1, .GlobalEnv)
  assign(paste("MSDrandG2", method, filtered, variable, sep = ""), MSDrand2, .GlobalEnv)
  assign(paste("MSDrand", method, filtered, variable, sep = ""), toSave, .GlobalEnv)
}

MSDrandom(300, 310, "All", "CQN", "fixedEst")
MSDrandom(300, 310, "All", "CQN", "fdr")
MSDrandom(300, 310, "Fil", "CQN", "fixedEst")
MSDrandom(300, 310, "Fil", "CQN", "fdr")
MSDrandom(300, 310, "Fil", "EDA", "fixedEst")
MSDrandom(300, 310, "Fil", "EDA", "fdr")

# EDA All "by hand"
together <- read.csv("C:/Users/Lauren/Documents/Research/Summer week 06 Extend/edaAll.csv")
MSDrandG1EDAAllfixedEst <- MSDrandG2EDAAllfixedEst <- MSDrandEDAAllfixedEst <- vector("numeric", 11)
MSDrandG1EDAAllfdr <- MSDrandG2EDAAllfdr <- MSDrandEDAAllfdr <- vector("numeric", 11)
for (i in 300:310){
  rand <- read.csv(paste("C:/Users/Lauren/Documents/Research/Summer Week 10 Edit/random split 2/EDA/All/edaAllseed", i, ".csv", sep = ""))
  # fixed effect estimate
  MSDrandG1EDAAllfixedEst[i - 299] <- mean((rand[which(rand$group == 1), "fixedEst"] - together[which(rand$group == 1), "fixedEst"])^2,
                                           na.rm = TRUE)
  MSDrandG2EDAAllfixedEst[i - 299] <- mean((rand[which(rand$group == 2), "fixedEst"] - together[which(rand$group == 2), "fixedEst"])^2,
                                           na.rm = TRUE)
  MSDrandEDAAllfixedEst[i - 299]<- MSDrandG1EDAAllfixedEst[i - 299] - MSDrandG2EDAAllfixedEst[i - 299]
  # fdr
  MSDrandG1EDAAllfdr[i - 299] <- mean((rand[which(rand$group == 1), "fdr"] - together[which(rand$group == 1), "fdr"])^2, na.rm = TRUE)
  MSDrandG2EDAAllfdr[i - 299] <- mean((rand[which(rand$group == 2), "fdr"] - together[which(rand$group == 2), "fdr"])^2, na.rm = TRUE)
  MSDrandEDAAllfdr[i - 299]<- MSDrandG1EDAAllfdr[i - 299] - MSDrandG2EDAAllfdr[i - 299]
}

# Plots
base <- ggplot() +
  theme_bw() + scale_y_continuous(breaks = NULL) + xlim(0, 0.25) +
  scale_color_manual(name = "",
                     labels = c("Protein coding genes", "Non-protein coding genes",
                                "Random group 1 genes", "Random group 2 genes"),
                     values = c(alpha("red", 0.5), alpha("black", 0.5), alpha("blue", 0.5), alpha("gold", 0.5))) +
  scale_shape_manual(name = "",
                     labels = c("Protein coding genes", "Non-protein coding genes",
                                "Random group 1 genes", "Random group 2 genes"),
                     values = c(17, 17, 19, 19))
cqnAllFE2 <- base +
```

```
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproCQNAllfixedEst[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproCQNAllfixedEst[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1CQNAllfixedEst, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2CQNAllfixedEst, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "CQN", x = "MSD", subtitle = "All Genes")
cqnFilFE2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproCQNFilfixedEst[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproCQNFilfixedEst[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1CQNFilfixedEst, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2CQNFilfixedEst, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "", x = "MSD", subtitle = "Filtered Genes")
edaAllFE2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproEDAAllfixedEst[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproEDAAllfixedEst[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1EDAAllfixedEst, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2EDAAllfixedEst, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "EDASeq", x = "MSD", subtitle = " ")
edaFilFE2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproEDAFilfixedEst[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproEDAFilfixedEst[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1EDAFilfixedEst, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2EDAFilfixedEst, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "", x = "MSD", subtitle = " ")
cqnAllFDR2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproCQNAllfdr[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproCQNAllfdr[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1CQNAllfdr, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2CQNAllfdr, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "CQN", x = "MSD", subtitle = "All Genes")
cqnFilFDR2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproCQNFilfdr[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproCQNFilfdr[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1CQNFilfdr, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2CQNFilfdr, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "", x = "MSD", subtitle = "Filtered Genes")
edaAllFDR2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproEDAAllfdr[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproEDAAllfdr[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1EDAAllfdr, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2EDAAllfdr, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "EDASeq", x = "MSD", subtitle = " ")
edaFilFDR2 <- base +
  geom_hline(yintercept = 0, size = 0.1, col = "darkgray") +
  geom_point(aes(x = MSDproEDAFilfdr[1], y = 0, col = "1", shape = "1"), size = 3) +
  geom_point(aes(x = MSDproEDAFilfdr[2], y = 0, col = "2", shape = "2"), size = 3) +
  geom_point(aes(x = MSDrandG1EDAFilfdr, y = 0, col = "3", shape = "3"), size = 3) +
  geom_point(aes(x = MSDrandG2EDAFilfdr, y = 0, col = "4", shape = "4"), size = 3) +
  labs(y = "", x = "MSD", subtitle = "")
grid.arrange(ggarrange(cqnAllFE2, cqnFilFE2,
                       edaAllFE2, edaFilFE2,
                       ncol = 2, nrow = 2, common.legend = TRUE, legend = "bottom"),
             top = "Comparing Mean Squared Distance Values from Log base 2 Fold Change Values")

grid.arrange(ggarrange(cqnAllFDR2, cqnFilFDR2,
                       edaAllFDR2, edaFilFDR2,
                       ncol = 2, nrow = 2, common.legend = TRUE, legend = "bottom"),
             top = "Comparing Mean Squared Distance Values from FDR Corrected P-Values")
```

Volcano Plots

After data is loaded, reading in only the gene names, log2FC estimates, and FDR corrected p-values

Change column names to match across all 32 test variations

```
pullGene <- function(EnsID) {
  df <- data.frame(rbind(wLog2Raw[wLog2Raw$gene == as.character(EnsID), ],
                         wLog2RawFil[wLog2RawFil$gene == as.character(EnsID), ],
                         wLog2RPKM[wLog2RPKM$gene == as.character(EnsID), ],
                         wLog2RPKMFil[wLog2RPKMFil$gene == as.character(EnsID), ],
                         cqnOG2[cqnOG2$gene == as.character(EnsID), ],
                         cqnOGFil2[cqnOGFil2$gene == as.character(EnsID), ],
                         cqnOGw2[cqnOGw2$gene == as.character(EnsID), ],
                         cqnOGFilw2[cqnOGFilw2$gene == as.character(EnsID), ],
                         cqnSplit2[cqnSplit2$gene == as.character(EnsID), ],
                         cqnSplitFil2[cqnSplitFil2$gene == as.character(EnsID), ],
                         cqnSplitw2[cqnSplitw2$gene == as.character(EnsID), ],
                         cqnSplitFilw2[cqnSplitFilw2$gene == as.character(EnsID), ],
                         cqnJ2[cqnJ2$gene == as.character(EnsID), ],
                         cqnJFil2[cqnJFil2$gene == as.character(EnsID), ],
```

```
                        cqnJw2[cqnJw2$gene == as.character(EnsID), ],
                        cqnJFilw2[cqnJFilw2$gene == as.character(EnsID), ],
                        edaOG2[edaOG2$gene == as.character(EnsID), ][1,],
                        edaOGFil2[edaOGFil2$gene == as.character(EnsID), ][1,],
                        edaOGw2[edaOGw2$gene == as.character(EnsID), ][1,],
                        edaOGFilw2[edaOGFilw2$gene == as.character(EnsID), ][1,],
                        edaSplit2[edaSplit2$gene == as.character(EnsID), ][1,],
                        edaSplitFil2[edaSplitFil2$gene == as.character(EnsID), ][1,],
                        edaSplitw2[edaSplitw2$gene == as.character(EnsID), ][1,],
                        edaSplitFilw2[edaSplitFilw2$gene == as.character(EnsID), ][1,],
                        edaGP2[edaGP2$gene == as.character(EnsID), ][1,],
                        edaGPFil2[edaGPFil2$gene == as.character(EnsID), ][1,],
                        edaGPw2[edaGPw2$gene == as.character(EnsID), ][1,],
                        edaGPFilw2[edaGPFilw2$gene == as.character(EnsID), ][1,],
                        edaPG2[edaPG2$gene == as.character(EnsID), ],
                        edaPGFil2[edaPGFil2$gene == as.character(EnsID), ][1,],
                        edaPGw2[edaPGw2$gene == as.character(EnsID), ][1,],
                        edaPGFilw2[edaPGFilw2$gene == as.character(EnsID), ][1,]),
                row.names = c("wLog2Raw", "wLog2RawFil", "wLog2RPKM", "wLog2RPKMFil",
                              "cqnOG", "cqnOGFil", "cqnOGw", "cqnOGFilw",
                              "cqnSplit", "cqnSplitFil", "cqnSplitw", "cqnSplitFilw",
                              "cqnJ", "cqnJFil", "cqnJw", "cqnJFilw",
                              "edaOG", "edaOGFil", "edaOGw", "edaOGFilw",
                              "edaSplit", "edaSplitFil", "edaSplitw", "edaSplitFilw",
                              "edaGP", "edaGPFil", "edaGPw", "edaGPFilw",
                              "edaPG", "edaPGFil", "edaPGw", "edaPGFilw"))

  return(df)
}

APC <- pullGene("ENSG00000134982")
APC <- dplyr::mutate(APC, sig = ifelse(fdr < 0.05, "Significant", "Not Significant"))
APC <- dplyr::mutate(APC, ttype = c(rep("wilcoxon", 4), rep(c("lme4", "wilcoxon"), 14)))
APCvol <- ggplot(APC, aes(x = log2fc, y = -log10(fdr))) +
  geom_point(aes(col = sig, shape = ttype), size = 2) +
  geom_hline(yintercept = -log10(0.05), col = alpha("red", 0.5), linetype = 5) +
  scale_color_manual("Significance", values = c(alpha("gray18", 0.5), alpha( "green4", 0.5))) +
  scale_shape("Differential\nExpression\nTest") +
  theme_bw() + xlim(-0.6, 0.6) +
  labs(x = "log2 fold change", y = "-log10 FDR corrected p-value", title = "Test Results for APC: ENSG00000134982")
get_legend(APCvol)
plot_grid(ggplot(APC, aes(x = log2fc, y = -log10(fdr))) +
  geom_point(aes(col = sig, shape = ttype), size = 2) +
  geom_hline(yintercept = -log10(0.05), col = alpha("red", 0.5), linetype = 5) +
  scale_color_manual("Significance", values = c(alpha("gray18", 0.5), alpha( "green4", 0.5))) +
  scale_shape("Differential\nExpression\nTest") +
  theme_bw() + xlim(-0.6, 0.6) + theme(legend.position = "none") +
  labs(x = "log2 fold change", y = "-log10 FDR corrected p-value", title = "Test Results for APC: ENSG00000134982"),
  as_ggplot(get_legend(APCvol)), rel_widths = c(0.77, 0.23))

CTNNB1 <- pullGene("ENSG00000168036")
CTNNB1 <- dplyr::mutate(CTNNB1, sig = ifelse(fdr < 0.05, "Significant", "Not Significant"))
CTNNB1 <- dplyr::mutate(CTNNB1, ttype = c(rep("wilcoxon", 4), rep(c("lme4", "wilcoxon"), 14)))
plot_grid(ggplot(CTNNB1, aes(x = log2fc, y = -log10(fdr))) +
  geom_point(aes(col = sig, shape = ttype ), size = 2) +
  geom_hline(yintercept = -log10(0.05), col = alpha("red", 0.5), linetype = 5) +
  scale_color_manual("Significance", labels = c("Significant", "Not Significant"),
                     values = c("Not Significant" = alpha( "gray18", 0.5), "Significant" = alpha( "green4", 0.5))) +
  scale_shape("Differential\nExpression\nTest") +
  theme_bw() +  xlim(-3.5, 3.5) + theme(legend.position = "none") +
  labs(x = "log2 fold change", y = "-log10 FDR corrected p-value", title = "Test Results for CTNNB1: ENSG00000168036"),
  as_ggplot(get_legend(APCvol)), rel_widths = c(0.77, 0.23))
```