Lab III: Pre-processing and Quality Assessment of Affymetrix GeneChip data

Ben Bolstad
bolstad@stat.berkeley.edu

June 30, 2005

Contents

1 Introduction

This lab introduces the basic tools provided for the low-level analysis of Affymetrix data. By the end of this lab you should be able to perform basic inspection of Affymetrix

datasets, process raw data to produce expression measures and assess the quality of the arrays. Some parts of this lab are marked *for reference only* and it is not necessary to execute any code given in those sections for successful completion of the lab. Instead these sections discuss techniques that are useful for an analyst wanting to apply these methods to there own data or carry out extended analysis. To begin this lab it is necessary to load the packages and dataset that will be used. Loading the affyPLM package will also load the other packages required in this analysis. The jsmHyperdip package contains the dataset which we use in this lab. Use:

```
> library(affyPLM)
> library(jsmHyperdip)
```

to load the packages. To make sure we have access to the data we most attach it to the current R session. This can be done by typing:

```
> data(jsmHyperdip)
```

More specific details about this dataset can be found by reading the help page: `?jsmHyperdip`.

This data is stored in an *AffyBatch* object. The *AffyBatch* object is used for storing raw probe-level data. It encapsulates raw probe-intensities and related phenotypic data into a single object. Many R functions can be applied to *AffyBatch* objects and this lab explores a subset of these methods. Typing the name of the object will show some of the information stored in the object:

```
> jsmHyperdip
```

To see the phenotypic data in this *AffyBatch* type:

```
> pData(jsmHyperdip)
```

which in this case contains sample numbers and the original filenames (in this lab the 6 arrays are referred to using the first 6 letters of the alphabet) for this data.

The `pm` and `mm` functions provide access to raw probe intensities. Calling the `pm` function with only the name of the `AffyBatch` returns all the PM probe intensities for all 6 arrays. However, displaying all the PM values on the screen may take considerable time and is not typical useful for interactive analysis. Often it is more useful to look at the probe intensities for a particular probeset. For instance, to see the PM intensities for all probes in the probeset *101_at* across all 6 arrays type:

```
> pm(jsmHyperdip, "101_at")
```

The PM probe intensities for a probeset can also be examined graphically. Specifically, for this same probeset use:

```
> par(mfrow = c(1, 2))
> matplot(pm(jsmHyperdip, "101_at"), type = "l",
+     ylab = "PM intensity", xlab = "Probe Number")
> matplot(t(pm(jsmHyperdip, "101_at")), type = "l",
+     ylab = "PM intensity", xlab = "Array Number")
```

to see the probe-intensity behavior for the probeset across Probe Number and across Array Number.

The functions `sampleNames` and `geneNames` show the names of the arrays and the names of the probesets on the array. Typing:

```
> sampleNames(jsmHyperdip)
> geneNames(jsmHyperdip)[1:10]
```

shows some of this information for the *jsmHyperdip* data.

1.1 Loading data using `ReadAffy` *for reference only*

For this lab the dataset has already been supplied to you in an R library and you don't need to worry about reading it into R. However, in general you start with CEL files (raw intensity data) and read it into R using the `ReadAffy` function. Reading all the CEL files in the current directory and storing can be accomplished using:

```
> my.Data <- ReadAffy()
```

the current working directory is shown by `getwd()` and may be set using `setwd()` (on Windows you may also do this using the *Change Dir ...* option in the File menu). The `filenames` argument is supplied to the `ReadAffy` function to read specific files only. For example:

```
> my.Data <- ReadAffy(filenames = c("A.CEL", "B.CEL",
+     "C.CEL"))
```

would read in 3 CEL files and store them in an *AffyBatch* called `my.Data`.

2 Exploratory Data Analysis

2.1 Raw Images

Before commencing more involved analysis it is advisable to examine the unprocessed data. A number of functions are provided for doing this in a variety of different ways. An examination of the raw images is the first step in the analysis. The `image` function can be used to examine log transformed PM probe intensities. To examine images for the first two chips in the dataset use:

```
> par(mfrow = c(1, 2))
> image(jsmHyperdip[, 1])
> image(jsmHyperdip[, 2])
```

Examining these two images we see that array B has a clear artifact.

Because of great differences in magnitude between intensities examining images of the untransformed data is less informative. Examining chip B in this manner obscures the previously obvious artifact:

```
> par(mfrow = c(1, 1))
> image(jsmHyperdip[, 2], transfo = function(x) {
+     x
+ })
```

2.2 Histogram and Boxplots

It is also often useful to graphically examine the distribution of probe intensities for each array in the dataset. The first method for doing this is to use the `hist` function. Typing:

```
> hist(jsmHyperdip, col = 1:6, main = "Histogram of log2 PM probe intensities")
> legend(13, 1.5, sampleNames(jsmHyperdip), col = 1:6,
+     lty = 1:5, lwd = 2)
```

gives the histograms for the *jsmHyperdip* dataset. Most differences in position and spread are typically removed by normalization and do not indicate potential problems. However, histograms that have significantly different shapes are often of lesser quality. In this case we see that the histogram for array B has a shape that is distinctly different from the others. This is the array observed earlier to have a large artifact.

Boxplots can also be used to examine probe-intensity distributions. Applying the `boxplot` function to the *jsmHyperdip* dataset using:

```
> boxplot(jsmHyperdip)
```

shows differences in the spread and center across the arrays. As we shall see momentarily, most of the differences visible on these boxplots are reduced after normalization.

2.3 MA plots

On two channel microarray platforms MA plots are used to compare the two color channels on each arrays. Since there is only a single channel on an Affymetrix chip MA plots can not be used in the same way. Instead, MA plots are used to compare between chips. In this context M values are log fold changes and A values are average log intensities between two arrays. While it would be possible to look at MA plots for every possible pair of arrays, this will be an immense number of plots for any dataset consisting

of more than arrays. To reduce the number of possible comparisons a probewise median array can be created and then each array compared to this pseudo-array. The `MAplot` function creates these plots. For the *jsmHyperdip* data type:

```
> par(mfrow = c(2, 3))
> MAplot(jsmHyperdip)
```

3 Producing Expression Values

All the functions that produce expression values take an *AffyBatch* object and return an *exprSet* object. Expression values, phenotypic and other related information are stored in *exprSet* objects. Routines that produce expression values usually carry out a specific sequence of pre-processing steps: background correction, normalization and finally summarization.

3.1 RMA

A popular expression measure known as RMA is computed using the `rma` function.

```
> eset.rma <- rma(jsmHyperdip)
```

Typing the name of the *exprSet* at the command line will show some basic information about the stored expression values. The computed expression values stored in an *exprSet* object can be accessed using the `exprs` function. Using:

```
> eset.rma
> exprs(eset.rma)[1:10, ]
```

shows some of this information for the RMA values computed for the *jsmHyperdip* dataset.

As we noted before, often normalization removes significant differences in position and spread between arrays. Using:

```
> par(mfrow = c(1, 2))
> boxplot(jsmHyperdip, col = "red")
> title("Raw PM probe intensities")
> boxplot(eset.rma, col = "blue")
> title("RMA expression values")
```

shows this to be true for the RMA expression values we computed.

3.2 GCRMA

Another popular expression measure is known as GCRMA. It replaces the background correction step in RMA with a more sophisticated algorithm which utilizes probe sequence information. To compute GCRMA expression values for the *jsmHyperdip* data use:

```
> eset.gcrma <- gcrma(jsmHyperdip)
```

3.3 Other expression measures *for reference only*

While RMA and GCRMA are the expression measures most widely used by users of this software, they are certainly not the only possibilities. Another popular expression measure algorithm, proposed by Affymetrix, is called MAS 5.0. In the BioConductor software this is implemented in the `mas5` function. For example to compute MAS 5.0 expression values for the *jsmHyperdip* dataset use:

```
> eset.mas <- mas5(jsmHyperdip)
```

Some users prefer to construct their own expression measure by combining their own sequence of pre-processing steps. Two functions `expresso` and `threestep` are available for this purpose. Of the two functions `expresso` provides slightly greater flexibility while `threestep` is faster and more memory efficient.

For example suppose you wanted expression values where you used the MAS 5.0 background correction, a quantile normalization step, subtract the Mismatch (in the manner implemented by MAS 5.0) and then averaged the PM probe intensities. You could do this using:

```
> eset1 <- expresso(jsmHyperdip, bgcorrect.method = "mas",
+     normalize.method = "quantiles", pmcorrect.method = "mas",
+     summary.method = "avgdiff")
```

An expression measure consisting of the GCRMA background correction, a scaling normalization and then summarization by taking the log of the 2nd highest PM probe intensity could be computed using:

```
> eset2 <- threestep(jsmHyperdip, background.method = "GCRMA",
+     normalize.method = "scaling", summary.method = "log.2nd.largest")
```

For fuller details on these functions the user is referred to the Vignettes named *affy: Built-in Processing Methods* and *affyPLM: the threestep function*. Both can be accessed by typing `openVignette()` or on the web at`http://www.bioconductor.org/viglistingindex.html`.

4 Assessing Dataset Quality

The probe-level modeling procedures provided by the affyPLM give useful tools for dataset quality assessment. The primary routine for this purpose is the function `fitPLM` which operates on `AffyBatch` object and returns a `PLMset` object. To fit the default model on our dataset using `fitPLM` you should type:

```
> Pset <- fitPLM(jsmHyperdip)
```

For quality assessment purposes we will use the standard errors, weights and residuals stored in the `PLMset` object. Raw access to these values are available using the `se()`, `weights()` and `resids()` functions. The quality assessment tools are based on visual representations of these quantities.

4.1 Images of weights and residuals

While images of the raw probe intensities make some artifacts clearly visible others are invisible. More useful are chip pseudo-images created using the weights and residuals stored in the `PLMset` object. These can be created using the `image()` function. By default the weights are imaged. To create images of all 6 chips in this dataset use:

```
> par(mfrow = c(2, 3))
> image(Pset)
```

For the weights images, darker green means lower weight. The artifact we saw earlier is clearly visible on the chip pseudo-image for array B. Instead of using the weights, we could look at residuals by using the `type="resids"` argument. Residual images for the six chips in our data set are created by using:

```
> par(mfrow = c(2, 3))
> image(Pset, type = "resids")
```

Residuals images are colored so that high positive residuals are red, low negative residuals are blue and residuals close to 0 are white. Look closely at the chip image for array A (you may need to enlarge the graphic window), do you notice anything?

Using the `which` argument we can more closely examine the images for array A. Using `type="pos.resids"` gives only the positive residuals, `type="neg.resids"` gives the negative residuals and `type="sign.resids"` looks only at the sign of the residuals. These images for array A can be produced using:

```
> par(mfrow = c(2, 2))
> image(Pset, which = 1, type = "resids")
> image(Pset, which = 1, type = "pos.resids")
> image(Pset, which = 1, type = "neg.resids")
> image(Pset, which = 1, type = "sign.resids")
```

The artifact identified on this array is not easily visible in the image of raw probe intensities.

4.2 NUSE: Normalized Unscaled Standard Errors

One method of deciding whether or not an array is problematics from a quality standpoint is NUSE. The goal of NUSE is to identify any arrays which have elevated standard errors relative to other arrays in the dataset. This is done by standardizing the SE across arrays to have median 1 for each probeset. Our graphical tool consists of boxplots of these quantities for each array. Type:

```
> par(mfrow = c(1, 1))
> NUSE(Pset)
> title("NUSE for jsmHyperdip dataset")
```

to create the plot. You will notice that array B has a discordant boxplot. This indicates it is of poorer quality relative to the rest of the dataset (not surprising given the large artifact). While we also saw a recognisble artifact on array A, because it was of small size, it did not have large appreciable effect on the overall quality of the array.

Instead of visually examining these quantities suitable numerical summaries such as the median and IQR NUSE could be used. These can be found using:

```
> NUSE(Pset, type = "stats")
```

4.3 RLE: Relative Log Expression

Another tool for making a decision about whether an array should be removed from subsequent analysis because of poor quality is RLE. These are the log-scale expression values relative to the median expression value computed on a probeset by probeset basis. As with NUSE, we examine these quantities using boxplots. For our data this is accomplished using:

```
> RLE(Pset)
> title("RLE for jsmHyperdip dataset")
```

As before, array B has a significantly different

The median and IQR RLE may also be used in place of the boxplots. To do this use:

```
> RLE(Pset, type = "stats")
```

5 Memory Usage *for reference only*

Probe-level analysis of Affymetrix data is very memory intensive. In general you will be able to handle more arrays with more memory. For this lab most activities should be successfully handled using 512MB of RAM.

Note that on Windows operating systems you have some control over the maximum memory available to R. Specifically, by default R will be restricted to the smaller of

the amount of physical RAM and 1 GB. The function `memory.limit()` can be used to alter this limit when R is running. Command line arguments can be supplied to change these limits at launch time. For further details refer to the R for Windows FAQ at `http://cran.r-project.org/bin/windows/base/rw-FAQ.html`.

On Linux and other Unix based operating systems you are not required to set the maximum memory size.

Two additional functions, `justRMA` and `justGCRMA` are available for computing RMA and GCRMA expression measures respectively. These compute the expression measures directly from CEL files. This makes them much more memory efficient but you will not be able to carry out additional probe-level analysis.

6 Where To Go For More Information

1. `http://www.bioconductor.org/viglistingindex.html`: BioConductor Vignettes
2. `http://www.stat.berkeley.edu/~bolstad/PLMImageGallery/`: An image gallery showing chip quality to images for many arrays and complete datasets.
3. `http://www.stat.berkeley.edu/users/~bolstad/RMAExpress/RMAExpress.html`: RMAExpress is a standalone GUI application for computing RMA expression values.
4. `http://www.affymetrix.com`: The Affymetrix website.
5. Bolstad, B. M., Irizarry, R. A., Astrand, M., and Speed, T. P., A comparison of normalization methods for high density oligonucleotide array data based on variance and bias, Bioinformatics, 19, 185 (2003). : Paper discussing normalization methods
6. Cope, LM, Irizarry, RA, Jaffee, H, Wu, Z, Speed, TP (2004) A Benchmark for Affymetrix GeneChip Expression Measures. Bioinformatics 20: 323-331.: Paper discussing how to compare different expression measures. See also the related website at `http://affycomp.biostat.jhsph.edu/`
7. Irizarry, RA, Bolstad BM, Collin, F, Cope, LM, Hobbs, B, and, Speed, TP (2003) Summaries of Affymetrix GeneChip Probe Level Data. Nucleic Acids Research. Vol. 31, No. 4 e15: Discusses specific comparisons between RMA, MAS 5.0 and dChip.
8. Wu, Z., Irizarry, R., Gentleman, R., Martinez Murillo, F. Spencer, F. A Model Based Background Adjustment for Oligonucleotide Expression Arrays. Journal of American Statistical Association 99, 909-917 (2004).: Discusses the GCRMA expression measure.