

# R/BioC Exercises : Two Color Microarrays

Emiel Ver Loren van Themaat and Perry Moerland

April 15, 2010

☞ This document and information on how to log on to a PC in the exercise room and the SARA server can be found here:

<http://bioinformaticslaboratory.nl/twiki/bin/view/BioLab/EducationBioinformaticsII>

## 1 Introduction

This tutorial guides you through the first steps of a two color microarray analysis. We will follow the structure of the analysis performed in Chapter 4 in the book. We will describe various tools and functions available in R/Bioconductor for reading and storing data from two-color spotted arrays and present packages for preprocessing these data sets. The term preprocessing is often used to refer to the quality assessment and normalization of the microarray data. You are advised to make full use of the *help* function (*help(topic)*) to obtain information about functions and *vignette(package)* to get introductory material for a certain package. Before starting the exercises it is useful to recall some of the properties of two color arrays. In Chapter 4.2 of the book a short summary is given.

### 1.1 The integrin $\alpha 4/\beta 7$ dataset

The integrin  $\alpha 4/\beta 7$  dataset is used in this tutorial. This experiment aims to study the cell adhesion molecule integrin  $\alpha 4/\beta 7$  which assists in directing the migration of blood lymphocytes to the intestine and associated lymphoid tissues. The goal of the study is to identify differentially expressed genes between the  $\alpha 4/\beta 7+$  and  $\alpha 4/\beta 7-$  memory T helper cells. Each hybridization involved  $\beta 7+$  cell RNA from a single subject (labeled with one dye) and  $\beta 7-$  cell RNA from same subject (labeled with the other dye). In total 6 microarrays are used in our analysis. See Chapter 4.2 for references with detailed information about the biology of the integrin  $\alpha 4/\beta 7$  dataset and the design of the microarray.

The type of microarray used for this experiment is called a cDNA microarray. The construction of this type of array is done by printing cDNA on spots on the array. Spots are printed in parallel using a print head containing 48 print tips (12 x 4). Printing proceeds from top to bottom and right to left in each print tip grid containing 23 rows and 21 columns. The print head takes up cDNA from standard 384-well plates, each individual plate supplying the DNA for 8 (384/48) consecutive spots within each tip group. On our array there are in total 483 spots per print tip.

## 1.2 Starting R

Before starting R, you will need to copy some files to your local directory. It is handy to create a separate directory on your account for this. After logging onto SARA type the following commands:

```
mkdir Integrin
cd Integrin
cp /data/home/stud00/Integrin/* ~/Integrin
```

Now start R version 2.10:

```
R 2.10
```

and load the libraries `limma`, `arrayQuality`, `sma`.

```
> library(limma)
> library(arrayQuality)
> library(sma)
```

## 1.3 Loading scan results

Raw scans of microarrays are stored as images. Bioconductor does not provide much image processing utilities and relies on other software. Usually each scanner is accompanied by software that can estimate intensity properties for each spot on the chip, such as average spot intensities for the red and green channel. Each image processing software program outputs its data in different file formats. Most common formats are supported by special *read* functions in Bioconductor, including GenePix's `.gpr`, Spot's `.spot`, SMD's `.xls`, and Agilent's `.txt` files. The Integrin arrays are scanned and pre-processed by GenePix.

In R special classes are designed which can store an entire experiment containing multiple microarrays. We already created such an R object that stores both scan results of all chips (6 in total), experimental information and design of the microarray. This object is stored in the package `beta7` and can be retrieved with the following commands:

```
> library(beta7)
> rawData <- beta7
```

This object contains all the information of the GenePix files of the integrin data. The class structure of this object is rather complex and is explained in Chapter 4.3.5. For example, `rawData` has an attribute (slot) `maGf` of class *matrix* which stores the green foreground channel for each array and can be accessed by either calling:

```
> rawData@maGf
```

or

```
> maGf(rawData)
```

**Exercise 1:** Try to retrieve answers to the following questions by examining the `rawData` object.

- What is the class type of the `rawData` object?
- Which type of cell is hybridized onto the green channel in the microarray with subjectID 8?
- Retrieve the expression value of the 6th spot on the 2nd array for the red foreground channel.
- How many spots does each array contain?
- Which five categories of spots are present on the array?

☞ Hint: Functions `summary`, `str`, `class`, and `getClass` might come in handy.

📖 Each class type in R requires that an object belonging to that class has a pre-defined structure containing a specific set of attributes (also called slots). For example, when making an object of class `matrix` its attribute `dim`, containing the number of rows and columns, needs to be instantiated. In an object of class `marrayRaw` the attributes `marrayRaw@maRf` and `marrayRaw@maGf` need to be matrices. Because of these requirements generic functions like `plot` and `summary` are customized for different classes and can thus be class-specific. The other way around also holds; most functions require arguments of a specific class. Function `help("function")` shows the required class type of the arguments of the “function”. Often functions are available to convert classes of objects (`as` and `new` for example).

## 2 Quality assessment

Before analyzing the data, it is important to check whether the experimental data is of acceptable quality or if there is a need to repeat some hybridizations. Functions in R have been developed to assess the quality of individual microarrays stored in `marrayRaw` objects. The purpose of this part is to show some graphical methods in context of two-color arrays. You can play around with the arguments given to the functions. Most of the code below is copied from Chapter 4.4.

### 2.1 Spatial inspection

The function `image` can be used to spatially visualize the arrays and thus may reveal artifacts on the array.

```
> image(rawData[, 3], xvar = "maGf")
```

Each spot within each grid in the image is spotted with the same print tip. The GenePix scanner also assigns a weight to each spot on each array. This weight can be viewed as a quality score and is stored in the `rawData` object. By assigning a vector of Booleans to the argument `overlay` in the function `image` the low-quality spots are highlighted.

```
> flags <- rawData@maW[, 3] < -50
> image(rawData[, 3], xvar = "maGf", overlay = flags)
```

## 2.2 Boxplots

Boxplots summarize data with five values: smallest observation, lower quartile (Q1), median, upper quartile (Q3), and largest observation (and outliers). Boxplots of spot measurements per print-tip-group, per plate or per slide can be used to identify hybridization artefacts or biases. To view differences in the distribution of intensities between arrays, use:

```
> boxplot(rawData, yvar = "maGf", las = 2, log = "y")
```

To view the differences between plates type:

```
> par(mar = c(5, 3, 3, 3), cex.axis = 0.7)
> boxplot(rawData[, 3], xvar = "maPlate", yvar = "maGf", outline = FALSE,
+         las = 2, log = "y")
```

As can be seen in the plots, the last plates have a much lower intensity distribution in this microarray. This is due to the overrepresentation of negative control spots on these plates.

**Exercise 2:** Try to highlight the spots with “Empty” controls using function *image* and argument *overlay*.

## 2.3 MA-plots

To see if a gene (that is, spot) has a higher expression in the green or red labeled sample we can look at ratios. First plot, for example, the Cy5 signal on the  $x$ -axis and the Cy3 signal on the  $y$ -axis for all spots in the third array:

```
> plot(x = rawData@maRf[, 3], y = rawData@maGf[, 3], pch = ".")
```

If a spot has an equal intensity in red and green, the point representing that spot would lie along the line  $x = y$ . We can plot the distribution of this ratio:

```
> hist(rawData@maRf[, 3]/rawData@maGf[, 3])
```

From this histogram one can see that taking the logarithm would make the distribution more symmetrical (for up and down regulation):

```
> hist(log2(rawData@maRf[, 3]/rawData@maGf[, 3]))
```

The log of a ratio is often referred to as the  $M$ -value. The letter  $M$  is used because a log of  $R/G$  is equal to  $\log(R)$  Minus  $\log(G)$ . A common effect in 2-dye microarrays is a dye effect: the intensity distribution of the dyes within an array are not equal, even when the same sample is used for red and green. This often results in a shift of means but also in a nonlinear effect, dependent on the actual intensity. To illustrate this we can plot the  $M$ -value on the  $y$ -axis, and the average log intensity (also known as  $A$  value) on the  $x$ -axis. This plot is actually a rotation of 45 degrees of the RG-plot and is known as the MA-plot.

```
> plot(y = log2(rawData@maRf[, 3]/rawData@maGf[, 3]), x = (log2(rawData@maRf[,
+ 3]) + log2(rawData@maGf[, 3]))/2, pch = ".")
```

In the next section we will try to correct for this effect and other technical effects.

**Exercise 3:** In Section 4.4.1 of the book the wrapper function *maQualityPlots* is described. This function creates sheets with MA-plots, density plots and images of the arrays. Use that function and examine the produced results. The results written to your directory can be viewed with either WinSCP (preferred) or by using the function *browseURL* (but this might be very slow):

```
> browseURL("file:///data/home/studXX/Integrin/")
```

In the browser you can click on the png files to view the diagnostic plots. Do you think it is reasonable to compare the arrays with each other, based on the MA-plots?

¶ For those interested, the actual code of some functions can be viewed by typing the function in R without any brackets at the end. For example, the code of *maQualityPlots* can be viewed in R by typing

```
> maQualityPlots
```

## 3 Normalization

### 3.1 RGLists

For normalization we usually use a different type of class called (*RGList*), which can also store batches of array data and its annotation. This class type is used by the package *limma* which contains many normalization functions. Luckily conversion methods are available between objects of class *marrayRaw* and *RGList*. Type the following commands:

```
> library(limma)
> library(convert)
> marrayRG <- as(rawData, "RGList")
```

**Exercise 4:** • *limma* also provides the function *read.maimages* to read the data from GenePix files and store it in an object of class *RGList*. Read the description of this function carefully and use this function to read in the .gpr files stored in your local directory. Remember that the filenames are stored somewhere in the `rawData` object.

- Can you think of a reason why there is a difference between the values for the red channel in `marrayRG` (created with function *read.GenePix*) and the object created with *read.maimages*?
- What is the main difference in the design of the two classes *marrayRaw* and *RGList* in terms of attaching attributes to the object?

The role of normalization is to remove technical variation while retaining biological signal. Possible sources of technical variations in 2-dye spotted arrays are: different labeling efficiencies and scanning properties of Cy3 and Cy5 dyes, different scanning parameters, print-tip, spatial and plate effects. We can divide the normalization procedures into two groups: within-array normalization and between-array normalization.

### 3.2 Within-array normalization

Within-array normalization adjusts the within-array contrasts  $M$  using values of  $A$  as well as other factors such as print-tip and spatial position of the plates. It adjusts the center and spread of distribution of the log-ratios ( $M$ ), while taking  $A$  into account. Most of these methods are based on loess (LOcal regrESSion). The approach can best be visualized in the  $M$  versus  $A$  plot:

```
> M <- log2(marrayRG$R[, 6]/marrayRG$G[, 6])
> A <- (log2(marrayRG$R[, 6]) + log2(marrayRG$G[, 6]))/2
> plot(A, M, pch = ".")
> lines(lowess(A, M), col = "red")
```

Normalization is performed by shifting the data such that the fitted regression line becomes equal to the line  $M = 0$ . So, basically, the assumption is made that log-ratios are equally distributed around 0, where the red intensity is equal to green. Given that most microarrays measure over 30,000 genes this assumption can be made for most experiments without inserting too much bias.

For the integrin dataset, we perform a loess normalisation for each print-tip using the built-in function *normalizeWithinArrays*. To get a new object containing the normalized data:

```
> normLimma <- normalizeWithinArrays(marrayRG, method = "printtiploess",
+   bc.method = "none", weight = NULL)
```

**Exercise 5:** Let us try to visualize the normalized data:

- Inspect the class structure and type of the `normLimma` object.
- Create a MA-plot of the normalized data for the sixth array.
- Add a loess line to the plot.
- Try to make one MA-plot for all arrays using function *plotMA3by2* (read the help of this function first and set argument *zero.weights* to TRUE).
- Do you think it is now reasonable to compare the arrays?

### 3.3 Between-array normalization

Location normalization centers log-ratios around  $M = 0$  by accounting for intensity and spatial bias. However, as can be seen in the last plot, it does not adjust for differences in scale of spread of  $M$ -values between arrays. These scale differences can be caused by small changes in scanner settings, for example. *Between-array normalization* addresses this effect by improving the comparability of the distributions of log intensities or log ratios between arrays. Information from the distributions in all arrays is used to normalize each array. One solution forces that the median absolute deviation of the  $M$ -values and  $A$ -values is the same across arrays <sup>1</sup>:

---

<sup>1</sup>For more information on the MAD procedure: type *?normalizeBetweenArrays* or see Section 6 and Figure 6 in: <http://www.statsci.org/smyth/pubs/normalize.pdf>.

```
> scaleLimma <- normalizeBetweenArrays(normLimma, method = "scale")
```

Another solution, quantile normalization on the separate channels, ensures that the intensities of the separate channels - instead of the log ratios - have the same distribution across arrays and across channels:

```
> quantileLimma <- normalizeBetweenArrays(normLimma, method = "q")
```

**Exercise 6:** Use the functions *plotDensities* and *plotMA3by2* (make sure you set argument *zero.weights* to TRUE) to visualize log intensities and log ratios in the *marrayRG*, *normLimma*, *scaleLimma* and *quantileLimma* objects (tip: use *x11()* to start a new graphics device). Answer the following questions:

- Try to explain the differences between the plots of *normLimma*, *scaleLimma* and *quantileLimma*.
- Which part of pre-processing did we leave out compared to the book?
- It is also possible to perform a quantile normalisation on the intensities in the red channel only. Why is this not particularly useful for the integrin  $\alpha4/\beta7$  experiment? Which kind of experimental design would benefit the most from such a normalization?

**i** When using all these kind of statistics to force the data into certain distributions, noise can be inserted and/or true biological signals might be removed. Experimental design and diagnostic plots are of great importance for choosing the right set of normalization procedures.

## 4 Extra questions

For those who have gone smoothly through the exercises and still have time left we created some extra exercises. These exercises are categorized into two parts: one open exercise and an exercise containing handy R tricks.

**Exercise 7:** The directory */data/home/stud00/Agilent/* contains four files. These files contain raw data from four Agilent arrays. Try to get them into a *marrayRaw* object and make diagnostic plots. When you examine the background intensities, do you notice anything?

**Exercise 8:** The R environment can be used for multiple purposes. The application of statistical algorithms and the visualization of data are among these functionalities. To improve calculation time of statistical algorithms on matrices, lists or dataframes, the functions *apply* and *lapply* are built into R. Function *apply* can apply some specified function to each row or column separately, and return a vector or list of the results. The apply functions basically speeds up *for* loops:

```
> x <- c()
> for (i in 1:6) {
+   x <- c(x, mean(marrayRG$R[, i]))
+ }
> x
```

```
[1] 1075.6665 1392.6667 944.6394 548.5286 988.0307 1574.0515
```

For large matrices the above for-loop is slower because it needs to be translated to a low-level language like C in every iteration, while `apply` does this immediately. To get the mean expression per gene (=row) in the matrix containing the red channel intensities using `apply`:

```
> gene.means <- apply(marrayRG$R, 1, mean)
```

or per array (=column):

```
> array.means <- apply(marrayRG$R, 2, mean)
```

- Try to use `apply` to get the median  $M$  expression per gene. You can use any function you like, including your own functions.
- Make a function which calculates the mean of logs and then use `apply`:  

```
> array.mean.logs <- apply(marrayRG$R, 2, my_function)
```
- Show that the mean of logs is different from the log of means.
- Use `apply` to calculate the standard deviation and mean of the  $R$ -value for every gene using your function `myFirstFunction` written yesterday.

Here is a little puzzle for the `apply` function: make a matrix `mat` with 10 rows and 6 columns which results in:

```
> apply(mat, 1, mean)
```

```
[1] 35 34 33 32 31 30 29 28 27 26
```

```
> apply(mat, 2, mean)
```

```
[1] 55.5 45.5 35.5 25.5 15.5 5.5
```

```
> sessionInfo()
```

```
R version 2.10.1 (2009-12-14)
```

```
i386-pc-mingw32
```

```
locale:
```

```
[1] LC_COLLATE=English_United Kingdom.1252
```

```
[2] LC_CTYPE=English_United Kingdom.1252
```

```
[3] LC_MONETARY=English_United Kingdom.1252
```

```
[4] LC_NUMERIC=C
```

```
[5] LC_TIME=English_United Kingdom.1252
```

```
attached base packages:
```

```
[1] grid      stats      graphics  grDevices  utils      datasets  methods
```

[8] base

other attached packages:

[1] beta7_1.0.0	sma_0.5.15	arrayQuality_1.24.0
[4] RColorBrewer_1.0-2	gridBase_0.4-3	hexbin_1.20.0
[7] lattice_0.17-26	convert_1.22.0	Biobase_2.6.1
[10] marray_1.24.0	limma_3.2.3	