Analysis of *E. coli* Time Course RNA-Seq Data

Annie Cohen[1,4], Ethan Ashby[2,4], Lian Morales[3,4], Prof. Jo Hardin[2], Prof. Dan Stoebel[3], Prof. Danae Schulz[3]

Affiliation: [1]Department of Mathematics, Scripps College [2]Department of Mathematics, Pomona College, [3]Department of Biology, Harvey Mudd College [4]NSF Data Science REU at Harvey Mudd College

Table of Contents

1 Introduction

1.1 Background on *E. coli*

E. coli possesses a general stress response to a variety of environmental stresses (Battesti *et al* 2011, Hengge 2011) ranging from osmotic shock to nutrient starvation. According to preliminary data from Professor Daniel Stoebel, a key transcription factor coordinating this response is RpoS, which regulates one quarter of the bacteria's genome. Simple interpretations of transcriptional networks often invoke an analogy of an on/off switch, in which the presence of a stimulus turns some genes on and other genes off. However, these simple interpretations don't adequately describe the complex, dyanmic processes underlying many transcriptional responses to stimuli. Currently, there exists a limited understanding regarding the dynamic nature of transcriptional responses, and the well-annotated, heavily-studied genome of E. coli presents an excellent model to study these intricate regulatory circuits. Past research has determined three different profiles for genes in relation to RpoS. Linear genes' expression increases linearly with an increase in RpoS. Genes are considered sensitive if low levels of RpoS cause them to react more than expected in the linear profile and insensitive genes are less expressed with low levels of RpoS than linear genes. This research examines whether genes that have previously been identified as linear, sensitive, or insensitive have similar profiles of expression over time under cell stress conditions.

1.2 RNA-Seq

RNA-Seq is the process of extracting, fragmenting, amplifying, sequencing, and mapping RNA to a reference gene in order to determine gene expression levels. Sequencing of RNA data allows us to determine how RpoS regulates genes in response to a stressor such as starvation. We use RNA-Seq to compare the gene expression levels across treatments and time points. We are interested in both the differences in expression between our treatment samples, wildtype (WT), and our control (dRpoS or knockout RpoS), as well as the differential expression of WT from 0 to 150 minutes of starvation.

1.3 Our data

The data we recieved from Professor Stoebel and his lab was a count matrix, with 37 columns. The column names specify the strain, timepoint, treatment, and replicate of the sample. "JH_01" or "JH_02" specifies the strain of *E. coli*, where "JH_01" is the wildtype under starvation, our treatment, and "JH_02" is delta-RpoS, our control in which RpoS is removed from the cells, under starvation. "A", "B", and "C" denotes the biological replicate and the number following it indicates the time point: "01" is 0, "02" is 30, "03" is 60, "04" is 90, "05" is 120, and "06" is 150 minutes. The original data came with a column called "Geneid", which denotes the specific gene. However, we needed a matrix of only count data in order to normalize our data, so we assigned this column as the row names and then deleted the column. In order to compare these genes with external gene lists, we parse the `Geneid` column into `genename`, which is a 3-4 digit character string, usually containing one capital letter.

```r
head(ec_rawcounts[,1:6],6)
```

```
##      JH01_A01 JH01_A02 JH01_A03 JH01_A04 JH01_A05 JH01_A06
## thrL     2761     1599     2451     1994     1055     4025
## thrA     2072      441     1203      739      605     1645
## thrB     1000      278      640      537      606     1349
## thrC     1775      530     1174      762      826     2072
## yaaX      119       36       40       42       28       45
## yaaA      428      424      550      553      265      509
```

2 Normalization

Normalization is an extremely influential step in the process of determining differentially expressed genes. It is necessary in order to minimize external factors such as gene length and sequencing depth. If a gene is longer, there is a higher probability of having more reads because of its longer sequence. Similarly, if one sample has disproportionately more reads than another, it will return a false positive. Therefore, we must normalize for both gene length and sequencing depth.

We tested out many normalization methods including `DESeq2` and `edgeR`. `DESeq2` is a normalization and differential expression analysis method that divides counts by sample specific size factors that are calculated as the median of the ratios of read counts to the geometric mean. `EdgeR` is similarly both a normalization and differential expression analysis method that employs the trimmed mean method (TMM), where gene expression is adjusted by a size factor that takes into account the fold change and relative expression level to its sample (Ciaran et. al). Through a comparison of size factors, we determined that the two methods produced very similar normalized counts. We ultimately decided to use `DESeq2` because of its abundant documentation.

DESeq2 normalizes data within the differential expression analysis function pipeline, so we first run the DESeq() function and then extract the normalized counts.

```
# Create DESeq dataframe
ec_dds <- DESeqDataSetFromMatrix(ec_rawcounts, colData = ec_coldata,
                                 design = ~ timetreat)

# Run DESeq and extract normalized counts
ec_dds_de <- DESeq(ec_dds)
ec_normcounts <- as.data.frame(counts(ec_dds_de, normalized=TRUE))

head(ec_normcounts[1:6],6)
```

```
##        JH01_A01   JH01_A02   JH01_A03   JH01_A04   JH01_A05   JH01_A06
## thrL 3442.1830 2874.70179 2562.00610 2098.83952 1526.87418 3087.26245
## thrA 2583.1956  792.83520 1257.48402  777.85477  875.60083 1261.75074
## thrB 1246.7160  499.79181  668.98568  565.23411  877.04811 1034.71231
## thrC 2212.9210  952.84049 1227.17061  802.06405 1195.44841 1589.26902
## yaaX  148.3592   64.72124   41.81161   44.20825   40.52367   34.51598
## yaaA  533.5945  762.27239  574.90957  582.07535  383.52764  390.41406
```

3 Differential Expression

Differential expression measures whether or not the gene expression of the same gene in different groups or treatments is significantly different. We apply three different differential expression analysis methods below to determine a list of differentially expressed genes in our data set.

3.1 DESeq2

According to Mistry, Meeta, *et al.* (2017), DESeq2 determines differential expression by:

1. Normalization of the raw counts by determining size factors and adjusting each sample by the size factor.
2. Measuring gene-wise dispersions, where dispersion, α, measures the variability within biological replicates of the same sample: $\text{Var} = \mu + \alpha\mu^2$.
3. Adjusting dispersion estimates to fit a curve created by the distribution of dispersion estimates. Use 'plotDispEsts()' to see fit estimates for yourself.
4. Comparing normalized counts with a negative binomial distribution with the sample mean, μ_j, and gene-wise dispersion parameter, α_j to determine the significance of differential expression.

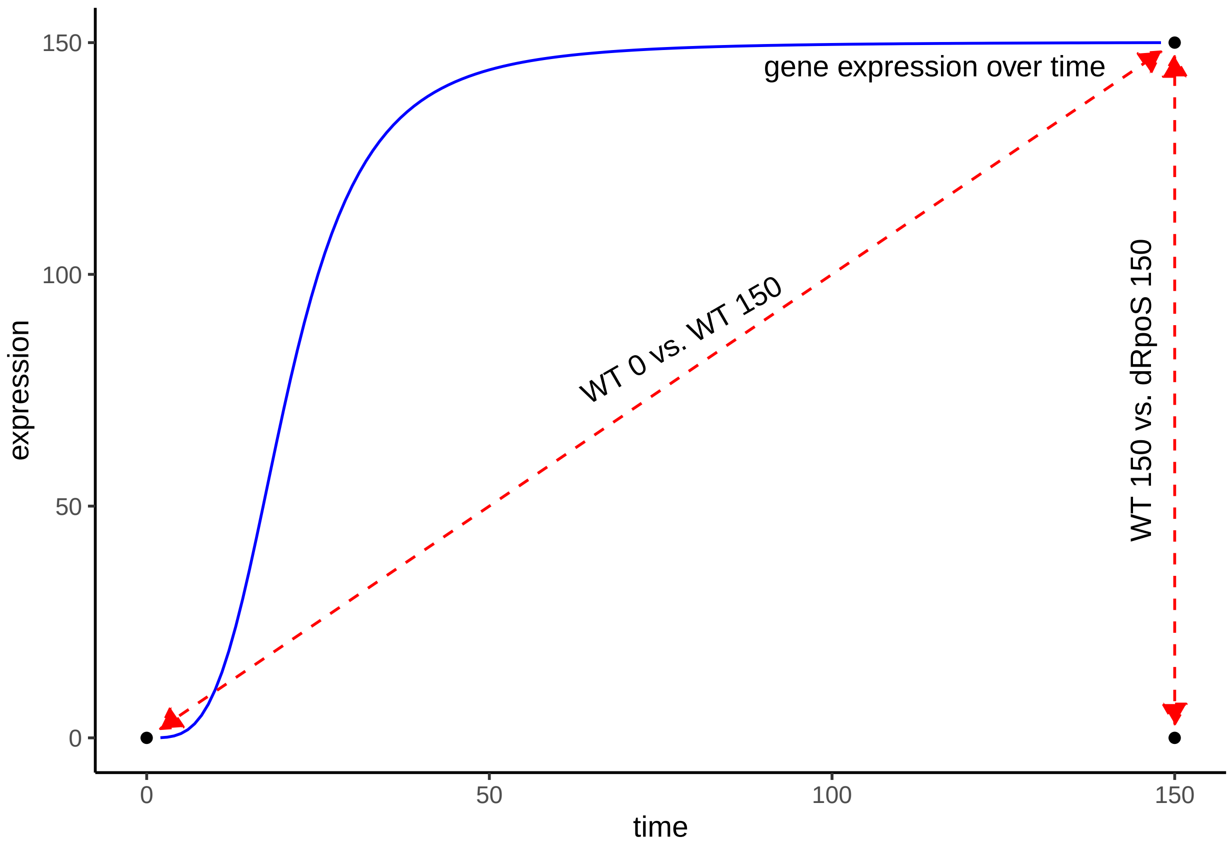
Because we already ran DESeq() above, we can now extract the significant results. We chose a p-value of 0.01 to be consistent across all of our work. DESeq2 is a two-sample comparison method, but we want to accurately determine differentially expressed genes across *all* the time points. Therefore, we compute:

1. A list of differentially expressed genes between the treatment group (wild-type *E. coli*) at 0 minutes and 150 minutes.

2. A list of differentially expressed genes between the treatment group and the control (delta-RpoS *E. coli*), both at 150 minutes.

3. The intersection of the gene lists, or the genes that appear in both lists, which we use as our differentially expressed gene list from ‘DESeq2‘.

```
x = data.frame("x" = c(2:148))
ggplot()+
  geom_line(data = x, aes(x = x, y = 150.1206 + (1.860341e-15 - 150.1206)/(1 + (x/20.62591)^3.591457)),
  geom_line(data = x,aes(x = x, y = x), color = "red", lty = 2, arrow = arrow(length = unit(0.03,"npc")
  geom_point(aes(x = 0, y = 0)) +
  geom_point(aes(x = 150, y = 0)) +
  geom_point(aes(x = 150, y = 150)) +
  geom_segment(aes(x = 150, y = 3, xend = 150, yend = 147), color = "red", lty = 2,arrow = arrow(length
  geom_text(aes(x = 78, y = 86, label = "WT 0 vs. WT 150", angle = 30)) +
  geom_text(aes(x = 145, y = 75, label = "WT 150 vs. dRpoS 150", angle = 90)) +
  geom_text(aes(x = 115, y = 145, label = "gene expression over time")) +
  theme(panel.background = element_blank(), axis.line = element_line(color = "black")) +
  labs(x= "time",y = "expression")
```

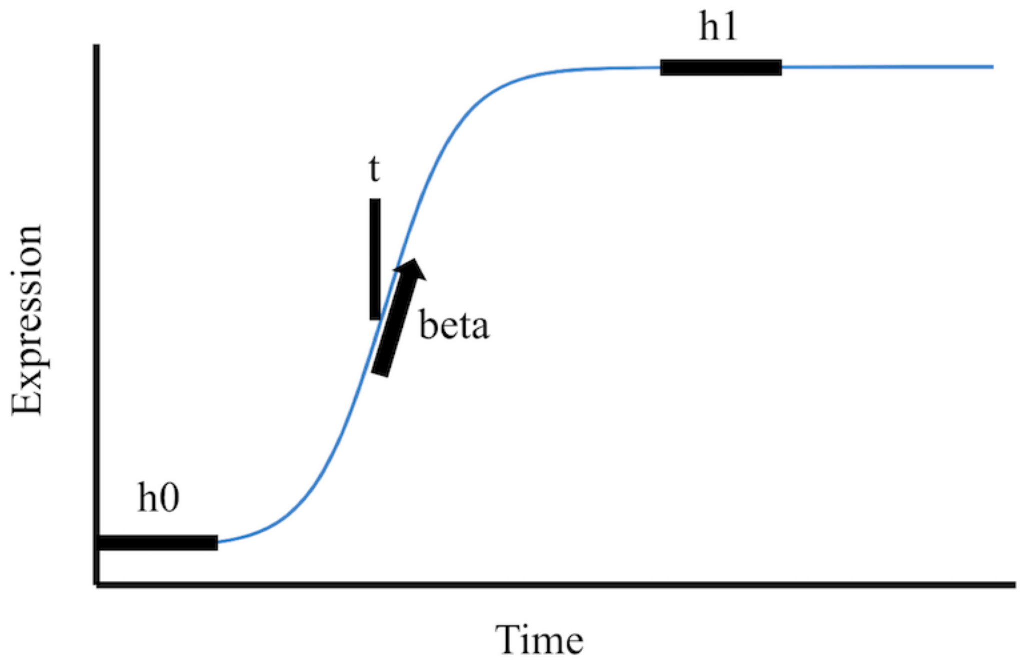


```
EC_sig_genes <- c()

# Determine the genes that are differentially expressed between WT0 and WT150
de_results <- results(ec_dds_de, contrast = c("timetreat", "WT150","WT0"))
# Filter for a p-value less than 0.01
resSig <- de_results[ which(de_results$padj < 0.01 ), ]
resSig$genename <- rownames(resSig)
```

```r
ind <- match(rownames(ec_normcounts),resSig$genename)
ec_normcounts$sig <- resSig$genename[ind]
EC_sig_genes$WT_compare <- ec_normcounts %>%
 filter(!is.na(ec_normcounts$sig))


# Determine the genes that are differentially expressed between WT150 and dRpoS150
de_results <- results(ec_dds_de, contrast = c("timetreat", "WT150","dRpoS150"))
# Again, filter by p-value.
resSig <- de_results[ which(de_results$padj < 0.01 ), ]
resSig$genename <- rownames(resSig)
ind <- match(rownames(ec_normcounts),resSig$genename)
ec_normcounts$sig <- resSig$genename[ind]
EC_sig_genes$treat_compare <- ec_normcounts %>%
 filter(!is.na(ec_normcounts$sig))

# Find the intersection of the two lists of differentially expressed genes
EC_sig_genes <- intersect(EC_sig_genes$WT_compare$sig, EC_sig_genes$treat_compare$sig)
```

3.2 ImpulseDE2

ImpulseDE2 is a serial method for differential expression that models data based on the impulse model, which has parameters $t_1, t_2, h_0, h_1,$ and b. The impulse model has a spike, characterized by its beginning height, h_0, inflection point, t_1, its height, h_1, and its slope at t_1, b. Then the model decreases into its steady state, characterized by the second inflection point, t_2, and its final height, h_2. The expression profiles of our data hint at a sigmoid trajectory, where the first four parameters (h_0, t, β, h_1) are the same, but h_1 is the final expression.



According to Fischer et al., ImpulseDE2 determines differential expression by:

1. Running ‘DESeq2‘ and adjusting gene-wise dispersion estimates from ‘DESeq2‘ to fit a curve.
2. Computing size factors and normalizing counts.
3. Fitting null and alternative model to each of the genes.
4. Fitting sigmoid model to the treatment condition – in our case, samples containing the RpoS translation factor.
5. Determining differential expression based on these models fits to each gene using a log-likelihood ratio (Fischer et al. 2018).

```r
# Set up design matrix with
ec_design <- data.frame(Sample=colnames(ec_rawcounts),
                        Condition=c(rep("case",6), rep("control",6), rep("case",6),
                                    rep("control",6), rep("case",6),
                                    rep("control",6)),
                        Time=rep(c(0,30,60,90,120,150),6))
```

```r
# Set `boolCaseCtrl = TRUE` because our data has both case and control conditions,
#`scaQThres = 0.01` specifies the level of significance,
#`scaNProc = 8` speeds up the function's runtime,
#and `boolIdentifyTransients = TRUE` indicates that we are also fitting the sigmoidal model.
impulse_ecoli <- ImpulseDE2::runImpulseDE2(as.matrix(ec_rawcounts),
                                           ec_design, boolCaseCtrl=TRUE,
                                           vecConfounders=NULL, scaQThres = 0.01,
                                           scaNProc = 8,
                                           boolIdentifyTransients = TRUE)
```

3.3 maSigPro

maSigPro is another serial method that determines genes with significant expression differences over time and between conditions. According to Conesa and Nueda (2017), the package performs differential expression analysis by:

1. Defining a regression model for the data with the experimental conditions identified by dummy variables.
2. Using least-squared method to adjust the model.
3. Selecting differentially expressed genes controlling for false discovery rates.
4. Applying stepwise regression to determine differential expression between groups (Conesa and Nueda 2017).

```r
# Set up design
Time <- rep(c(0,30,60,90,120,150),6)
Replicate <- rep(1:3, 12)
Control <- c(rep(0, 6), rep(1, 6), rep(0, 6), rep(1, 6), rep(0, 6), rep(1,6))
d_Rpos <- c(rep(1, 6), rep(0, 6), rep(1, 6), rep(0, 6), rep(1, 6), rep(0, 6))
ec_design <- cbind(Time, Replicate, Control, d_Rpos)
```

```r
rownames(ec_design) <- colnames(ec_rawcounts)

# Get DESeq2 normalized counts and create a design matrix
ec_normcounts <- as.data.frame(counts(ec_dds_de, normalized=TRUE))
ec_design <- make.design.matrix(ec_design, degree=5)
```

```r
# perform a regression fit for our gene expression data.
# Set `Q = 0.01` for our significance level,
# set `MT.adjust = "BH"` to adjust our p-values using the Benajamini & Holderberg correction method
fits <- p.vector(ec_normcounts, ec_design, Q=0.01, MT.adjust = "BH",
                 min.obs=6, counts=TRUE)

# perform a stepwise regression fit for our gene expression data.
# Set `step.method = "backward"` and `alfa = 0.01` to specify significance level
tsep <- T.fit(fits, step.method = "backward", alfa=0.01)

# Get significant genes between delta_RpoS and Control
sigs <- get.siggenes(tsep, rsq = 0.6, vars = "groups")
maSigProGenes <- sigs$summary$d_RposvsControl
```

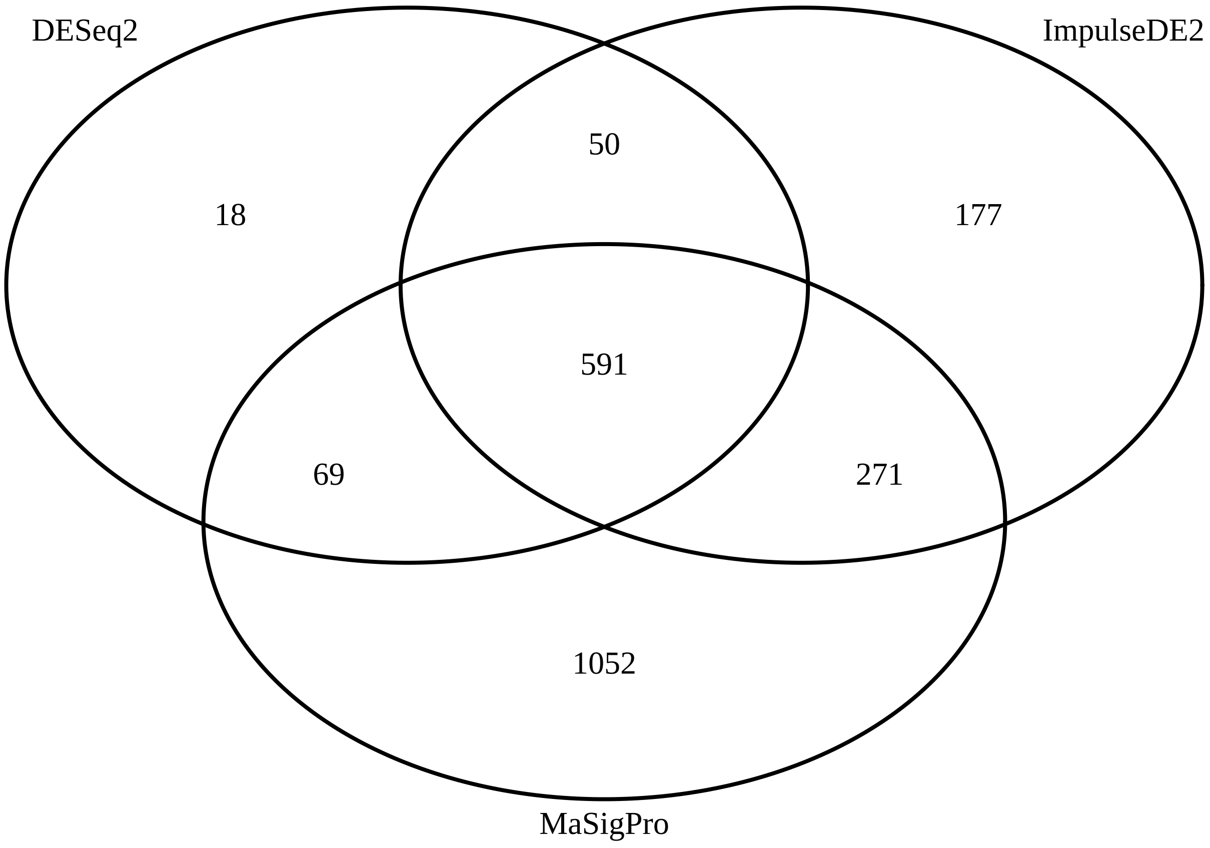
3.4 Find the Intersection

In order to balance our sensitivity with false discovery rates, we decided to consider genes differentially expressed if they were identified in at least 2 of the 3 differential expression methods. For the Impulse trajectory analysis, we extracted genes identified by ImpulseDE and one other method, but for the analysis in this write up, we used genes from any 2 of the 3 methods.

First, we made lists of genes from all of the combinations of intersections between each method and then took the union of that list.

```r
# Intersect each gene list with another method
DS_IM <- distinct(data.frame("genename" = intersect(EC_sig_genes,
                                                    impulse_ecoli$vecDEGenes)))
IM_MA <- distinct(data.frame("genename" = intersect(impulse_ecoli$vecDEGenes,
                                                    maSigProGenes)))
MA_DS <- distinct(data.frame("genename" = intersect(maSigProGenes, EC_sig_genes)))

# Take the union of all three of these intersections, making sure to delete duplicate genes.
EC_DEGs <- distinct(bind_rows(DS_IM, IM_MA, MA_DS))
```

Next we can visualize this overlap with the `vennDiagram` package.



```
## (polygon[GRID.polygon.67], polygon[GRID.polygon.68], polygon[GRID.polygon.69], polygon[GRID.polygon.
```

Here we can see that the overlap of all of the methods has 591 differentially experssed genes and the overlap of 2 of the 3 methods has 981 differentially expressed genes. Choosing which intersections to use depends on research intentions and optimal numbers of differentially expressed genes.

4 Visualization

4.1 Tidying Our Data

```r
# Join list of differentially expressed gene names with their normalized counts
ec_normcounts$genename <- rownames(ec_normcounts)
EC_DEGs <- left_join(EC_DEGs, ec_normcounts, by = "genename")

head(EC_DEGs[,1:7])
```

```
##   genename   JH01_A01  JH01_A02   JH01_A03  JH01_A04  JH01_A05  JH01_A06
## 1      mog 1466.13806 1898.4897 1197.90249  967.3187  700.4807  569.8971
## 2     nhaA 2256.55602 1197.3430 1108.00753 1507.2910 2654.3007 3535.9702
## 3     kefF   74.80296  106.0709   78.39676  101.0474  247.4839  180.2501
## 4     kefC  160.82637  145.6228  205.92215  299.9846  978.3573  716.3983
## 5     polB  430.11703  494.3984  424.38779  677.8599 1026.1173 1316.9763
## 6     leuD  268.04395  147.4206  202.78628  278.9330  490.6259  595.2089
```

As you can see from the preview of our data, it is currently in a format where each gene has one row with all of the treatments and time points as columns. However, in order to plot our gene expression data, we want each gene at each time point and condition to have a row. We must tidy our data using `tidyr` and `dplyr` functions. We also average our data over the three samples for each time frame in order to minimize variability from the samples.

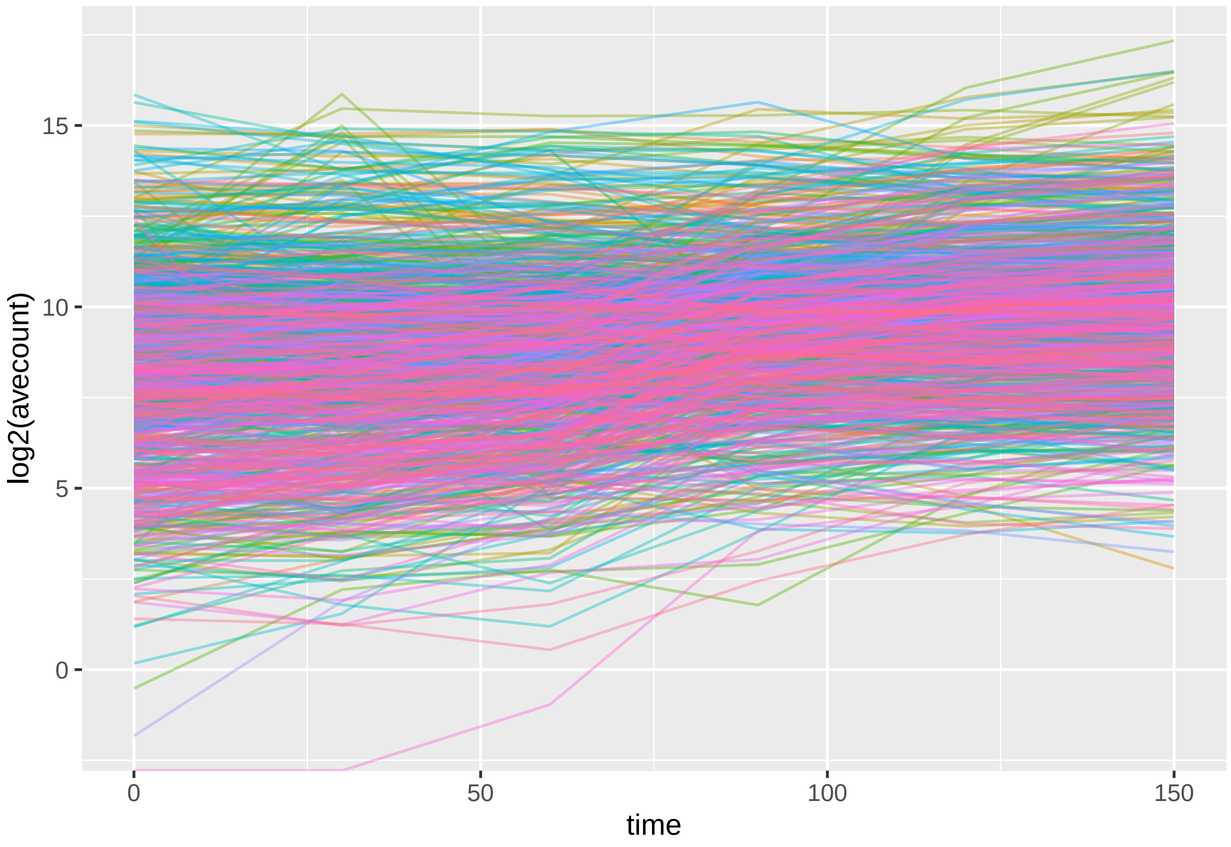
```r
EC_counts_gath <- EC_DEGs %>%
  # Gather count data for each sample by gene
  tidyr::gather(-genename, key="sample", value = "rawcount") %>%
  # Separate columns into treat and rep_time
  tidyr::separate(sample, c("treat", "rep_time"), "_") %>%
  # Create rep column from rep_time column
  dplyr::mutate(rep = ifelse(substr(rep_time,start=1,stop = 1) == "A", 1,
                       ifelse(substr(rep_time,start=1,stop = 1) == "B", 2, 3))) %>%
  # Create time column from rep_time column
  dplyr::mutate(time = ifelse(substr(rep_time,start=3,stop = 3) == "1", 0,
                     ifelse(substr(rep_time,start=3,stop = 3) == "2", 30,
                            ifelse(substr(rep_time,start=3,stop = 3) == "3", 60,
                                   ifelse(substr(rep_time,start=3,stop = 3) == "4", 90,
                                          ifelse(substr(rep_time,start=3,stop = 3) == "5", 120, 150)))
  # Rename treatment with more descriptive value
  dplyr::mutate(treat = ifelse(treat == "JH01", "WT","dRpoS")) %>%
  # Delete rep_time combo column
  dplyr::select(-c(rep_time))

EC_counts_ave <- EC_counts_gath %>%
  # Group data frame by genename, treatment, and time
  dplyr::group_by(genename, treat, time) %>%
  # Take the average of the rawcounts across the grouped variables
  dplyr::summarise(avecount = mean(rawcount)) %>%
  # Select only WT (our case) average count data
  filter(treat == "WT")

# Rename
colnames(EC_counts_ave) <- c("genename", "treat", "time", "avecount")
head(EC_counts_ave)
```

```
## # A tibble: 6 x 4
## # Groups:   genename, treat [1]
##   genename treat  time avecount
##   <chr>    <chr> <dbl>    <dbl>
## 1 accD     WT        0    6170.
## 2 accD     WT       30    6757.
## 3 accD     WT       60    4703.
## 4 accD     WT       90    3080.
## 5 accD     WT      120    2497.
## 6 accD     WT      150    2283.
```

4.2 ggplot2

Finally, we can plot the tidy average expression data using `ggplot()`.

```
EC_counts_ave %>%
  # We also apply a `log2()` transformation to the y-axis of our expression data in order to view all o
  ggplot(aes(x=time, y=log2(avecount))) +
  # We use `geom_line()`, with the parameter `color = genename`
  # in order to color the gene names separately and create a line plot with each gene's expression
  geom_line(aes(color = genename), alpha = 0.4) +
  # Make sure to specify `legend.position = "none"`
  theme(legend.position = "none")
```



In our case, these plots are not extremely helpful because they are so overcrowded and it is difficult to see any trends. In Section 5, we will jump into clustering methods and other analyses that can help to identify specific gene profiles.

4.3 Shiny App

`Shiny` is an R package that allows us to create interactive web apps inside of R. While our Shiny code is not interactive or compatible with the PDF export format, this code runs within R and creates a pop-up app that allows the viewer to visualize the gene expression data. The code is included in the .Rmd version of this file. `Shiny` is especially helpful when trying to determine optimal cut-offs for p-values or fold changes, or when switching between comparisons of the same data.

5 Analysis

As seen in 4.2, our data is still messy and fairly meaningless. We use PAM clustering, Gene Ontology, and Gene Set Enrichment Analysis in order to identify unique profiles within the mass of differentially expressed genes and extract biologically significant results. With Professor Schulz's *T. brucei* data, we used Fuzzy clustering as well, which is a potential future research area for the *E. coli* data.

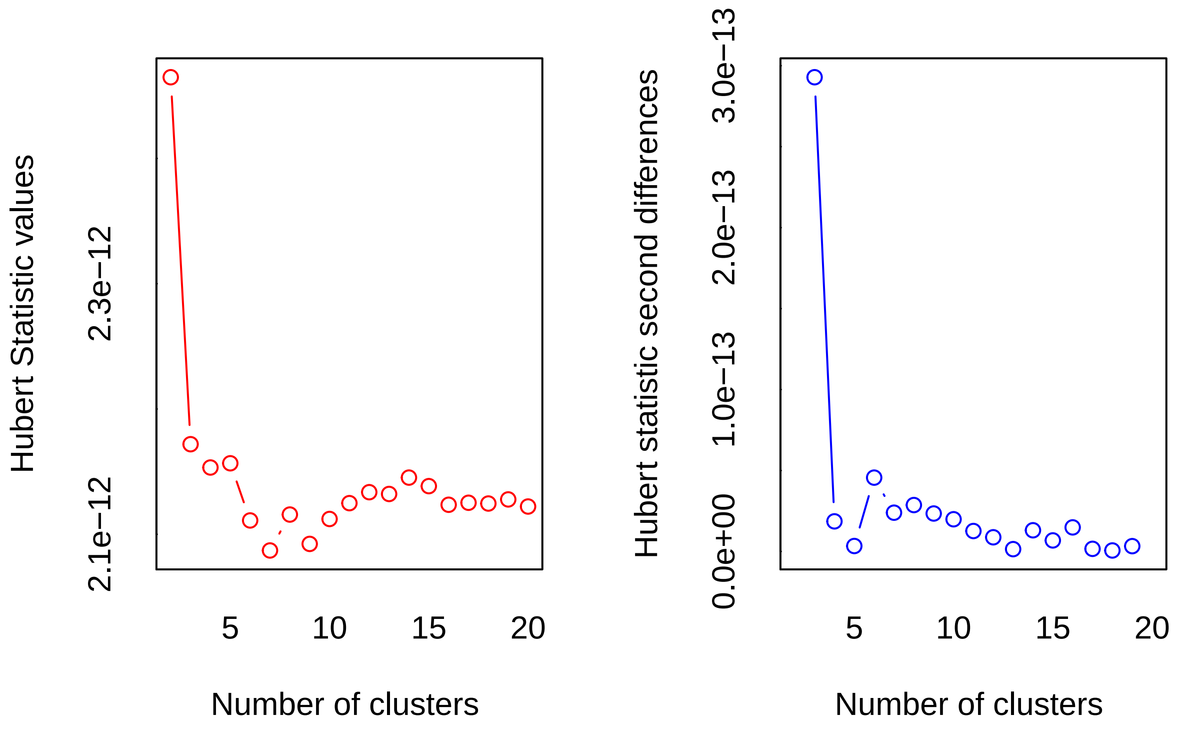
5.1 PAM Clustering

Partitioning Around Medoids (PAM) is a clustering method that iteratively groups genes into clusters of genes with similar profiles. For our code, we input a dissimilarity matrix, which is the transposed matrix of the correlation between each gene, based on their counts. We transpose the count matrix because we want all of the genes as columns in order to calculate the correlation. To calculate correlation, we specified the method as "spearman", however "pearson" returns similar results. The PAM method determines clusters by:

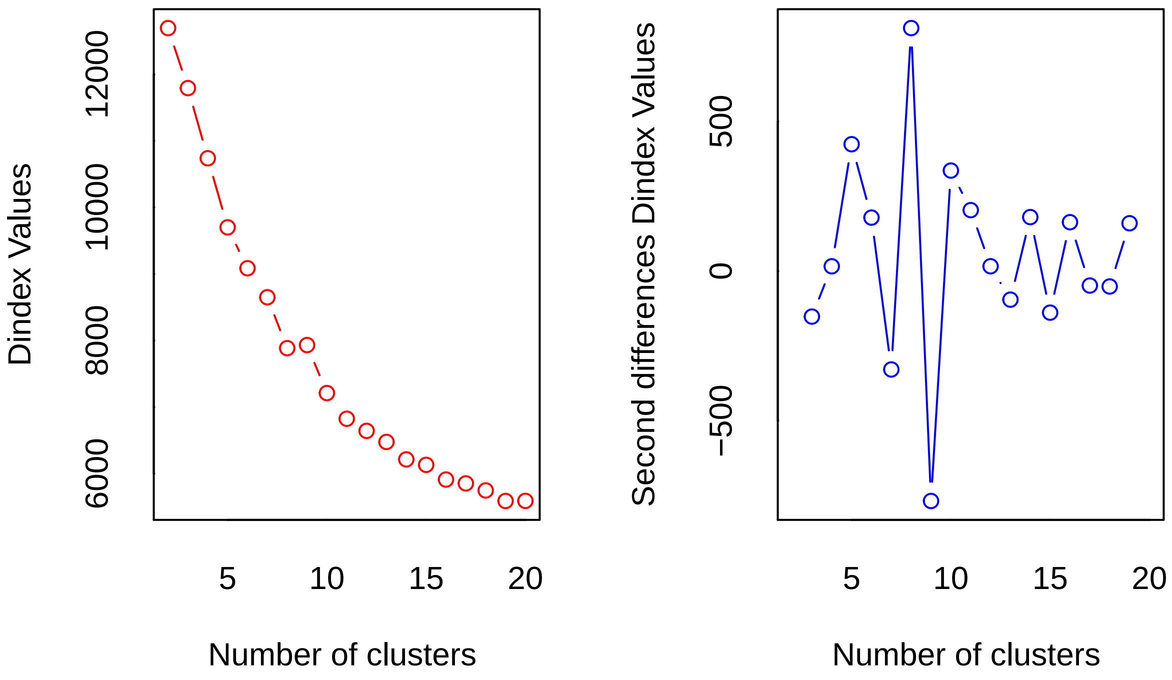
1. Choosing k center genes (medoids).
2. Assigning all of the rest of the genes to the closest medoids.
3. Calculating new medoids from these clusters – in our case, this is based off of the dissimilarity matrix created at the beginning.
4. Repeating steps 2 and 3 until the clusters and medoids stay the same between iterations.

First however, we compute the optimal k number of clusters. This package `NbClust` runs through 22 different tests that each determine the optimal k and plots the distribution of all 22 of the optimal k results in order to determine the most common optimal k. Different values for k will categorize the data differently, thus it is importatnt to determine an optimal k value.

```r
rownames(EC_DEGs) <- EC_DEGs$genename
EC_DEGs <- EC_DEGs %>%
  dplyr::select(-genename)

# Set `diss = NULL` because our data is not a dissimilarity matrix.
# We set `distance = euclidean` because euclidean distance is the distance measurement
# used by the kmeans method, which we specify as the method
k_test <- NbClust(data = EC_DEGs, diss = NULL, distance = "euclidean",
                  min.nc = 2, max.nc = 20, method = "kmeans")
```



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```



```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                   second differences plot) that corresponds to a significant increase of the value of
##                   the measure.
##
## *******************************************************************
## * Among all indices:
## * 1 proposed 2 as the best number of clusters
## * 13 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 3 proposed 16 as the best number of clusters
## * 1 proposed 18 as the best number of clusters
## * 1 proposed 19 as the best number of clusters
## * 2 proposed 20 as the best number of clusters
##
##                      ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

In these plots, `NbClust` determines k=3 to be optimal, because 13 of the 22 methods identify 3 as the optimal number of clusters. However, we take the results of this function with a grain of salt, because `NbClust` uses a different clustering method than the PAM method that we use. We ended up using k = 6 so that we did not miss any unique profiles. This is a potential area for more research, looking into the best k, the outcomes of the clusters, and the significance of their GO terms depending on the number of clusters.

Then we use the PAM method to cluster the DEGs' count data.

```
k = 6

# Calculate the dissimilarity matrix
dissimilarity_matrix <- 1 - cor(t(EC_DEGs), method = "spearman")

# Run the pam() function which clusters the data
# `pamonce = 3` speeds up the function by maximizing short cuts
EC_DESeq_clusters <- cluster::pam(x = dissimilarity_matrix, k = k, pamonce = 3)

# Create a data table with the cluster name and genename
gene_cluster <- data.frame("cluster" = EC_DESeq_clusters$clustering,
                           "genename" = names(EC_DESeq_clusters$clustering))

# Make a data frame with the names of the medoid for each of the k clusters
EC_PAM_medoids <- data.frame("genename" = rownames(EC_DESeq_clusters$medoids))

ec_norm_counts <- ec_normcounts %>%
  mutate("genename" = rownames(ec_normcounts))

# Join the normalized counts with the clusters of genes by their genename
EC_cluster_df <- left_join(gene_cluster, ec_norm_counts, by = "genename")
```

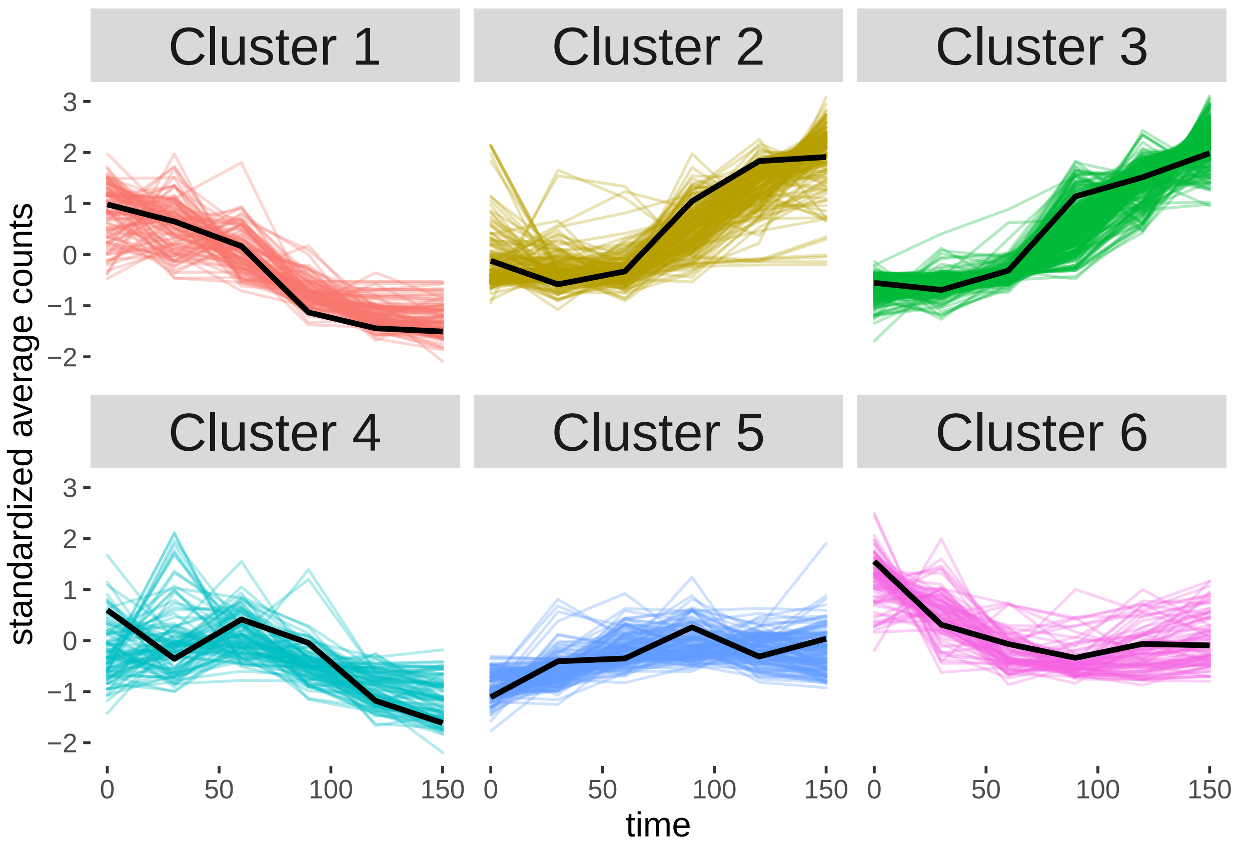
If we plot the clusters from PAM, the different profiles are not very distinct. So we scale our data using the `Mfuzz` package's `standardise()` function. The function standardizes the data so that each gene has a mean expression of 0 and a standard deviation of 1. We standardize our data with the Fuzzy package in order to visualize and understand the profiles within each cluster and scale the plots appropriately in order to compare expression dynamics.

```r
cluster_standard <- EC_cluster_df %>%
  dplyr::select(-cluster)
rownames(cluster_standard) <- cluster_standard$genename
cluster_standard <- cluster_standard %>%
  dplyr::select(-genename)

# Create a phenotype data frame, which describes the conditions/columns of the mnain data frame.
metadata <- data.frame(labelDescription = c("treat", "rep", "time", "timetreat"))
phenoData <- new("AnnotatedDataFrame", data=ec_coldata, varMetadata=metadata)

# Convert the gene expression data frame to a matrix
clusters <- as.matrix(cluster_standard)

# Create an object of type `ExpressionSet`
eset <- Biobase::ExpressionSet(assayData = clusters, phenoData = phenoData)

# Plug `eset` into the `standardise()` function
# and extract the standardized matrix and convert it into a data frame
cluster_standardized <- Mfuzz::standardise(eset = eset)
standard_clusters <- data.frame(cluster_standardized@assayData$exprs)
standard_clusters$genename <- rownames(standard_clusters)

# Join the cluster data with the standardized expression counts
clusters <- left_join(standard_clusters, gene_cluster, by = "genename")
```

Then, we can use the tidying pipeline from Section 4.1 to tidy the standardized data.

Next, we extract the medoid genes and tidy the medoid gene expression data in order to plot the medoids on top of the gene clusters. This step can be helpful to determine a general trend for each cluster.

Now we can plot our clusters using `ggplot`.



Finally, we create a list of data frames that contain each clusters list of genes.

```
cluster_genes <- data.frame("genename" = gene_cluster$genename)
rownames(cluster_genes) <- cluster_genes$genename
cluster_genes <- left_join(cluster_genes, gene_cluster, by = "genename") %>%
  filter(!is.na(genename))
rownames(cluster_genes) <- cluster_genes$genename

# List of data frames with gene names for each cluster called `EC_cluster`
EC_cluster <- c()
EC_cluster$cluster1 <- cluster_genes %>%
  filter(cluster == 1) %>%
  dplyr::select(genename)
# Repeat for rest clusters 2-6
```

5.2 Gene Ontology Analysis

Gene Ontology (GO) analysis is a tool used in order to determine whether specified groups of genes have unique biological significance. Gene Ontology Analysis relies on a list of GO terms, which is a list of all of the genetic functions within the organism and the respective genes involved in each of those functions (Alexa and Rahnenfuhrer, 2019). For each of the specified groups of genes, the analysis determines whether the list of genes is disproportionately distributed across genetic functions. Compared to the null (uniform) distribution of the group's genes across all genetic functions, the analysis tests whether or not the group's genes are significantly enriched within specific functions.

To answer this question, we use a package called `TopGO` which has a database with all of *E. coli*'s GO terms. We can specify the statistical method used to determine whether a GO term is significant within a group of genes. Below, I provide code for the Fisher method, but the other option, a Kolmogorov-Smirnov test, is a simple substitution. GO terms are classified within three different categories: Biological Processes (BP), Molecular Functions (MF), and Cellular Components (CC). After consultation with Professor Stoebel, Biological Process appears to be the most relevant in this research. However, the other two types have positives that could be more useful in other work: Molecular Function can offer specific, molecular-level information and Cellular Component can provide broad information about cell parts and structure.

```r
# Create a function that runs GO analysis,
# with the two parameters being the type of GO and the cluster number.
go_enrich <- function(go_type, k) {
  # Create a function within the function that
  # returns a logical list marking whether or not a gene is in cluster k
  selection <- function(k) {
    gene_clust <- cluster_genes$cluster == k
    return(gene_clust)
  }
  # Create a factor with 2 levels:
  # (1) integer versions (TRUE = 1 and FALSE = 0) of the logical result
  # from the above `selection()` function
  # and (2) their respective gene name
  GeneList <- factor(as.integer(selection(k)))
  names(GeneList) <- cluster_genes$genename

  # Map gene IDs to GO terms, both from database.
  # Set `whichOnto = go_type` so that the function specifies the type of GO terms
  # and set `ID = "symbol"` because our gene data is identified
  # by their symbol, the four letter combination associated with each gene
  annot <- annFUN.org(whichOnto = go_type,
                      feasibleGenes = NULL, mapping = "org.EcK12.eg", ID = "symbol")

  # Create a `topGOdata` object. Again, specify `ontology = go_type`.
  # Set `allGenes = GeneList` which is our factor object with 1s and 0s and gene names.
  # Then we set `annot = annFUN.GO2genes` and `GO2genes = annot`,
  # our previous object of mapped gene IDs and their GO terms.
  GOdata <- new("topGOdata", ontology=go_type, allGenes=GeneList,
                annot=annFUN.GO2genes, GO2genes = annot,
                geneSelectionFun = selection, nodeSize = 5)

  # Run the GO analysis, specifying the algorithm and statistic.
  # If a Kolmogorov-Smirnov test is preferred,
  # change `statistic = "fisher"` to `statistic = "ks"` and rename object as `resultKS`.
  resultFis <- runTest(GOdata, algorithm = "classic", statistic = "fisher")

  # Generate a table with the results ranked by their p-value.
  # If a Kolmogorov-Smirnov test is preferred,
  # change `fisherPval = "resultFis"` to `ksPval = "resultKS"`
  goEnrich <- GenTable(GOdata, fisherPval = resultFis, topNodes = 30)

  # Create a column that identifies the genes from the cluster
  # within each GO term. Mostly NULL values
  ind <- match(goEnrich$GO.ID, names(annot))
  goEnrich$Gene <- annot[ind]
  return(goEnrich)
}
```

```r
# Run the `go_enrich` function for each type within each cluster
# and bind the results, specifying the type of GO term in each case
bp <- go_enrich("BP", 1) %>%
  mutate("Type" = "Biological Process")
mf <- go_enrich("MF", 1) %>%
  mutate("Type" = "Molecular Function")
cc <- go_enrich("CC", 1) %>%
  mutate("Type" = "Cellular Component")
cluster1_genes <- bind_rows(bp, mf, cc)

# Repeat for clusters 2-6
```

```r
# Create a list of data frames with the results from each of the clusters
GO_clusters <- c()
GO_clusters$cl1 <- cluster1_genes
# Repeat for clusters 2-6
```

5.3 Gene Set Enrichment Analysis

Gene Set Enrichment Analysis (GSEA) is a tool that allows us to test whether or not a determined set of genes is enriched in a list of genes with different levels of differential expression, noted by a test statistic. GSEA requires this list of genes to have a ranking statistic. In our case, we used a p-value statistic which describes the significance of the gene's differential expression. GSEA results with high positive enrichment scores indicate enrichment at the top of the ranked gene list, while large negative enrichment scores indicate enrichment at the bottom of the list. Small enrichment scores, regardless of the sign, are not as significant.

We were interested in determining whether or not sensitive or insensitive genes were over represented in a particular cluster, so we compared each the 6 clusters to the lists of sensitive, insensitive, and linear genes to determine whether the sensitive, insensitive, or linear genes were enriched in any of our clusters. For example, were there more sensitive genes in cluster 1 than would be expected under a uniform distribution of sensitive genes? Our results were not very significant, however further analysis could potentially check other gene sets for enrichment.

```r
# create data frame for ranked list with all genenames and pvalues
# I used the gene list output by DESeq2 because this list has
# an adjusted pvalue associated with each gene
# and seemed to capture a lot of the genes from the venn diagram
ec_Stat <- data.frame("genename" = resSig$genename, "padj" = resSig$padj)

stat_all <- ec_Stat$padj
names(stat_all) <- ec_Stat$genename

cluster1 <- left_join(EC_cluster$cluster1,ec_Stat, by = "genename") %>%
  filter(!is.na(genename))
stat_cluster1 <- cluster1$padj
names(stat_cluster1) <- cluster1$genename

# Repeat for clusters 2-6
```

```r
sens <- read_csv("past_sensitivity.csv") %>%
  dplyr::select(c(geneName,bNum,
                  `Differentially expressed between 0% and 100% RpoS`,
                   `Direction of regulation`, sensitivity)) %>%
  filter(`Differentially expressed between 0% and 100% RpoS` == TRUE) %>%
  dplyr::select(-`Differentially expressed between 0% and 100% RpoS`)

sens_name <- c()
sens_name$sensitive <- sens$geneName[which(sens$sensitivity=="sensitive")]
sens_name$insensitive <- sens$geneName[which(sens$sensitivity=="insensitive")]
sens_name$linear <- sens$geneName[which(sens$sensitivity=="linear")]

fgsea(sens_name, stat_all, minSize = 1, maxSize = 500, nperm = 10000)
```

```
##        pathway      pval      padj         ES      NES nMoreExtreme size
## 1:   sensitive 0.2252775 0.4505549  0.7157443 1.031338         2252   89
## 2: insensitive 1.0000000 1.0000000 -0.6753731      NaN            0   18
##                          leadingEdge
## 1:  yjhP,yiaW,ugpB,crr,asnA,pepT,...
## 2: mdtF,hdeD,gadC,yhiM,ybgS,gadA,...
```

```
fgsea(sens_name, stat_cluster1, minSize = 1, maxSize = 500, nperm = 10000)
```

```
##    pathway      pval      padj        ES       NES nMoreExtreme size
## 1:  linear 0.7685686 0.7685686 0.5240021 0.8968567         7677   20
##                leadingEdge
## 1: ompT,chiP,ygdG,fimC,tsx
```

```
# Repeat for clusters 2-6
```

```
##        pathway       pval       padj         ES        NES nMoreExtreme
## 1:   sensitive 0.49135086 0.59444056  0.8960404  1.0065010         4913
## 2: insensitive 0.02696456 0.08089368 -0.8631579 -1.3897488           34
## 3:      linear 0.59444056 0.59444056  0.8780041  0.9937833         5944
##    size                          leadingEdge
## 1:   28                        crr,ihfA,yhhX
## 2:    3                       hdeD,yhiM,aroM
## 3:  101 fruA,pmbA,yhcH,fruK,kefF,ybiU,...
```

```
##        pathway      pval      padj        ES       NES nMoreExtreme size
## 1:   sensitive 0.7531247 0.9994001 0.7976242 0.9739734         7531   55
## 2: insensitive 0.9988001 0.9994001 0.5551723 0.6614341         9988   15
## 3:      linear 0.9994001 0.9994001 0.7201791 0.8876963         9994  139
##                          leadingEdge
## 1:          ugpB,pepT,acnA,ybeL,ytfQ
## 2:                         appA,cbdA
## 3: yiiS,ybbY,grxB,ycgZ,yiiM,gabD,...
```

```
##      pathway      pval      padj       ES       NES nMoreExtreme size
## 1: sensitive 0.1435897 0.2871795 0.940000 1.2667783          727    1
## 2:    linear 0.9079092 0.9079092 0.663353 0.8796214         9079   48
##                          leadingEdge
## 1:                              asnA
## 2: yehU,ymfD,proX,yhfA,intD,cyoE,...
##    pathway       pval       padj       ES      NES nMoreExtreme size
## 1:  linear 0.01089891 0.01089891 0.822279 1.202394          108   77
##                          leadingEdge
## 1: yfbS,yaiZ,pmrD,yjhH,ycbJ,ygfZ,...
```

```
##      pathway      pval      padj         ES        NES nMoreExtreme size
## 1: sensitive 0.3062589 0.6125177 -0.8481013 -1.1323543         1511    1
## 2:    linear 0.6630337 0.6630337  0.6723633  0.9604224         6630   46
##                          leadingEdge
## 1:                              folA
## 2: wecF,fecR,lptC,trmI,wzyE,efeB,...
```

6 Conclusion

This pipeline provides the structure for understanding noisy RNA-Seq data through normalization, differential expression analysis, tidying, clustering, gene ontology, and gene set enrichment analysis.

Gene set enrichment analysis allowed us to compare our differentially expressed gene clusters to the lists of sensitive and insensitive genes from past research. We used gene ontology analysis to compare our clusters with genetic functions to determine whether specific functions are enriched in certain profiles.

Further research is mainly centered around the ImpulseDE trajectories and parameters from 3.2 for previously labeled sensitive and insensitive genes. Other research could look into clustering – specifically, looking at what value of k is optimal and which clustering method produces the clearest results for downstream analysis such as gene ontology or gene set enrichment analysis.

7 References

Alexa A, Rahnenfuhrer J (2019). topGO: Enrichment Analysis for Gene Ontology. R package version 2.36.0.

Battesti, et al. "The RpoS-Mediated General Stress Response in Escherichia Coli." Annual Reviews, 2011, www.annualreviews.org/doi/10.1146/annurev-micro-090110-102946.

Conesa A and Nueda MJ (2017). maSigPro: Significant Gene Expression Profile Differences in Time Course Gene Expression Data. R package version 1.50.0, http://bioinfo.cipf.es/.

Evans, Ciaran, et al. "Selecting between-Sample RNA-Seq Normalization Methods from the Perspective of Their Assumptions." OUP Academic, Oxford University Press, 27 Feb. 2017, academic.oup.com/bib/article/19/5/776/3056951.

Fischer, David S., et al. "Impulse Model-Based Differential Expression Analysis of Time Course Sequencing Data." BioRxiv, Cold Spring Harbor Laboratory, 1 Jan. 2017, doi.org/10.1101/113548.

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2019). dplyr: A Grammar of Data Manipulation. R package version 0.8.3. https://CRAN.R-project.org/package=dplyr

Hengge, Regine. "Stationary-Phase Gene Regulation in Escherichia Coli §." EcoSal Plus, America, www.asmscience.org/content/journal/ecosalplus/10.1128/ecosalplus.5.6.3.

Mistry, Meeta, et al. "Gene-Level Differential Expression Analysis with DESeq2." Introduction to DGE, 12 May 2017, hbctraining.github.io/DGE_workshop/lessons/04_DGE_DESeq2_analysis.html.

Struyf, Anja, Mia Hubert, & Peter Rousseeuw. "Clustering in an Object-Oriented Environment." Journal of Statistical Software [Online], 1.4 (1997): 1 - 30. Web. 17 Sep. 2019

Ross, Zev. "R Powered Web Applications with Shiny (a Tutorial and Cheat Sheet with 40 Example Apps)." Technical Tidbits From Spatial Analysis & Data Science, 13 Sept. 2017, zevross.com/blog/2016/04/19/r-powered-web-applications-with-shiny-a-tutorial-and-cheat-sheet-with- 40-example-apps/.