Lab 7: Analyzing Microarray Data: From Images to List of Candidate Genes

June 4, 2003

In this lab, we demonstrate how to use R and Bioconductor to 1) read-in Affymetrix cel files, 2) do some quality control checks, 3) pre-process the cel level data to obtain expression measures, 4) obtain scores for differential expression and cut-offs to create list, and finally 5) create user-friendly web pages to report results.

```
> library(Biobase, warn.conflicts = FALSE)
> library(affy, warn.conflicts = FALSE)

image is already generic, could be a problem.

> library(ctest)
> library(multtest)
> library(bioclabs)
> library(annotate)
> library(hgu95av2)
```

1 Read-In Cel Files

The function `ReadAffy` can be used to read in cel files and enter phenotypic data as well as MIAME information.

```
R> spikein <- ReadAffy(phenoData="phenodata.txt",description="miame.txt",
                       verbose=TRUE)
R> phenodata <- read.phenoData("phenodata.txt",check.names=FALSE)
R> phenoData(spikein) <- phenodata
```

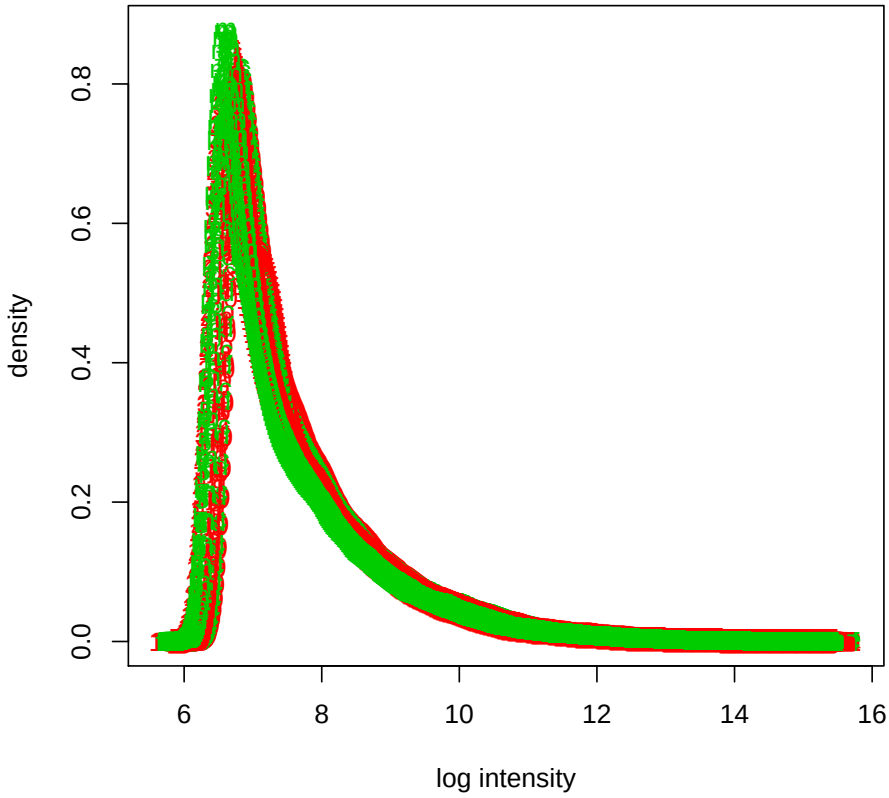
The data one would obtain by doing this is available from the `bioclabs` package.

```
> data("spikein")
```

2 Quality Control

Various functions exist in the `affy` package that can be used for quality control. Let's try a few...

```
> pops <- pData(spikein)[, 1] + 2
> hist(spikein, col = pops, type = "l")
```



The different colors represent the two different populations.
We can also use `image`, `boxplot`, among others...

3 Pre-processing

Now we need to convert the probe-level data into expression measures. Various methods are available through the function `expresso` and one can easily create a new one using the function `express`.

Now we will use the function `rma` which is implemented in C and is therefore quite fast.

```
> eset <- rma(spikein)

Background correcting
Normalizing
Calculating Expression
```

4 Differential Expression

In this section we will compute the average log ratio for between the two populations and the t-test as well. We will then obtain adjusted p-values and create a list with genes that are statistically significant.

First, notice there are two populations and 12 replicates in each.

```
> eset$population

 [1] 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

> Index1 <- which(eset$population == 0)
> Index2 <- which(eset$population == 1)
```

Let's get average intensities, average log ratios, t-tests, and p-values using the `t.test` function.

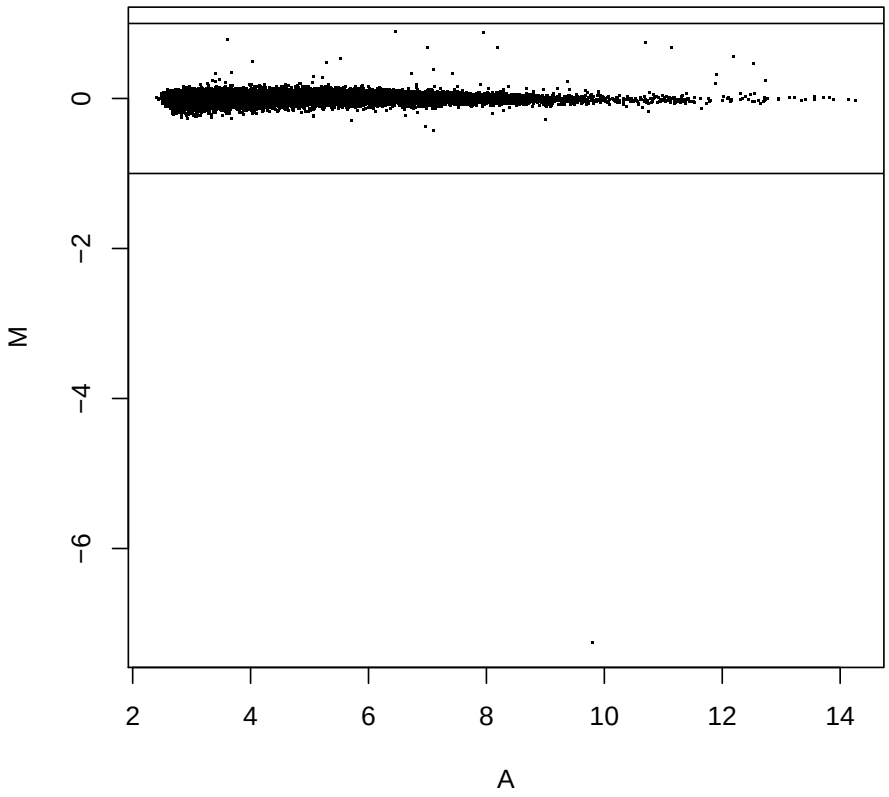
```
> scores <- esApply(eset, 1, function(x) {
+     tmp <- t.test(x[Index2], x[Index1], var.equal = TRUE)
+     c(mean(tmp$estimate), -diff(tmp$estimate), tmp$statistic,
+         tmp$p.value)
+ })
```

Now let's make the genes be in the rows and give appropriate names to the columns:

```
> scores <- t(scores)
> colnames(scores) <- c("a", "m", "t.test", "p.value")
```

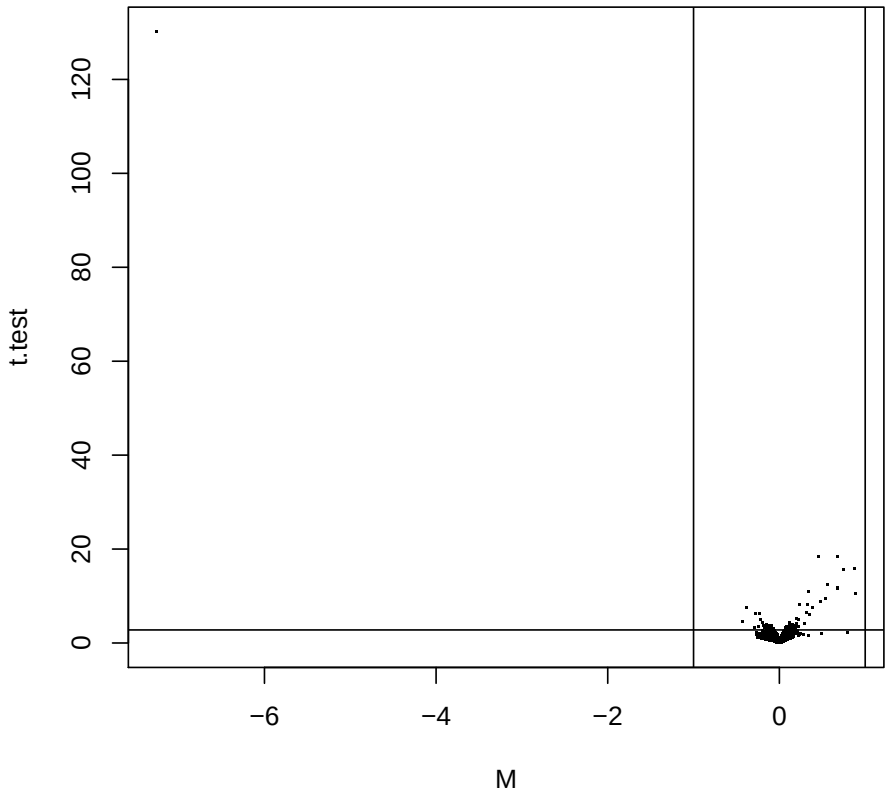
Now let's make an M (average log ratio) vs A (average intensity plot). The horizontal line shows the typical two-fold-change cutoff.

```
> plot(scores[, 1], scores[, 2], xlab = "A", ylab = "M", pch = ".")
> abline(h = c(-1, 1))
```



Should we take the variability of the estimates into account? It's only 3 replicates but we can try a t-test. The following is a so-called volcano plot which plots the t-test versus the estimate.

```
> plot(scores[, 2], abs(scores[, 3]), xlab = "M", ylab = "t.test",
+     pch = ".")
> abline(v = c(-1, 1))
> a <- qt(0.975, 4)
> abline(h = a)
```



How many genes have p-values less than 0.05? How about 0.01?

```
> sum(scores[, 4] <= 0.05)
```

```
[1] 489
```

```
> sum(scores[, 4] <= 0.01)
```

```
[1] 126
```

Maybe we should adjust the p-values. Let's use the `multtest` package to obtain adjusted p-values using Benjamini and Yekutieli's procedures for (strong) control of the false discovery rate (FDR).

The function `mt.rawp2adjp` gives adjusted p-values according to various methods using only the raw p-values.

```
> tmp <- mt.rawp2adjp(scores[, 4], proc = "BH")
> adj.p.values <- tmp$adjp[order(tmp$index), ]
> scores <- cbind(scores, adj.p.values[, -1])
> colnames(scores)[5] <- "FDR"
```

This assumes that the t-test are actually t-distributed. If we had more time we could try a non-parametric method such as maxT using the function `mt.maxT`.

At what FDR would we be happy? 0.01 is pretty conservative. Let's try it anyway. In the next section we will make a

5 Creating Report

First let's pick the AffyIDs corresponding to the genes with adjusted p-values of less than 0.01.

```
> Names <- geneNames(eset)[scores[, 5] <= 0.01]
> Names <- Names[order(scores[Names, 5])]
```

Now using the data available through the metadata package `hgu95av2` lets find the corresponding gene names and locus link IDs.

```
> ll <- multiget(Names, env = hgu95av2LOCUSID)
> sym <- multiget(Names, env = hgu95av2SYMBOL)
```

We can now make a nice web-page

```
> res <- data.frame(unlist(sym), signif(scores[Names, ], 2))
> ll.htmlpage(ll, filename = "report.html", title = "HTML report",
+     othernames = res, table.head = c("Locus ID", "Gene Symbol",
+         colnames(scores)), table.center = TRUE)
```

Use a web browser to look at this page.

How we faired with the known to be differentially expressed genes? Of the genes we called how many where actually differential expressed?

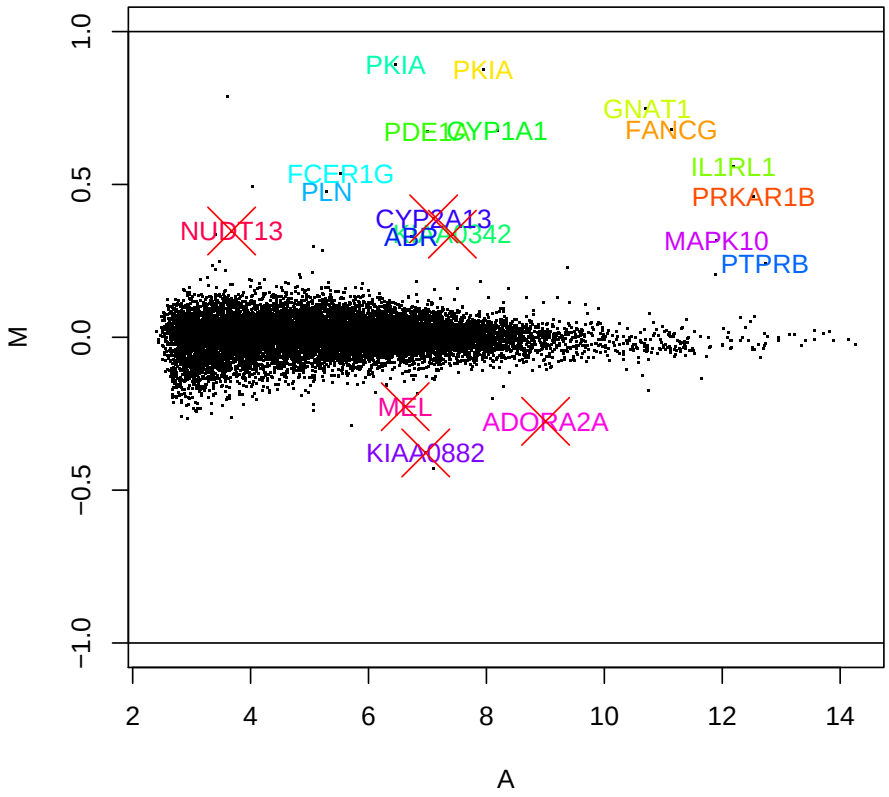
```
> true <- colnames(pData(eset))[-1]
> tp <- sum(Names %in% true)
> cat(tp, "true positives", length(Names) - tp, "false positives.\n")
```

```
14 true positives 6 false positives.
```

Not bad... but not 0.01 FDR either.

Let's make an MVA plot with the gene names.

```
> plot(scores[, 1], scores[, 2], xlab = "A", ylab = "M", pch = ".",
+     ylim = c(-1, 1))
> text(scores[Names, 1], scores[Names, 2], sym, pch = 16, col = rainbow(length(Names)
> abline(h = c(-1, 1))
> fp <- Names[!Names %in% true]
> points(scores[fp, 1], scores[fp, 2], pch = 4, cex = 4, col = "red")
```



Was it worth using a t-tests over the more simple fold change estimates?
Let's see which one does better at ranking the truly differentially expressed genes:

```
> m.ranks <- rank(-abs(scores[, 2]))
> names(m.ranks) <- rownames(scores)
> t.ranks <- rank(-abs(scores[, 3]))
> names(t.ranks) <- rownames(scores)
> cbind(m.ranks, t.ranks)[true, ]
```

```
         m.ranks t.ranks
37777_at      33      14
684_at         1       1
1597_at       97     117
38734_at      12      12
39058_at      20      13
36311_at       8       7
36889_at      10      11
1024_at        7       8
```

36202_at	3	4
36085_at	5	5
40322_at	9	6
407_at	56	21
1091_at	13	2
1708_at	21	17
33818_at	6	3
546_at	2	10