**RNA-seq workshop: from FASTQ files to differential expression**

**1. Designing a RNA-seq experiment with the intention of testing for differential expression**

Ideally, all biological replicates across conditions will be reared under the same conditions and collected at the same time/developmental stage. Even with these controls in place, RNA-seq data suffers from an overdispersion problem (meaning the variance > mean).

To identify whether genes are differentially expressed, the difference in expression between two conditions (or species) has be greater than the uncertainty surrounding a gene’s expression level within a condition (or species). Power for this test is achieved in two ways: **coverage** and **replication**.

**Higher coverage versus more replicates**

Uncertainty surrounding biological variation between replicates can be reduced by increasing the number of replicates. While a high number of biological replicates (at the expense of coverage) will help you call more differentially expressed genes with programs like DESeq or EdgeR, you will likely not be able to identify differential expression in genes with lower read counts. Cost-wise, replicates are also frequently more expensive (through library preparation and time to prepare or rear new samples).

A useful web based tool to help calculate whether you should invest more in coverage or in biological replicates is **Scotty:** <http://bioinformatics.bc.edu/marthlab/scotty/scotty.php>

Ultimately you will need to weigh the costs of increasing coverage vs. increasing the number of replicates given the goals of your experiment. If you anticipate a large degree of biological variation (like if, for example, you are using samples from animals collected in the wild, animals of disparate ages or developmental times, or animals reared in different facilities) favoring replicates over coverage may be the best way to proceed. Conversely, if you are interested in the expression of specific gene families or alleles in a heterozygous individual, higher coverage may be preferential.

**2. Mapping to a genome or transcriptome with Tophat2**

<http://ccb.jhu.edu/software/tophat>

TopHat2 is a splice junction mapper for RNA-Seq reads. Tophat2 uses the short-read aligner Bowtie and analyzes the resulting mapped reads to uncover splice junctions between exons. To map reads with Tophat2 you will need either a transcriptome or genome to map to, but you will not need a reference annotation (however, having a reference annotation will make the program run substantially faster!).

To run Tophat2, you will need **bowtie** (1 or 2) and **samtools** installed and in your path (Note: to temporary add a program to you path, type “export PATH=$PATH:/path/to/the/directory” in your terminal window. These are already added to your path in the virtualbox.)

The input for Tophat2 are FASTQ files. Both unpaired and paired end data can be mapped with Tophat, though it is recommended that you run paired and unpaired data separately.

To start, you will need to build an index for your reference genome or transcriptome. The command **bowtie2-build** builds an index file from your reference that Tophat2 and bowtie2 use to efficiently map reads.

If your reference is a fasta file, you can input the following command to build your index:

bowtie2-build <reference\_file> <basename>

Where <reference\_file> is the path to your reference genome or transcriptome fasta file and <basename> is what you want the prefix of your output index files to be.

As an example, we will be creating an index for an *E. coli* strain. Type the following command to index the genome of the an *E. coli* strain:

bowtie2-build Ecoli\_sampledata/ecoli\_genome.fa ecoli\_index/ecoli\_bw2

After bowtie build has finished running, you should have index files in your index folder.

Once you’ve built the index, you are ready to map your reads. The basic usage for Tophat2 is:

tophat [options]\* <genome\_index\_base> PE\_reads\_1.fq.gz,SE\_reads.fa PE\_reads\_2.fq.gz

Where genome\_index\_base is the basename for the reference followed by your reads. Reads should be in fastq format. We will test this with reads from an *E. coli* dataset (Ecoli\_sampledata/SRR1783109.fastq) and the index we just built. These reads are from an experiment on *E. coli* expression under multiple oxygen conditions.

tophat ecoli\_index/ecoli\_bw2 Ecoli\_sampledata/SRR1783109.fastq

You should see a new directory called tophat\_out. In this directory is the output of our tophat run. The file “accepted\_hits.bam” are the aligned reads. To check the mapping quality, click on align\_summary.txt. In here, we see that 84.4% of our input mapped to the genome (not bad!). The other files in this folder include splice juntions, insertions, and deletions that tophat identified. We will ignore these for now but know that these files are of use when looking for alternative splicing.

**3. Using samtools to manipulate BAM files**

SAM (Sequence Alignment/Map) is a format for storing alignment data. Tophat2 will output data as either a SAM file or a BAM file (the compressed version of a SAM file) depending on your input settings. Our tophat output is in BAM format.

Samtools is a command line utility for manipulating BAM an SAM files. Samtools is all sorts of great applications (mpileup for creating pileup files to find variants, for one), but for now we only need to know how to convert between BAM and SAM formats and sort files to prepare for our next step (counting reads). To sort a BAM file by chromosomal coordinates, you can type:

samtools sort <in.bam> <out.prefix>

To sort a file by read names rather than coordinates:

samtools sort -n <in.bam> <out.prefix>

Let’s sort our bam files now by read names:

samtools sort –n tophat\_out/accepted\_hits.bam ecoli\_sortn

As SAM files are human readable and BAM files are not, it is often necessary to convert between them. You can do this with the following commands:

samtools view -h -o outputfile.sam inputfile.bam

samtools -b -S inputfile.sam > outputfile.bam

Let’s do this now with our bam file:

samtools view -h -o ecoli\_sortn.sam ecoli\_sortn.bam

**4. Counting reads**

***This step is unnecessary if you map your reads to a transcriptome.***

If you have aligned to a genome, you will need to count the reads mapping to each exon. An easy way to do this is to use HTSEQ-count:

[**http://www-huber.embl.de/users/anders/HTSeq/doc/count.html#count**](http://www-huber.embl.de/users/anders/HTSeq/doc/count.html#count)

HTSeq-count requires a GTF/GFF (genome annotation file) with annotated exons to count reads. As long as you have the coordinates for your exons, a GTF file will be easy to create yourself (click here to see the file specification: <http://www.ensembl.org/info/website/upload/gff.html>)

A gene, as counted by HTSeq, is a union of all of its exons (if you are interested in alternative splicing, exons can instead be counted as individual features, however, for identify differential expression between replicates we will be using a sum of the reads mapping to each exon associated with a given gene).

When HTSeq is installed, you can run the following command to count reads:

python -m HTSeq.scripts.count --stranded=no <your alignment\_file> <your gff or gtf file>

If you have paired-end data, your bam files will need to be sorted (the default for HTSeq is by name, which you learned how to do in the previous section).

HTSeq-count has three running modes (option denoted as –m or **--mode**=, options are: union, intersection\_strict, and intersection\_nonempty) that determine how it handles read data. Which mode you choose depends on how conservative you want to be and the quality of your genome annotation. “Union” is the default mode and is applicable to most datasets.

We will not try this here with our *E. coli* dataset in the interest of time. As an example, however, there is an ecoli .gtf file in Ecoli\_sampledata/ folder called ecoli\_APEC.2.gtf. This is what a typical gft file looks like. This gft file and the sorted sam input file tou created are all you need to count reads with HTSeq-count.

Once we have a text file with read counts for all our genes, its time to call differential expression.

**5. Choosing how you call differential expression**

Choosing how to test for differential expression not only depends on the purpose of your experiment, but also the kind of data you have collected for your analysis. RNAseq data suffers from an overdispersion problem, meaning that it has substantial variability and it cannot usually be modeled as a Poisson distribution. Programs designed for RNAseq analysis attempt to model dispersion to find genes that are differentially expressed between treatments. DESeq and EdgeR are two of the most popular programs and use slightly different statistics to call differential expression. Both of these programs are R packages that are easy to install and run and take in the count tables you learned to create in the previous step.

**DESeq –** DESeq models the variance in counts using a negative binomial distribution. DESeq performs best with multiple replicates.

[**http://bioconductor.org/packages/release/bioc/html/DESeq.html**](http://bioconductor.org/packages/release/bioc/html/DESeq.html)

**EdgeR –** EdgeR also models variance based on a negative binomial distribution. The main difference between DESeq and EdgeR is how they handle normalization and model dispersion. Where DESeq normalizes based on a hypergeometric mean, EdgeR normalizes based on a trimmed mean. Practically, this translates to EdgeR frequently being more permissive in what qualifies as differentially expressed, especially for low read count genes. DESeq Differences between the results of DESeq and EdgeR are greatest when there is no biological replication.

[**http://www.bioconductor.org/packages/release/bioc/html/edgeR.html**](http://www.bioconductor.org/packages/release/bioc/html/edgeR.html)

**Do it yourself:** Under some circumstances, it may be best to test for differential expression with your own scripts and statistical-know-how. If the way you have set up your experiment violates the assumptions of the above packages, it is best to test for differential expression on your own.Anything that changes the signal to noise ratio that the above programs expect prior to testing for differential expression will make them underperform and can distort you signal. For example, working with tissues that require pooling prior to sequencing (as with insects) or filtering out specific genes or gene families can cause problems.

**6. Using DESeq to identify differentially expressed genes**

In this example we are going to test for differential expression between light mice and dark mice in the hopes of finding differentially expressed genes related to this color polymorphism. Change directories to look at the sample data:

cd Mus\_sampledata/

head Sample\_dataset\_Mus.txt

Open R (you can just type “R” on the command line if you have this program installed) and then load the DESeq library:

library(DESeq)

Read in your table of count data, which we have named “Sample\_dataset\_Mus.txt”:

countsTable <- read.delim("Sample\_dataset\_Mus.txt",header=TRUE, sep=" ")

rownames(countsTable) <- countsTable$gene

countsTable <- countsTable[,-1]

Let's check that our table has been read in correctly:

head(countsTable)

Now we will enter the conditions of our experiment based on the columns in our count table. In the sample dataset provided, we have two conditions ("L" and "D", which corresponds to light and dark mice):

conds <- factor( c( "L", "L", "L", "L","L","L","L","D","D","D","D","D","D","D" ) )

cds <- newCountDataSet( countsTable, conds )

Note that at this step you can input more than two conditions, but you can only test between two conditions at a given time with the negative binomial test.

Now we normalize the read counts to account for differences in library sizes:

cds <- estimateSizeFactors( cds )

Notice that the read counts change:

head( counts( cds, normalized=TRUE ) )

We will now estimate dispersion. You can think of dispersion as the square of biological variation. It is a per gene estimate of the variability within the dataset:

cds = estimateDispersions( cds )

We can also create a plot of these dispersion values:

str( fitInfo(cds) )

plotDispEsts( cds )

#To save the plot

jpeg("DispEsts\_DESeq.jpg")

plotDispEsts(cds)

dev.off()

The dispersion values (pooled by gene) calculated by DESeq look like this:

head (fData(cds) )

#Save the dispersion values:

write.csv(fData(cds), file="Sample\_dispersionnumbers.csv")

Now to test for differential expression. Perform a negative binomial test between the light mouse and dark mouse replicates:

res = nbinomTest( cds, "L", "D" )

Plot the Log2 fold changes against the mean of the normalized counts:

jpeg("plotMA.jpg")

plotMA(res)

dev.off()

We also look at this as a histogram of p-values:

hist(res$pval, breaks=100, col="skyblue", border="slateblue", main="")

jpeg("histogrampvalues.jpg")

We can now find out how many differentially expressed genes we have at the 1% or 5% false discovery rate between the light and dark mice:

nrow(res[res$padj<0.05 & !is.na(res$padj),]) # at the .05 level

nrow(res[res$padj<0.01 & !is.na(res$padj),]) # at the .01 level

We can save all the genes with their p-values to a file:

write.csv( res, file="allgenes\_sample.csv" )

Or just save genes that are differentially expressed at a 5% false discovery rate:

resSig05 = res[ res$padj < 0.05, ]

write.csv( resSig05, file="sample\_Sig05.csv" )

Above we applied a simple negative binomial between replicates to unfiltered RNAseq data. However, it is often advantageous to filter your data before applying the binomial test. Below we try two filtering methods.

A. First we will filter our read count by dropping genes in the lowest then retesting for differential expression with our negative binomial test:

rs = rowSums ( counts ( cds))

theta = 0.4

use = (rs > quantile(rs, probs=theta))

resFilt = nbinomTest( cdsFilt, "L", "D")

table(use)

cdsFilt = cds[ use, ]  
resFilt = nbinomTest( cdsFilt, "L", "D" )

Dropping the lowest quantile decreases the number of genes we are testing for differential expression. Compare the number of genes that we tested previously with the number we are testing now:

nrow(res[!is.na(resFilt$padj),]) #unfiltered data

nrow(resFilt[!is.na(resFilt$padj),]) #filtered data

You'll notice we now get a greater number of differentially expressed genes because the FDR correction is on a lower number of p-values:

nrow(resFilt[resFilt$padj<0.05 & !is.na(resFilt$padj),])

Obviously this method is not advisable is you are interested in whether there is differential expression in this lower quantile. An alternative is to filter based on the mean base count.

B. Filtering based on mean base count.

We want to eliminate any gene with fewer than 10 reads:

res\_10<-res[res$baseMean>10, ]

We then adjust the p-value (Benjamini and Hochberg/FDR) after base count filtering:

res\_10$padj <- p.adjust(res\_10$pval, method="BH"))

How many differentially expressed genes does that leave us with?

nrow(res\_10[res\_10$padj<0.01 & !is.na(res\_10$padj),])