Lab 4 – Gene Expression Analysis I

[Software needed: web access]

In this lab we will look at one of NCBI's online repositories of gene expression data, the Sequence Read Archive (SRA), followed by a brief introduction to RNA-seq data analysis and statistical testing using the BioConductor analysis environment in R, a statistical programming language.

With the advent of high-throughput technologies capable of assessing the global transcriptional state of an organism in a single experiment detailed transcriptional profile information has become available on a massive scale, first via microarrays and more recently through RNA-seq. Online repositories of microarray experiments are now abundant and provide a valuable resource for hypothesis driven research in the post-genomic era. While more recent RNA-seq technology can provide more information about how a gene is expressed (e.g. alternative splicing), the large repository of gene expression data based on microarray experiments represents a valuable resource for trying to figure out a gene's function.

Transcriptome abundance data are noisier than DNA sequence data, and much care needs to be taken in not only the experimental design of profiling experiments but in the analysis of the resulting data. This includes the proper normalization of results across multiple replicates and conditions, as well as using the proper statistics to select significantly differentially expressed genes. We will touch on both of these issues after exploring some RNA-seq data from the SRA.

Box 1. Generation of global gene expression (transcriptome) data

There are two main platforms used to generate transcriptome data: array-based and sequencing-based. Each offers advantages and disadvantages in terms of reproducibility, cost, and data post-processing requirements.

Spotted cDNA or long-oligonucleotide microarrays are made by generating gene-specific ~80 nucleotide-long oligonucleotides or PCR-amplifying individual cDNAs from an organism, typically in 96 or 384 well microtiter plates, and then spotting these using a pin tool onto glass slides in some sort of logical manner. Because the spots can be of different sizes due to spotting artifacts, it is necessary to perform a competitive hybridization using one's "treatment" RNA and a "control or reference" RNA. These are each labelled with dyes, such as Cy3 and Cy5, that fluoresce with different colours when excited by a laser. After the labelled RNAs are allowed to hybridize to their cognate cDNAs on the glass slide, the slide is scanned using

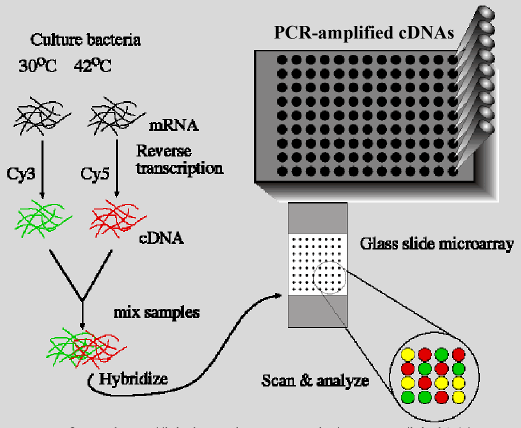
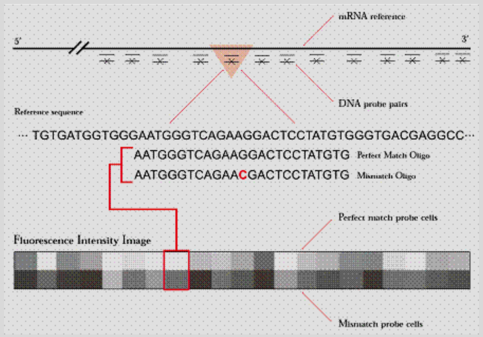


Image from http://biology.kenyon.edu/courses/biol14/image3/Microarray.gif a laser and fluorescence intensities are quantified. Based on the amount of a given RNA message in the treatment versus the control the corresponding spot will fluoresce yellow (if the amounts of that RNA species are equal in both) or green or red, if they are more abundant in the treatment or the control. Typically the ratio of the "red" signal to the "green" signal is given. Several technical replicates are needed for each hybridization, as chip-to-chip reproducibility is not that high.

The second type of microarray platform is manufactured by Affymetrix. With this type of platform, the "GeneChip", short 25 base oligos are synthesized directly onto a silicon wafer, using photolithographic methods adapted from the semiconductor industry.

The top figure at the right shows an exemplary gene-specific oligonucleotide, called a "probe". For each gene to be queried, there are 10-12 probes that are typically towards the 3' end of the gene. These constitute a "probe set". In some versions of Affymetrix microarrays, for each probe there is a corresponding mismatch probe, which is used to help assess non-specific hybridization in some algorithms.


mRNA reference
5'
3'
DNA probe pairs
Reference sequence
··· TGTGATGGTGGGAATGGGTCAGAAGGACTCCTATGTGGGTGACGAGGCC···
AATGGGTCAGAAGGACTCCTATGTG Perfect Match Oligo
AATGGGTCAGAACGACTCCTATGTG Mismatch Oligo
Perfect match probe cells
Fluorescence Intensity Image
Mismatch probe cells
Image from http://www.med.yale.edu/wmkeck/affymetrix/images/affy.h2.gif

Because it is possible to manufacture Affymetrix chips very precisely, typically a single sample is hybridized per chip, as per the lower figure to the right. The mRNA is reverse transcribed into cDNA and then biotin-labelled cRNA is generated from the cDNA using an *in vitro* transcription (IVT) step. The cRNA is hybridized to the microarray, which is subsequently washed and stained. The amount of a given mRNA species in the applied sample is determined by the fluorescence intensity of a given probe set after scanning by a laser. Raw signal intensities are adjusted through a normalization step.

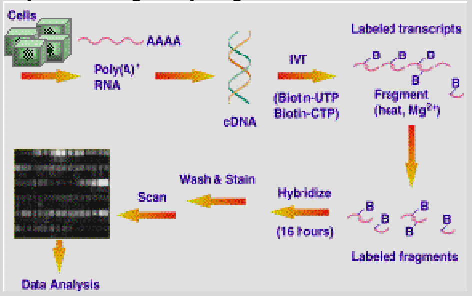
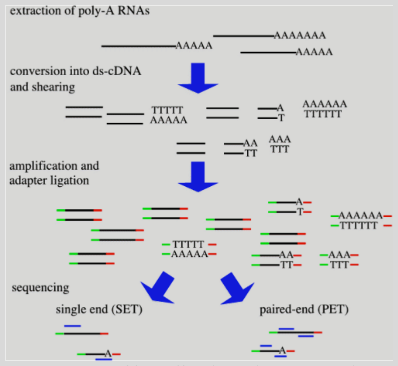


Image from www.tmri.org/gene_exp_web/oligoarray_4.gif

Sequencing-based methods for assessing global genes expression levels exist, as exemplified by RNA-seq (image to the right), MPSS (massively parallel signature sequencing) and DGE (digital gene expression). These methods are useful even in the case of organisms with no reference genome sequence. With these methods, we're counting the number of tags/fragments for a given gene to assess abundance.


extraction of poly-A RNAs
AAAAA
AAAAAAA
AAAAA
conversion into ds-cDNA and shearing
TTTTT
AAAAA
A
T
AAAAAA
TTTTTT
AA
TT
AAA
TTT
amplification and adapter ligation
sequencing
single end (SET)
paired-end (PET)
Image courtesy of http://cmb.molgen. mpg.de

Clearly, the type of platform used to assess RNA abundance, how RNA samples were prepared, from which tissues they were isolated etc. is vital to the proper interpretation of a transcriptomics experiment. That is why the "MicroArray and Gene Expression" markup language (MAGE-ML) was adopted. At online expression database like GEO or the SRA, you should be able to access such meta-information, in addition to the gene expression levels for a given experiment.

Gene Expression Omnibus (GEO) and the Sequence Read Archive (SRA)

The Gene Expression Omnibus gathers the results of microarray experiments, while the Sequence Read Archive is for raw sequences from RNA-seq and other kinds of sequencing experiments. Data come from a variety of sources, including submissions from individual researchers, as well as collections of data from larger gene expression profiling efforts such as ENCODE or the AtGenExpress effort (multinational efforts to document transcriptomes and other 'omes in human and transcriptomes in *Arabidopsis thaliana*). We will use the Short Read Archive (SRA) to examine RNA-seq experiments pertaining to floral development in the important reference plant *Arabidopsis thaliana*.

Connect to the SRA at http://www.ncbi.nlm.nih.gov/sra and search for SRA023501 in the search field. The SRA identifier acts as a "container" for a number of different accessions related to a particular set of sequences. The SRP identifiers are for records of *P*roject metadata; SRX records contain meta-information about the platform, library and processing of the particular sequencing e*X*periment; SRR records contain the actual *R*ead, i.e. sequence, information (these files can be huge, with millions of short reads in them); SRS records detail the *S*ample metadata; and SRZ records are for compressed BAM files (more on this later).

Scroll down to record "SRX025400 – Illumina sequencing of Arabidopsis thaliana AP3 domain translatome at flower stages 4, replicate 1" and click on it to read the Study summary description. Be sure to click on 'show abstract…' to see the complete abstract. Move back to the list of records associated with SRA023501 if necessary to answer the following questions.

a. In what tissues were samples collected?

b. How many samples total where taken in this experiment?

c. How many replicates were done for each treatment?

d. What next-generation sequencing platform was used in this experiment?

e. Look at the PubMed record (or Study summary) associated with these data. What kinds of comparisons can be done between samples?

Let's identify those genes whose expression levels are differentially expressed in the translatome of the B domain of the developing flower, where *AP3* is expressed, relative to the whole flower at Stage 4. To do this, we will use samples from the AP3 translatome data sets, as well as files for the appropriate control treatments (whole flower translatome at Stage 4).

f. What data sets would you download and what are the names of the SRR files for each set?

Note: As you probably determined above, each sample corresponds to one RNA-seq experiment. There are two biological replicates per "condition". The results of an RNA-seq experiment create huge files. We'll be using data files trimmed to the first 100 kb of each of the 5 *Arabidopsis* chromosomes so that you can carry out the analyses on your laptop if necessary, so **don't actually download the 4 SRR files from the SRA using the SRA Toolkit**, rather use trimmed versions of these we'll provide (see later).

BioConductor

BioConductor is the leading environment for the analysis of gene expression data. It runs within the R statistical programming environment and possesses methods for background reduction, normalization, differential analysis, coexpression analysis and visualization of expression data. Although R and BioConductor are complex and can be intimidating, this exercise will serve as a superficial introduction to how gene expression data analysis can be performed with this great suite of tools. Another more detailed tutorial is referenced at the end of this lab.

You can use our cloud instance of R and BioConductor as per the **where to get it** section. R is similar to Matlab or SAS. It is dynamically typed so if you have experience with either MATLAB, SAS, or programming in general, then you should feel pretty comfortable fairly quickly in R. Our instance uses RStudio running on Amazon Web Services, and this permits you to use a nicer web interface to interact with R. Create a login/login at http://bar.utoronto.ca/BioC/. Note your work won't be saved if you close your browser but you can save your workspace at any time by typing **`save.image("Lab4.RData")`** at the prompt. You can download this file and upload it to continue at a later point. (If you like, you can also download and install R and Bioconductor onto your own computer, as per the Appendix. We can't, however, provide any technical help if you choose to do this.) If you don't have any programming experience then here are two things about R:

1) Variable assignment
Variable assignment is analogous to the algebraic assignment of variables, e.g. x = 3. However, the assignment of complex data structures to variables is possible and variable names can be of any length.

Try entering the following commands into the R Console in the left pane of RStudio (note the '**>**' is just the prompt and not typed, and also don't forget to hit *Enter* after each **bolded** command):

```
> mydata = c(3,2,8,1)
> mydata
[1] 3 2 8 1
```

Here we created a variable '**`mydata`**' and assigned a vector of 4 numbers to it. A vector in mathematical terms is simply a series of numbers. As you may have gathered, the typing of the variable name and hitting enter returns the variables contents. But what about the **`c( )`**?

2) Methods
Methods are predefined programs that take data and parameters (placed within parentheses), perform some calculation or operation on that data (within the context of the provided parameters) and return a result. This result is usually assigned to a variable but doesn't have to be. Here **`c( )`** is the combine method. It combines comma delimited numbers into a vector of numbers and returns the vector, which we assigned to the variable '**`mydata`**'. *Type* **`'?c'`** *within R to get a detailed description of the* **`c( )`** *method.*

Now we'll tell R where to look for the data using the "set working directory" command…(hit *Enter* after copy-pasting or typing the bolded command, and use the *[username]* you created when logging in to AWS; don't include the "[ ]" when you enter your own username!):

```
> setwd("/home/[username]") #the data you need are in the Rrnaseq directory!
# If you're using BioConductor on your desktop, you'll need to download the
# Rrnaseq.zip file from BOX, unzip it and use the setwd command to specify the
# directory it's in, e.g. setwd("[C:/path/to/directory/for/[username]")
# Download URL: https://box.xjtlu.edu.cn/f/3cf4157365fa4723846a/?dl=1
```

We've preinstalled a specific version of Bioconductor on our RStudio instance (see page 14 for details). If you are installing Bioconductor on your own computer, you'd specify "https://bioconductor.org/ biocLite.R".

We'll load some information about the samples we'll be using from a file that defines this. Basically, with the targets.txt file, we're telling R how to deal with the 4 samples in the Rrnaseq/data directory that's provide for you (either in RStudio or in the Rrnaseq.zip file):

```
> targets <- read.delim("./Rrnaseq/data/targets.txt", comment.char = "#")
> targets
                        FileName SampleName Factor Factor_long
1 ./Rrnaseq/data/SRR064154.fastq   AP3_fl4a    AP3     AP3_fl4
2 ./Rrnaseq/data/SRR064155.fastq   AP3_fl4b    AP3     AP3_fl4
3 ./Rrnaseq/data/SRR064166.fastq    Tl_fl4a    TRL      Tl_fl4
4 ./Rrnaseq/data/SRR064167.fastq    Tl_fl4b    TRL      Tl_fl4
```

The first step in generating expression level information from these millions of short reads is to align them to a reference genome. Several Unix-based algorithms exist to do this, or we could use an online resource such as Galaxy, which "wraps" these into something we can use as part of a workflow see https://usegalaxy.org/u/jeremy/p/galaxy-rna-seq-analysis-exercise). We'll use systemPipeR (Backman & Girke, 2016) in our exercise, which is a way to automate specifying parameters for several useful RNA-seq tools, instead of having to enter the same command for multiple files. We'll be using HISAT2 for the read-mapping aspect (Langmead et al., 2015), without pre-trimming the reads using Trimmomatic or a similar program (Bolger et al., 2014).

`> library(systemPipeR)` (a reminder that the '>' is just the prompt, not typed; and to hit *Enter* after copy-pasting or typing this command into the R Console of RStudio!)

Type/paste the following commands to tell systemPipeR where to put the results:

```
> dir.create("results")  # create a results directory
> results <- "./results" # defines location where to write results
```

We'll automate the execution of readmapping using HISAT2 by providing a template file (hisat2.param) into which the samples and filenames in the targets.txt file are interpolated:

```
> args <- systemArgs(sysma="./Rrnaseq/param/hisat2.param",
  mytargets="./Rrnaseq/data/targets.txt")
> sysargs(args)[1] # check parameters for running 1st sample using HISAT2

                              AP3_fl4a
"hisat2 -p 1 -k 1 --min-intronlen 30 --max-intronlen 3000 -S
/home/provartn/results/SRR064154.fastq.hisat.sam
/home/provartn/Rrnaseq/data/tair10chr.fasta -U /Rrnaseq/data/SRR064154.fastq"
```

Check out the HISAT2 parameters page – what are we telling it to do? Now let's align our reads to the TAIR10 chromosome sequences (we are effectively issuing 4 hisat2 commands, each specific for one of the 4 data sets we're using, with this single command):

```
> runCommandline(args=args) #ignore warnings
```

Wait a few minutes for the command prompt to reappear…now let's see how many of the reads mapped to the TAIR10 genome (the mapping results are the .bam files in the /results directory). Remember that R is case-sensitive if you're typing this command (that's a capital "S" on Stats):

```
> read_statsDF <- alignStats(args) #generate alignment statistics
> read_statsDF # display alignment statistics
         FileName  Nreads  Nalign Perc_Aligned     Nalign_1° Perc_Aligned_1°
AP3_fl4a AP3_fl4a 1633256 1591064     97.41669     1591064      97.41669
AP3_fl4b AP3_fl4b 1669046 1627935     97.53686     1627935      97.53686
Tl_fl4a   Tl_fl4a  210407  193113     91.78069      193113      91.78069
Tl_fl4b   Tl_fl4b  289021  282267     97.66315      282267      97.66315
```

g. How many reads mapped to the TAIR10 genome in each of the 4 samples?

h. How many reads didn't map?

Visualizing our RNA-seq data

We will now examine the quality of these RNA-seq data by looking at the distributions for each data set using boxplots, followed by RPKM normalization and heatmapping.

First we will need to count the number of mapped reads per genomic feature in which we are interested. Genome features are typically stored in GFF3 files. GFF stands for general feature format and is a text-based, tab-delimited format for describing features in a genome (indexed by nucleotide position along a chromosome), such as genes, introns, exons, UTRs, etc. See http://gmod.org/wiki/GFF#GFF3_Annotation_Section for a complete specification. We'll use the easy-to-use **makeTxDBFromGFF** function of the **GenomicFeatures** library to store the genome feature information in an object to use for counting the reads mapped to the gene features.

```
> library(GenomicFeatures)
> txdb <- makeTxDbFromGFF(file="./Rrnaseq/data/TAIR10_GFF3_trunc.gff",format=
  "gff3",dataSource="TAIR",organism="Arabidopsis thaliana") #One line! Ignore
  warnings

> eByg <- exonsBy(txdb, by="gene") # retrieve exon features grouped by genes
> bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
> counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg, x,
  mode="Union", ignore.strand=TRUE, inter.feature=TRUE, singleEnd=TRUE))# one
  line! Note: for strand-specific RNA-Seq set 'ignore.strand=FALSE' and for
  paired-end data set 'singleEnd=FALSE'…our data set is not paired-end, so we set
  the flag as TRUE

> countDFeByg <- sapply(seq(along=counteByg), function(x)
  assays(counteByg[[x]])$counts) #one line
> rownames(countDFeByg) <- names(rowRanges(counteByg[[1]]));
  colnames(countDFeByg) <-names(bfl) #one line

> countDFeByg[1:4,] # display the first four rows of gene counts
          AP3_fl4a AP3_fl4b Tl_fl4a Tl_fl4b
AT1G01010       51       25      56      74
AT1G01020      144       74      78      62
AT1G01030        5        1      13      13
AT1G01040      474      335     264     335

> write.table(countDFeByg, "results/countDFeByg.xls", col.names=NA,
  quote=FALSE, sep="\t") # store as a table if desired
```

Boxplots are an easy means of visualizing the distribution of our data. The top and bottom of the box are the 75th and 25th percentiles respectively, the middle of the box is the Median and the top and bottom lines can be used to denote the minimum and maximum of the data, or the 9th and 91st percentiles, or some other measure of variability outside the 75th and 25th percentiles (upper and lower quartiles; see Figure 1 and Box 2). Outliers beyond these lines can be plotted with circles.

MA plots are a means of making a quick assessment of the quality of a transcriptomics experiment. They reveal intensity dependent effects and are usually judged by the overall cloud shape. Essentially, in assessing MA plots we're looking for a straight horizontal "loess" curve and a uniform scatterplot of points horizontally, centered around 0. MA plots plot M, the log fold change on the y-axis relative to a pseudo-reference sample, against the average log intensity for the gene (A), on the x-axis. Most genes in an experiment will exhibit little change (see Figure 1 and Box 2).

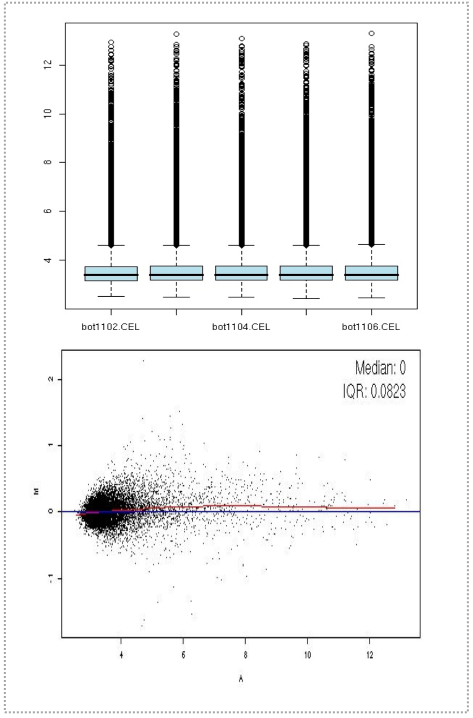


Figure 1 : Boxplots of some microarray data (top) and the corresponding MA plot for after normalization (bottom).

Let's look at the distribution of the number of reads per gene data with:

```
> boxplot(countDFeByg[,1:4]) # plot columns 1 through 4
```

i. Would you say the data in each file are identically distributed? Use **`boxplot(countDFeByg [,1:4],ylim=c(0,2000))`** *if you can't distinguish the upper and lower quartile box boundaries.*

j. What do the first 2 boxes represent? And the last 2?

In the case of microarray data, MA plots are a useful diagnostic, but for RNA-seq data they can't practically be used due the presence of many 0 values which throw a wrench into the works when computing the "M" value. If you are working with microarray data, there are a couple of MA plot packages such as **`MAplot()`** in the **`affy`** library or **`plotMA()`** in the **`limma`** library. For RNA-seq data, we'll do a diagnostic clustering after a simple normalization procedure.

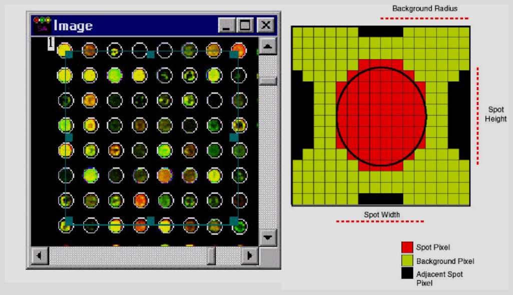
Background reduction, normalization and summarization

In the case of microarray data, before we can proceed with a differential analysis we need to perform background reduction on our samples and normalize our distributions across all chips. Although a variety of methods exist for accomplishing this, the rma procedure is a popular method for Affymetrix data and you could use the **`affy`** package and a command like **`affy.sum = rma(affydat)`** `# affydat contains data` to achieve all three steps in one command.

In the case of RNA-seq data, several methods have been developed for normalization. The dynamic range of the method is at least 10^6 and the response is almost perfectly linear. There is no background (a count of zero reads really means no reads), obviating the need to do background reduction and fancier quantile normalization. The simplest method conceptually is RPKM (or FPKM for paired-end read data).

Box 2. Background reduction and normalization

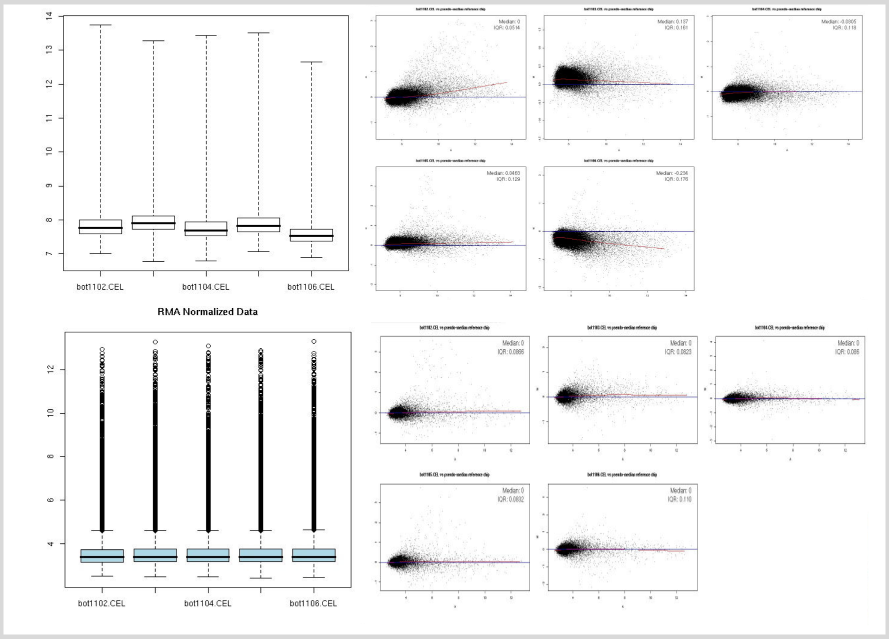
For microarrays, after scanning of the hybridized slide, fluorescence intensities are captured in an image file. Image processing algorithms are used to identify spots, as shown to the right. There is always some background level of hybridization, and this value needs to be subtracted from the computed signal intensity. This step is clearly more important for signals towards the lower end of the intensity range. Various ways of assessing the background level of hybridization have been developed: one popularly used procedure is the Robust Multiarray Analysis (RMA) method for Affymetrix data.



Images f. http://rana.lbl.gov/manuals/ScanAlyzeDoc.pdf

Once background values have been subtracted, the signal intensities need to be normalized. This is also true of raw read counts for RNA-seq data. The reason for this is that a lot of variability is introduced into the "system" over the course of signal generation. For instance, the amount of RNA that is input to the microarray or RNA-seq experiment can vary, as can the efficiency of cDNA synthesis, fluorescence yield (microarrays)/library generation (RNA-seq), amount of sequence generated (RNA-seq) and so on.

Basically the goal of normalization is to adjust the signal intensities by some factor so that the average level of expression is constant across all experiments – the assumption being that the vast majority of the genes won't change in expression levels substantially between control and treated samples, and that if there are changes, there will be roughly equal numbers of genes up or down.



For microarrays, several different normalization methods exist, within such packages as RMA, MAS5.0/GCOS, or dCHIP. Compare the boxplots before and after RMA normalization of 5 microarray experiments (top and bottom left, respectively, in the panel above). The array-to-array variation is dramatically reduced after normalization, allowing accurate between array comparisons. The same effect is visible in the "MA" plots (log2 of [signal/pseudo-median value] versus log2 signal) – one sees the majority of genes showing little change relative to the pseudo-median control, with most being centered around the horizontal blue line (the "no change" line, bottom right) after normalization.

For RNA-seq data, commonly used methods of normalization (including a summarization step) include RPKM (*R*eads *P*er *K*ilobase per *M*illion reads mapped), or its paired-end equivalent, FPKM (*F*ragments *P*er *K*ilobase per *M*illion reads mapped). In the case of RNA-seq, the more sequence we generate, the more reads we would expect for each gene. Thus to standardize everything to a *per million reads mapped* basis makes sense when comparing samples with different total numbers of reads. The rationale behind standardizing values to a per kilobase basis is that longer genes would generate more reads. This standardization is useful when comparing expression levels of different genes, which we don't often do. Other methods, such as the *T*rimmed *M*ean of *M* values (TMM) by Robinson and Oshlack (2010), exist that try to correct for RNA compositional bias where rRNAs might not be as efficiently removed in some samples, and thus might "swamp out" more weakly expressed genes.

For RNA-seq data, note that RPKM/FPKM/TMM etc.-normalized values aren't typically used for testing differential expression, as it is useful to know that more absolute counts were generated when doing the testing. Consider the following example: in two libraries, each with one million reads, gene X may have 10 reads for treatment A and 5 reads for treatment B, while these values would be 100 times greater after sequencing 100 million reads from each library. In the second case, we can be much more confident that there is a true difference between the two treatments than in the first one. However, the RPKM-normalized values would be the same for both scenarios.

Let's RPKM-normalize out RNA-seq data with a single command:

```
> rpkmDFeByg <- apply(countDFeByg, 2, function(x) returnRPKM(counts=x,
  ranges=eByg)) # one line!
```

Here we're using a function that effectively takes the read counts for each gene divided by the size of each gene in kilobases to give Reads per Kilobase, and then scaling this value relative to the total number of reads in the experiment on a per million read basis. We can also write this to a file if desired:

```
> write.table(rpkmDFeByg, "results/rpkmDFeByg.xls", col.names=NA, quote=FALSE,
  sep="\t") # store as a table
```

What does this do to the distribution of the expression levels? Let's use the boxplot function again to find out.

```
> boxplot(rpkmDFeByg)
> boxplot(rpkmDFeByg,ylim=c(0,2000)) # Examine the upper/lower quartiles
```

j. Would you say the data after RPKM normalization have equal distributions?

Hmmm…not really! Keep in mind that we're looking at a very small slice of the original sequence data, those reads that map to the first 100kb of each of the 5 *Arabidopsis* chromosomes. Some of the chromosomes are 20 Mb or larger, so we might not expect the distributions to be the same in this somewhat artificial data set.

We can assess quality by plotting the "fastq" scores (which are measures of read quality) on per position basis for all of the reads for each of the samples (note: we could have executed this command at the bottom of page 5, and checked the quality there…)

```
> fqlist <- seeFastq(fastq=infile1(args), batchsize=1000, klength=8)
> pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
> seeFastqPlot(fqlist)
> dev.off()
```

Open the fastqReport.pdf (this file can be seen under the File tab in the bottom right pane of RStudio, in the "results" folder). The first page of the quality control report shows the distribution of fastq scores according to the base position in the short read. Ideally, we'd want all positions to have the same (high) quality scores.

k. Which sample has particularly bad quality scores towards the end of the short reads?

Keep in mind we can also look at the number of reads that map as a measure of quality, as we saw at the top of page 6. Perhaps the lower percentage of reads mapping for the third sample is due to the not-so-great fastq scores towards the ends of the reads for this sample?

Let's do a reproducibility check between biological replicate samples using the following commands. We'll talk more about hierarchical clustering next lab, but suffice it to say, the idea here is to group samples that have similar expression patterns across the ~145 or so genes in our particular example data set.

```
> library(ape)
> d <- cor(rpkmDFeByg, method="spearman")
> hc <- hclust(dist(1-d))
> plot.phylo(as.phylo(hc), type="p", edge.col=4, edge.width=3,
   show.node.label=TRUE, no.margin=TRUE) # all on one line!
```

l. Do the samples group as expected, at least according to the expression levels of the 100 plus genes we're looking at in our example data set?

Identifying differentially expressed genes

A common objective in gene expression profiling experiments is to identify those genes that exhibit differential expression (an increased or decreased steady state abundance) under a particular treatment relative to the expression levels of those genes in the control samples. In the simplest case, this has often involved the identification of genes exhibiting a relative fold change greater than some arbitrary cutoff, for example, say 3-fold or greater expression difference in the treatment gene's expression level relative to its level in the control. We can't really generate any sort of *p*-value for such a calculation. A more statistically sound approach is to use one of the several packages available, such as **`DESeq2`** or **`edgeR`**, to carry out such analyses. We'll use **`DESeq2`** here, which uses a negative binomial distribution along with the appropriate statistical tests. For microarray data, we might want to use the **`genefilter`** library, which encompasses several statistical tests appropriate for these kinds of data.

Here's a **`DESeq2`** analysis to identify genes with > 2-fold change and a false discovery rate (FDR) of 10%, or, to be more stringent, of 1% FDR:

```
> library(DESeq2)
> cmp <- readComp(file="./Rrnaseq/data/targets.txt", format="matrix", delim="-")
> # With the above command we're defining what sort of comparisons to make…
> cmp[[1]] # check we're comparing AP3 samples to the whole translatome samples
     [,1]  [,2]
[1,] "AP3" "TRL"

> degseqDF <- run_DESeq2(countDF=countDFeByg, targets=targets, cmp=cmp[[1]],
independent=FALSE)
> # We defined the targets (samples) on page 5…we're telling DESeq2 (running
  using systemPipeR's run_DESeq2 wrapper) to do the comparison(s) as above
> countDFeByg[1:4,] # let's check first 4 rows of countDFebyg, as on page 6.

          AP3_fl4a AP3_fl4b Tl_fl4a Tl_fl4b
AT1G01010       51       25      56      74
AT1G01020      144       74      78      62
AT1G01030        5        1      13      13
AT1G01040      474      335     264     335

> degseqDF[1:4,] # let's check out the DESeq2 results for the same genes

          AP3-TRL_baseMean AP3-TRL_logFC AP3-TRL_lfcSE AP3-TRL_stat AP3-TRL_pvalue AP3-TRL_FDR
AT1G01010        50.437482    -0.8615482     0.4236712    -2.033530     0.04199898  0.07733145
AT1G01020        87.525848     0.5140377     0.3582196     1.434979     0.15129306  0.22499993
AT1G01030         7.972584    -2.2502458     1.0602298    -2.122413     0.03380305  0.06428121
AT1G01040       345.599083     0.3711985     0.1659133     2.237304     0.02526648  0.05328930
```

You can see that DESeq2 has computed the fold change (shown as $\log_2$ fold change in the `AP3-TRL_logFC` column) for each gene, and has provided a *p*-value and FDR (false discovery rate) that we can use to filter genes for further analysis – not all genes are significantly differentially expressed.

```
> DEG_list2 <- filterDEGs(degDF=degseqDF, filter=c(Fold=2, FDR=10)) # limit
  output to genes only showing 2-fold change up or down, and having a 10% FDR
  or better…you'll get a nice graph with the number of genes "up" and "down"
```

m. How many of the genes in our original data set are differentially expressed as defined above?

```
> nrow(countDFeByg) # number of genes in original data set, incl. genes with
  no counts…the nrow function returns the number of rows
[1] 145
> lengths(DEG_list2$UporDown[]) # number with fold-change > 2 and at a 10%
  false discovery rate; note "s" on lengths function
AP3-TRL
     49
```

Now, let's take the "top hits" and plot them in a heatmap. Our definition for "top hits" will be a more stringent FDR cutoff of 1% so that we pull out just the most differentially expressed genes (at least of those present in our example data set) between the translatomes of the AP3 regions of stage 4 flower buds versus the translatomes of whole flower buds of the same stage:

```
> DEG_list3 <- filterDEGs(degDF=degseqDF, filter=c(Fold=2, FDR=1)) # first we
have to create a new list…
```

n. If we set our FDR to 1% how many genes pass the test of differential expression in the AP3 samples versus those in the control treatment? (Hint, you can use the same **`lengths`** *function we just used, but on the* **`DEG_list3$UporDown[]`** *list).*

o. What row number of the first gene that is differentially expressed in our original **`countDFeByg`** *data matrix? (Hint: use* **`DEG_list3$UporDown`** *to see the IDs of the differentially expressed genes and then count the rows down to that gene in* **`countDFeByg`***).*

Box 3. Significance and the problem of multiple testing

Your background in the sciences will hopefully have introduced you to the concept of *p*-value for statistical tests. Typically, for a given experiment, we hear people saying things like "this is significant at p=0.05". What this means is that 1 time out of 20 such a result **could** be observed by chance alone – this would be a false positive.

With transcriptomics data, we are typically performing 10,000 t-tests or more. If we set our *p*-value cutoff to be 0.05, and 1000 genes are considered "significantly different" at *p*=0.05, then 50 of these will likely be false positives. This is the problem of multiple testing. We need to adjust the *p*-value in some way to be able to reduce the number of false positives. One very strict correction is called the **Bonferroni correction**, which sets the *p*-value for significance at the *p*-value considered acceptable for a single test divided by the number of tests. In the case of 10,000 tests, we'd divide *p*=0.05 by 10,000 – which is 5e-06. Clearly only those genes showing huge changes in gene expression would be identified using this cutoff, and we'd likely discard biologically meaningful differences.

Another way to address this issue is to use the **false discovery rate** (FDR), as suggested in 1995 by Benjamini & Hochberg and modified later by others. Essentially, we can permute the sample labels and then perform significance tests on these permuted data sets to assess how many "significant" or "falsely discovered" genes we are likely to identify with the scrambled data. We can specify the percentage of false discoveries we'd be happy with in our non-scrambled data set. Depending on whether the experiment if a "fishing expedition" or will be used to guide wet-lab research, we can adjust our false discovery rate appropriately.

Let's look at the raw expression values for the genes with significant differential expression.

```
> results.top <- countDFeByg[DEG_list3$UporDown[[1]], ]
> # Generate a subset of the raw read counts of our significant gene rows
> # from the DEG_list3 data object, using the list of the $UporDown IDs
> # to select the row names in the countDFeByg matrix, and then
> # assign them to a matrix called results.top
```

```
> results.top
```

```
           AP3_fl4a AP3_fl4b Tl_fl4a Tl_fl4b
AT1G01050       283      258     753     927
AT1G01060       219      132     342     470
AT1G01070         8        4      45      53
AT2G01008        28       36       1       4
AT2G01010    426562   491019   56759   80216
…
```

Now let's finally display the expression levels of our significantly differentially expressed genes in a heatmap.

```
> heatmap(results.top, Colv=NA, Rowv=NA, col=rev(heat.colors(256)))
```

Setting the parameters Colv and Rowv to NA (i.e. Not Available) turns off clustering by row and column. The last argument just provides us with some nice colouring that expresses high transcript abundance as red and lower levels of abundance as white and everything in-between.

p. Which genes are increased in abundance in the AP3 region-derived transcriptome in our heatmap?

We can also examine the RPKM-normalized expression levels of our top significant genes:

```
> results_rpkm.top <- rpkmDFeByg[DEG_list3$UporDown[[1]], ]
> heatmap(results_rpkm.top, Colv=NA, Rowv=NA, col=rev(heat.colors(256)))
```

Go to http://bar.utoronto.ca/ntools/cgi-bin/ntools_agi_converter.cgi. Enter the AGI IDs in the box, select From:AGI and To:Gene Alias and Annotation (click to select these), and click 'Submit'. Hint: use > `results.top[,0]` to retrieve just the AGI IDs (AGI stands for Arabidopsis Genome Initiative, and AGI IDs have the format of At[1-5CM]*ddddd*, where *d* can be any digit. These start at At1g01010 and typically increase by 10 along the chromosome).

q. What are these genes annotated as?

r. Can you place any biological meaning to the genes which have increased abundance in the AP3 translatome data set?

Save your workspace to the Rrnaseq directory by typing `> save.image("Lab4.RData")` and download it using the Export function under "More" in the RStudio file panel for use in the next module.

End of lab (Objectives at the end of Lab 5)!

Where to get it: Get a login for this course's RStudio server running on Amazon Web Services at http://bar.utoronto.ca/BioC/.

This lab was developed with R version 3.5.2, and BioConductor version 3.8. Thomas Girke's excellent systemPipeR documentation (the sections on RNA-seq analysis) on Bioconductor.org at https://www.bioconductor.org/packages/release/bioc/vignettes/systemPipeR/inst/doc/systemPipeR.html, was used with permission as a basis for this lab. It contains many more tips and tricks that are useful for RNA-seq analysis.

Further Reading

Bolger AM, Lohse M, Usadel B (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics 30: 2114–2120. doi: 10.1093/bioinformatics/btu170.

Backman TWH and Girke T (2016). systemPipeR: NGS workflow and report generation environment. BMC Bioinformatics 17:388. doi: 10.1186/s12859-016-1241-0.

Jiao Y and Meyerowitz E (2010). Cell-type specific analysis of translating RNAs in developing flowers reveals new levels of control. Mol Syst Biol. 6:419. doi: 10.1038/msb.2010.76.

Kim D, Langmead B & Salzberg SL (2015). HISAT: a fast spliced aligner with low memory requirements. Nature Methods 12, 357–360. doi:10.1038/nmeth.3317

Robinson M and Oshlack A (2010). A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology 11:R25 doi:10.1186/gb-2010-11-3-r25

Appendix – Installing R and Bioconductor on your own computer

You can get R at http://r-project.org > Getting Started box > download R link then choose your closest CRAN mirror (or any one, for that matter). On the mirror page, click on the Download R for … whatever operating system you are using, then choose to install the "base" then follow the download link on the page that you're taken to.

To install BioConductor (http://bioconductor.org) start R then type the following. Don't type the > sign, that just denotes the command prompt character in R. And don't forget to press ENTER after typing a command! Tip: you can copy and paste any of the commands in this lab (without the leading >) into the R command window.

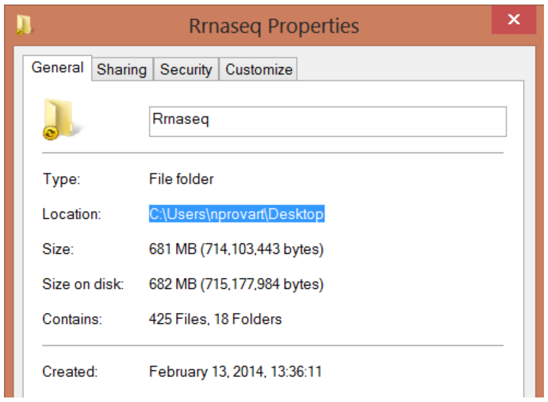
```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

R will ask you if you want to use a personal library – say yes if prompted and wait for the packages to be installed.

If you want to use R and Bioconductor on your own computer for *this* lab then type the following to install the necessary packages (you'll also need to install HISAT2 separately):

```
BiocManager::install(c("systemPipeR", "ape", "DESeq2", "GenomicFeatures"))
```

Again, if you want to use your own computer for this lab, finally download the Rrnaseq.zip file from ICE and unzip it. Find out the absolute path to the unzipped Rrnaseq directory or folder. In Windows, right click on the directory in File Explorer and choose properties. In Mac, highlight the Rrnaseq directory in Finder then hit the ⌘ and "i" keys together to "inspect" that folder. The path is after Where or Location. Use this information to set the working directory with the `setwd` command. Replace the backslashes (\) with forward ones (/) if necessary.



Rrnaseq

Type: File folder
Location: C:\Users\nprovart\Desktop
Size: 681 MB (714,103,443 bytes)
Size on disk: 682 MB (715,177,984 bytes)
Contains: 425 Files, 18 Folders
Created: February 13, 2014, 13:36:11

```
> setwd("[C:/path/to/working_directory/for/username]")
# replace […] with your path!
# Windows:     […] example: "C:/Users/nprovart/Desktop/"
# Mac:         […] example: "/Users/nprovart/Desktop/"
```

Note that this lab was developed with R version 3.3.2, and BioConductor version 3.4, so the commands shown in it might not work if you're using different versions of these suites. In addition, the data sets for this lab have been trimmed so that they can run on most laptops/desktops. If you try to process "real" RNA-seq data sets on your own computer, you might run out of RAM.