Differential Expression Analysis for RNA-Seq Data

Installing edgeR

edgeR is a Bioconductor package that performs differential gene expression analysis using count data under a negative binomial model. The software works on a table of integer read counts, with rows corresponding to genes and columns to independent libraries. The counts represent the total number of reads aligning to each gene. The methods used in edgeR do not support FPKM, RPKM or other types of data that are not counts. Similarly, edgeR is not designed to work with estimated expression levels, for example as might be output by Cufflinks.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("edgeR")
```

Loading Data

We will use a data from *Saccharomyces Cerevisiae* experiment [1]. Ten strains of *Saccharomyces Cerevisiae* grown in three media, namely YPD, Delft and Glycerol each with 3-4 biological replicates were sequenced using Illumina's Genome Analyzer II. The sequencing yielded 36 bp-long single-end reads. Reads were mapped to the reference genome (SGD release 64) using Bowtie, considering only unique mapping and allowing up to two mismatches.

First, download data files from *Data.zip* from Blackboard to your R working directory, then load the datasets into R.

```
> load ("geneLevelCounts.rda")
> load ("geneInfo.rda")
> load ("laneInfo.rda")

> class (geneLevelCounts)
[1] "matrix"
> dim(geneLevelCounts)
[1] 6575   14
> head(geneLevelCounts, 3)
        Y1_1 Y1_2 Y2_1 Y2_2 Y7_1 Y7_2 Y4_1 Y4_2   D1   D2   D7    G1   G2   G3
YAL062W   11    4    6    8   12    9   41   43   54   38   44  1628   57  256
YAL061W   33   17   50   20   77   51  177  166  311  338  301 29951 1310 2208
YAL060W  209  129  216  181  387  286 1328 1386 3316 1262 1130 16548 5222 3482


> head(geneInfo, 3)
          length        GC
YAL069W      315 0.4349206
YAL068W-A    255 0.3529412
YAL068C      363 0.4958678
```

```
>laneInfo
     lib_prep growth_cond flow_cell lib_prep_proto
Y1_1       Y1         YPD     428R1      Protocol1
Y1_2       Y1         YPD     4328B      Protocol1
Y2_1       Y2         YPD     428R1      Protocol1
Y2_2       Y2         YPD     4328B      Protocol1
Y7_1       Y7         YPD     428R1      Protocol1
Y7_2       Y7         YPD     4328B      Protocol1
Y4_1       Y4         YPD     61MKN      Protocol2
Y4_2       Y4         YPD     61MKN      Protocol2
D1         D1         Del     428R1      Protocol1
D2         D2         Del     428R1      Protocol1
D7         D7         Del     428R1      Protocol1
G1         G1         Gly     6247L      Protocol2
G2         G2         Gly     62OAY      Protocol1
G3         G3         Gly     62OAY      Protocol1
```

We want to filter out the non-expressed genes. For simplicity, we consider only the genes with an average read count of 10 or more.

```
> means <- rowMeans(geneLevelCounts)
> filter <- means >= 10
> table(filter)
filter
FALSE  TRUE
 1041  5534

> geneLevelCounts <- geneLevelCounts[filter,]
> dim (geneLevelCounts)
[1] 5534   14
```

One of the main characteristics of RNA-seq data is that each library will generally have a different sequencing depth. We can visualize the total number of mapped reads to known genes with the barplot function. To check for systematic effects we can color-code the plot by different biological or technical variables.

```
> library(RColorBrewer)
> colors <- brewer.pal(9, "Set1")
> totCounts <- colSums(geneLevelCounts)
> barplot(totCounts, las=2, col=colors[laneInfo[,2]])
> barplot(totCounts, las=2, col=colors[laneInfo[,4]])
```

The boxplot function provides an easy way to visualize the difference in distribution between each experiment.

```
> boxplot(log2(geneLevelCounts+1), las=2, col=colors[laneInfo[,4]])
```

Comparison between Delft and Glycerol yeast libraries.

Building the edgeR object

DGEList is the function that coverts the count matrix into an edgeR object. In addition to the counts, we need to group the samples according to the variable of interest in our experiment. We can then see the elements that the object contains by using the names function

```
> library(edgeR)
> group <- laneInfo[9:14,2]
> group <- droplevels(group)
> counts <- geneLevelCounts[, 9:14]
> cds <- DGEList( counts , group = group )
> names(cds)
[1] "counts"  "samples"
```

These elements can be accessed using the $ symbol.

```
>head(cds$counts, 3) # original count matrix
          D1   D2   D7    G1   G2   G3
YAL062W   54   38   44  1628   57  256
YAL061W  311  338  301 29951 1310 2208
YAL060W 3316 1262 1130 16548 5222 3482

>cds$samples # contains a summary of your samples
   group lib.size norm.factors
D1   Del  6048367            1
D2   Del  2851240            1
D7   Del  2562678            1
G1   Gly 21568559            1
G2   Gly  4597674            1
G3   Gly  4830425            1
```

Normalization

We can compute effective library sizes using TMM normalization to account for compositional difference between the libraries.

```
> cds <- calcNormFactors(cds)
> cds$samples
   group lib.size norm.factors
D1   Del  6048367    1.0161315
D2   Del  2851240    0.8774570
D7   Del  2562678    0.8738484
G1   Gly 21568559    0.9423705
G2   Gly  4597674    1.2645464
G3   Gly  4830425    1.0770397
```

The effective library sizes are then the product of the actual library sizes and these factors.

# effective library sizes
```
> cds$samples$lib.size*cds$samples$norm.factors
[1]  6145937  2501840  2239392 20325574  5813972  5202559
```

By default, the function calcNormFactors normalize the data using the "weighted trimmed mean of M-values" (TMM) method, proposed by [2]. Other options are RLE [3] and upper-quartile [4]. If we want to use the upper-quartile to normalize, we can add an extra argument to the function

```
> cds <- calcNormFactors(cds, method="upperquartile")
> cds$samples
   group lib.size norm.factors
D1   Del  6048367    0.9520041
D2   Del  2851240    0.8565882
D7   Del  2562678    0.8504893
G1   Gly 21568559    1.0664895
G2   Gly  4597674    1.2137072
G3   Gly  4830425    1.1139090
```

Estimating Dispersion

The first dispersion type to calculate is the common dispersion. In the common dispersion setting, each gene gets assigned the same dispersion estimate. The output of the estimation will include the estimate as well as some other elements added to the edgeR object, cds.

```
> cds <- estimateCommonDisp(cds)
> names(cds)
[1] "counts"          "samples"          "common.dispersion"
[4] "pseudo.counts"   "AveLogCPM"        "pseudo.lib.size"
```

We can see the estimate of the common dispersion.
```
>  cds$common.dispersion
[1] 0.1084335
```

To understand what this value means, recall the parameterization for the variance of the negative binomial is $v(\mu) = \mu+\mu^2 \cdot\varphi$. For poisson it's $v(\mu) = \mu$. The implied standard deviations are the square-roots of the variances. Now, suppose a gene had an average count of 200. Then the standard deviation under the two models would be

```
> sqrt(200) # poisson sd
[1] 14.14214
> sqrt(200 + 200^2 * cds$common.dispersion) # negative binomial
[1] 67.35977
```

In real experiments, the assumption of common dispersion is almost never met. Often, we observe a relation between mean counts and dispersion, i.e., the more expressed genes have less dispersion.

The way edgeR estimates a tagwise (i.e. gene-wise) dispersion parameter is by "shrinking" the gene-wise dispersions toward a common value (the common dispersion estimated in the previous step). Alternatively, one can shrink the gene-wise estimates to a common trend, by estimating a smooth function prior to the shrinkage (using the *estimateTrendedDisp* function). Here we keep things simple and shrink the estimates to the common value

```
> cds <- estimateTagwiseDisp(cds)
> plotBCV(cds)
```

The genewise dispersions show a decreasing trend with expression level. At low logCPM (a log2 counts per million, normalized for library sizes), the dispersions are very large indeed:

Now that we've estimated the dispersion parameters we can see how well they fit the data by plotting the mean-variance relationship.

```
> meanVarPlot <- plotMeanVar( cds ,show.raw.vars=TRUE ,
show.tagwise.vars=TRUE , show.binned.common.disp.vars=FALSE ,
show.ave.raw.vars=FALSE , dispersion.method = "qcml" , NBline = TRUE , nbins
= 100, pch = 16 , xlab ="Mean Expression (Log10 Scale)" , ylab = "Variance
(Log10 Scale)" , main = "Mean-Variance Plot" )
```

Four things are shown in the plot: the raw variances of the counts (grey dots), the variances using the tagwise dispersions (light blue dots), the variances using the common dispersion (solid blue line), and the variance = mean a.k.a. poisson variance (solid black line). The plot function outputs the variances which will be stored in the data set meanVarPlot.

Testing for Differentially Expressed (DE)

The function *exactTest* performs pair-wise tests for differential expression between two groups. The important parameter is pair which indicates which two groups should be compared. The output of *exactTest* is a list of elements: we can get the table of the results with the *topTags* function.

```
> et <- exactTest(cds, pair=levels(group))
> topTags(et)             #extract top DE  genes, ranked by P-value
> top <- topTags(et, n=nrow(cds$counts))$table
```

We can store the ID of the DE genes and look at the distribution of the *p*-values

```
> de <- rownames(top[top$PValue<0.01,])
> hist(top$PValue, breaks=20)
```

We can use the function *plotSmear* to produce a plot that shows the relationship between concentration and fold-change across the genes. The differentially expressed genes are colored red and the non-differentially expressed are colored black. The blue line is added at a log-FC of 2 to represent a level for biological significance.

```
> plotSmear(cds , de.tags=de)
> abline(h=c(-2, 2), col="blue")
```

We can use the "volcano plot" to visualize the relationship between log-fold-changes and *p*-values.

```
> plot(top$logFC, -log10(top$PValue), pch=20, cex=.5, ylab="-log10(p-value)",
 xlab="logFC", col=as.numeric(rownames(top) %in% de)+1)
> abline(v=c(-2, 2), col="blue")
```

Outputting the results

```
> write.table(top, file="Del_Gly_Comp.txt", sep='\t', quote=FALSE)
```

References

1. Risso D, Schwartz K, Sherlock G, Dudoit S (2011). GC-content normalization for RNA-Seq data. BMC Bioinformatics 12:480
2. Robinson MD, Oshlack A (2010). A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25.
3. Anders S, Huber W (2010). Differential expression analysis for sequence count data. Genome Biology 11:R106.
4. Bullard JH, Purdom E, Hansen KD, Dudoit S (2010). Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. BMC Bioinformatics 11:94