GS01 0163
Analysis of Microarray Data

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

20 September 2007

Lecture 7: Affymetrix, R, and BioConductor

- Reading Affymetrix data with BioConductor
- Processing Affymetrix data
- Quantification = summarization
- Description of quantification methods
  - MAS 5.0
  - RMA
  - PDNN
- Quality control assessment

Reading Affymetrix data with BioConductor

Once upon a time, the BioConductor `affy` package included a graphical interface (via the function `ReadAffy()`) that made it easier to read in Affymetrix data and contruct `AffyBatch` objects. This method started to break in version in 2.4, and is now essentially useless in version 2.5. If we are really lucky, they may fix it in version 2.6.

In the meantime, we have to do everything by writing scripts by hand. Since we are interested in docuemting things and making them reproducible, it may be to our advantage that the (irreproducible) GUI no longer works.

Getting started

We start by loading the basic Affymetrix R library.

```
> library(affy)

Loading required package: Biobase
Loading required package: tools

Welcome to Bioconductor

  Vignettes contain introductory material. To view, type
  'openVignette()'. To cite Bioconductor, see
  'citation("Biobase")' and for packages 'citation(pkgname)'

Loading required package: affyio
```

File locations

As we did with dChip, we have to let the system know where the files are located. In this example (as in most examples), all the CEL files are stored in a single directory. The associated sample information files are nearby.

```
> datapath <- "G:/Public/Singh-Prostate-Affymetrix"
> celpath <- file.path(datapath, "CelFiles")
```

There is a function that makes it easy to get a list of the CEL files

```
> filenames <- list.celfiles(celpath)
> filenames[1:3]
```

```
[1] "N01__normal.CEL" "N02__normal.CEL"
[3] "N03__normal.CEL"
```

AnnotatedDataFrame

In earlier version of BioConductor, the "sample information file" was loaded into an object of class `phenoData`. In the most recent version, this has been replaced by the `AnnotatedDataFrame` class. Both objects basically represent a single data frame, with auxiliary information that expands on the column names.

We can read the same sample information file that we used for dChip into R and make it into an AnnotatedDataFrame:

```
> adf <- read.AnnotatedDataFrame(file.path(datapath,
+     "/subsamples.txt"), header = TRUE, sep = "\t",
+     row.names = 2)
> adf

  rowNames: N01A, N58A, ..., T49B (20 total)
```

varLabels and varMetadata:

- Array.name: read from file
- Status: read from file
- Batch: read from file
- Cluster: read from file

Actually creating the annotations

As you just saw, the extra annotations are not created when you read a file this way. We prepared another file describing the columns.

```
> tmp <- read.table(file.path(datapath, "explain.txt"),
+     header = TRUE, sep = "\t", quote = "",
+     row.names = "Id")
> varMetadata(adf) <- tmp
> rm(tmp)
> adf
```

```
  rowNames: N01A, N58A, ..., T49B (20 total)
  varLabels and varMetadata:
    Array.name: The CEL file name, without extension
    Status: Either "Normal" prostate or prostate "Tumor"
```

```
    Batch: One of four experimental run batches, from the
    Cluster: One of two clusters based on our analysis in
```

MIAME

MIAME = "Minimum Information About a Microarray Experiment"

Here is a MIAME object for the Singh experiment:

```
> miame <- new("MIAME", name = "Dinesh Singh",
+     lab = "William F. Sellers", title = paste("Gene",
+         "expression correlates of clinical",
+         "prostate cancer behavior"), pubMedIds = c("120
```

Creating an AffyBatch

R (on a 32-bit machine) is unable to read all 103 CEL files into an AffyBatch at once. (We will talk later about how to get around some of these limitations.) In this case, the sample file we read in earlier actually only contains descriptions of 20 of the CEL files, which is an amount we can easily read.

```
> filenames <- paste(pData(adf)$Array.name,
+     "CEL", sep = ".")
> abatch <- read.affybatch(filenames = file.path(celpath,
+     filenames), phenoData = adf, description = miame)
> rm(adf, miame)
```

The AffyBatch

Now we can look at a summary of what we have so far:

```
> abatch

AffyBatch object
size of arrays=640x640 features (11 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=20
number of genes=12625
annotation=hgu95av2
notes=
```

The Dilution DataSet

FOr the rest of this lecture, we are going to use the sampe dataq set that ships with BioConductor.

```
> library(affydata)
> data(Dilution)
> Dilution
```

```
AffyBatch object
size of arrays=640x640 features (27210 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=4
number of genes=12625
annotation=hgu95av2
notes=
```

Processing Affymetrix Data

BioConductor breaks down the low-level processing of Affymetrix data into four steps. The design is highly modular, so you can choose different algorithms at each step. It is highly likely that the results of later (high-level) analyses will change depending on your choices at these steps.

- Background correction
- Normalization (on the level of features = probes)
- PM-correction
- Summarization

Background correction

The list of available background correction methods is stored in a variable:

```
> bgcorrect.methods

[1] "mas"  "none" "rma"  "rma2"
```

none Do nothing

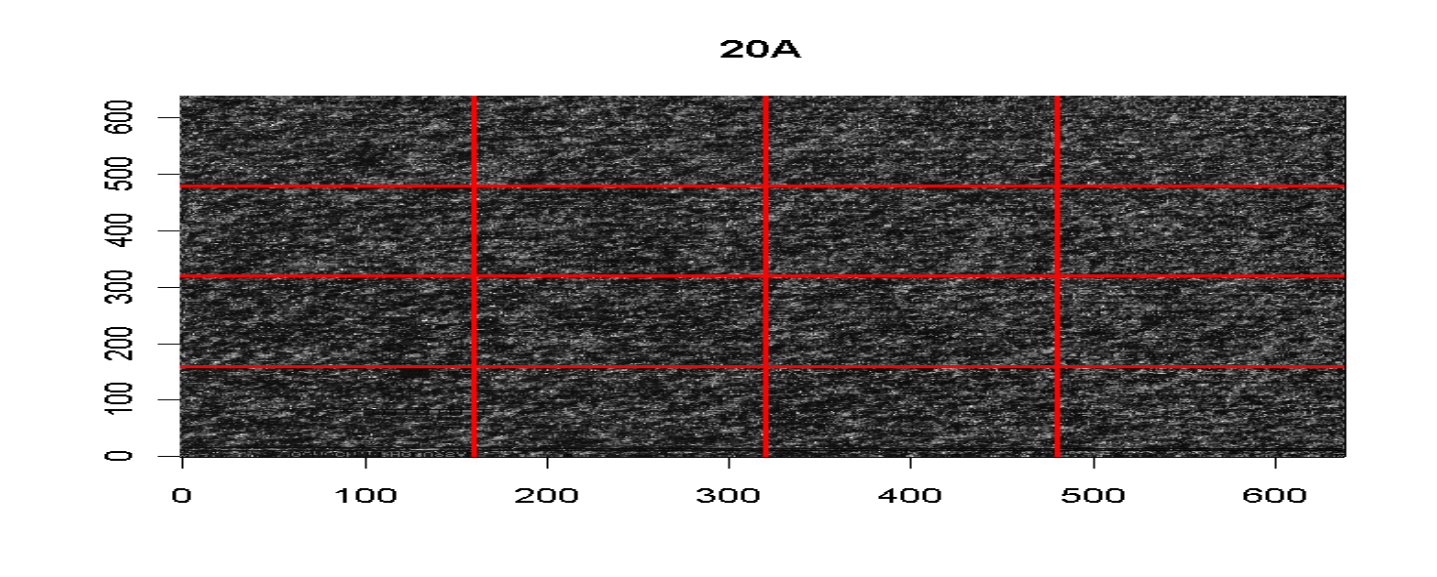
mas Use the algorithm from MAS 5.0

rma Use the algorithm from the current version of RMA

rma2 Use the algorithm from an older version of RMA

Background correction in MAS 5.0

MAS 5.0 divides the microarray (more precisely, the CEL file) into 16 regions. In each region, the intensity of the dimmest 2% of features is used to define the background level. Each probe is then adjusted by a weighted average of these 16 values, with the weights depending on the distance to the centroids of the 16 regions.



Background correction in RMA

RMA takes a different approach to background correction. First, only PM values are adjusted, the MM values are not changed. Second, they try to model the distribution of PM intensities as a sum of

- exponential signal with mean λ
- normal noise with mean μ and variance σ^2 (truncated at 0 to avoid negatives).

If we observe a signal $X = x$ at a PM feature, we adjust it by

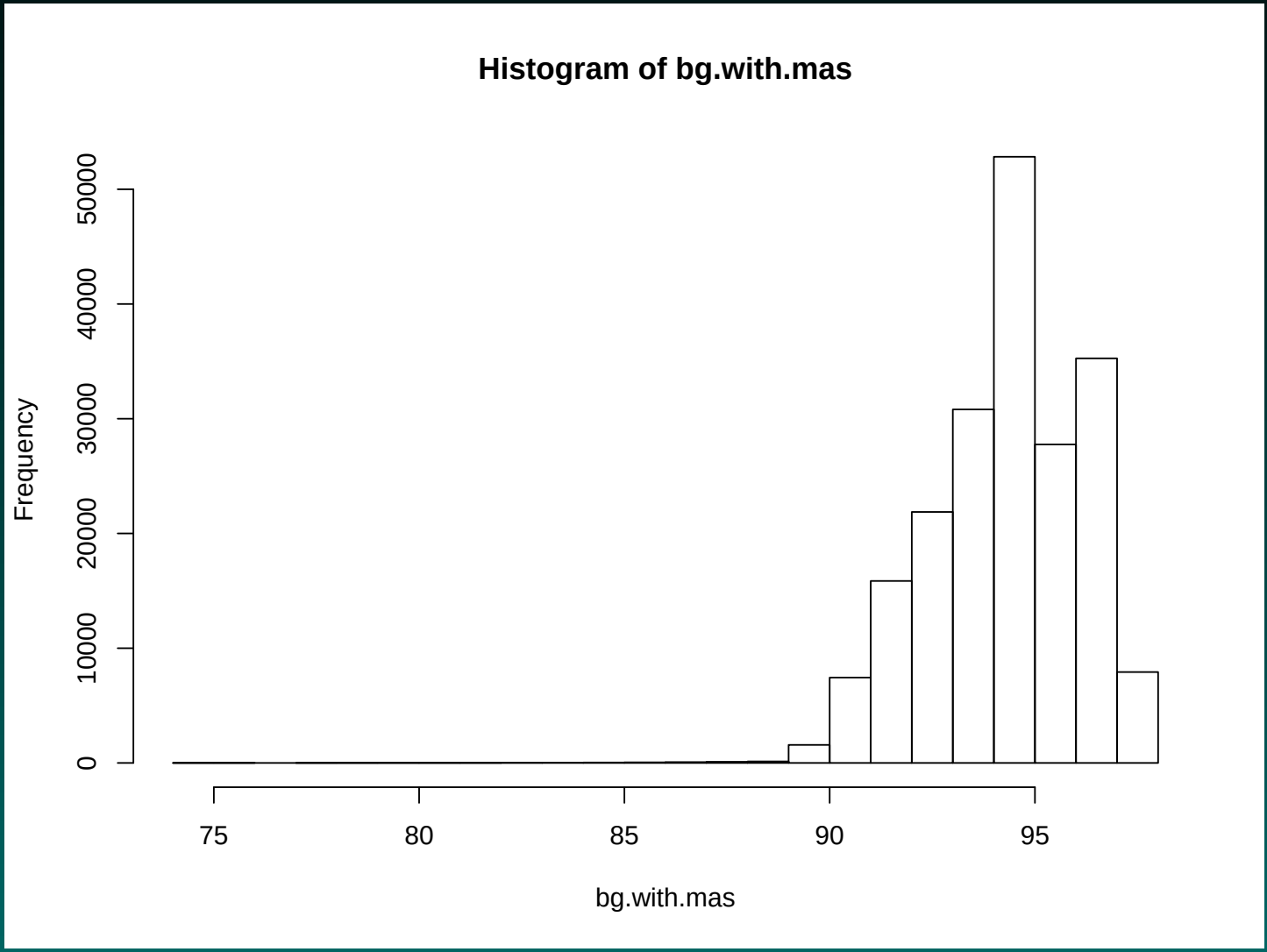
$$E(s|X = x) = a + b\frac{\phi(a/b) - \phi((x-a)/b)}{\Phi(a/b) + \Phi((x-a)/b) - 1}$$

where $b = \sigma$ and $a = s - \mu - \lambda\sigma^2$.

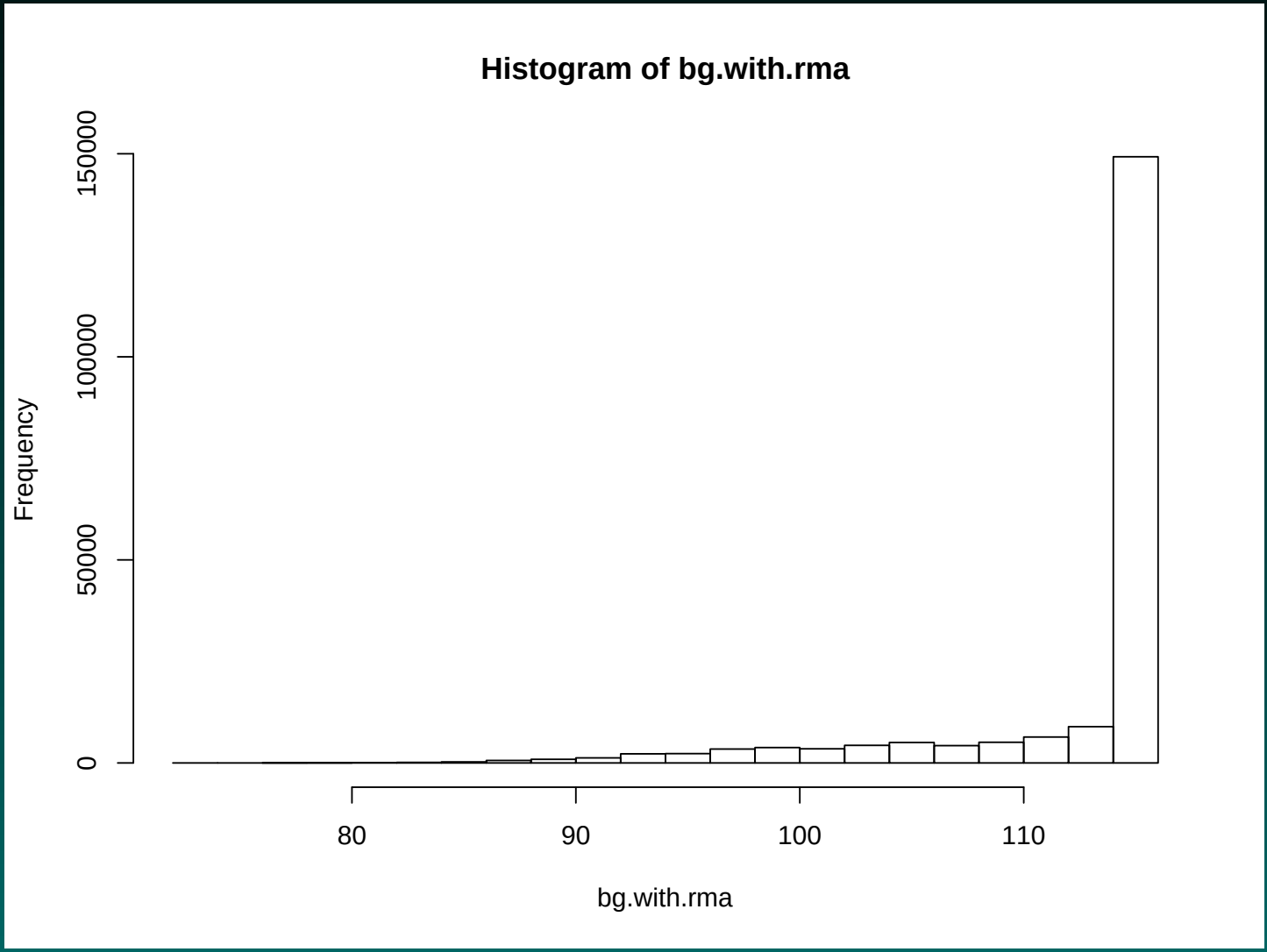
Comparing background methods

```
> d.mas <- bg.correct(Dilution[, 1], "mas")
> d.rma <- bg.correct(Dilution[, 1], "rma")
> bg.with.mas <- pm(Dilution[, 1]) - pm(d.mas)
> bg.with.rma <- pm(Dilution[, 1]) - pm(d.rma)
```

```
> hist(bg.with.mas)
```



```
> hist(bg.with.rma)
```



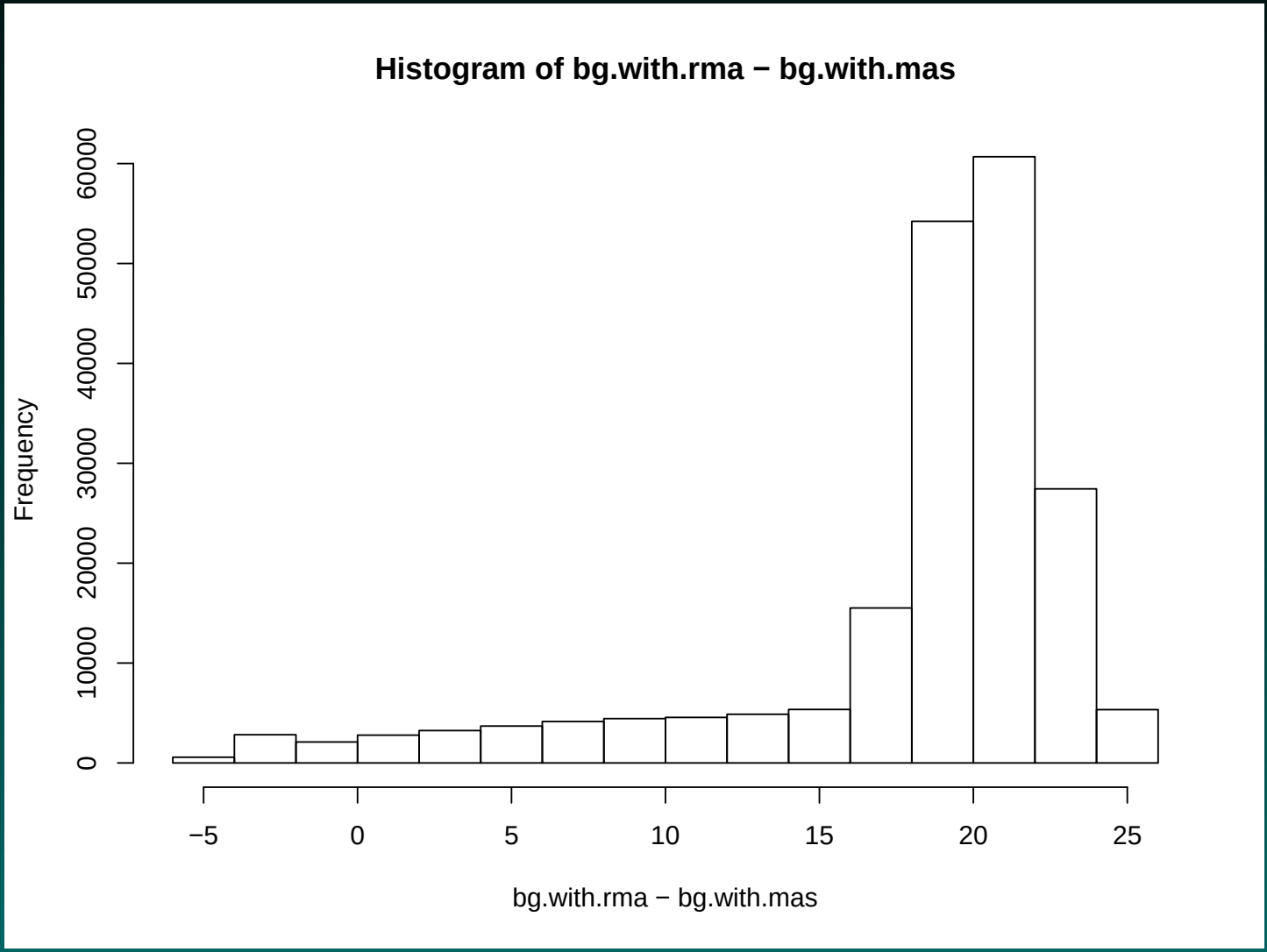
```
> summary(data.frame(bg.with.mas, bg.with.rma))

      X20A           X20A.1
 Min.   :74.53   Min.   : 72.4
 1st Qu.:93.14   1st Qu.:113.7
 Median :94.35   Median :114.9
 Mean   :94.27   Mean   :112.1
 3rd Qu.:95.80   3rd Qu.:114.9
 Max.   :97.67   Max.   :114.9
```

Difference in background estimates

On this array, RMA gives slightly larger background estimates, and gives estimates that are more nearly constant across the array. The overall differences can be displayed in a histogram.

```
> hist(bg.with.rma - bg.with.mas)
```



How big is 20 units?

```
> tmp <- data.frame(pm(Dilution[, 1]), mm(Dilution[,
+     1]))
> colnames(tmp) <- c("PM", "MM")
> summary(tmp)
```

```
       PM                MM
 Min.   :   76.0   Min.   :   77.3
 1st Qu.:  137.0   1st Qu.:  120.3
 Median :  225.0   Median :  164.5
 Mean   :  507.3   Mean   :  323.5
 3rd Qu.:  489.0   3rd Qu.:  313.0
 Max.   :23356.3   Max.   :17565.3
```

Quantification = summarization

I'm going to avoid talking about normalization and PM correction for the moment, and jump ahead to summarization. This step is the critical final component in analyzing Affymetrix arrays, since it's the one that combines all the numbers from the PM and MM probe pairs in a probe set into a single number that represents our best guess at the expression level of the targeted gene.

The available summarization methods, like the other available methods, can be obtained from a variable.

```
> express.summary.stat.methods

[1] "avgdiff"      "liwong"       "mas"
[4] "medianpolish" "playerout"    "pdnn"
```

expresso

The recommended way to put together all the steps for processing Affymetrix arrays in BioConductor is with the function `expresso`. Here is an example that blocks everything except the summarization:

```
> tempfun <- function(method) {
+     expresso(Dilution, bg.correct = FALSE,
+         normalize = FALSE, pmcorrect.method = "pmonly",
+         summary.method = method)
+ }
> ad <- tempfun("avgdiff")
> al <- tempfun("liwong")
> am <- tempfun("mas")
> ar <- tempfun("medianpolish")
```

Bland-Altman (M-versus-A) plots

Early in the study of microarrays, several groups (including ours) introduced what have come to be known in the microarray world as "M-versus-A" plots or sometimes just MA-plots. Statisticians knew these as "Bland-Altman" plots long before anyone started studying microarrays, since they were among the first people to use them.

The problem being solved by a Bland-Altman MA-plot is that of providing a useful graphical display of two vectors of data, x and y, which typically represent two measurements that should (almost always) be the same. The first thing that comes to mind is to plot y against x in the usual way, and see how well the points follow the identity line $y = x$.

Bland-Altman (M-versus-A) plots

The difficulties with this simple approach are

1. Humans can recognize horizontal lines much more easily than the can recognize diagonal lines.

2. Different aspect ratios (i.e., different scales along the axes) can move the line away from the diagonal.

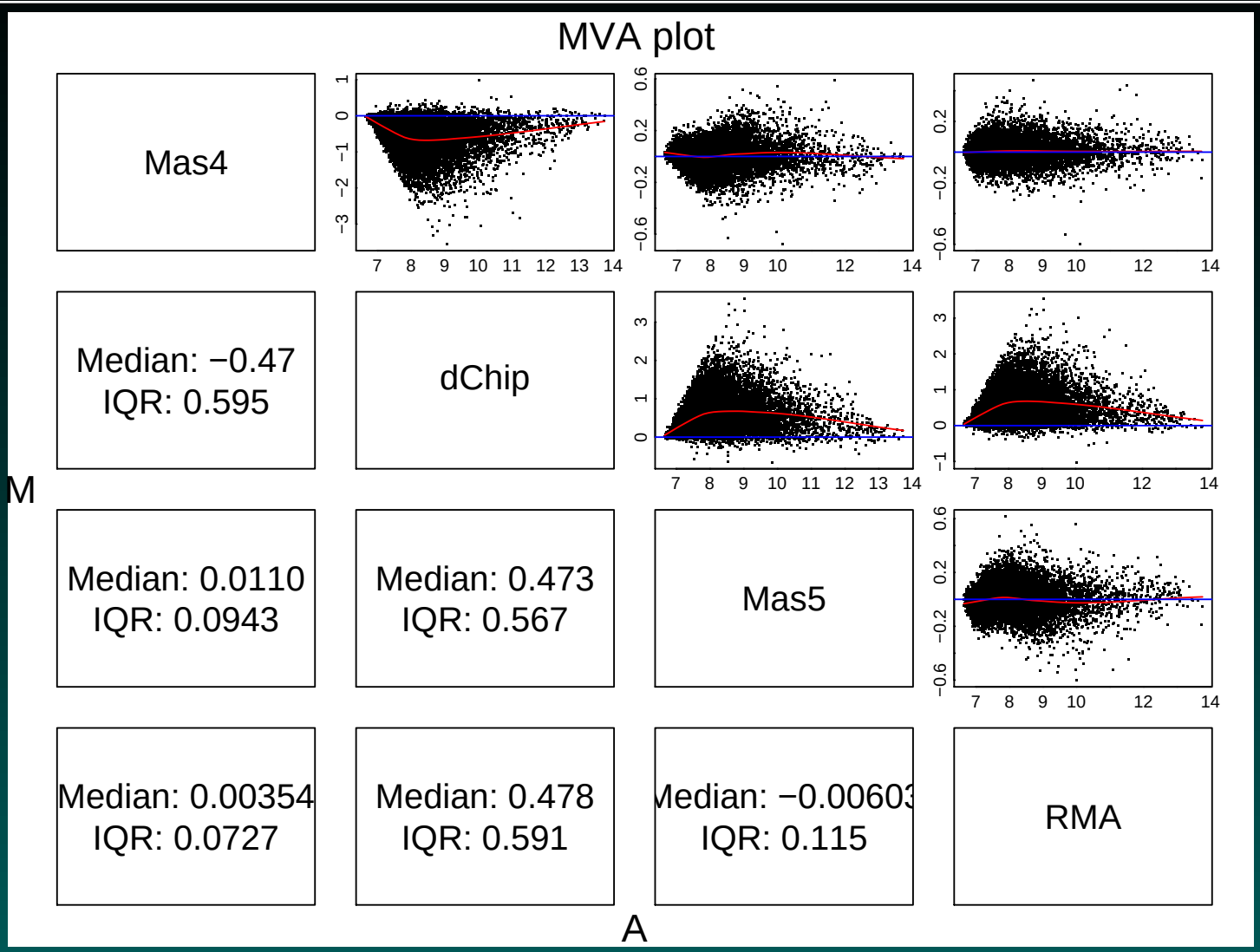
3. Deviations from a tilted diagonal line are hard to estimate accurately by eye.

The Bland-Altman solution is to rotate the plot by 45 degrees, which turns the diagonal line into a horizontal line. To do this, they plot the average $((x + y)/2)$ along the horizontal axis and the difference $(y - x)$ along the vertical axis.

MA-plots in BioConductor

The affy package includes a function called mva.pairs to make it easier to generate these plots. (You should also check out the MAplot function.) We are going to use this to compare the different quantification/summary methods.

```
> temp <- data.frame(exprs(ad)[, 1], exprs(al)[,
+     1], exprs(am)[, 1], 2^exprs(ar)[, 1])
> dimnames(temp)[[2]] <- c("Mas4", "dChip",
+     "Mas5", "RMA")
> mva.pairs(temp)
```



Alternate preprocessing

It is possible that the differences we see in the MA-plots are caused because we did no processing before summarization. We will try again, but this time we will use `expresso` to correct background with the RMA method, perform quantile normalization, and just use the PM values for summarization.

```
> tempfun <- function(method) {
+     expresso(Dilution, bgcorrect.method = "rma",
+         normalize.method = "quantiles",
+         pmcorrect.method = "pmonly", summary.method = m
+ }
```

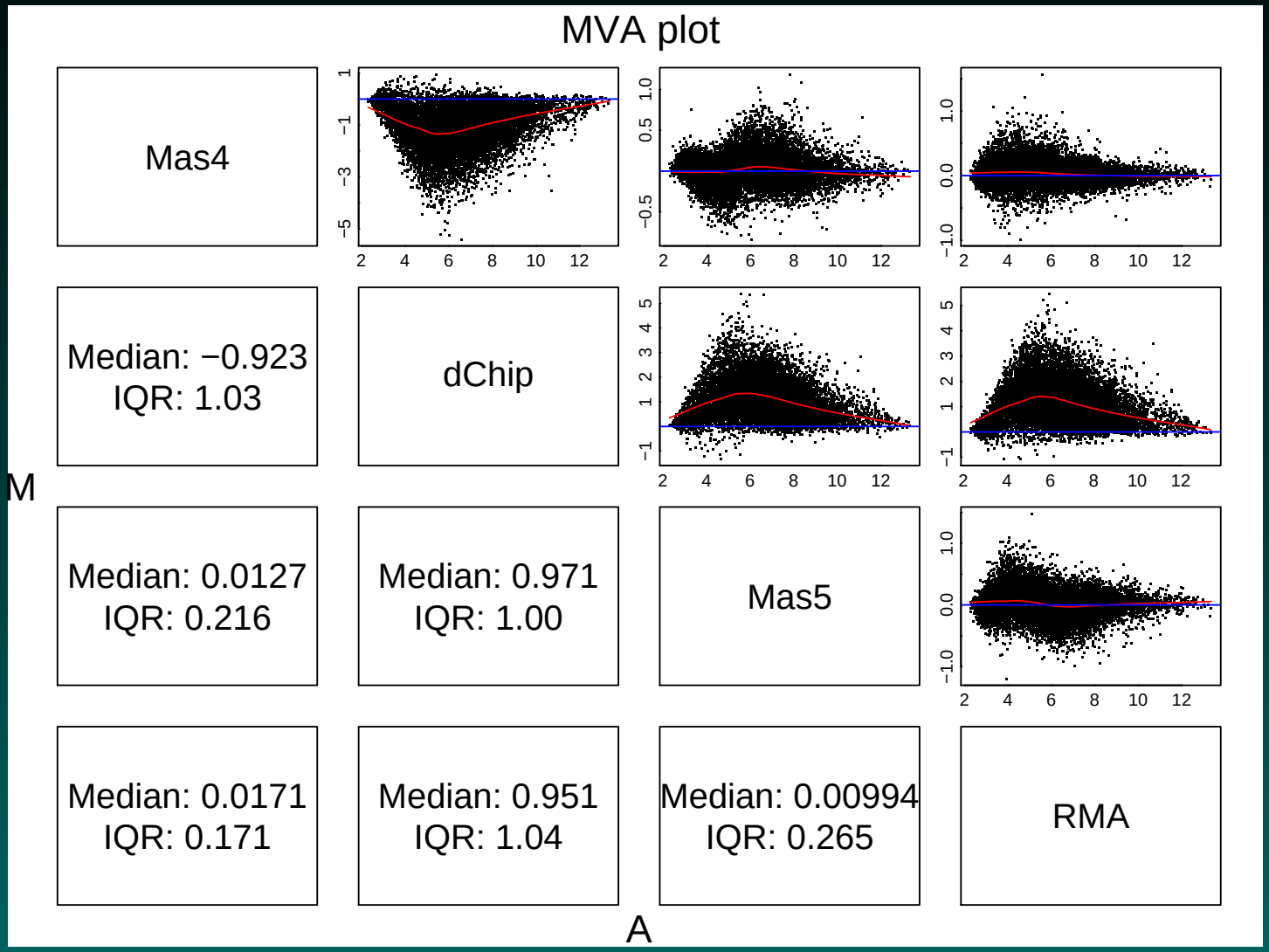
Alternate preprocessing

Now we repeat the same commands as before, which use the four different summarization methods on the same array and put them into a temporary data frame for display.

```
> ad <- tempfun("avgdiff")
> al <- tempfun("liwong")
> am <- tempfun("mas")
> ar <- tempfun("medianpolish")

> temp <- data.frame(exprs(ad)[, 1], exprs(al)[,
+     1], exprs(am)[, 1], 2^exprs(ar)[, 1])
> dimnames(temp)[[2]] <- c("Mas4", "dChip",
+     "Mas5", "RMA")
> mva.pairs(temp)
```

Comparison of summarization methods



How do the summarization methods work?

Recall first what we know about the oldest method for processing Affymetrix data, the AvDiff method of MAS4.0. This method

1. Uses the background-correction method described above, based on the bottom 2% of probes.
2. Normalizes by scaling the median intensity to a fixed value.
3. Computes the PM − MM differences.
4. Trims outliers and computes the average (mean) of the differences.

Review of dChip

We have also looked previously at the dChip method:

1. Normalizes using an "invariant set" method (described later).
2. Optionally uses either both PM and MM values or PM-only.
3. Fits a statistical model for sample i, and probe j,

$$MM_{ij} = \nu_j + \theta_i \alpha_j + \epsilon$$

$$PM_{ij} = \nu_j + \theta_i \alpha_j + \theta_i \phi_j + \epsilon$$

Focusing on the PM $-$ MM differences, this model estimates the probe affinities ϕ_j and the expression values θ_i.

Improving Robustness: MAS 5.0

Affymetrix learned something from the modelling process. In particular, they noted the importance of multiplicative adjustments and statistical measures with some means of identifying outliers. They also noted that negative values from AvDiff just were not well received by biologists or statisticians.

They modifed their algorithm in several ways. Instead of the straight MM value, they subtract a "change threshold" (CT) which is guaranteed to be lower than the PM value. Basically, they "fix" the MM value when it is smaller than PM. Next, they shifted to the log scale to capture multiplicative effects. Finally, they used a robust statistical method to downweight outliers instead of their earlier *ad hoc* method.

$$\text{signal} = \exp(\text{Tukey Biweight}(\log(PM_j - CT_j)))$$

MAS 5.0 vs MAS 4.0

It was at this stage that Affy decided it wasn't going to fight to have the best algorithm; it would let others play that game. Indeed, it could reap the benefits of better algorithms by selling more chips.

To let people test their own models, they created and posted a test dataset: The Affy Latin Square Experiment.

Using the test set, they could demonstrate that the MAS5 signal statistic is an improvement on AvDiff. It tracks nominal fold changes better, and it is less variable.

What it still doesn't do is use information across chips.

Robust Multichip Analysis: RMA

RMA (Irizarry et al, Biostatistics 2003) tries to take the better aspects of both dChip and MAS 5.0, and to add some further twists.

Earlier in this lecture, we described the statistical model used by RMA to perform background correction.

They normalize using the "median polish" method, which we will describe in a later lecture.

They throw away the MM values entirely. They contend that there are too many cases where MM $>$ PM, and hence including the MMs introduces more variability than the correction is worth. (They are probably correct.)

Robust Multichip Analysis: RMA

As with dChip, the RMA summarization method is built around a model:

$$log(medpol(PM_{ij} - BG)) = \mu_i + \alpha_j + \epsilon_{ij}$$

(array i, probe j).

The parameters of this model are fit using multiple chips.

Unlike dChip, the random jitter (epsilon) is introduced on the log scale as opposed to the raw scale. This more accurately captures the fact that more intense probes are more variable.

Incorporating other information: PDNN

The above methods are all mathematical, in that they focus solely on the observed values without trying to explain those values.

Why should some probes give consistently stronger signals than others?

What governs nonspecific binding?

In general, these will depend on the exact sequence of the probe, and the thermodynamics of the binding.

Fitting the thermodynamics

Li Zhang introduced the Position-Dependent Nearest Neighbor (PDNN) model (*Nat Biotech*, 2003; 21:818). Unlike dChip and RMA, the parameters for the PDNN model can all be estimated from a single chip, in large part because the number of parameters is much smaller. He posits a scenario where the chance of binding is dictated by the probe sequence, and shifts the mathematical modeling back from the expression values to the probe sequences.

The model parameters are:

1. The position k of a base pair in the sequence.

2. Interactions with nearest neighbors: knowing k, we must also know what is at $k-1$ and $k+1$.

What is the model?

The observed signal Y_{ij} for probe i in the probe set for gene j is modeled as

$$Y_{ij} = \frac{N_j}{1 + exp(E_{ij})} + \frac{N^*}{1 + exp(E^*_{ij})} + B$$

In this model, there are two global terms that need to be estimated: the background, B, and the number, N^*, of RNA molecules contributing to non-specific binding (NSB).

The quantity of interest is N_j, the number of expressed mRNA molecules contributing to gene-specific binding (GSB).

The binding energies E_{ij} and E^*_{ij} are sums over contributions from each position.

What are the model parameters?

For example, consider a probe with sequence

CACCCAGCTGGTCCTGTGGATGGGA

- We write this as an ordered list of neighboring pairs:

1. C, A energy $= w_1 \nu(b_1, b_2)$
2. A, C energy $= w_2 \nu(b_2, b_3)$
3. C, C energy $= w_3 \nu(b_3, b_4)$
4. C, C energy $= w_4 \nu(b_4, b_5)$
5. C, A energy $= w_5 \nu(b_5, b_6)$
6. A, G energy $= w_6 \nu(b_6, b_7)$
7. G, C energy $= w_7 \nu(b_7, b_8)$
8. C, T energy $= w_8 \nu(b_8, b_9)$
9. etc.

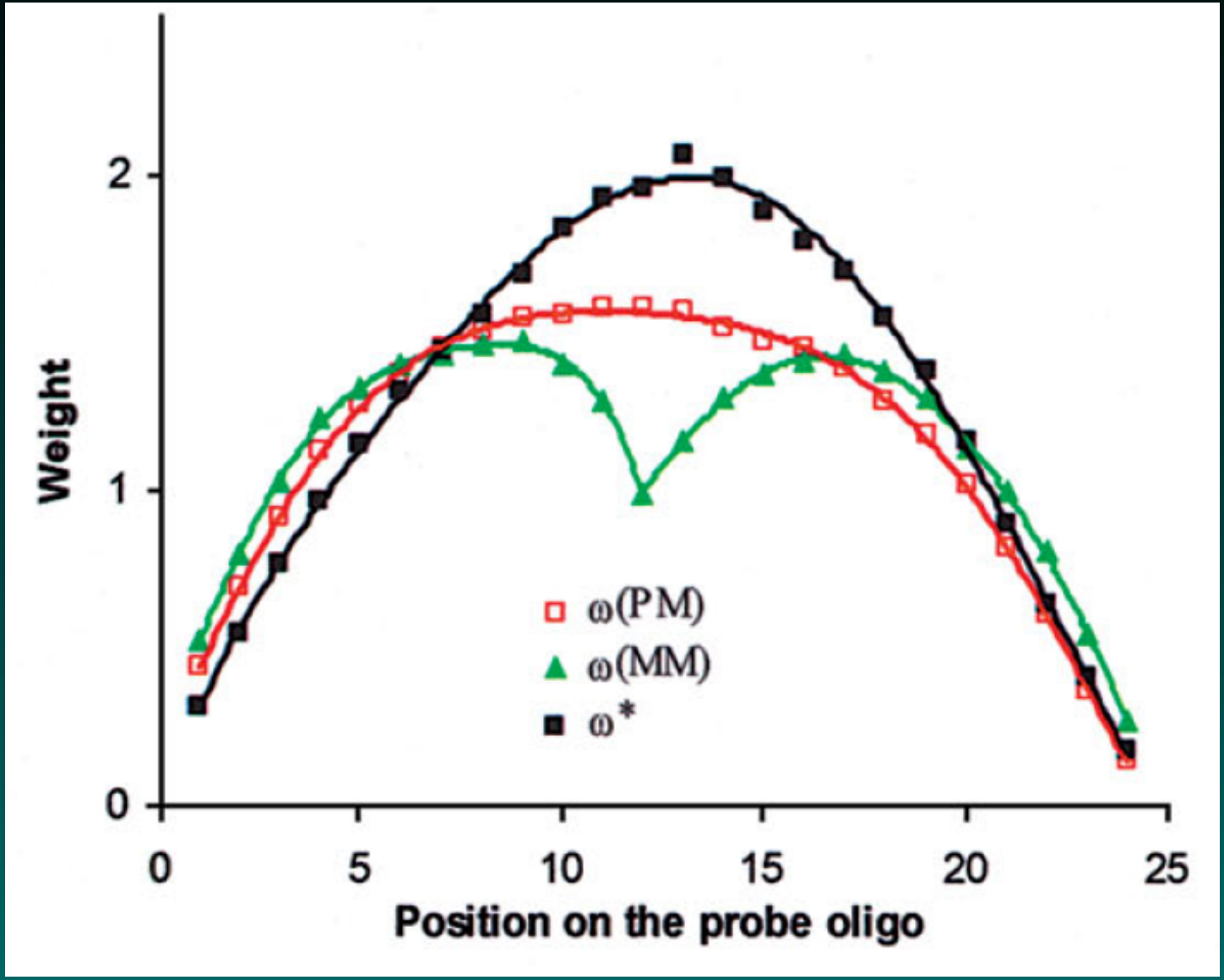
Free energy

The free energy for a perfect match is

$$E_{ij} = \sum_k w_k \nu(b_k, b_{k+1})$$

There is a similar formula for binding in the presence of manu mismatches. In total, there are 2*24 weight parameters, 2*16 neighboring base pair parameters, 2 global parameters, plus one expression parameter per probe set. Because there are many probes in each probe set, we can fit all these parameters with a single chip.

Fitted weight parameters



Using PDNN in R

The implementation of the PDNN method is contained in a separate BioConductor package. When you load the package libary, it updates the list of available methods.

```
> library(affypdnn)
> express.summary.stat.methods
```

```
[1] "avgdiff"      "liwong"       "mas"
[4] "medianpolish" "playerout"    "pdnn"
```

Using PDNN in R

One should note that the PDNN method does not follow the standard four-step procedure used by `expresso`. Instead of background correction, the method starts immediately with quantile normalization. The model can be fit separately on the PM and MM probes, or the MM probes can be discarded. Background is estimated along with the energy parameters and expression parameters as part of a single model.

In particular, you must use a variant of `expresso` called `expressopdnn`.

Which method is best?

Well, all of the above methods are implemented in Bioconductor.

we're going to try a few head to head comparisons later. In this context, it's worth thinking about how we can define a measure of "goodness". Hmm?

Quality control assessment

A critical part of the analysis of any set of microarray experiments is ensuring that the arrays are of reasonable quality. BioConductor includes several methods to assist with the QC effort for Affymetrix projects.

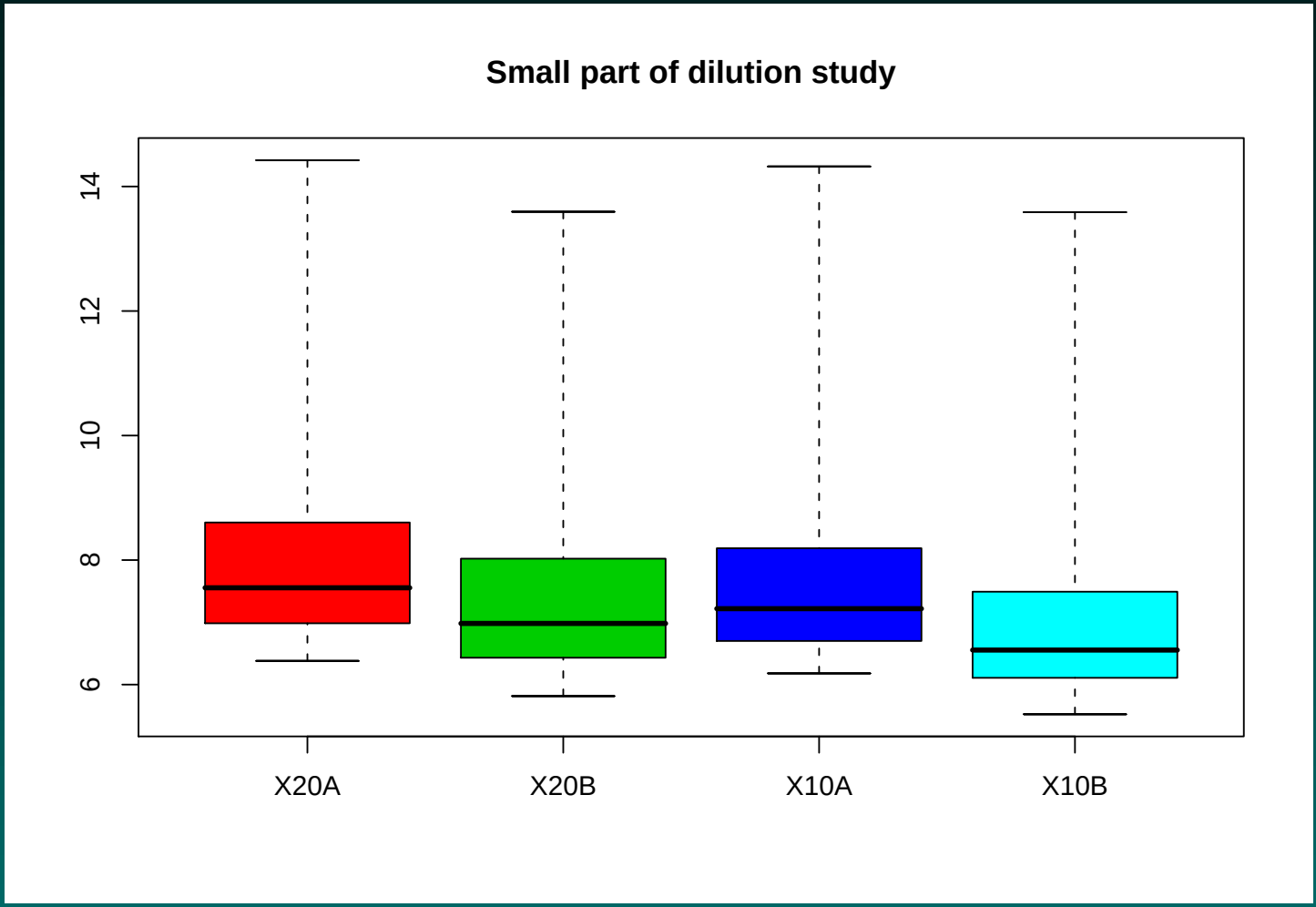
The first step is typically to look at the images, which we did in the previous lecture.

We can also look at the distributions of intensities to se how well they match.

BioConductor includes tools to compute some summary statistics that tell us about the relative quality and comparability of arrays

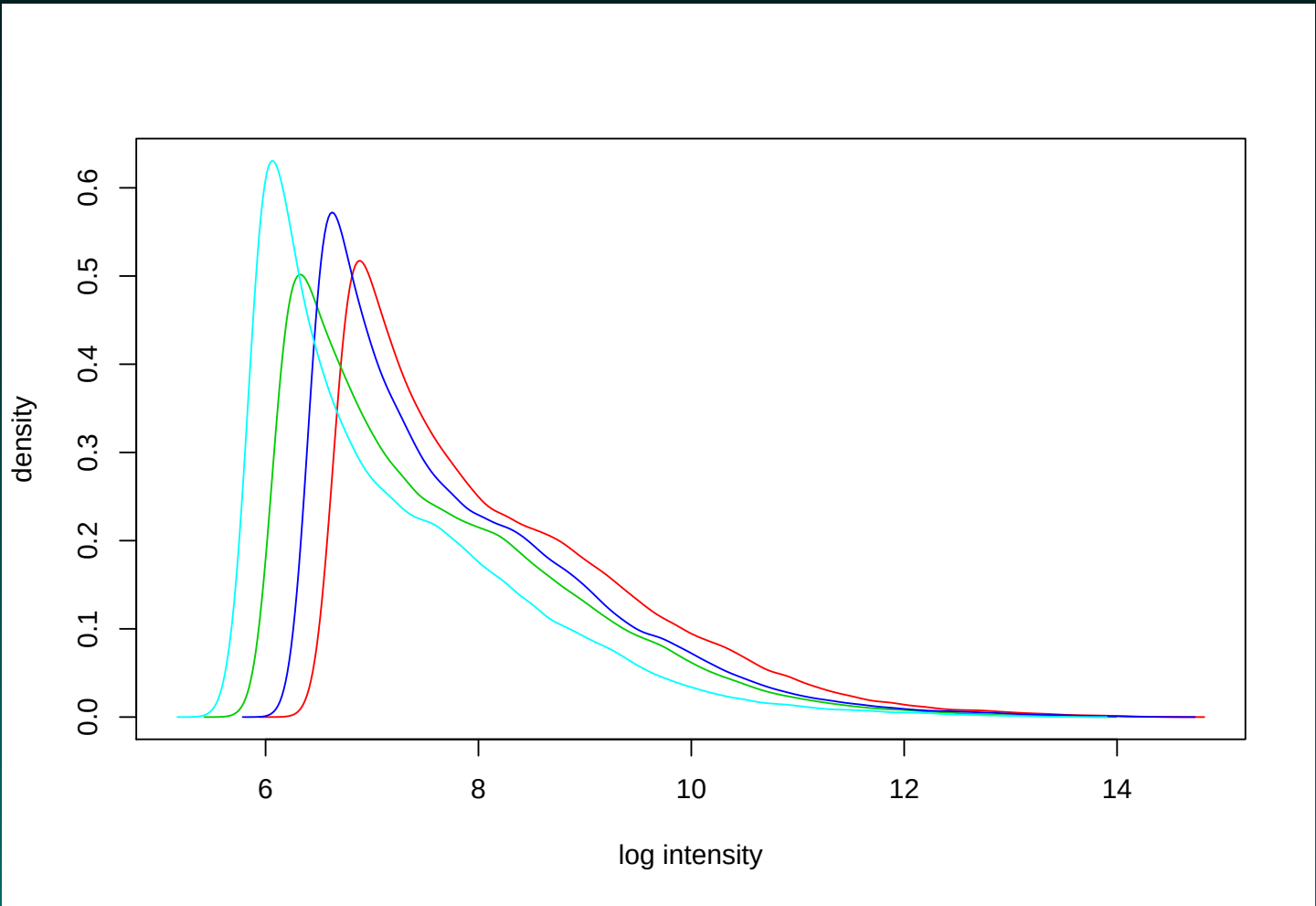
A summary view of four images

```
> boxplot(Dilution, col = 2:5)
```



The distribution of feature intensities

```
> hist(Dilution, col = 2:5, lty = 1)
```



Simple Affy

We can use the `simpleaffy` package to compute some QC summary statistics.

```
> require(simpleaffy)
```

```
[1] TRUE
```

```
> Dil.qc <- qc(Dilution)
```

Computing the metrics will take a little time, and then we can start to look at them.

Background

The first check we make is that the background values across the arrays should be roughly comparable. In the four arrays from the dilution experiment, that seems to be the case.

```
> avbg(Dil.qc)

     20A      20B      10A      10B
94.25323 63.63855 80.09436 54.25830

> maxbg(Dil.qc)

     20A      20B      10A      10B
97.66280 68.18998 83.24646 57.62283

> minbg(Dil.qc)
```

```
      20A       20B       10A       10B
89.52555 60.01397 77.32196 49.22574
```

Global scaling factors

As mentioned above, the standard Affymetrix normalization procedure involves globally rescaling the arrays to set the median probe intensity to the same level. Affymetrix says that the scaling factors should not difffer by more than 3-fold if we want to compare arrays.

```
> sfs(Dil.qc)

[1] 0.8934013 1.2653627 1.1448430 1.8454067
```

Percent present calls

Extremely low (below about 30%) or high (above about 60%) values for the percentage of probes called present also signal potential quality problems.

```
> percent.present(Dil.qc)
```

```
20A.present 20B.present 10A.present 10B.present
   48.79208    49.82178    49.37822    49.75842
```

3'/5' ratios

Affymetrix includes probes at the 3' and 5' ends of some control genes; the ratios should be less than 3.

```
> ratios(Dil.qc)
    AFFX-HSAC07.3'/5' AFFX-HUMGAPDH.3'/5'
20A         0.6961423          0.4429746
20B         0.7208418          0.3529890
10A         0.8712069          0.4326566
10B         0.9313709          0.5726650
    AFFX-HSAC07.3'/M AFFX-HUMGAPDH.3'/M
20A        0.1273385         -0.0602414
20B        0.1796231         -0.0136629
10A        0.2112914          0.4237527
10B        0.2725534          0.1125823
```

RNA degradation

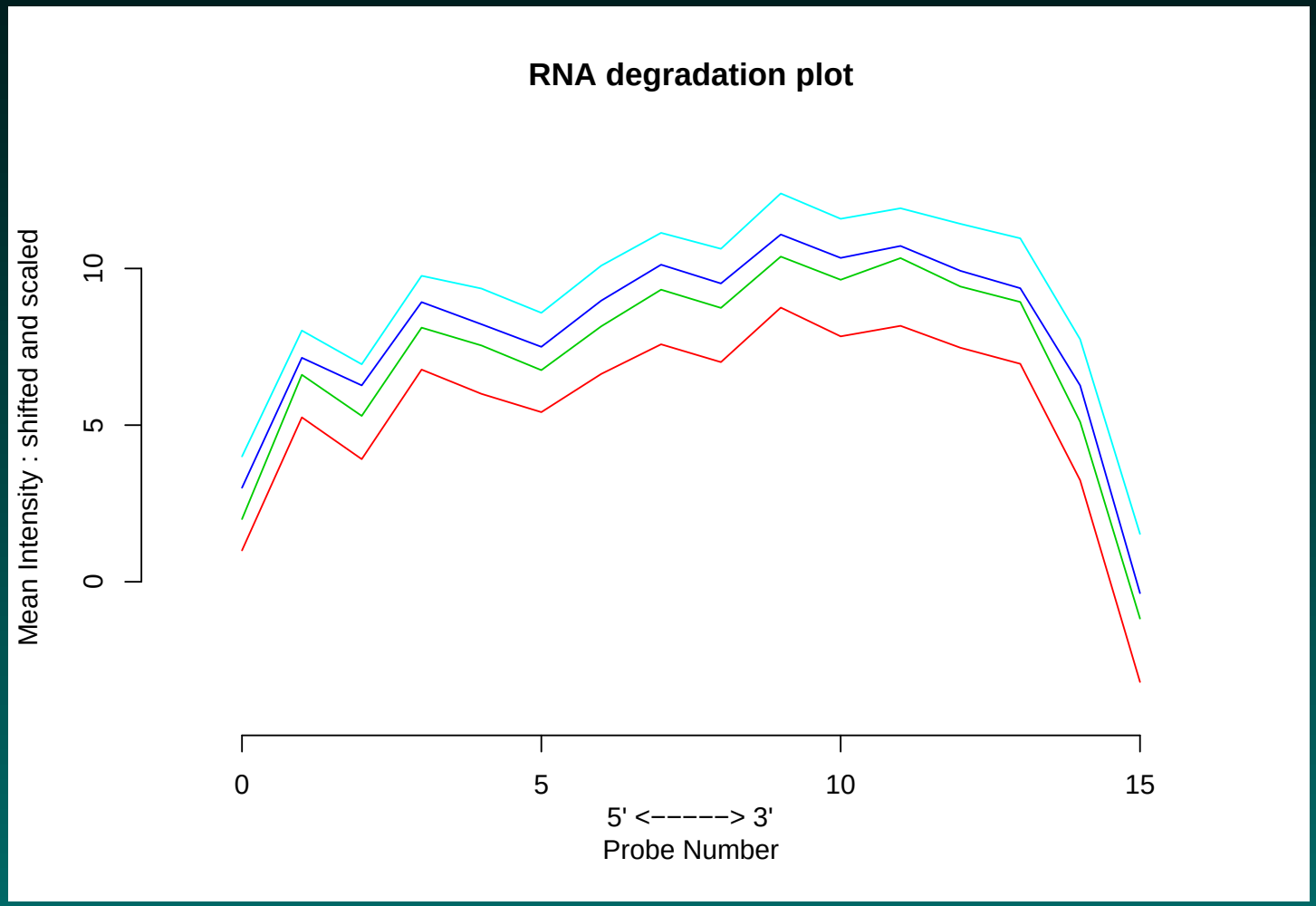
Individual (perfect match) probes in each probe set are ordered by location relative to the 5' end of the targeted mRNA molecule. We also know that RNA degradation typically starts at the 5' end, so we would expect probe intensities to be lower near the 5' end than near the 3' end.

The `affy` package of BioConductor includes functions to summarize and plot the degree of RNA degradation in a series of Affymetrix experiments. These methods pretend that something like "the fifth probe in an Affymetrix probe set" is a meaningful notion, and they average these things over all probe sets on the array.

```
> degrade <- AffyRNAdeg(Dilution)
```

Visualizing RNA degradation

```
> plotAffyRNAdeg(degrade, col = 2:5)
```



Model-based QC

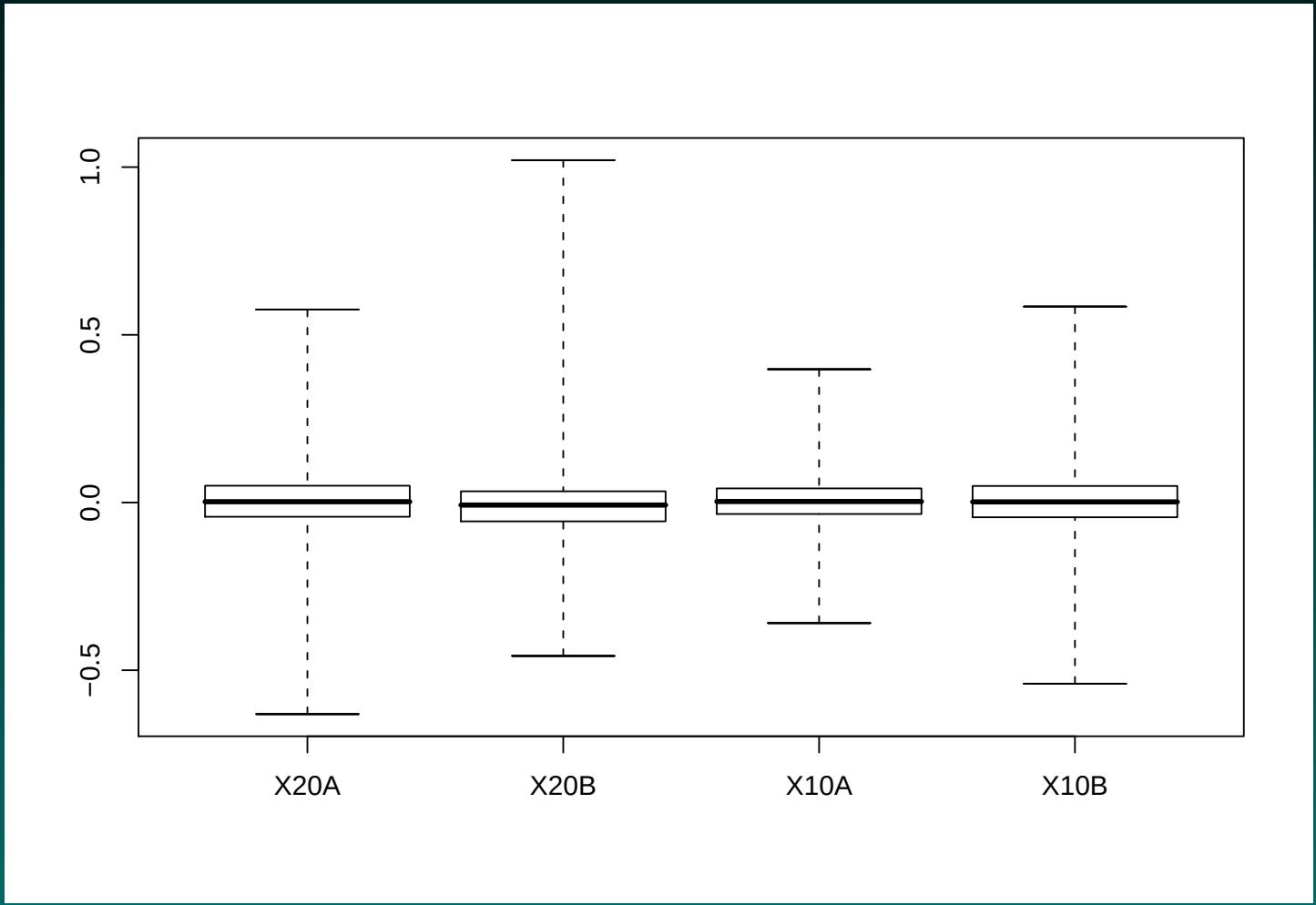
As we have seen, methods like dChip and RMA fit statistical models to the probe intensities in order to summarize gene expression values. The quantities associated with such models can also provide QC information. The BioConductor package `affyPLM` fits a probe-level model (PLM) similar to RMA.

```
> library("affyPLM")
> pset <- fitPLM(Dilution)
```

The residuals (unexplained variation) from the model can be plotted using the `image` function. Patterns here typically indicate spatial flaws in the image that are not captured by the model. No such features were noted in the Dilution experiment, so I will not reproduce the pictures.

Relative Log Expression plots

```
> Mbox(pset)
```



Normalized Unscaled Standard Error

```
> boxplot(pset)
```