GS01 0163
Analysis of Microarray Data

Keith Baggerly and Kevin Coombes
Department of Bioinformatics & Computational Biology
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org
kcoombes@mdanderson.org

27 November 2007

Lecture 25: Linear Models for Two-Color Microarrays

- XML is your friend
- Converting to limma
- Fitting data with a linear model
- Making tables
- Comments on Replication

XML is Your Friend

Last time, we started looking at six two-color glass arrays that had been selected from a larger study. The data from the full study is available at GEO in two formats: SOFT and MINiML.

We'd like to look at the larger data, so we'll grab one of these from GEO.

We're going to focus on the MINiML format, which we find somewhat easier to work with. This supplies data as a gzipped tarball, which uncompresses (typically) into 1 file/array of quantifications, 1/(array type) of gene identifiers, and one overview table in XML.

To understand what we have, we need to explore R's XML library. (This year, a straight install works.)

Creating the XML parse tree

We begin by trying to figure out how to parse the XML file that describes all of this data. First we load the library and create the parse tree.

```
> library(XML)
> xmlsource <- file.path("GSE1039", "GSE1039_family2.xml
> mytree <- xmlTreeParse(xmlsource)
> rm(xmlsource)
```


The Root Node

The root of the tree identifies this as a MINiML document, which contains 35 child nodes.

```
> root <- xmlRoot(mytree)
> xmlName(root)

[1] "MINiML"

> xmlSize(root)

[1] 35
```


A Child Node

The first child node describes one of the contributors (Michael E. Salazar) of the data set.

```
> root[[1]]
```

```
<Contributor iid="contrib1">
 <Person>
  <First>Michael</First>
  <Middle>E.</Middle>
  <Last>Salazar</Last>
 </Person>
 <Phone>(415) 514-4371</Phone>
 <Laboratory>Functional Genomics Core Laboratories</Labora
 <Organization>University of California, San Francisco</Or
```



```
 <Address>
  <Line>1550 Fourth Street, RM 545</Line>
  <City>San Francisco</City>
  <State>CA</State>
  <Zip-Code>94158</Zip-Code>
  <Country>USA</Country>
 </Address>
 <Web-Link>arrays.ucsf.edu</Web-Link>
</Contributor>
```

Kinds of Child Nodes

We can easily determine the type of all the nodes. Note that there are only five kinds of nodes.

```
> as.character(xmlSApply(root, xmlName))
```

```
 [1] "Contributor" "Contributor" "Contributor"
 [4] "Contributor" "Database"    "Platform"
 [7] "Platform"    "Sample"      "Sample"
[10] "Sample"      "Sample"      "Sample"
[13] "Sample"      "Sample"      "Sample"
[16] "Sample"      "Sample"      "Sample"
[19] "Sample"      "Sample"      "Sample"
[22] "Sample"      "Sample"      "Sample"
[25] "Sample"      "Sample"      "Sample"
```



```
[28] "Sample"       "Sample"       "Sample"
[31] "Sample"       "Sample"       "Sample"
[34] "Sample"       "Series"
```


A Platform Node

The important objects for analyzing the data are Platforms and Samples. Both of these contain references to "external data", which is to say, those files we read in above. We start by looking more closely at one of the platforms.

```
> plat1 <- root[[6]]
> as.character(xmlSApply(plat1, xmlName))
```

```
 [1] "Status"                "Title"
 [3] "Accession"             "Technology"
 [5] "Distribution"          "Organism"
 [7] "Manufacturer"          "Manufacture-Protocol"
 [9] "Description"           "Contributor-Ref"
[11] "Contributor-Ref"       "Contact-Ref"
```



```
[13] "Data-Table"
```

What Kind of Platform?

From the accession and the title, we see that the GEO identifier GPL976 represents the UCSF version 4 human oligo array.

```
> plat1["Accession"][[1]]
```

```
<Accession database="GEO">GPL976</Accession>
```

```
> plat1["Title"][[1]]
```

```
<Title>UCSF 4Hs Human v.2 Oligo Array</Title>
```


Data Tables as External Data

At this point, we do not really care about most of the platform information, but we do care about the data table. As you can see, this contains a lot of "Column" information and an "External-Data" entry. The external data gives a file reference that will not be immediately useful to us, since the path describes where the data is stored at the GEO web site. The actual file name (but not the directories) is the same as the name of the corresponding file that we extracted from the tarball.

```
> plat1DT <- plat1["Data-Table"][[1]]
> as.character(xmlSApply(plat1DT, xmlName))
```

```
 [1] "Column"          "Column"          "Column"
 [4] "Column"          "Column"          "Column"
```



```
 [7] "Column"          "Column"          "Column"
[10] "Column"          "Column"          "External-Data"
```

Column Header Information is in the XML File

```
> plat1DT[[5]]

<Column position="5">
 <Name>Operon_ID</Name>
 <Description>Operon assigned Oligo ID</Description>
</Column>

> xmlValue(plat1DT["External-Data"][[1]])

[1] "/am/ftp-geo/DATA/MINiML/by_series/GSE1039/GPL976-tbl-
```

A Sample Node

Samples have a similar structure:

```
> samp1 <- root[[8]]
> as.character(xmlSApply(samp1, xmlName))
```

```
 [1] "Status"                "Title"
 [3] "Accession"             "Type"
 [5] "Channel-Count"         "Channel"
 [7] "Channel"               "Description"
 [9] "Data-Processing"       "Platform-Ref"
[11] "Contact-Ref"           "Supplementary-Data"
[13] "Data-Table"
```


A Sample Node

```
> samp1["Accession"][[1]]

<Accession database="GEO">GSM16665</Accession>

> samp1["Title"][[1]]

<Title>Hs_004_187_2</Title>
```


Extracting Information from the Sample Nodes

Next, we create an object that identifies which of the child nodes of the root object represent samples. We can use this to immediately extract interesting sample information.

```
> idxSamp <- which(xmlSApply(root, xmlName) ==
+     "Sample")
> ArrayID <- sapply(idxSamp, function(x) {
+     xmlValue(root[[x]]["Title"][[1]])
+ })
> GSMID <- sapply(idxSamp, function(x) {
+     xmlValue(root[[x]]["Accession"][[1]])
+ })
> PlatformID <- sapply(idxSamp, function(x) {
+     xmlAttrs(root[[x]]["Platform-Ref"][[1]])
```



```
+ })
> data.frame(GSMID, ArrayID, PlatformID)[1:10,
+     ]
```

	GSMID	ArrayID	PlatformID
1	GSM16665	Hs_004_187_2	GPL976
2	GSM16675	Hs_004_186_2	GPL976
3	GSM16679	Hs_004_235	GPL976
4	GSM16680	Hs_004_189_1	GPL976
5	GSM16681	Hs_004_188	GPL976
6	GSM16685	6Hs.094	GPL978
7	GSM16686	6Hs.195.1	GPL978
8	GSM16687	6Hs.168	GPL978
9	GSM16688	6Hs.169.1	GPL978
10	GSM16689	6Hs.166	GPL978

Sample Descriptions Contain Important Unstructured Data

Now we take a look at the "Description" field for one of the samples.

```
> xmlValue(samp1["Description"][[1]])
```

```
"Total RNA from beta7 cells isolated from human PBMC
(CD4+ and CD45RA- population)\nPatient sex: female;
Patient age: 30\nSubject ID# 001\n2 rounds amplification
(Ambion's MessageAmp aRNA Kit)"
```

In this case, the contributors have structured the "Description" of each sample so that it contains potentially useful information such as the gender, age, and ID of each patient. The design of MINiML does not give a structured way to include patient covariates. Nevertheless, we can still extract the relevant information from the descriptions.


```
> d <- sapply(idxSamp, function(x) {
+     xmlValue(root[[x]]["Description"][[1]])
+ })
> click <- regexpr("sex: ", d)
> clack <- regexpr("male; ", d)
> Gender <- substring(d, click + 5, clack + 3)
> click <- regexpr("Patient age: ", d)
> Age <- as.numeric(substring(d, click + 13,
+     click + 15))
> click <- regexpr("Subject ID# ", d)
> Subject <- paste("S", substring(d, click +
+     12, click + 14), sep = "")
> rm(click, clack)
```


Reviewing the Patient Characteristics by Array

```
> data.frame(PlatformID, Subject, Age, Gender)
```

	PlatformID	Subject	Age	Gender
1	GPL976	S001	30	female
2	GPL976	S001	30	female
3	GPL976	S006	27	female
4	GPL976	S009	23	female
5	GPL976	S009	23	female
6	GPL978	S001	30	female
7	GPL978	S001	30	female
8	GPL978	S003	25	female
9	GPL978	S003	25	female
10	GPL978	S004	37	female
11	GPL978	S004	37	female


```
12      GPL978    S006  27 female
13      GPL978    S006  27 female
14      GPL978    S006  27 female
15      GPL978    S006  27 female
16      GPL978    S007  45   male
17      GPL978    S007  45   male
18      GPL978    S007  45   male
19      GPL978    S007  45   male
20      GPL978    S008  28   male
21      GPL978    S008  28   male
22      GPL978    S010  31 female
23      GPL978    S010  31 female
24      GPL978    S011  28 female
25      GPL978    S011  28 female
26      GPL978    S011  28 female
```



```
27      GPL978     S011  28 female
```

Why Do We Need Both Platforms?

We see that samples S001 and S006 were run on both platforms; sample S009 was only run on the older platform, and all the other samples were only run on the newer platform. Thus, we expect eventually to only use the data from the newer platform for our analysis.

Channel Information

We also know that these were two color experiments, and that the $\beta 7+$ and $\beta 7-$ cell populations from the same individual were used in the two channels of a single slide. So, we still need to extract this channel information. Looking to future applications, we have tried to write this code so it applies to different numbers of channels.

```
> idxChan <- which(xmlSApply(samp1, xmlName) ==
+     "Channel")
> for (i in 1:length(idxChan)) {
+     assign(paste("Source", i, sep = ""), sapply(idxSamp
+         function(x, i) {
+             xmlValue(root[[x]][[idxChan[i]]]["Source"]
+         }, i))
+ }
```



```
> rm(i)
> data.frame(Subject, Source1, Source2)

   Subject Source1 Source2
1     S001  beta7-  beta7+
2     S001  beta7+  beta7-
3     S006  beta7-  beta7+
4     S009  beta7-  beta7+
5     S009  beta7+  beta7-
6     S001  beta7-  beta7+
7     S001  beta7-  beta7+
8     S003  beta7+  beta7-
9     S003  beta7-  beta7+
10    S004  beta7+  beta7-
11    S004  beta7-  beta7+
12    S006  beta7-  beta7+
```



```
13    S006  beta7+  beta7-
14    S006  beta7+  beta7-
15    S006  beta7-  beta7+
16    S007  beta7-  beta7+
17    S007  beta7+  beta7-
18    S007  beta7+  beta7-
19    S007  beta7-  beta7+
20    S008  beta7+  beta7-
21    S008  beta7-  beta7+
22    S010  beta7+  beta7-
23    S010  beta7-  beta7+
24    S011  beta7+  beta7-
25    S011  beta7-  beta7+
26    S011  beta7+  beta7-
27    S011  beta7-  beta7+
```

Which Files Contain the Data?

The final information that we need for each sample is the name of the external file that contains the data. As mentioned above, this is the last component of the value of the "External-Data" entry in the "Data-Table". So, we can extract that piece with the following code.

```
> dt <- sapply(idxSamp, function(x) {
+     root[[x]]["Data-Table"]
+ })
> File <- sapply(dt, function(x) {
+     y <- x["External-Data"][[1]]
+     z <- xmlValue(y)
+     w <- strsplit(z, "/")[[1]]
+     w[length(w)]
+ })
```


The File Mapping is Straightforward

```
> data.frame(Subject, GSMID, File)[1:9, ]
```

	Subject	GSMID	File
1	S001	GSM16665	GSM16665-tbl-1.txt
2	S001	GSM16675	GSM16675-tbl-1.txt
3	S006	GSM16679	GSM16679-tbl-1.txt
4	S009	GSM16680	GSM16680-tbl-1.txt
5	S009	GSM16681	GSM16681-tbl-1.txt
6	S001	GSM16685	GSM16685-tbl-1.txt
7	S001	GSM16686	GSM16686-tbl-1.txt
8	S003	GSM16687	GSM16687-tbl-1.txt
9	S003	GSM16688	GSM16688-tbl-1.txt

Putting the Sample Information Together

Next, we assemble all the information into a single data frame, and throw away the copies of the individual pieces that we no longer need.

```
> sampleInfo <- data.frame(File, ArrayID, PlatformID,
+     GSMID, Subject, Age, Gender, Source1, Source2)
> rm(File, ArrayID, PlatformID, GSMID, Age, Gender,
+     Subject, Source1, Source2)
```


The Data Table

It is now time to look at the rest of the data table. As with the platform example above, the data table for each sample contains a bunch of column entries and one external data reference. Here is what a column entry looks like.

```
> dt[[1]][[1]]

<Column position="1">
 <Name>ID_REF</Name>
 <Description>the unique identifier of the feature derive
</Column>
```


Column Entries in the XML File

As you can see, the column entry contains an integer attribute that defines the position of the column, a "Name" that can be used as the column header, and an (optional) "Description" that allows us to interpret the column. The next block of code extracts the column names for each sample.

```
> sampColNames <- sapply(dt, function(dt1) {
+     idxCol <- which(xmlSApply(dt1, xmlName) ==
+         "Column")
+     posn <- as.numeric(xmlSApply(dt1, xmlAttrs)[idxCol]
+     if (any(diff(posn) != 1))
+         stop("missing column names")
+     name <- as.character(xmlSApply(dt1, function(x) {
+         xmlValue(x[[1]])
```



```
+     })[idxCol])
+     if (any(is.na(name)))
+         stop("empty column name")
+     name
+ })
> as.integer(sapply(sampColNames, length))

 [1] 78 78 78 78 78 44 44 44 44 44 44 44 44 44 44 44 44
[18] 44 44 44 44 44 44 44 44 44 44
```

Notice that the tables have different numbers of columns, corresponding exactly to the two different platforms that were used for the experiments.

Reading the Actual Data

Now we can use the file names and column header information to read in the external data (i.e., the actual quantifications). This entire operation took about three minutes on my computer, which is several orders of magnitude faster than processing the SOFT files of the same data set with GEOquery.


```
> home <- "GSE1039"
> for (i in 1:nrow(sampleInfo)) {
+     fn <- file.path(home, as.character(sampleInfo[i,
+         "File"]))
+     vn <- as.character(sampleInfo[i, "GSMID"])
+     print(paste("Reading", vn, "from", fn))
+     temp <- read.table(fn, header = FALSE,
+         sep = "\t", quote = "", comment.char = "")
+     colnames(temp) <- sampColNames[[i]]
+     assign(vn, temp)
+ }
```

```
[1] "Reading GSM16665 from GSE1039/GSM16665-tbl-1.txt"
[1] "Reading GSM16675 from GSE1039/GSM16675-tbl-1.txt"
[1] "Reading GSM16679 from GSE1039/GSM16679-tbl-1.txt"
```

```
[1] "Reading GSM16680 from GSE1039/GSM16680-tbl-1.txt"
[1] "Reading GSM16681 from GSE1039/GSM16681-tbl-1.txt"
[1] "Reading GSM16685 from GSE1039/GSM16685-tbl-1.txt"
[1] "Reading GSM16686 from GSE1039/GSM16686-tbl-1.txt"
[1] "Reading GSM16687 from GSE1039/GSM16687-tbl-1.txt"
[1] "Reading GSM16688 from GSE1039/GSM16688-tbl-1.txt"
[1] "Reading GSM16689 from GSE1039/GSM16689-tbl-1.txt"
[1] "Reading GSM16690 from GSE1039/GSM16690-tbl-1.txt"
[1] "Reading GSM16691 from GSE1039/GSM16691-tbl-1.txt"
[1] "Reading GSM16692 from GSE1039/GSM16692-tbl-1.txt"
[1] "Reading GSM16693 from GSE1039/GSM16693-tbl-1.txt"
[1] "Reading GSM16694 from GSE1039/GSM16694-tbl-1.txt"
[1] "Reading GSM16695 from GSE1039/GSM16695-tbl-1.txt"
[1] "Reading GSM16699 from GSE1039/GSM16699-tbl-1.txt"
[1] "Reading GSM16700 from GSE1039/GSM16700-tbl-1.txt"
```



```
[1] "Reading GSM16704 from GSE1039/GSM16704-tbl-1.txt"
[1] "Reading GSM16705 from GSE1039/GSM16705-tbl-1.txt"
[1] "Reading GSM16706 from GSE1039/GSM16706-tbl-1.txt"
[1] "Reading GSM16719 from GSE1039/GSM16719-tbl-1.txt"
[1] "Reading GSM16720 from GSE1039/GSM16720-tbl-1.txt"
[1] "Reading GSM16724 from GSE1039/GSM16724-tbl-1.txt"
[1] "Reading GSM16725 from GSE1039/GSM16725-tbl-1.txt"
[1] "Reading GSM16726 from GSE1039/GSM16726-tbl-1.txt"
[1] "Reading GSM16727 from GSE1039/GSM16727-tbl-1.txt"
```

```
> rm(i, vn, fn, temp)
```

```
> rm(d, dt, idxSamp, idxChan, samp1, sampColNames,
+     plat1, plat1DT)
```


Parsing the Platform Information

The next block of code extracts the information describing the two platforms.

```
> idxPlat <- which(xmlSApply(root, xmlName) ==
+     "Platform")
> Title <- sapply(idxPlat, function(x) {
+     xmlValue(root[[x]]["Title"][[1]])
+ })
> GSMID <- sapply(idxPlat, function(x) {
+     xmlValue(root[[x]]["Accession"][[1]])
+ })
> dt <- sapply(idxPlat, function(x) {
+     root[[x]]["Data-Table"]
+ })
```



```
> File <- sapply(dt, function(x) {
+     y <- x["External-Data"][[1]]
+     z <- xmlValue(y)
+     w <- strsplit(z, "/")[[1]]
+     w[length(w)]
+ })
> platformInfo <- data.frame(GSMID, Title, File)
> rm(File, Title, GSMID)
> platformInfo
```

```
   GSMID                                  Title                File
1 GPL976 UCSF 4Hs Human v.2 Oligo Array GPL976-tbl-1.txt
2 GPL978 UCSF 6Hs Human v.2 Oligo Array GPL978-tbl-1.txt
```

We now get the corresponding column names, and then read the external files containing the gene information for each platform.


```
> platColNames <- lapply(dt, function(dt1) {
+     idxCol <- which(xmlSApply(dt1, xmlName) ==
+         "Column")
+     posn <- as.numeric(xmlSApply(dt1, xmlAttrs)[idxCol]
+     if (any(diff(posn) != 1))
+         stop("missing column names")
+     name <- as.character(xmlSApply(dt1, function(x) {
+         xmlValue(x[[1]])
+     })[idxCol])
+     if (any(is.na(name)))
+         stop("empty column name")
+     name
+ })
```



```
> for (i in 1:nrow(platformInfo)) {
+     fn <- file.path(home, as.character(platformInfo[i,
+         "File"]))
+     vn <- as.character(platformInfo[i, "GSMID"])
+     print(paste("Reading", vn, "from", fn))
+     temp <- read.table(fn, header = FALSE,
+         sep = "\t", quote = "", comment.char = "")
+     colnames(temp) <- platColNames[[i]]
+     assign(vn, temp)
+ }

[1] "Reading GPL976 from GSE1039/GPL976-tbl-1.txt"
[1] "Reading GPL978 from GSE1039/GPL978-tbl-1.txt"

> rm(i, vn, fn, temp)
```

```
> rm(dt, idxPlat, platColNames)
```

Converting to limma

We now have to convert the raw files (which have been loaded into R) into the `RGList` data structure needed by the `limma` package. We start by loading the package.

```
> require(limma)
```

```
[1] TRUE
```

```
> library(marray)
```


CrossCut: A Utility Function

Because the data structures used by `limma` cut across the raw files produced by the quantifications, we are going to write a function that extracts them based on the column name.

```
> crossCut <- function(sampleNames, columnName) {
+     temp <- lapply(sampleNames, function(x,
+         cn) {
+         data <- eval(as.name(x))
+         data[, cn]
+     }, columnName)
+     temp <- as.matrix(as.data.frame(temp))
+     colnames(temp) <- sampleNames
+     temp
+ }
```


The Four Basic Limma Components

The GEO web page for this data set asserts (indirectly) that channel 1 = F635 = Cy5 = Red. You can also get this information from the descriptions of the columns in the sample data table entries. We need that information to make sure we later match "R" with "Source1".

```
> isGPL978 <- sampleInfo[, "PlatformID"] == "GPL978"
> si <- as.character(sampleInfo[isGPL978, "GSMID"])
> R <- crossCut(si, "F635 Mean")
> G <- crossCut(si, "F532 Mean")
> Rb <- crossCut(si, "B635 Median")
> Gb <- crossCut(si, "B532 Median")
```


Limma Lets Us Keep Other Components in a List

```
> other <- list(FPixels = crossCut(si, "F Pixels"),
+     BPixels = crossCut(si, "B Pixels"), Rsd = crossCut
+         "F635 SD"), Gsd = crossCut(si, "F532 SD"),
+     Rbsd = crossCut(si, "B635 SD"), Ggsd = crossCut(si
+         "B532 SD"))
> myData <- new("RGList", list(R = R, G = G,
+     Rb = Rb, Gb = Gb, other = other, genes = GPL978,
+     printer = getLayout(GPL978)))
```

We like to clean up after ourselves:

```
> rm(R, Rb, G, Gb, other, si)
> rmlist <- c(as.character(sampleInfo[, "GSMID"]),
```

```
+      as.character(platformInfo[, "GSMID"]))
> rm(list = rmlist)
```

Normalizing in Limma

The default for normalizing data in limma is to use print-tip loess normalization on each array separately. By default, this function also performs local background subtraction before normalizing.

```
> normData <- normalizeWithinArrays(myData)
```

Fitting Data with a Linear Model

```
> b7 <- rep(1, sum(isGPL978))
> b7[sampleInfo[isGPL978, "Source1"] == "beta7-"] <- -1
> LMres <- lmFit(normData, design = b7)
```

Okay, what did we just do?

`lmFit` is the main workhorse function of the `limma` package, and it fits LInear Models to MicroArrays. But what is a linear model?

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \epsilon$$

Inside LMres

What numbers are being played with here?

```
> class(LMres)
```

```
[1] "MArrayLM"
attr(,"package")
[1] "limma"
```

```
> slotNames(LMres)
```

```
[1] ".Data"
```

Surprise! While this is an object of type "MArrayLM", its contents are contained in a simple data frame. So, what things do we have here?

Fitted Numbers

```
> length(LMres)

[1] 23184

> LMres[1, ]

An object of class "MArrayLM"
$coefficients
            [,1]
[1,] -0.02296491

$stdev.unscaled
          [,1]
```



```
[1,] 0.2294157

$sigma
[1] 0.7546736

$df.residual
[1] 18

$cov.coefficients
           [,1]
[1,] 0.04545455

$pivot
[1] 1
```



```
$genes
  ID Block Column Row  Operon_ID    GB_ACC Unigene
1  1     1      1   1 H200000297 NM_002557    1154
                                        Description
1 oviductal glycoprotein 1, 120kDa (mucin 9, oviductin)
  Symbol
1  OVGP1

1 AAAGGTGACTGTCCCCTCCAGAAACATATCAGTCACCCCTGAAGGGCAGACTATGC
  SPOT_ID
1

$Amean
[1] 5.371024
```



```
$method
[1] "ls"

$design
[1] -1 -1  1 -1  1
17 more rows ...

$contrasts
     [,1]
[1,]    1
```


Behind the Curtain, 1

```
> normData$M[1, 1:12]
```

```
   GSM16685    GSM16686    GSM16687    GSM16688
         NA -0.23798421  0.15978895  0.04970296
   GSM16689    GSM16690    GSM16691    GSM16692
 0.21717200  0.02243622  1.43137206  1.11211550
   GSM16693    GSM16694    GSM16695    GSM16699
 1.17259869 -0.23260778  1.39032408          NA
```

These are (some of) the log ratios that we have available for this gene.


```
> mean(normData$M[1, ], na.rm = TRUE)

[1] 0.325289

> mean(normData$M[1, ] * LMres[1, ]$design, na.rm = TRUE)

[1] -0.02296491
```

This is the coeff value!

So, why are there NAs in the `M` field? Why do we need to multiply by the design vector? What do the 1's and -1's indicate?

Behind the Curtain, 2

This is "sigma":

```
> sqrt(var(normData$M[1, ] * LMres[1, ]$design,
+     na.rm = T))

          [,1]
[1,] 0.7546736
```

This is the number of valid reads:

```
> 1/LMres[1, ]$stdev.unscaled^2

     [,1]
[1,]   19
```


The `lmFit` call is summarizing the individual `M` values in order to highlight a specified contrast. By changing the design matrix, different contrasts can be seen, and tested for.

Testing Significance

```
> LMresEB <- eBayes(LMres)
```

So, what do we get? (long list)

```
> slotNames(LMresEB)
```

```
[1] ".Data"
```

```
> summary(LMresEB)
```

```
                 Length Class      Mode
coefficients     23184  -none-     numeric
stdev.unscaled   23184  -none-     numeric
```

```
sigma                23184  -none-      numeric
df.residual          23184  -none-      numeric
cov.coefficients         1  -none-      numeric
pivot                    1  -none-      numeric
genes                   11  data.frame  list
Amean                23184  -none-      numeric
method                   1  -none-      character
design                  22  -none-      numeric
df.prior                 1  -none-      numeric
s2.prior                 1  -none-      numeric
var.prior                1  -none-      numeric
proportion               1  -none-      numeric
s2.post              23184  -none-      numeric
t                    23184  -none-      numeric
p.value              23184  -none-      numeric
```



```
lods          23184  -none-     numeric
F             23184  -none-     numeric
F.p.value     23184  -none-     numeric
```


Making Tables

At this point, we have some test statistic values and associated p-values. The test-stat values were computed by borrowing strength across the genes available on the array to get more stable estimates of "null variation", so we have "moderated" t-tests as opposed to the plain vanilla variety.

Still, given these, we would like to extract a small number of them and report them in a fairly illustrative fashion.

```
> shortTable <- topTable(LMresEB, number = 10,
+     resort.by = "M")
```

What Do We Get?

```
> shortTable[9:10, ]
```

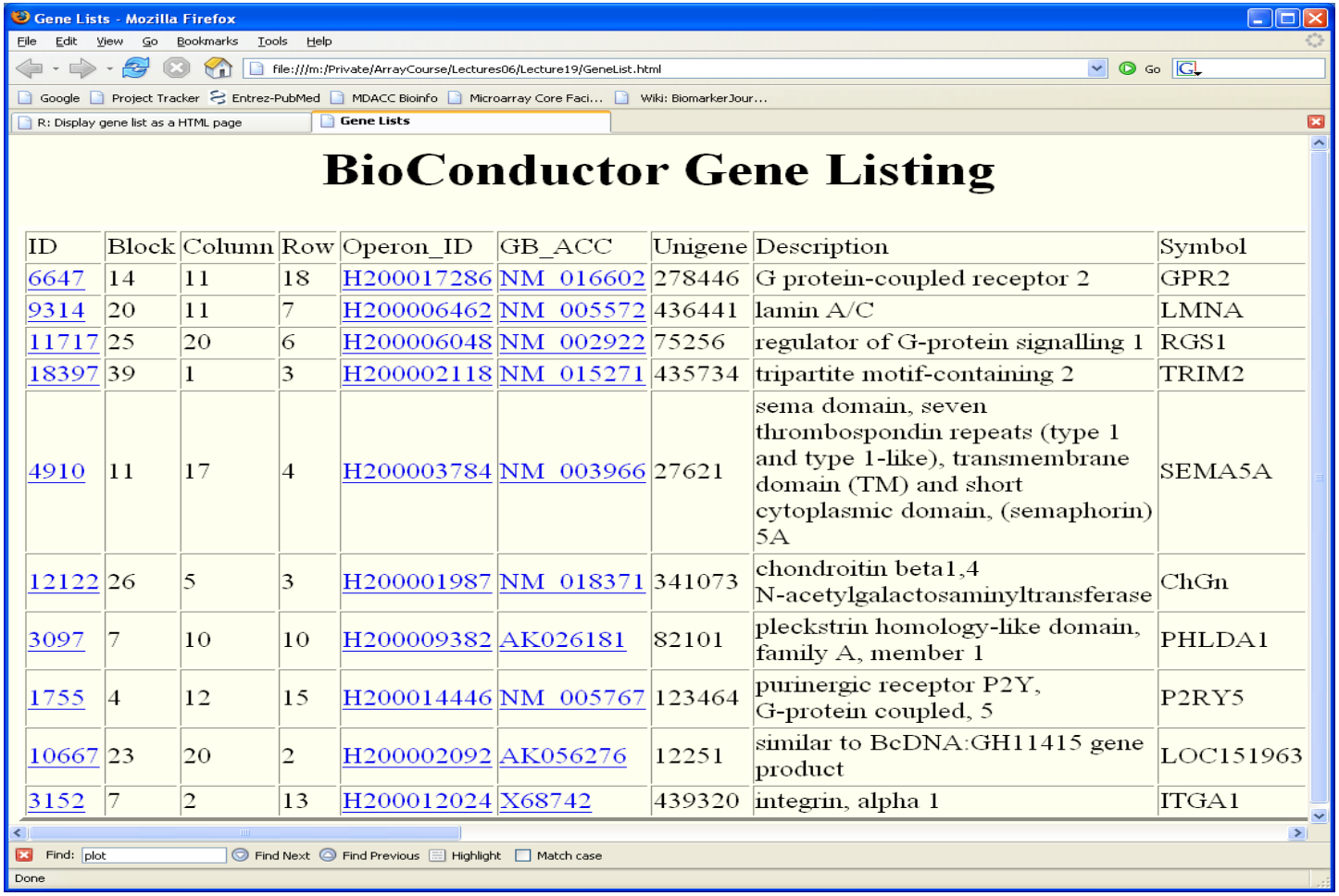
```
       ID Block Column Row  Operon_ID    GB_ACC Unigene
1755 1755     4     12  15 H200014446 NM_005767  123464
3152 3152     7      2  13 H200012024    X68742  439320
                                      Description
1755 purinergic receptor P2Y, G-protein coupled, 5
3152                              integrin, alpha 1
     Symbol
1755  P2RY5
3152  ITGA1

1755 TGCACCCGCCGTTTTTGTTCAGTCTACCCACTCTCAGGGTAACAATGCCTCAC
3152 GAGCTTGCTATTCAAATATCCAAAGATGGGCTACCGGGCAGAGTGCCATTATC
```

```
      SPOT_ID      logFC   AveExpr         t      P.Value
1755          0.8956644 11.966292  13.15216 2.275701e-12
3152         -1.1016176  6.934712 -12.02233 1.460089e-11
        adj.P.Val        B
1755 2.637992e-08 17.04423
3152 7.670386e-08 15.53534
```

Hey, integrin made it to the list!

The Full Table...



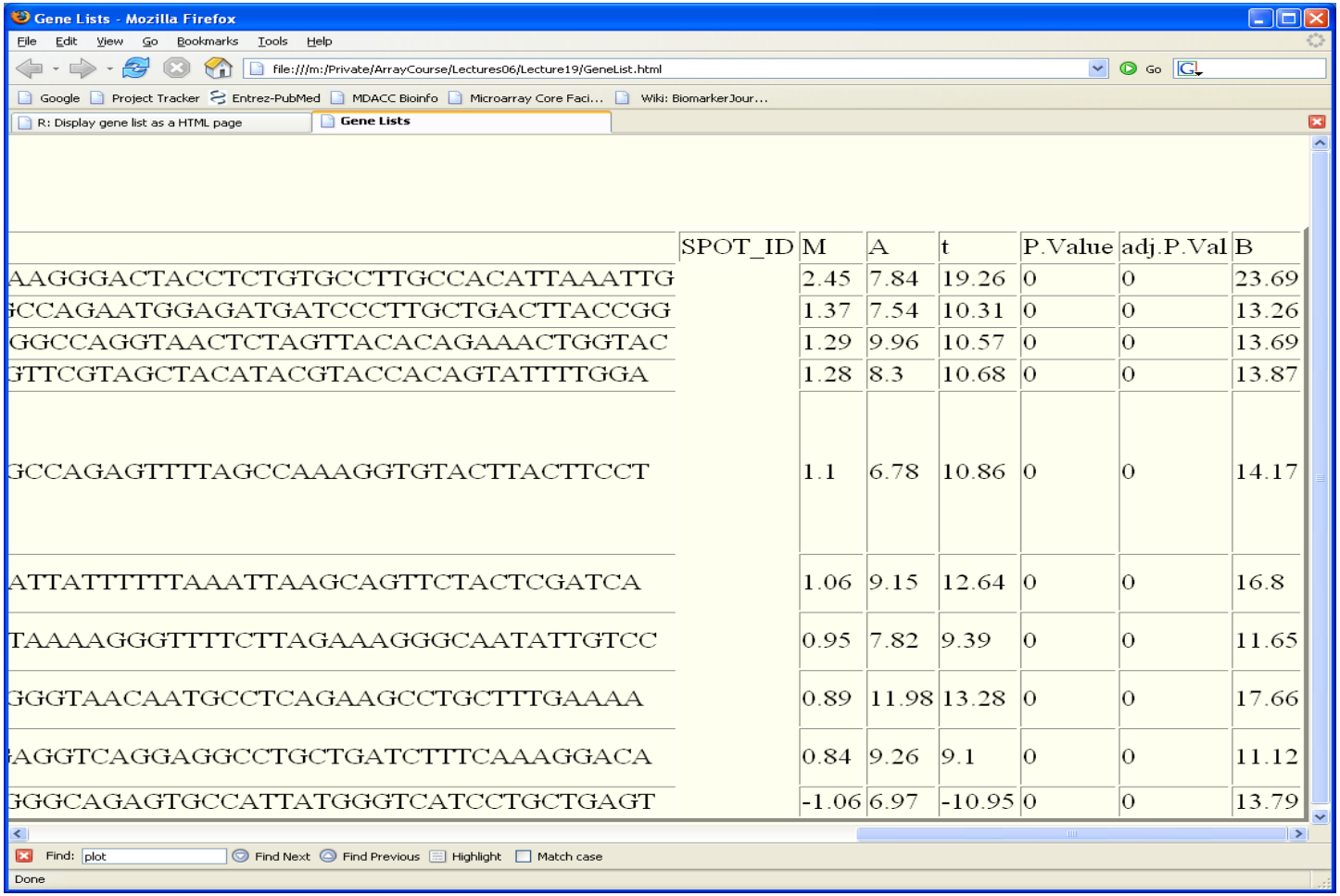
BioConductor Gene Listing

ID	Block	Column	Row	Operon_ID	GB_ACC	Unigene	Description	Symbol
6647	14	11	18	H200017286	NM_016602	278446	G protein-coupled receptor 2	GPR2
9314	20	11	7	H200006462	NM_005572	436441	lamin A/C	LMNA
11717	25	20	6	H200006048	NM_002922	75256	regulator of G-protein signalling 1	RGS1
18397	39	1	3	H200002118	NM_015271	435734	tripartite motif-containing 2	TRIM2
4910	11	17	4	H200003784	NM_003966	27621	sema domain, seven thrombospondin repeats (type 1 and type 1-like), transmembrane domain (TM) and short cytoplasmic domain, (semaphorin) 5A	SEMA5A
12122	26	5	3	H200001987	NM_018371	341073	chondroitin beta1,4 N-acetylgalactosaminyltransferase	ChGn
3097	7	10	10	H200009382	AK026181	82101	pleckstrin homology-like domain, family A, member 1	PHLDA1
1755	4	12	15	H200014446	NM_005767	123464	purinergic receptor P2Y, G-protein coupled, 5	P2RY5
10667	23	20	2	H200002092	AK056276	12251	similar to BcDNA:GH11415 gene product	LOC151963
3152	7	2	13	H200012024	X68742	439320	integrin, alpha 1	ITGA1

```
> table2html(shortTable, disp = "file")
```

NULL

...Rest of The Full Table



	SPOT_ID	M	A	t	P.Value	adj.P.Val	B
AAGGGACTACCTCTGTGCCTTGCCACATTAAATTG		2.45	7.84	19.26	0	0	23.69
GCCAGAATGGAGATGATCCCTTGCTGACTTACCGG		1.37	7.54	10.31	0	0	13.26
GGCCAGGTAACTCTAGTTACACAGAAACTGGTAC		1.29	9.96	10.57	0	0	13.69
GTTCGTAGCTACATACGTACCACAGTATTTTGGA		1.28	8.3	10.68	0	0	13.87
GCCAGAGTTTTAGCCAAAGGTGTACTTACTTCCT		1.1	6.78	10.86	0	0	14.17
ATTATTTTTTAAATTAAGCAGTTCTACTCGATCA		1.06	9.15	12.64	0	0	16.8
TAAAAGGGTTTTCTTAGAAAGGGCAATATTGTCC		0.95	7.82	9.39	0	0	11.65
GGGTAACAATGCCTCAGAAGCCTGCTTTGAAAA		0.89	11.98	13.28	0	0	17.66
AGGTCAGGAGGCCTGCTGATCTTTCAAAGGACA		0.84	9.26	9.1	0	0	11.12
GGGCAGAGTGCCATTATGGGTCATCCTGCTGAGT		-1.06	6.97	-10.95	0	0	13.79

Why are there two forms of p-value? And what is B?

What is Going On?

They report both raw p-values and p-values that are adjusted for multiple testing (how?). The adjusted p-values can reach 1 when there are only a few samples.

The value B is the "log odds" that the gene is differentially expressed – if the value of B is 0.59, then the odds that the gene is differentially expressed are $\exp(0.59) = 1.803$ to 1, and the probability of differential expression is $1.803/(1.803 + 1) = 0.643$. With log odds of 11, the probability is greater than 99.9%.

Is this the Right Contrast?

Not quite. Leading question – why were there at least two arrays run for each patient?

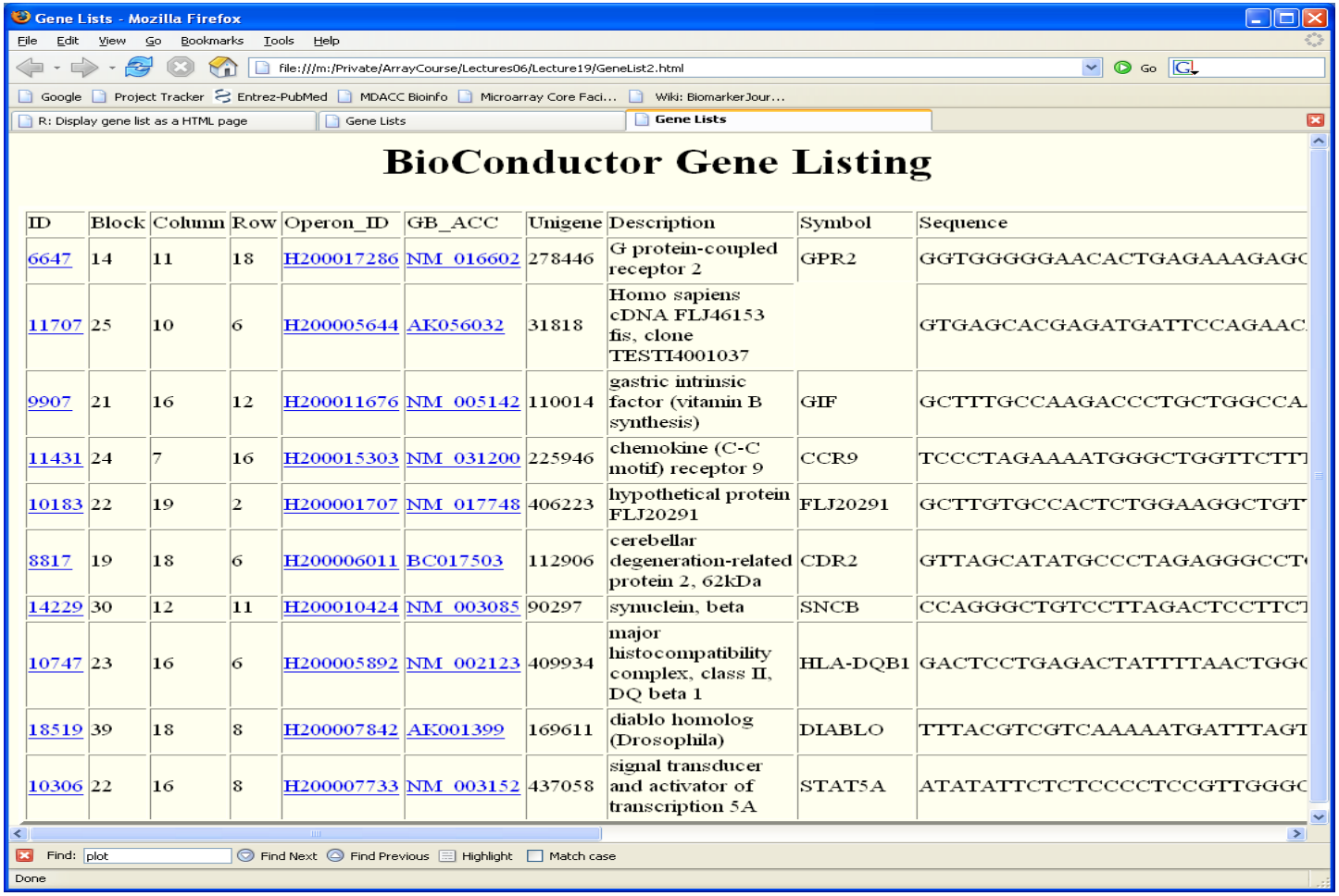
Dye Swaps.

There may be a dye effect present, and it may be possible to account for this. This requires adjusting the design matrix.

```
> design <- cbind(Dye = 1, b7)
> LMres2 <- lmFit(normData, design)
> LMres2EB <- eBayes(LMres2)
> shortTable2 <- topTable(LMres2EB, adjust = "fdr",
+     number = 10, resort.by = "M")
> table2html(shortTable2, filename = "GeneList2.html",
+     disp = "file")
```


NULL

Accounting for Dye...



BioConductor Gene Listing

ID	Block	Column	Row	Operon_ID	GB_ACC	Unigene	Description	Symbol	Sequence
6647	14	11	18	H200017286	NM_016602	278446	G protein-coupled receptor 2	GPR2	GGTGGGGGGAACACTGAGAAAGAGC
11707	25	10	6	H200005644	AK056032	31818	Homo sapiens cDNA FLJ46153 fis, clone TESTI4001037		GTGAGCACGAGATGATTCCAGAAC
9907	21	16	12	H200011676	NM_005142	110014	gastric intrinsic factor (vitamin B synthesis)	GIF	GCTTTGCCAAGACCCTGCTGGCCA
11431	24	7	16	H200015303	NM_031200	225946	chemokine (C-C motif) receptor 9	CCR9	TCCCTAGAAAATGGGCTGGTTCTT
10183	22	19	2	H200001707	NM_017748	406223	hypothetical protein FLJ20291	FLJ20291	GCTTGTGCCACTCTGGAAGGCTGT
8817	19	18	6	H200006011	BC017503	112906	cerebellar degeneration-related protein 2, 62kDa	CDR2	GTTAGCATATGCCCTAGAGGGCCT
14229	30	12	11	H200010424	NM_003085	90297	synuclein, beta	SNCB	CCAGGGCTGTCCTTAGACTCCTTCT
10747	23	16	6	H200005892	NM_002123	409934	major histocompatibility complex, class II, DQ beta 1	HLA-DQB1	GACTCCTGAGACTATTTTAACTGGC
18519	39	18	8	H200007842	AK001399	169611	diablo homolog (Drosophila)	DIABLO	TTTACGTCGTCAAAAATGATTTAGT
10306	22	16	8	H200007733	NM_003152	437058	signal transducer and activator of transcription 5A	STAT5A	ATATATTCTCTCCCCTCCGTTGGGC

```
> table2html(shortTable2, disp = "file");
```

The table changes a lot...

Is it what they used in the paper?

Not quite. There, they also decided to not subtract background, with the result that they did not have to deal with those pesky NA values.

They also dealt with several replicates per person. This can be accomodated in lmFit by defining a more extensive model matrix.

The general lesson here is that the answers that we get change rather drastically as we change the nature of the question being asked.

This is addressed in considerable detail in Chapter 23 of Gentleman et al on `limma` by Gordon Smyth. We will revisit this in later lectures.

Comments on Replication

Why do we need to treat replicates differently than other samples?

They're not measuring "independent" quantities. If we measure 10 replicates from a sick person and 10 replicates from a healthy person, then contrasting these 20 arrays, we're still contrasting just one person with another. Replications in the form of dye-swaps, however, is still useful in that it allows us to preclude certain biases from affecting our results.

An Example

(adapted from Smyth's chapter)

Let's say that we have two patients that we want to compare with two controls. How many possible pairwise combinations are there?

Here, there are 8: 2 controls * 2 patients * 2 dye orderings.

Now, what we really want to say something about is the difference between disease states: avg disease - avg control.

This in turn is given by

$$(D1 + D2)/2 - (C1 + C2)/2$$

But how do we get these?

Levels of Contrasts

These are average levels for each individual, so that the results for one individual do not dominate the results by simply being present in more of the samples.

These average levels can be estimated, using an appropriately defined design matrix.

We can lay things out more precisely.

What Goes Where

```
> fakeSamples <- read.table("fakeSamples.txt",header=T)
> fakeSamples
  FileName Cy3 Cy5
1 F1.gpr   D1  C1
2 F2.gpr   D1  C2
3 F3.gpr   D2  C1
4 F4.gpr   D2  C2
5 F5.gpr   C1  D1
6 F6.gpr   C2  D1
7 F7.gpr   C1  D2
8 F8.gpr   C2  D2
```

Coming up with the design is fairly easy, for one relative to all of the others

The Model Matrix

```
> design <- modelMatrix(fakeSamples, ref="D1")
> design
  C1 C2 D2
1  1  0  0
2  0  1  0
3  1  0 -1
4  0  1 -1
5 -1  0  0
6  0 -1  0
7 -1  0  1
8  0 -1  1
```

+1 if the sample is in Cy5, -1 if it is in Cy3.

Fit the Model and Contrast

```
> design <- cbind(Dye = 1, design);
> fakeFit <- lmFit(fakeMA, design);
```

Given things that are directly estimable, define the contrast of interest in terms of the values found, and fit the contrast.

```
> contrast.matrix <- makeContrasts(
    DvsC = (D2/2) - ((C1+C2)/2),
    levels = design)
> fakeFitCont <- contrasts.fit(fakeFit,
    contrast.matrix)
> fakeFitContEB <- eBayes(fakeFitCont)
```

Things We Glossed Over, 1

How did we normalize the data?

We made a call to `normalizeWithinArrays`, but what does that do?

By default, this deals with print-tip loess, but there are two things to consider here. First, print-tip loess makes stronger assumptions that loess. Ratios from each print tip are assumed to have the same distributions, and this is a dangerous assumption if the allocation of clones to plates is nonrandom. Spots grouped by function may be brighter.

Things We Glossed Over, 2

The second aspect of normalization that's a bit more difficult in the initial analysis is the question of how to deal with multiple array layouts.

I would tend to use some type of print-tip or spatial loess within each array to correct for overall trends, and then use quantile normalization to line up the ratios for genes spotted on both platforms.

