Introduction to Bioconductor (BaRC Hot Topics)

George Bell

October 6, 2011

This document accompanies the slides from BaRC's Introduction to Bioconductor and shows how to generate all analyses and figures. See the accompanying slides and data files at `http://iona.wi.mit.edu/bio/education/R2011/` or material from other Hot Topics talks at `http://iona.wi.mit.edu/bio/hot_topics/`. All of these commands should work similarly if run on Windows/Mac (using the typical R installation or RStudio) or Linux operating systems (such as tak).

1 Getting the datasets used in the vignette

Start by downloading all files from the Hot Topics web page. Create a class directory (`BaRC_class`) and inside that, directories called `Affymetrix`, `Agilent`, and `RNASeq`. Put the files for each dataset in the corresponding directory. To get started, make `BaRC_class` your "working directory". To find where R is now (for my analysis; your directory will be different), try

```
> getwd()
```

```
[1] "/nfs/BaRC/Hot_Topics_2011-2012/R_Bioconductor/class_3/vignette"
```

To change the working directory on your computer, go to File => Change dir... menu. On tak (or you can do it this way on your computer, too), use the `setwd()` command, like `setwd("/lab/solexa_public/Graceland_Lab/Elvis")`. We're going to enter each directory to get the data but then print output files up one level in the directory hierarchy in `BaRC_class` (hence the `setwd("../")` commands).
You can begin in any of these three sections:

- 2 Affymetrix arrays
- 3 Agilent arrays
- 4 RNA-Seq

2 Affymetrix arrays

The initial processing of the two types of arrays start out quite differently, but the statistics are quite similar. Feel free to skip sections that appear redundant.

2.1 Getting started with Affymetrix arrays

Go to the Affymetrix directory you just created and see what input data files are there.

```
> setwd("Affymetrix")
> dir()
```

```
 [1] "Affy_targets.txt"        "GSM230387.cel"
 [3] "GSM230388.cel"           "GSM230389.cel"
 [5] "GSM230390.cel"           "GSM230397.cel"
 [7] "GSM230398.cel"           "GSM230399.cel"
 [9] "GSM230400.cel"           "GSM230407.cel"
[11] "GSM230408.cel"           "GSM230409.cel"
[13] "GSM230410.cel"           "GSM230417.cel"
[15] "GSM230418.cel"           "GSM230419.cel"
[17] "GSM230420.cel"           "hgu133plus2.symbols.txt"
```

The directory should include 16 CEL files, each of which contain probe-level measurements for one biological sample. These arrays are a subset of a human study (`http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE9103`) that examines "Skeletal muscle transcript profiles in trained or sedentary young and old Subjects". You have four CEL files each from 4 groups: young sedentary, young trained, old sedentary, and old trained. Running quality control on the set of arrays can be helpful, and this will be covered in a future Hot Topics talk.
Let's load the Bioconductor packages `affy` and `limma` (which include many of the commands we'll be using).

```
> library(affy)
> library(limma)
```

If you get an error that `affy` or `limma` can't be found, it's probably not installed. In that case you can install it by adding the Bioc repositories to the usual CRAN repository (with the command `setRepositories()`) and then install the package with a command like `install.packages("limma")`. After installation, you still need to run the command `library(affy)` or `library(limma)` to access the library.

2.2 Preprocessing Affymetrix arrays

Our first task with the set of Affymetrix arrays (CEL files) is to process, normalize, and summarize them into probeset-level RNA levels with the `affy` package. We start by reading the current directory of CEL files and creating an "AffyBatch" object that includes probe-level data for a set of arrays. They all need to be of the same array design (such as our sample data of U133 Plus 2.0 arrays). At this point we'll also need an R package describing the array design we're using. Usually R will install the package automatically, but if it doesn't you can always use a command like `install.packages("hgu133plus2cdf")`. If we want to use an array design (CDF) other than the standard Affy design, that would need to be installed by hand. We can also get some general information about our set of arrays.

```
> CELs = ReadAffy()
> CELs

AffyBatch object
size of arrays=1164x1164 features (22 kb)
cdf=HG-U133_Plus_2 (54675 affyids)
number of samples=16
number of genes=54675
annotation=hgu133plus2
notes=

> setwd("../")
```

Now we can preprocess this set of arrays using algorithms such as MAS5 (the original Affymetrix algorithm), RMA, or GCRMA. Be aware that MAS5 outputs actual probeset values, whereas RMA and GCRMA output log2-transformed values. For this vignette we'll use RMA to create an "ExpressionSet". This step takes a few minutes.

```
> eset = rma(CELs)

Background correcting
Normalizing
Calculating Expression
```

If we wanted to use another algorithm, we could have instead used a command like `eset = gcrma(CELs)` or `eset = mas5(CELs)`.

At this point we may want to print a matrix of all log2 expression values. The `eset` we just created includes more than just expression levels, and to access them we need to use the `exprs` command. Once we have that matrix (rounded to 4 decimal places) we can easily print it as a tab-delimited text file. We add the rownames as an extra column so the column headers match up correctly when viewed in Excel.

```
> eset.values = round(exprs(eset),4)
> write.table(cbind(rownames(eset.values), eset.values), file="Affy_log2_rma_values.txt",
+         sep="\t", row.names=F, quote=F)
```

From the AffyBatch object we can also get Absent/Present calls, the Affymetrix method (for chips with exact match and mismatch probes) for calling gene status, also including M for Marginal. As with expression values, the output is more complex than a simple matrix so we need the `exprs` command to get a matrix of As, Ms, and Ps.

```
> mas5calls = mas5calls(CELs)

Getting probe level data...
Computing p-values
Making P/M/A Calls

> mas5calls.calls = exprs(mas5calls)
> write.table(cbind(rownames(mas5calls.calls), mas5calls.calls), file="Affy_AP_calls.txt",
+         sep="\t", row.names=F, quote=F)
```

We can use this metric in a variety of ways, such as filtering out probesets that are absent across all samples. We start by concatenating all AP calls for a probeset and then can compare the number of rows (probesets) before and after filtering.

```
> mas5calls.merged = apply(mas5calls.calls, 1, paste, collapse="")
> eset.filtered = eset[mas5calls.merged!="AAAAAAAAAAAAAAAA",]
> nrow(eset)

Features
   54675

> nrow(eset.filtered)

Features
   29565
```

We have the choice of proceeding with either the whole or filtered datasets. For now, let's continue with the whole dataset.

2.3 Identifying differentially expressed probesets from Affymetrix arrays

Now come the differential expression statistics, answering questions about which probesets indicate different RNA levels in different groups. We have the processed expression data, and now we have to describe our experimental design to the computer. The first step is to read a tab-delimited file including columns for FileName (of each CEL file) and Target (a name of the sample used for the hybridization). You should have a file called `Affy_targets.txt` in the `Affymetrix` directory (or download it from the Hot Topics page). Read the file and then use `model.matrix()` to convert the targets matrix into a design matrix.

```
> targets = read.delim("Affymetrix/Affy_targets.txt")
> design = model.matrix(~0+targets$Target)
> colnames(design) = sort(unique(targets$Target))
> rownames(design) = targets$FileName
> head(design)

             Old_sedentary Old_trained Young_sedentary Young_trained
GSM230387.cel             1           0               0             0
GSM230388.cel             1           0               0             0
GSM230389.cel             1           0               0             0
GSM230390.cel             1           0               0             0
GSM230397.cel             0           1               0             0
GSM230398.cel             0           1               0             0
```

While we're making matrices, let's make a contrast matrix, which describes the comparisons we'd like to make. Since we're starting with log-transformed values, subtraction is really division of the untransformed values. We can make a list of all desired comparisons, optionally naming any of them. Then let's look at our output to be sure it makes sense.

```
> contrast.matrix = makeContrasts(
+ Old_trained - Old_sedentary,
+ Young_trained - Young_sedentary,
+ TrainedVsSedentary = ((Old_trained - Old_sedentary) + (Young_trained - Young_sedentary))/2,
+ levels=design)
> contrast.matrix
```

```
                Contrasts
Levels           Old_trained - Old_sedentary Young_trained - Young_sedentary
  Old_sedentary                          -1                               0
  Old_trained                             1                               0
  Young_sedentary                         0                              -1
  Young_trained                           0                               1
                Contrasts
Levels           TrainedVsSedentary
  Old_sedentary                -0.5
  Old_trained                   0.5
  Young_sedentary              -0.5
  Young_trained                 0.5
```

Now we're set to run the statistics we want. This is done in three steps:

- Given an expression matrix, fit a linear model for each "gene" (really probeset) (with `lmFit`)
- Given a linear model fit to microarray data, compute estimated coefficients and standard errors for a given set of contrasts (with `contrasts.fit`)
- Given a series of related parameter estimates and standard errors, compute moderated t-statistics, moderated F-statistic, and log-odds of differential expression by empirical Bayes shrinkage of the standard errors towards a common value (with `eBayes`)

```
> fit = lmFit(eset, design)
> fit2 = contrasts.fit(fit, contrast.matrix)
> fit2.ebayes = eBayes(fit2)
```

At this point we can look at the most differentially expressed probesets for each comparison or simply output all the data and browse it in Excel. For the former, `topTable` prints the top genes for a comparison (of number given by `coef`) and corrects/adjusts for multiple hypothesis testing (such as with False Discovery Rate).

```
> topTable(fit2.ebayes, coef=1, adjust="fdr")
```

```
               ID      logFC  AveExpr         t      P.Value   adj.P.Val
27523 218237_s_at  2.2789750 4.389281  9.808534 2.923527e-08 0.001598438
33839   224579_at  2.4804347 4.121275  7.208343 1.829193e-06 0.050005571
12184   202735_at  0.7602897 4.201023  6.124738 1.327239e-05 0.241889237
35134   225876_at  1.1684080 4.048014  5.785053 2.552826e-05 0.256469498
35447   226190_at  1.1685729 3.521960  5.684068 3.109962e-05 0.256469498
29702 220417_s_at  0.6212211 6.596310  5.586758 3.766296e-05 0.256469498
38530   229275_at  4.2891931 6.206719  5.558406 3.983298e-05 0.256469498
34401   225143_at  1.0196675 6.225151  5.489743 4.564004e-05 0.256469498
36211   226955_at -1.2948454 5.079199 -5.360829 5.902110e-05 0.256469498
12327 202878_s_at  0.8313796 5.841246  5.318093 6.430143e-05 0.256469498
             B
27523 6.041159
33839 3.693731
12184 2.392942
35134 1.941102
35447 1.802686
29702 1.667546
38530 1.627849
34401 1.531108
36211 1.347192
12327 1.285571
```

```
> topTable(fit2.ebayes, coef=2, adjust="fdr")
```

```
               ID      logFC  AveExpr         t      P.Value adj.P.Val
27523 218237_s_at  1.4452683 4.389281  6.220324 1.107207e-05 0.6053655
36211   226955_at -1.3574197 5.079199 -5.619895 3.528066e-05 0.9644851
35572   226315_at -0.8380764 4.935372 -5.184157 8.423082e-05 0.9725178
```

```
337    1552731_at -1.1920607 5.807456 -5.124706 9.501911e-05 0.9725178
10518   201069_at -0.9944146 5.186304 -4.755419 2.026544e-04 0.9725178
22215   212910_at -0.7344058 6.799115 -4.666012 2.439610e-04 0.9725178
18198   208782_at -1.3085801 6.189398 -4.627318 2.644176e-04 0.9725178
21889   212583_at -0.7480880 2.527804 -4.574729 2.950617e-04 0.9725178
39094   229839_at -1.2300552 3.588568 -4.502213 3.433639e-04 0.9725178
27178 217892_s_at -0.7797577 4.309477 -4.429156 4.002041e-04 0.9725178
             B
27523 -2.826021
36211 -2.961004
35572 -3.073210
337   -3.089519
10518 -3.196468
22215 -3.223855
18198 -3.235891
21889 -3.252427
39094 -3.275568
27178 -3.299278
```

```
> topTable(fit2.ebayes, coef=3, adjust="fdr")
```

```
              ID      logFC  AveExpr         t      P.Value    adj.P.Val
27523 218237_s_at  1.8621216 4.389281 11.334114 3.660698e-09 0.0002001487
36211   226955_at -1.3261325 5.079199 -7.764544 7.034561e-07 0.0192307315
33839   224579_at  1.6670962 4.121275  6.851463 3.451763e-06 0.0629083828
19290   209883_at -0.9646000 4.167576 -6.074986 1.459269e-05 0.1724894456
35813   226556_at  0.5088451 2.088021  6.034277 1.577407e-05 0.1724894456
35572   226315_at -0.6686346 4.935372 -5.849225 2.253426e-05 0.2053434888
39747 230492_s_at  0.5575371 3.755427  5.731002 2.836854e-05 0.2215785540
38732   229477_at -1.4453968 4.031406 -5.655578 3.288876e-05 0.2247741246
39944   230689_at -0.5868236 3.481043 -5.366766 5.832370e-05 0.2880939207
36563   227307_at  0.4862545 4.253737  5.355187 5.969177e-05 0.2880939207
             B
27523 4.1094626
36211 2.3776628
33839 1.7015770
19290 1.0247479
35813 0.9864728
35572 0.8088671
39747 0.6922711
38732 0.6165998
39944 0.3175489
36563 0.3052517
```

```
> write.fit(fit2.ebayes, file="Affy_limma_fdr.txt", digits=8, adjust="fdr")
```

Now we can browse all the output in `Affy_limma_fdr.txt`, given the following key:

- A = mean log2 level across all arrays
- Coef = log2 ratio
- t = t-statistic (can usually be ignored)
- p.value = moderated t-test raw p-value
- p.value.adj = adjusted p-value => Use this to select a threshold
- F = ANOVA F-statistic (can usually be ignored)
- F.p.value = ANOVA p-value => If this is low, there's something interesting in at least one comparison
- ID = probeset ID

How do you interpret the statistics? What probesets look like they're differentially expressed? The actual study (`http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE9103`) profiled more than 4 individuals per group. Was that larger sample size a good idea? Why?

2.4 Creating figures from Affymetrix microarray analysis

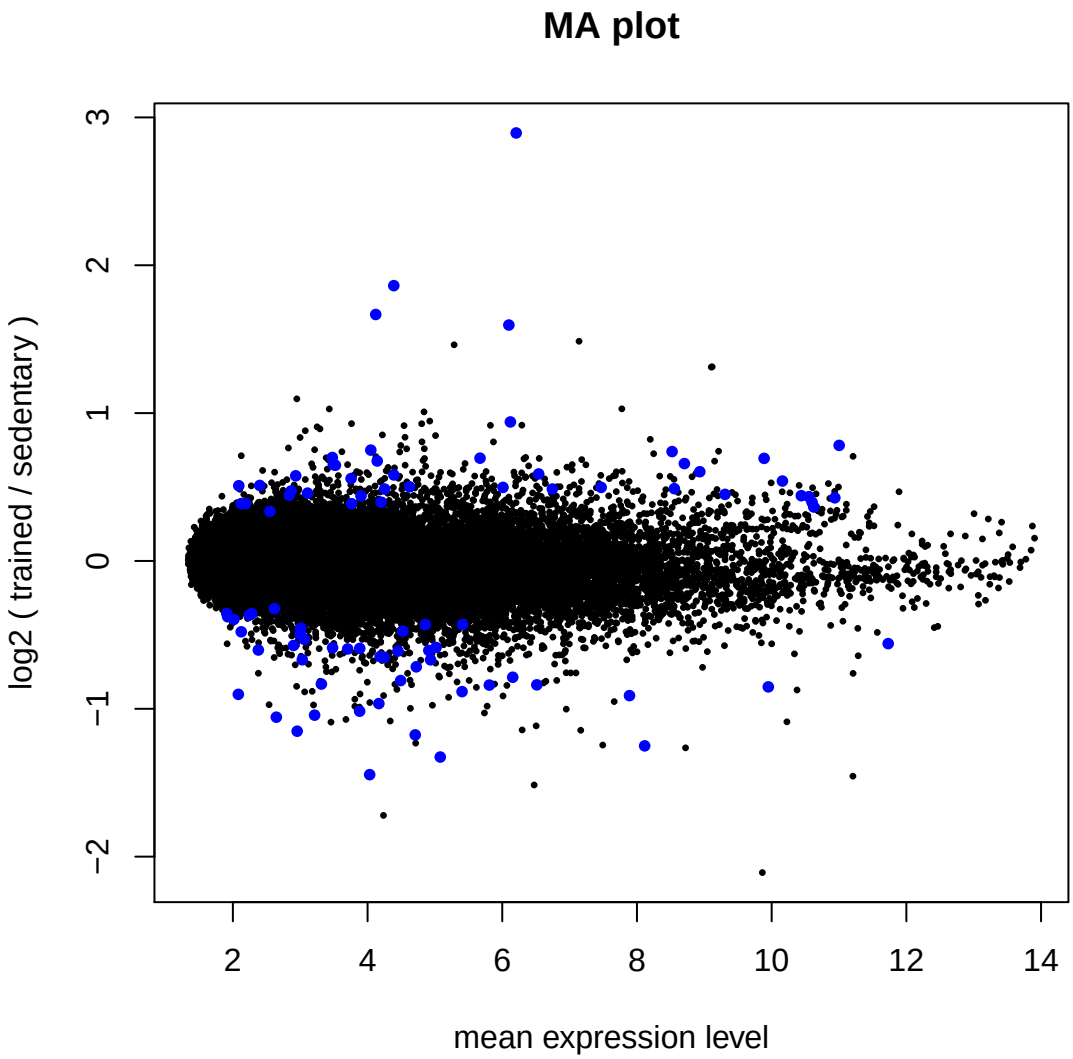
How can we create figures like scatterplots, MA plots, or volcano plots? More importantly, why would we want to? We find that these figures of all the data provide a good summary of the experiment and can help us put any differentially expressed genes in the context of all genes. They may also help us choose sensible thresholds and a definition of differential expression. We already have all the necessary data in computer memory, so we just need to access it. As with most R plots, it's easy to just plot the data, but optimizing the figure often means adding a lot of details to our commands. We'll concentrate on the all trained vs all sedentary subjects, which was the third comparison of contrast.matrix above. Since we can't find many differentially expressed genes using a typical FDR threshold, let's choose to define them as any gene with a raw p-value less than 1e-3. If we want, we can plot those genes in another color.

```
> comparison = 3
> DE = fit2.ebayes$p.value[,comparison] < 1e-3
> sum(DE)
```

```
[1] 89
```

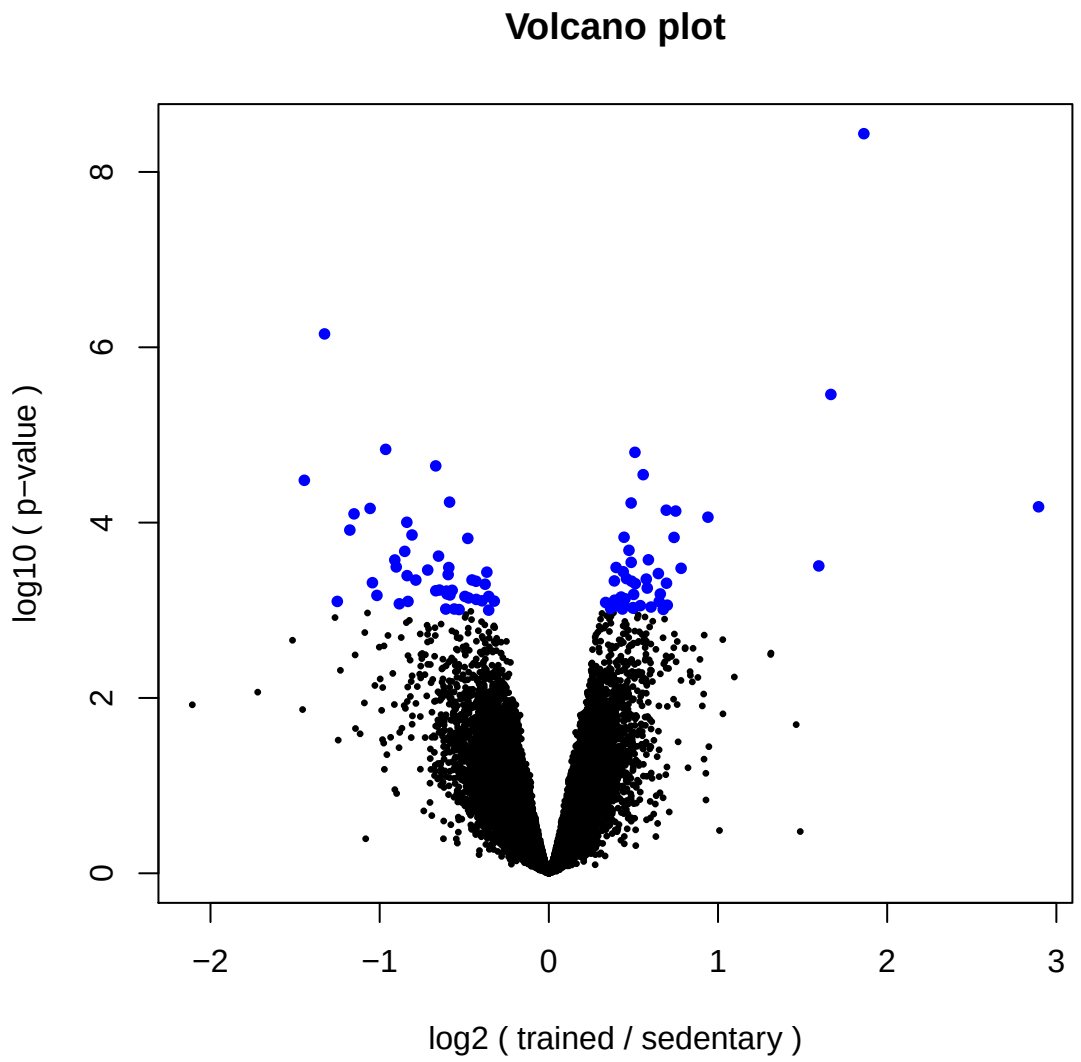
The variable DE is a list of TRUEs (differentially expressed, with n = the output of the previous command) and FALSEs (not DE) that we can use to subset the array. We can draw a MA (ratio-intensity) plot, which is like a scatterplot that's been rotated clockwise 45 degrees.

```
> plot(fit2.ebayes$Amean, fit2.ebayes$coeff[,comparison], main="MA plot",
+         ylab="log2 ( trained / sedentary )", xlab="mean expression level",
+         pch=20, cex=0.5)
> points(fit2.ebayes$Amean[DE], fit2.ebayes$coeff[DE,comparison], col="blue", pch=20, cex=1)
```



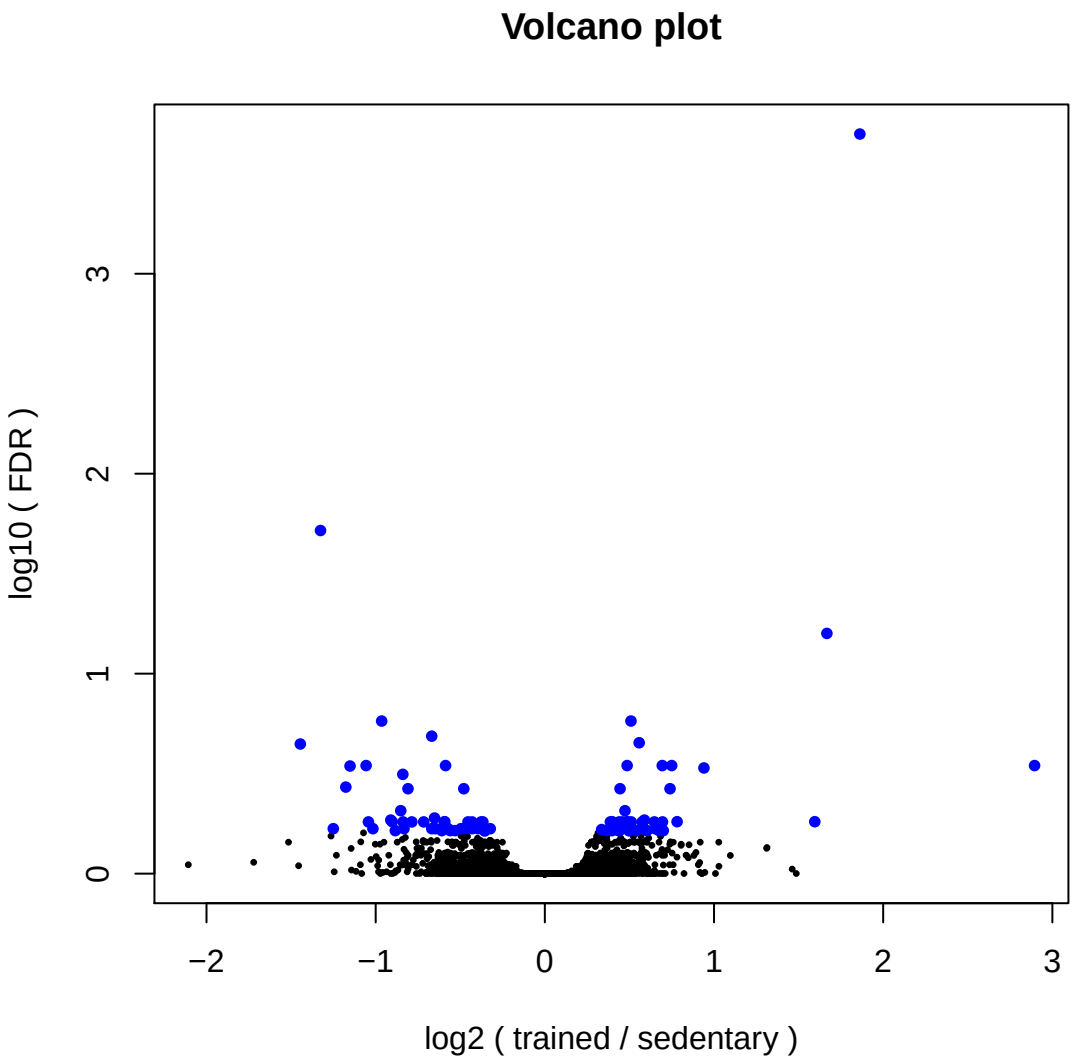
To draw a volcano plot, we would normally use p-values after FDR correction, but these aren't very low, so we get a more informative figure using raw p-values.

```
> plot(fit2.ebayes$coeff[,comparison], -log10(fit2.ebayes$p.value[,comparison]), main="Volcano plot",
+         xlab="log2 ( trained / sedentary )", ylab="log10 ( p-value )", pch=20, cex=0.5)
> points(fit2.ebayes$coeff[DE,comparison], -log10(fit2.ebayes$p.value[DE,comparison]),
+         col="blue", pch=20, cex=1)
```



Nevertheless, if we want to plot FDRs, we first need to calculate them from the raw p-values in `fit2.ebayes`.

```
> FDR = p.adjust(fit2.ebayes$p.value[,comparison], method="fdr")
> plot(fit2.ebayes$coeff[,comparison], -log10(FDR), main="Volcano plot",
+         xlab="log2 ( trained / sedentary )", ylab="log10 ( FDR )", pch=20, cex=0.5)
> points(fit2.ebayes$coeff[DE,comparison], -log10(FDR[DE]),
+         col="blue", pch=20, cex=1)
```



2.5 [Skip for now]: Getting Affymetrix probeset annotations

We can link probsets to gene symbols and other information using a variety of resources. All of these involve big downloads, so we are going to hold off doing this as a class. One source is the Affymetrix arrays web page (`http://www.affymetrix.com/support/technical/byproduct.affx?cat=arrays`). If we choose "Human Genome Arrays +" (under "3' Gene Expression Analysis Arrays") and go to "Human Genome U133 Plus 2.0 Array", we can download the file described as "HG-U133_Plus_2 Annotations, CSV format" to get a big Excel file (after registering for a free account). Another source is Bioconductor. After installing the `hgu133plus2.db` package (or a comparable package for our array design), we can get a gene symbol for most probes using the following commands:

```
> library(hgu133plus2.db)
> library(annotate)
> symbols = getSYMBOL(rownames(eset.values), "hgu133plus2")
> write.table(cbind(rownames(eset.values), symbols), file="hgu133plus2.symbols.txt",
+         sep="\t", row.names=F, quote=F)
```

3 Agilent arrays

The initial processing of the two types of arrays start out quite differently, but the statistics are quite similar. Feel free to skip sections that appear redundant.

3.1 Getting started with Agilent arrays

Go to the Agilent directory you just created and see what input data files are there.

```
> setwd("Agilent")
> dir()
```

```
 [1] "Agilent_targets.txt" "GSM522121.txt"       "GSM522122.txt"
 [4] "GSM522123.txt"       "GSM522124.txt"       "GSM522125.txt"
 [7] "GSM522126.txt"       "GSM522161.txt"       "GSM522162.txt"
[10] "GSM522163.txt"       "GSM522164.txt"       "GSM522165.txt"
[13] "GSM522166.txt"       "GSM522211.txt"       "GSM522212.txt"
[16] "GSM522213.txt"       "GSM522214.txt"       "GSM522215.txt"
[19] "GSM522216.txt"       "GSM522260.txt"       "GSM522261.txt"
[22] "GSM522262.txt"       "GSM522263.txt"       "GSM522264.txt"
[25] "GSM522265.txt"
```

The directory should include 24 2-color Agilent txt files, each containing probe-level measurements for two biological samples. These arrays are a subset of a human study (`http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE20881`) that examines "Colon biopsies from Crohns patients and healthy controls". You have several txt files each from 4 groups: healthy ileum, healthy descending colon, ileum with Crohn's disease, and descending colon with Crohn's disease. The intestinal samples are always hybridized on the Cy3 (green) channel, whereas the Cy5 (red) channel is always for Universal Human Reference RNA. Why is this design less than optimal? Running quality control on the set of arrays can be helpful, and this will be covered in a future Hot Topics talk.

Let's load the Bioconductor package `limma` (if you haven't already). If you get an error that `limma` can't be found, it's probably not installed. In that case you can install it by adding the Bioc repositories to the usual CRAN repository (with the command `setRepositories()`) and then install the package with the command `install.packages("limma")`. After installation, you still need to run the command `library(limma)` to access the library.

```
> library(limma)
```

We're first going to read all the array files. These are tab-delimited text files that can be opened in Excel, if you ever wonder what they look like. Just in case we have other txt files in the directory, we're going to limit ourselves to files that start with "GSM". These files have lots of data, but all Bioconductor needs are the columns gMeanSignal and rMeanSignal (for foreground) and gBGMedianSignal and rBGMedianSignal (for background).

```
> agilentFiles = dir(pattern = "GSM.*.txt$")
> RG = read.maimages(agilentFiles, source="agilent")
```

```
Read GSM522121.txt
Read GSM522122.txt
Read GSM522123.txt
Read GSM522124.txt
Read GSM522125.txt
Read GSM522126.txt
Read GSM522161.txt
Read GSM522162.txt
Read GSM522163.txt
Read GSM522164.txt
Read GSM522165.txt
Read GSM522166.txt
Read GSM522211.txt
Read GSM522212.txt
Read GSM522213.txt
Read GSM522214.txt
Read GSM522215.txt
Read GSM522216.txt
Read GSM522260.txt
Read GSM522261.txt
Read GSM522262.txt
Read GSM522263.txt
Read GSM522264.txt
Read GSM522265.txt
```

```
> setwd("../")
```

3.2 Normalizing Agilent arrays

We want to transform the raw array data so we can make the most valid comparisons between samples. These transformations can include steps such as background subtraction, normalization within arrays (between a red and corresponding green channel) and between arrays. The first session of BaRC's Microarray Analysis course (`http://jura.wi.mit.edu/bio/education/bioinfo2007/arrays/`) describes normalization in more detail. For even more detail, check out the limma User's Guide, linked from `http://www.bioconductor.org/packages/release/bioc/html/limma.html`. We're going to skip background subtraction, as some researchers find that background subtraction adds more noise and doesn't appear to make replicate samples more consistent. Then we're going to normalize within arrays (across channels) with loess (locally-weighted scatterplot smoothing) to try to minimize dye-dependent effects, even if they're intensity-dependent. Finally, we going to run "Aquantile" normalization between arrays, a method which involves quantile normalization of the A values (mean probe intensities across both channels).

```
> RG.nobg.0 = backgroundCorrect(RG, method="none", offset=0)
> MA.loess.0 = normalizeWithinArrays(RG.nobg.0, method="loess")
> MA.loess.q.0 = normalizeBetweenArrays(MA.loess.0, method="Aquantile")
```

It may be helpful to know that this process turns a RGList (with an array described in terms of Red and Green intensities) into a MAList (with an array described in terms of M (log2(Cy5/Cy3)) and A (mean of Cy5 and Cy3 intensities)). These data structures include each several matrices combined together, and if you ever want to know the names of each underlying variable, use the `names()` command. Once you know the underlying variables, it's easy to access them.

```
> names(RG.nobg.0)

[1] "G"       "R"       "targets" "genes"   "source"

> RG.nobg.0$R[1:5,1:5]

     GSM522121 GSM522122 GSM522123 GSM522124 GSM522125
[1,] 153.50000 140.46670  100.1111 117.57140 111.70370
[2,]  69.62069  63.82759   62.6129  73.86207  58.72414
[3,] 126.42860 148.36000  117.6786 111.03700 124.32140
[4,] 120.18520 142.71880  119.0345 104.38710 114.73330
[5,] 204.00000 202.41380  136.8929 141.96550 152.85190

> names(MA.loess.0)

[1] "targets" "genes"   "source"  "M"       "A"

> MA.loess.0$M[1:5,1:5]

       GSM522121   GSM522122   GSM522123  GSM522124    GSM522125
[1,] -4.68418366 -4.39969784 -4.65798271 -4.2325219 -5.188262292
[2,] -0.06994817 -0.05703049  0.01222401 -0.2264735 -0.005352078
[3,] -0.05570601  0.13241709  0.08152189 -0.1469588  0.132393704
[4,] -0.28353851 -0.06487891  0.06440928 -0.2402458 -0.045137713
[5,] -0.46355283 -0.11068345 -0.16382304 -0.4892521 -0.275981192
```

We can create a file with normalized data from all arrays, which can be useful for creating a heatmap or simply checking out the data from each array (before future summarization by `limma`). `names(MA.loess.q.0)` contains different types of data, but let's just print the M values (log2 fold changes) for now, together with probe IDs and gene symbols from `MA.loess.q.0$genes`.

```
> log2.ratios = round(MA.loess.q.0$M, 4)
> probe.info = MA.loess.q.0$genes[,c("ProbeName","GeneName")]
> write.table(cbind(probe.info, log2.ratios), file="Agilent_log2_ratios.txt",
+   sep="\t", row.names=F, quote=F)
```

3.3 Identifying differentially expressed probesets from Agilent arrays

We're going to use the normalized M and A values (a MAList) for differential expression analysis. Now we have to describe our experimental design to the computer. The first step is to read a tab-delimited file including columns for FileName (of each txt file) and target (a name of the sample used for the hybridization) for each Cy5 and Cy3 channel of each array. You should have a file called `Agilent_targets.txt` in your current directory (or download it from the Hot Topics page). Read the file and then use `model.matrix()` to convert the targets matrix into a design matrix. To do this we choose "UHR" as our refence sample.

```
> targets = read.delim("Agilent/Agilent_targets.txt")
> design = modelMatrix(targets, ref="UHR")

Found unique target names:
 crohns_descending_colon crohns_terminal_ileum healthy_descending_colon healthy_terminal_ileum UHR

> rownames(design) = agilentFiles
> head(design)

              crohns_descending_colon crohns_terminal_ileum
GSM522121.txt                       0                     0
GSM522122.txt                       0                     0
GSM522123.txt                       0                     0
GSM522124.txt                       0                     0
GSM522125.txt                       0                     0
GSM522126.txt                       0                     0
              healthy_descending_colon healthy_terminal_ileum
GSM522121.txt                        0                    -1
GSM522122.txt                        0                    -1
GSM522123.txt                        0                    -1
GSM522124.txt                        0                    -1
GSM522125.txt                        0                    -1
GSM522126.txt                        0                    -1
```

The design table has a "1" if the non-reference sample is on the Cy5 channel or a "-1" if it's on the Cy3 channel. Our experiment has all intestinal samples on the Cy3 channel, so we only see the latter.

While we're making matrices, let's make a contrast matrix, which describes the comparisons we'd like to make. Since we're starting with log-transformed ratios, subtraction is really division (of untransformed ratios). We can make a list of all desired comparisons, optionally naming any of them. then let's look at our output to be sure it makes sense.

```
> contrast.matrix = makeContrasts(
+ Colon = crohns_descending_colon - healthy_descending_colon,
+ Ileum = crohns_terminal_ileum - healthy_terminal_ileum,
+ CrohnsVsHealthy = ((crohns_descending_colon - healthy_descending_colon) +
+         (crohns_terminal_ileum - healthy_terminal_ileum))/2,
+ levels=design)
> contrast.matrix

                        Contrasts
Levels                   Colon Ileum CrohnsVsHealthy
  crohns_descending_colon     1     0             0.5
  crohns_terminal_ileum      0     1             0.5
  healthy_descending_colon  -1     0            -0.5
  healthy_terminal_ileum     0    -1            -0.5
```

This design is just like we set up for the Affymetrix experiment above. Many other designs are possible, especially for more complex experiments, and the limma User's Guide has examples for common designs. Often multiple designs are possible, and your choice is worth some careful thought.

Now we're set to run the statistics we want. This is done in three steps:

- Fit linear model for each gene (probeset) given an expression matrix (with `lmFit`)
- Given a linear model fit to microarray data, compute estimated coefficients and standard errors for a given set of contrasts (with `contrasts.fit`)
- Given a series of related parameter estimates and standard errors, compute moderated t-statistics, moderated F-statistics, and log-odds of differential expression by empirical Bayes shrinkage of the standard errors towards a common value (with `eBayes`)

```
> fit = lmFit(MA.loess.q.0, design)
> fit2 = contrasts.fit(fit, contrast.matrix)
> fit2.ebayes = eBayes(fit2)
```

At this point we can look at the most differentially expressed probesets for each comparison or simply output all the data and browse it in Excel. For the former, `topTable` prints the top genes for a comparison (of number given by `coef`) and corrects/adjusts for multiple hypothesis testing (such as with False Discovery Rate). To reduce the annotation information in fit2.ebayes, we're going to drop everything but ProbeName and GeneName (symbol).

```
> fit2.ebayes$genes = fit2.ebayes$genes[,c("ProbeName","GeneName")]
> topTable(fit2.ebayes, coef=1, adjust="fdr")

         ProbeName  GeneName      logFC  AveExpr         t      P.Value
3047   A_23_P27285     MPPE1 -0.4397415 8.610479 -5.028687 5.056127e-05
5998  A_23_P212870     MADH1 -0.3450835 8.664701 -4.584170 1.481566e-04
10996  A_23_P43476     VLDLR  0.3291409 7.395841  4.498770 1.823116e-04
4655   A_24_P21447     SURF6  0.4346200 7.214387  4.481538 1.901090e-04
36429  A_23_P63847   SUPV3L1  0.5026690 8.408700  4.367369 2.509333e-04
23158 A_24_P934135  AK092791 -0.5061809 7.655851 -4.210624 3.673887e-04
35952  A_32_P76441 I_1959765  0.2418376 8.743558  4.033449 5.651055e-04
15193  A_23_P48897      CPR8 -0.3333778 8.242222 -4.000634 6.119635e-04
42015    (+)eQC-38       eQC  0.1352048 6.005968  3.955003 6.836040e-04
24075 A_24_P105283      SFPQ  0.4181653 7.970554  3.905796 7.702084e-04
      adj.P.Val          B
3047  0.9960887 -0.2730818
5998  0.9960887 -0.7801145
10996 0.9960887 -0.8805097
4655  0.9960887 -0.9008758
36429 0.9960887 -1.0366897
23158 0.9960887 -1.2255046
35952 0.9960887 -1.4418612
15193 0.9960887 -1.4822373
42015 0.9960887 -1.5385264
24075 0.9960887 -1.5994041

> topTable(fit2.ebayes, coef=2, adjust="fdr")

         ProbeName     GeneName      logFC  AveExpr         t      P.Value
42308  A_23_P10025        NELL2  1.1530383 6.940010  7.899458 7.813107e-08
21568 A_32_P113584     AB011102  0.4102003 7.554871  7.304647 2.745415e-07
33578 A_23_P155837     I_958050  0.3811986 6.697331  7.129611 4.009836e-07
17984  A_32_P95541 A_32_BS95541  0.6015552 7.620763  7.037055 4.907193e-07
22333 A_23_P434430       ZNF439  0.5847760 7.195110  6.381807 2.116102e-06
20986 A_23_P126075        KCNK1 -1.4159275 9.119713 -6.268206 2.741301e-06
42503   A_32_P9816     AI590869  0.2731203 6.608594  5.947805 5.735841e-06
26788 A_23_P154379         NAT8  1.5397269 8.605989  5.914063 6.203835e-06
30138 A_23_P155835     I_958050  0.3971465 8.745790  5.805771 7.986317e-06
9214  A_23_P141394     FLJ10055 -0.9852119 9.593362 -5.737895 9.362092e-06
        adj.P.Val        B
42308 0.003432376 7.412342
21568 0.005389447 6.393650
33578 0.005389447 6.082326
17984 0.005389447 5.915592
22333 0.018592495 4.694136
20986 0.020071346 4.475245
42503 0.034067584 3.847113
26788 0.034067584 3.780067
30138 0.037175578 3.563780
9214  0.037175578 3.427369

> topTable(fit2.ebayes, coef=3, adjust="fdr")

         ProbeName        GeneName      logFC  AveExpr         t      P.Value
42308  A_23_P10025           NELL2  0.5904919 6.940010  5.721139 9.737441e-06
42015    (+)eQC-38             eQC  0.1377022 6.005968  5.696533 1.031662e-05
36295 A_24_P935902       I_3549574  0.1890874 6.450098  5.541686 1.486118e-05
```

```
33578 A_23_P155837        I_958050  0.2064238 6.697331  5.459958 1.803521e-05
39458 A_24_P129834            TPH2  0.1485352 6.064550  5.278675 2.776652e-05
24048    (-)3xSLv1 NegativeControl -0.1221310 6.019019 -5.226331 3.146696e-05
30541 A_24_P911259        AK023557 -0.4867591 7.752537 -5.185566 3.469230e-05
35952  A_32_P76441       I_1959765  0.2184358 8.743558  5.152187 3.758157e-05
14653    (-)3xSLv1 NegativeControl -0.1284890 6.020543 -5.130521 3.958627e-05
38608 A_24_P540560       XM_293353  0.1531962 6.657913  5.109038 4.168100e-05
      adj.P.Val        B
42308 0.1431994 3.117274
42015 0.1431994 3.071099
36295 0.1431994 2.778459
33578 0.1431994 2.622616
39458 0.1431994 2.273686
24048 0.1431994 2.172140
30541 0.1431994 2.092822
35952 0.1431994 2.027722
14653 0.1431994 1.985396
38608 0.1431994 1.943373
```

```
> write.fit(fit2.ebayes, file="Agilent_limma_fdr.txt", digits=8, adjust="fdr")
```

Note that these arrays include gene annotation in the actual scanner files (which isn't the case for Affymetrix arrays). Now we can browse all the output in `Agilent_limma_fdr.txt`, given the following key:

- A = mean log2 level across all arrays
- Coef = log2 ratio
- t = t-statistic (can usually be ignored)
- p.value = moderated t-test raw p-value
- p.value.adj = adjusted p-value => Use this to select a threshold
- F = ANOVA F-statistic (can usually be ignored)
- F.p.value = ANOVA p-value => If this is low, there's something interesting in at least one comparison
- Genes.ProbeName = Probe ID
- Genes.GeneName = Gene symbol (from original array file (so these may be out of date))

How do we interpret the statistics? What probesets look like they're differentially expressed? Does Crohn's disease affect the descending colon (towards the end of the large intestine) in a similar manner to the ileum (the last section of the small intestine)? The actual study (`http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE20881`) profiled more than these individuals per group. Note that different groups have differing numbers of samples. Why was the experiment designed in this way? Are there any drawbacks to this design?

3.4 Creating figures from Agilent microarray analysis

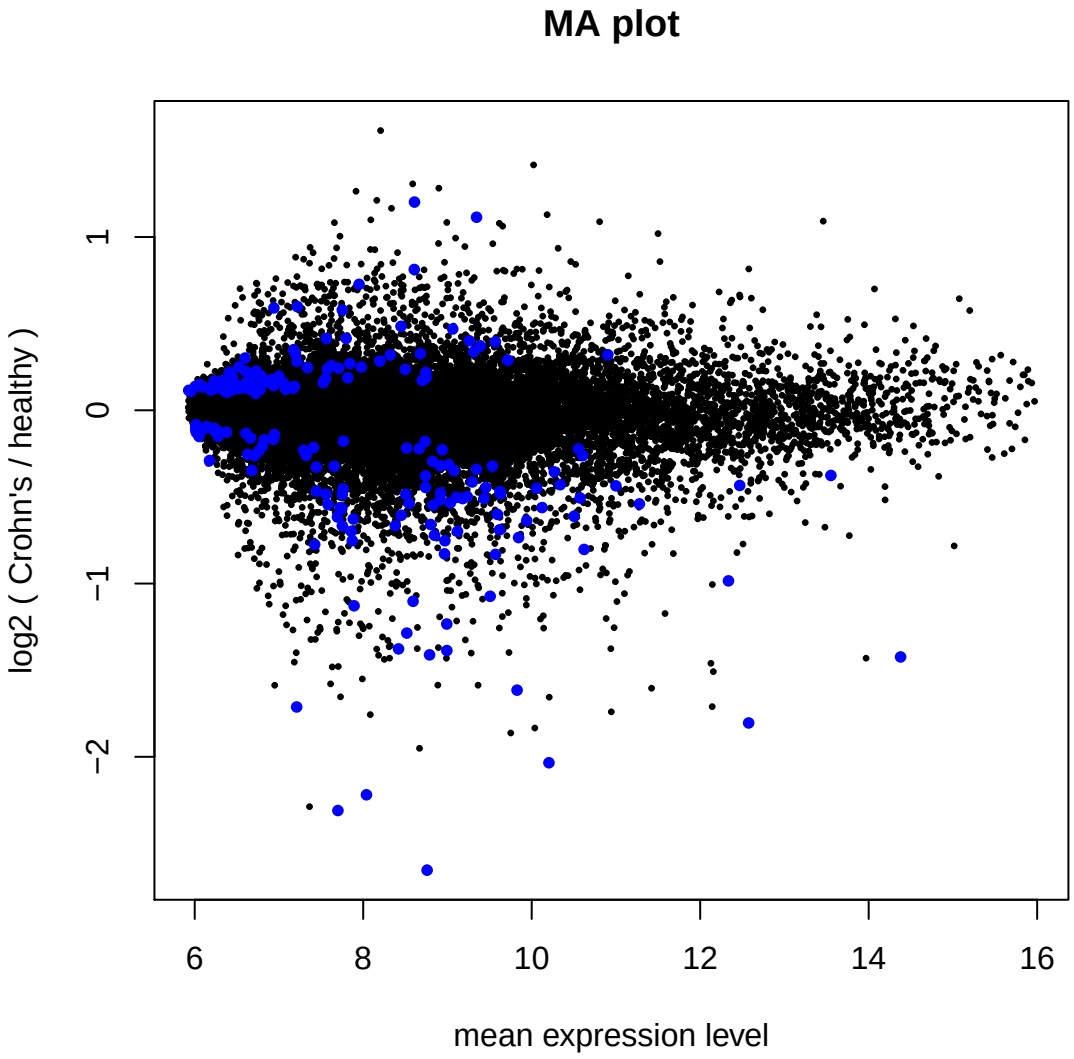
How can we create figures like scatterplots, MA plots, or volcano plots? More importantly, why would we want to? We find that these figures of all the data provide a good summary of the experiment and can help us put any differentially expressed genes in the perspective of all genes. They may also help us choose sensible thresholds and a definition of differential expression. We already have all the necessary data in computer memory, so we just need to access it. As with most R plots, it's easy to just plot the data, but optimizing the figure often means adding a lot of details to our commands. We'll concentrate on all Crohn's vs all healthy subjects, which was the third comparison of contrast.matrix above. Since we can't find many differentially expressed genes using a typical FDR threshold, let's choose to define them as any gene with a raw p-value less than 1e-3. If we want, we can plot those genes in another color.

```
> comparison = 3
> DE = fit2.ebayes$p.value[,comparison] < 1e-3
> sum(DE)
```

```
[1] 204
```

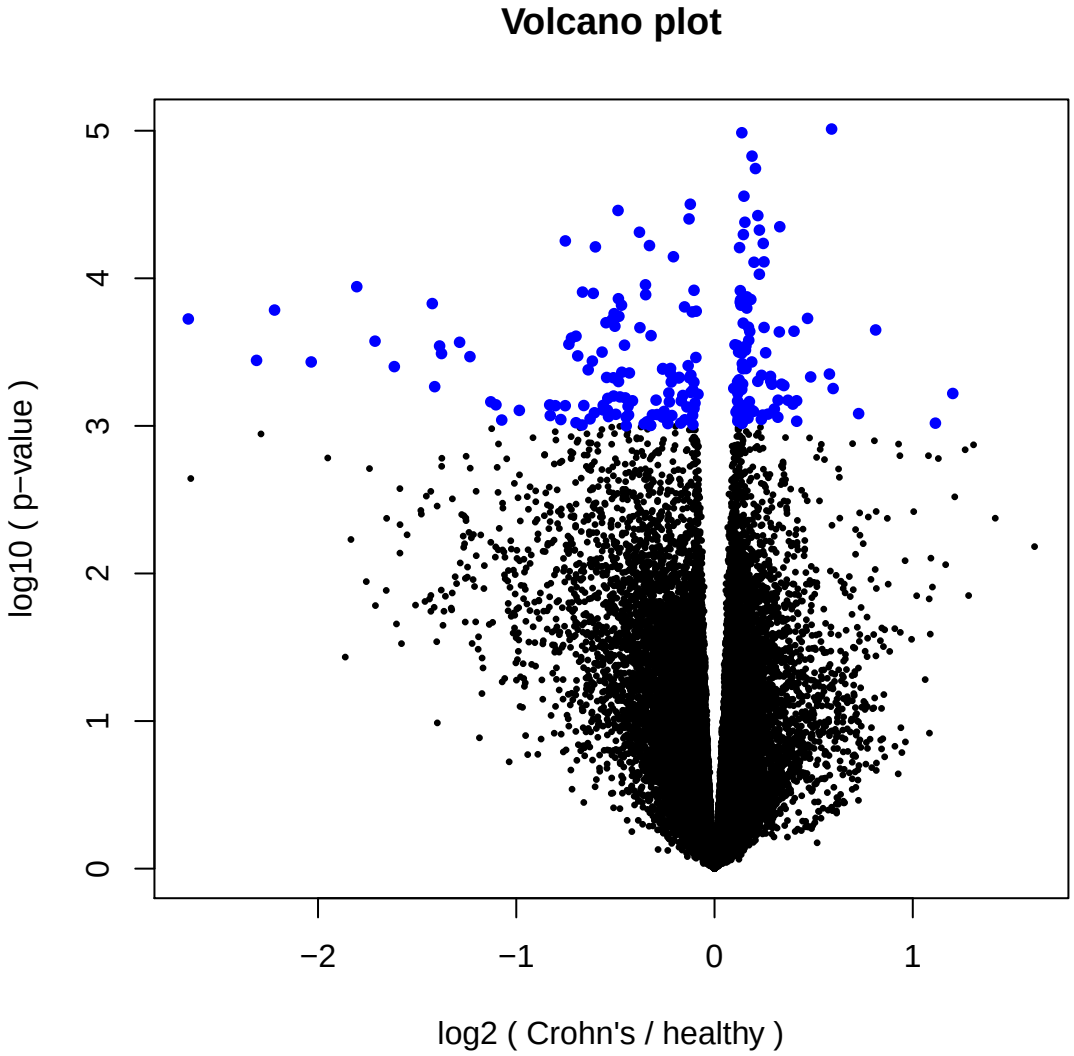
The variable DE is a list of TRUEs (differentially expressed, with n = the output of the previous command) and FALSEs (not DE) that we can use to subset the array. Let's draw a MA (ratio-intensity) plot, which is like a scatterplot that's been rotated clockwise 45 degrees.

```
> plot(fit2.ebayes$Amean, fit2.ebayes$coeff[,comparison], main="MA plot",
+         ylab="log2 ( Crohn's / healthy )", xlab="mean expression level",
+         pch=20, cex=0.5)
> points(fit2.ebayes$Amean[DE], fit2.ebayes$coeff[DE,comparison], col="blue", pch=20, cex=1)
```



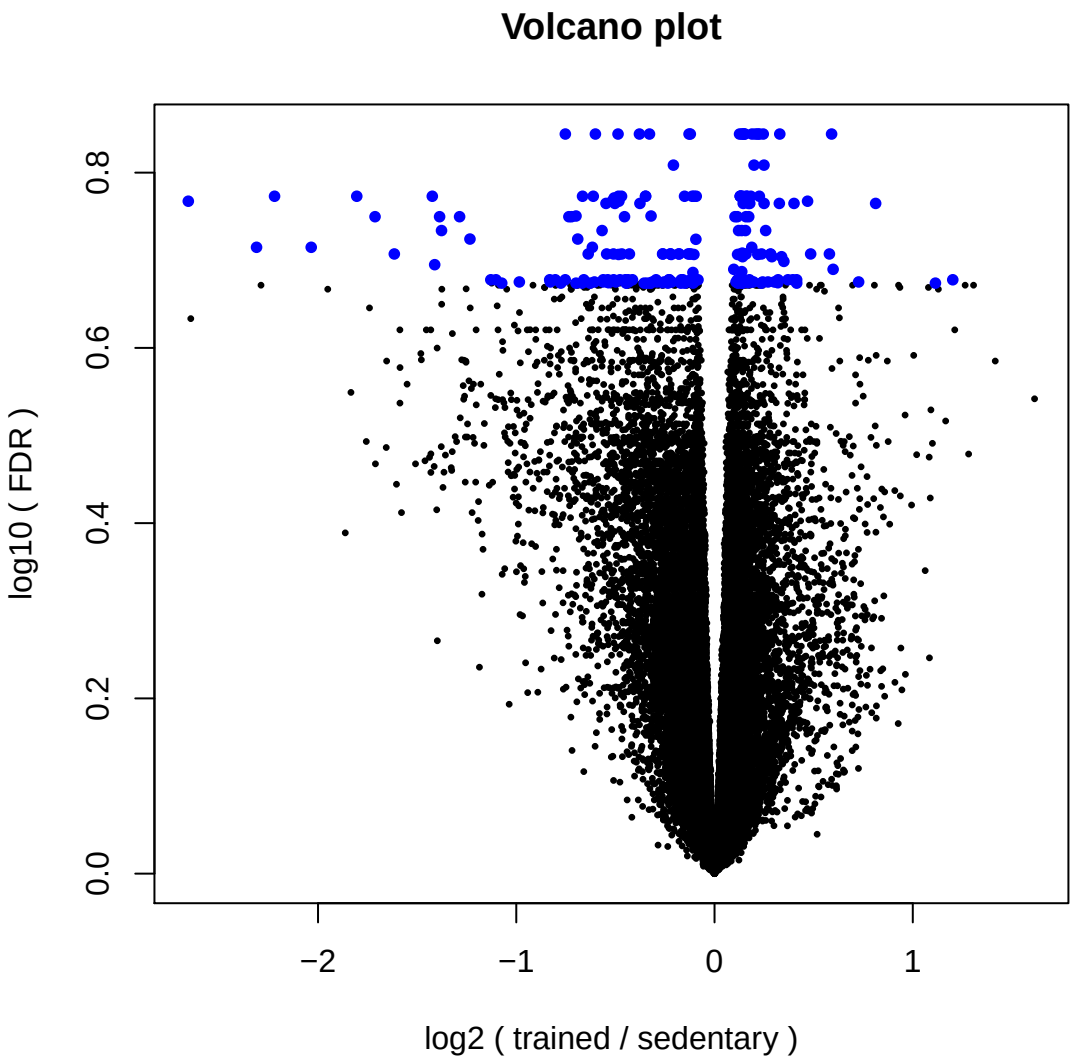
To draw a volcano plot, we would normally use p-values with the FDR correction, but these aren't very low, so we get a more interesting figure using raw p-values.

```
> plot(fit2.ebayes$coeff[,comparison], -log10(fit2.ebayes$p.value[,comparison]), main="Volcano plot",
+         xlab="log2 ( Crohn's / healthy )", ylab="log10 ( p-value )", pch=20, cex=0.5)
> points(fit2.ebayes$coeff[DE,comparison], -log10(fit2.ebayes$p.value[DE,comparison]),
+         col="blue", pch=20, cex=1)
```



If we want to plot FDRs, we first need to calculate them from the raw p-values in `fit2.ebayes`.

```
> FDR = p.adjust(fit2.ebayes$p.value[,comparison], method="fdr")
> plot(fit2.ebayes$coeff[,comparison], -log10(FDR), main="Volcano plot",
+         xlab="log2 ( trained / sedentary )", ylab="log10 ( FDR )", pch=20, cex=0.5)
> points(fit2.ebayes$coeff[DE,comparison], -log10(FDR[DE]),
+         col="blue", pch=20, cex=1)
```



How do these plots compare to those for the Affymetrix study? Every microarray experiment has a particular level of variability within and between samples that influences the optimal sample size and, together with one's choice of statistical threhsolds, the relative numbers of false positives and false negatives from the analysis.

4 RNA-Seq

4.1 Getting started with RNA-Seq

Go to the RNASeq directory you created and see what input data files are there.

```
> setwd("RNASeq")
> dir()
```

```
[1] "Brain_UHR_duplicates_counts.txt"
```

The main file is a matrix of counts derived from a subset of an RNA-Seq experiment performed by the MicroArray Quality Control (MAQC) project, which has expanded to include high-throughput sequencing. This study looked at RNA levels in the brain (from Ambion's human brain reference) and the Universal Human Reference (UHR, from Stratagene). The original short reads came from `http://www.ncbi.nlm.nih.gov/sra?term=SRA010153`, of which we analyzed duplicate samples from each RNA source. Note that this experiment is atypical in that brain and UHR are much more different than most biomedically relevant comparisons. RNA-Seq data can be fully processed in R, but we prefer to use stand-alone applications for short-read mapping and counting. For each samples, we started our analysis by mapping with tophat, the spliced-read mapper (`http://tophat.cbcb.umd.edu/`) and counting reads overlapping genes with htseq-count (`http://www-huber.embl.de/users/anders/HTSeq/doc/count.html`) using commands like these:

- `bsub tophat -o brain_1 -solexa-quals -G /nfs/genomes/human_gp_feb_09/gtf/hg19.refgene.gtf /nfs/genomes/human_gp_feb_09_no_random/bowtie/hg19 SRR037452.fastq`
- `bsub samtools view -h -o brain_1/accepted_hits.sam brain_1/accepted_hits.bam`
- `bsub "htseq-count brain_1/accepted_hits.sam /nfs/genomes/human_gp_feb_09/gtf/hg19.refgene.gtf > brain_1_gtf.htseq-count.out"`

Output of this set includes short read counts (not RPKM values) for each gene ID. After running this set of commands on all of our four samples, we merged them together into one matrix, with gene symbols in the first column.

4.2 Preprocessing RNA-Seq counts

Let's read the matrix, using the first column as row names, and see what our matrix looks like. After we read the file, we'll go up one level to the starting directory

```
> counts = read.delim("Brain_UHR_duplicates_counts.txt", row.names=1)
> setwd("../")
> nrow(counts)

[1] 22131

> head(counts)

      brain_1 brain_2 UHR_1 UHR_2
A1BG       46      64    95   113
A1CF        0       0    59    61
A2BP1    1011    1042     4     5
A2LD1       9      12    14    22
A2M       872     966  3938  4127
A2ML1      14      14     2     2
```

We have column headers (the first row of the file) but have to tell R which columns are replicates (i.e., name our biological groups). We can use the `rep()` command, which repeats a value a given number of times.

```
> groups = c(rep("brain",2), rep("UHR", 2))
> groups

[1] "brain" "brain" "UHR"   "UHR"
```

We could use the current counts matrix for differential expression analysis, but at BaRC we found a couple of additional steps that can be helpful. First, we remove all genes that have no counts in any sample. These can't possibly be interesting for our analysis, and removing them will at least slightly minimize the False Discovery Rate correction of our p-values. Second, we add pseudocounts (at least 1 count) to all positions in the matrix. This has the obvious effect of preventing any division by 0 (when we calculate fold changes) and is also similar to an offset for array intensities, which reduces the noise in genes with low expression. For our sample analysis we'll add 1 pseudocount. To calculate the total counts per gene, we use the `apply()` function, which applies a function (such as `sum()`) across all rows (if the second argument is 1) or across all columns (if the second argument is 2).

```
> sum.by.gene = apply(counts, 1, sum)
> counts.no0genes = counts[sum.by.gene > 0,]
> counts.no0genes.offset.1 = counts.no0genes + 1
> nrow(counts.no0genes.offset.1)

[1] 19069

> head(counts.no0genes.offset.1)

      brain_1 brain_2 UHR_1 UHR_2
A1BG       47      65    96   114
A1CF        1       1    60    62
A2BP1    1012    1043     5     6
A2LD1      10      13    15    23
A2M       873     967  3939  4128
A2ML1      15      15     3     3
```

Note how the output from the last command compares to the output from `head(counts)`, and how the number of genes (from `nrow()`) compare between the two matrices. Also note that we haven't yet normalized between samples, so one sample may include many more reads than another. Most RNA-Seq statistics packages are aware of this and expect input data to be unnormalized.

4.3 Identifying differentially expressed genes from RNA-Seq

Now we're going to need an R package to help us identify the differentially expressed genes. In BaRC we've had success with three packages that assay for differential expression: `edgeR`, `DESeq`, and `baySeq`. `edgeR` and `DESeq` are quite similar, and we've decided to use sample code for `DESeq`.

Let's load the Bioconductor package `DESeq` which includes many of the commands we'll be using.

```
> library(DESeq)

locfit 1.5-6          2010-01-20
```

If you get an error that `DESeq` can't be found, it's probably not installed. In that case you can install it by adding the Bioc repositories to the usual CRAN repository (with the command `setRepositories()`) and then install the package with a command like `install.packages("DESeq")`. After installation, you still need to run the command `library(DESeq)` to access the library. We can get lots more theoretical details about `DESeq` from the publication (`http://www.ncbi.nlm.nih.gov/pubmed/20979621`) and practical details from the vignette (`http://www.bioconductor.org/packages/2.8/bioc/vignettes/DESeq/inst/doc/DESeq.pdf` or the R command `vignette("DESeq")`)

We start by creating a CountDataSet, which is DESeq's terminology for a data frame of RNA-Seq counts and other associated information.

```
> cds = newCountDataSet(counts.no0genes.offset.1, groups)
```

We want to calculate a normalization factor to account for each sample having a different number of reads. The most obvious way of doing this is to calculate this from the total number of mapped reads that are associated with genes (so the total of each column of our counts matrix). This method can be unduly influenced by a few very-highly-expressed or very-differentially-expressed genes, and `DESeq` instead recommends a method called `estimateSizeFactors()` that uses geometric means. We'll look at how the scaling factors from each method differs for our data.

```
> # Naive way
> sum.by.sample = apply(counts.no0genes.offset.1, 2, sum)
> sum.by.sample

brain_1 brain_2   UHR_1   UHR_2
4901896 5421676 5748016 5993249

> sum.by.sample/mean(sum.by.sample)

  brain_1   brain_2     UHR_1     UHR_2
0.8886349 0.9828626 1.0420228 1.0864796

> # DESeq recommendation
> cds = estimateSizeFactors(cds)
> sizeFactors(cds)

  brain_1   brain_2     UHR_1     UHR_2
0.7994247 0.8495221 1.1892071 1.2366183
```

A key step of assaying for differential expression with `DESeq` is to use the actual counts data to calculate the variability (variance) for each gene.

```
> cds = estimateVarianceFunctions(cds)
```

Once we have the variability for each gene, we can figure out how confident we are that each gene is differentially expressed. As with microarrays, the same major caveat applies: we keep saying "differential expression" but really we mean "differential RNA abundance". We have no way of knowing if changing RNA levels are due to changes in transcription, RNA degradation, regulation by miRNAs, and/or some other mechanism. In any case, `DESeq` uses a method based on the negative binomial distribution. Since we may have an experiment that compares multiple different samples, we need to identify which two samples we want to compare, with the reference sample first (since `nbinomTest(cds, "A", "B")` calculates log2 ratios for B/A).

```
> results = nbinomTest(cds, "UHR", "brain")
```

We can take a look at the results of our statistical analysis. We can also print the output for all genes, together with our preprocessed, normalized counts for each sample. To get the latter, we divide our preprocessed matrix by the scaling factors.

```
> head(results)
```

```
     id   baseMean   baseMeanA   baseMeanB   foldChange log2FoldChange
1  A1BG   77.05471   86.456474   67.652940   0.78250867     -0.3538214
2  A1CF   25.75464   50.295258    1.214016   0.02413778     -5.3725631
3 A2BP1  625.67894    4.528212 1246.829674 275.34702721      8.1051072
4 A2LD1   14.75607   15.606278   13.905858   0.89104256     -0.1664337
5   A2M 2220.18727 3325.213415 1115.161130   0.33536528     -1.5761948
6 A2ML1   10.34228    2.474330   18.210239   7.35966452      2.8796400
          pval         padj      resVarA      resVarB
1 5.895285e-02 8.199051e-02 0.4520037072 0.9386538091
2 5.800018e-24 2.245696e-23 0.0006058216 0.0009230316
3 0.000000e+00 0.000000e+00 0.0018549887 0.2125819729
4 6.406008e-01 7.130292e-01 0.6958712267 0.1155807502
5 7.136553e-76 6.285770e-75 0.0478036558 0.1491655872
6 1.976859e-07 4.270617e-07 0.0011459467 0.0138454756
```

```
> counts.normalized = round(t(t(counts(cds))/sizeFactors(cds)), 2)
> write.table(cbind(results, counts.normalized), file="RNA-Seq_DESeq_output.txt",
+         sep="\t", quote=F, row.names=F)
```

We can browse all the output in `RNA-Seq_DESeq_output.txt`, given the following key. Unless indicated, all summary values are normalized (scaled to account for different numbers of reads in each sample), with groups labeled as A or B as indicated by the command `nbinomTest(cds, "A", "B")`

- id = Gene identifier
- baseMean = Mean counts across all samples
- baseMeanA = Mean counts across replicates of first sample
- baseMeanB = Mean counts across replicates of second sample
- foldChange = baseMeanB/baseMeanA
- log2FoldChange = log2 (foldChange)
- pval = raw p-value for statistical test asking, "Is baseMeanA = baseMeanB?"
- padj = False Discovery Rate correction for p-values (Use this when determining threshold)
- resVarA = measure of the variance of this gene in group A. If this number is very high, the hit may be a false positive.
- resVarB = [same for group A]
- counts for each each input sample (multiple columns; preprocessed and normalized)

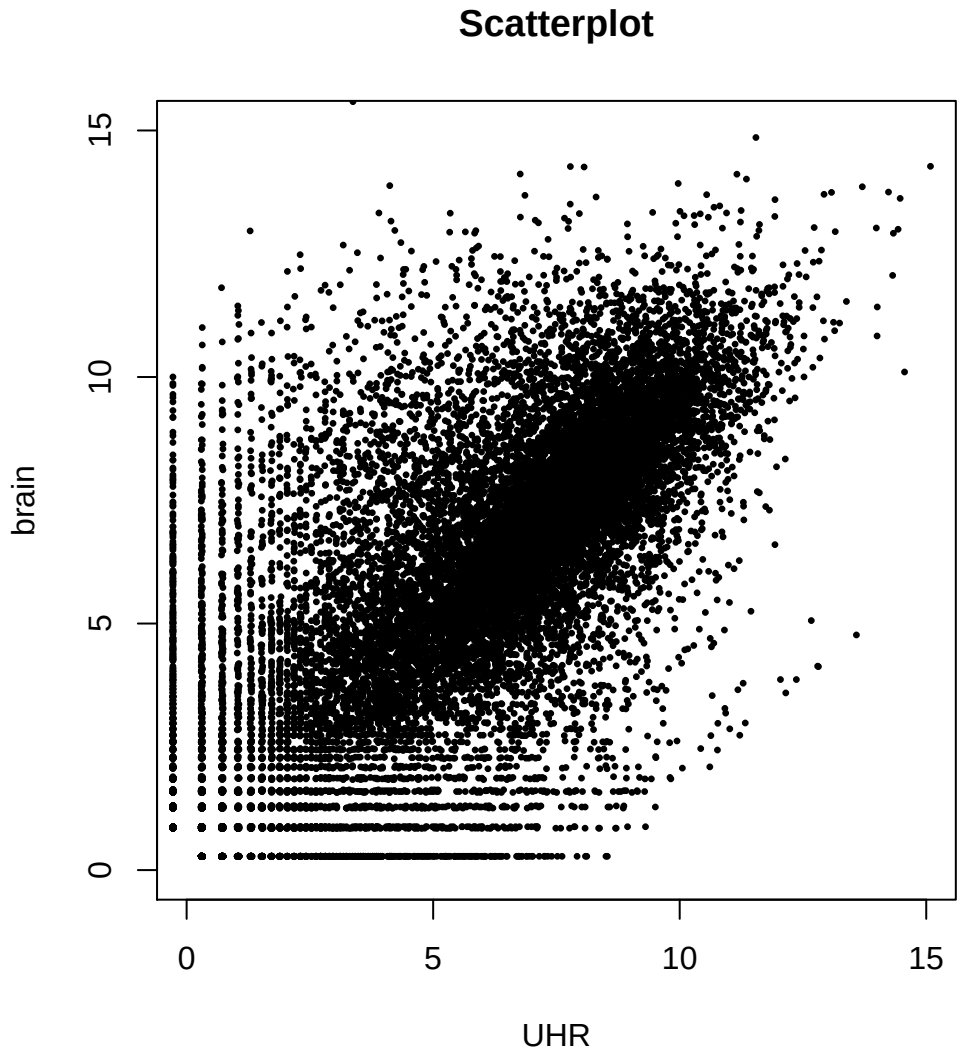
A few comments may help explain some of these results. First, the groups of brain and a human reference RNA are much more different than most groups that one would typically compare in an experiment. Second, the duplicate samples aren't biological replicates because they both came from the same commericial RNA source. Biological replicates would be expected to show more variability than we see here. For both reasons we see a huge number of genes that appear to be differentially expressed, which is not the usual result of a RNA-Seq experiment.

4.4 Creating figures from RNA-Seq analysis

We can draw figures that are similar to those we've done for microarrays.

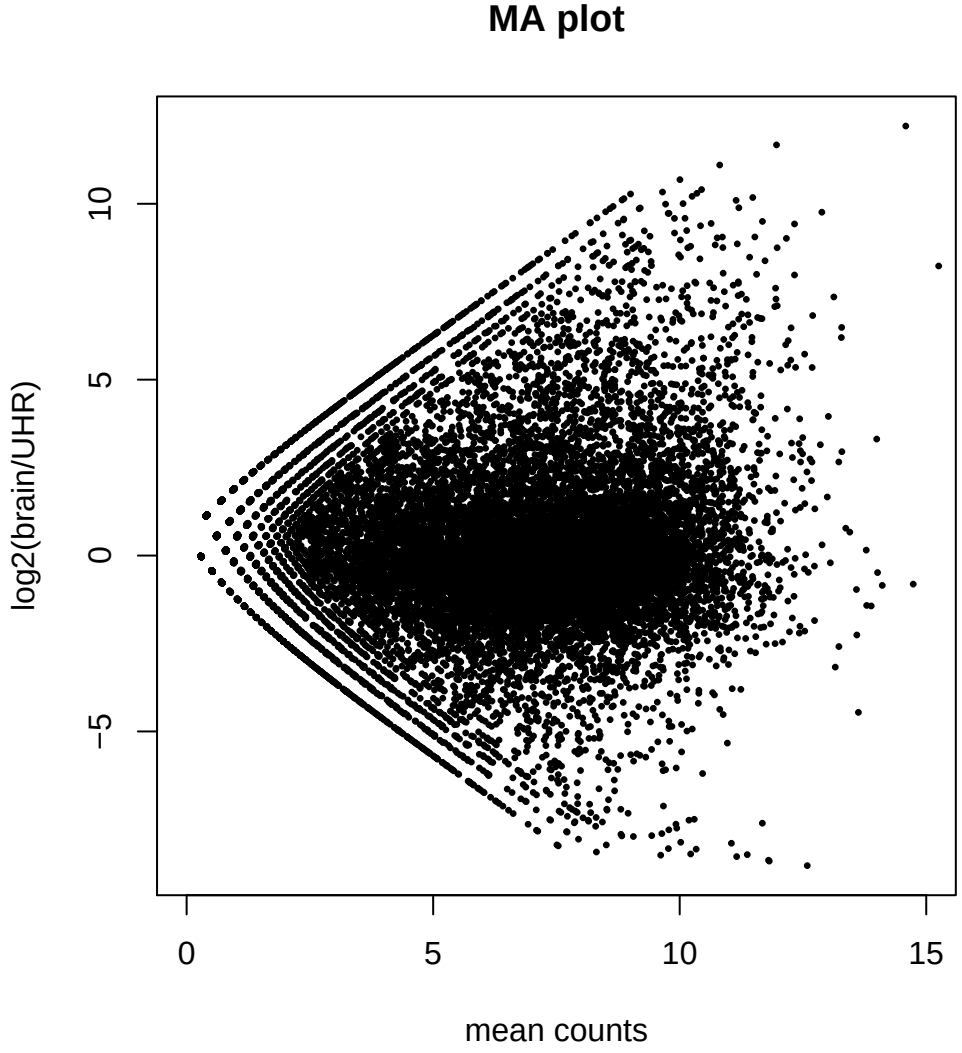
We have easy access to actual levels for each group so can do a typical scatterplot. For all of these figures, `par(pty="s")` forces the figure to be square, which makes the figures easier to interpret.

```
> par(pty="s")
> plot(log2(results$baseMeanA), log2(results$baseMeanB), main="Scatterplot",
+         xlim=c(0,15), ylim=c(0,15), xlab="UHR", ylab="brain", pch=20, cex=0.5)
```



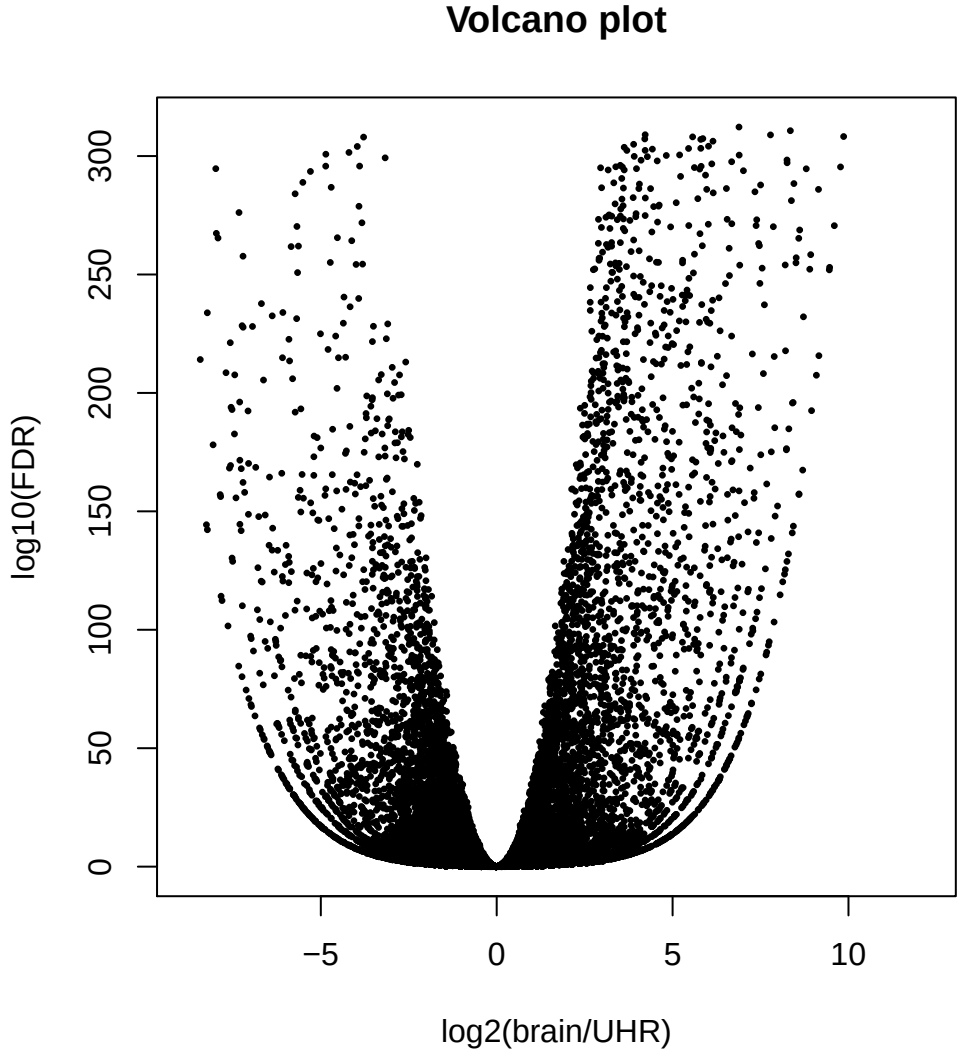
A MA plot (like the figure above, but rotated 45 degrees) is just as easy.

```
> par(pty="s")
> plot(log2(results$baseMean), results$log2FoldChange, main="MA plot",
+         ylab="log2(brain/UHR)", xlab="mean counts", pch=20, cex=0.5, xlim=c(0,15))
```



A volcano plot highlights the very low FDR p-values from this analysis.

```
> par(pty="s")
> plot(results$log2FoldChange, -log10(results$padj), main="Volcano plot",
+         xlab="log2(brain/UHR)", ylab="log10(FDR)", pch=20, cex=0.5)
```



The confidence associated with the actual FDR p-values is hard to believe, but the FDR rank of genes can help us choose which we think really are differentially expressed.