

Lab 1 - Pre-processing and Normalization

In this lab we will examine a collection of **R** packages for exploratory analysis and normalization of two-color cDNA microarray fluorescence intensity data. Some of these packages were developed as part of the Bioconductor project, which is distributed under an open source licenses such as GPL2 or BSD and may be downloaded from the project website <http://www.bioconductor.org>. Source code, binaries, and documentation for R and other R packages can be obtained via the "Comprehensive R Archive Network" (CRAN): <http://cran.r-project.org/>. As with any other R package, detailed information on the functions and their arguments and values can be obtained in the help files. For instance, to view the help file for the function `maNorm` in a browser, use `help.start()` followed by `?maNorm`, or simply type `help(maNorm)`.

For Bioconductor packages, an extended tutorial is provided in the `/docs` subdirectory of each package. These tutorials are generated using the `Sweave` function from the package `tools`, which allows automatic report generation mixing text and R code. In this lab, we will examine a data set `swirl` provided by Katrin Wuennenberg-Stapleton from the Ngai Lab at UC Berkeley. This experiment contains two sets of dye-swap experiments for a total of 4 replicate hybridizations. For each of these hybridizations, a comparison is made between the zebrafish mutant `swirl` and the corresponding *wild-type* (`wt`). The raw data is available in the package `marrayInput` and at <http://www.stat.berkeley.edu/~terry/zarray/Course/index.html>. This tutorial is based on S. Dudoit and Y. H. Yang (2002). *Bioconductor R packages for exploratory analysis and normalization of cDNA microarray data*. In G. Parmigiani, E. S. Garrett, R. A. Irizarry and S. L. Zeger, editors, *The Analysis of Gene Expression Data: Methods and Software*, Springer, New York (To appear)

1 Part I - Data Input

1. To begin, we need to load a few libraries:

```
R> library(marrayNorm)
R> library(marrayPlots)
```

2. Getting Class definition for the 4 packages. There are a few classes defined in order to keep track of the large amount of information associated with the microarray. Some of the more frequently use classes are:

- **marrayLayout:** The class `marrayLayout` was designed to keep track of various layout parameters such as the dimensions of the spot and grid matrices, and, for each probe on the array, its grid matrix and spot matrix coordinates. In addition, it is used to keep track of gene names, plate origin of the probes, and information on the spotted control sequences.
- **marrayInfo:** Information on the target mRNA samples co-hybridized to the arrays are stored in objects of the class `marrayInfo`. Such objects may include the names of the arrays, the names of the Cy3 and Cy5 labeled samples, notes on the hybridization and scanning conditions, and other textual information.
- **marrayRaw:** Pre-normalization intensity data for a batch of arrays which contain slots for the Cy3 and Cy5 foreground and background intensities (`maGb`, `maRb`, `maGf`, `maRf`), spot

quality weights (`maW`), layout parameters of the arrays (`marrayLayout`), description of the probes spotted onto the arrays (`maGnames`) and mRNA samples hybridized to the arrays (`maSamples`).

- **marrayNorm:** Post-normalization intensity data are stored in similar objects of class `marrayNorm`. These objects store the normalized intensity log-ratios `maM`, the location and scale normalization values (`maMloc` and `maMscale`), and the average log-intensities (`maA`). In addition, the `marrayNorm` class has a slot for the function call used to normalize the data, `maNormCall`.

A graphical representation of these classes can be found in the file `labT1.ppt`. The following functions get the Class definition:

```
R> getClassDef("marrayLayout")
R> getClassDef("marrayInfo")
R> getClassDef("marrayRaw")
R> getClassDef("marrayNorm")
```

3. In general, the function `new` from the `methods` package may be used to create new objects from a given class. For example, to create an object of class `marrayLayout` describing the array layout of an experiment, one could use the following code

```
R> my.layout <- new("marrayLayout", maNgr = 4, maNgc = 4, maNsr = 24,
+      maNsc = 30)
```

4. A more complicated example is to create an object of class `marrayInfo` describing the *swirl* experiment.

```
R> zebra.RG <- as.data.frame(cbind(c("swirl", "WT", "swirl", "WT"),
+      c("WT", "swirl", "WT", "swirl")))
R> dimnames(zebra.RG)[[2]] <- c("Cy3", "Cy5")
R> zebra.samples <- new("marrayInfo", maLabels = paste("Swirl array ",
+      1:4, sep = ""), maInfo = zebra.RG, maNotes = "NO NOTES")
R> zebra.samples
```

5. Print methods for microarray objects. Since there is usually no need to print out thousands of fluorescence intensities, the `print` method was overloaded for microarray classes by simple report generators. For example, summary statistics for an object of class `marrayRaw`, such as `swirl`, can be obtained by:

```
R> data(swirl)
R> swirl
```

6. Subsetting methods for microarray objects. To access only a subset of arrays in a batch and/or spots in an array. Subsetting methods `[` were defined for this purpose. For instance, to access the first 10 probe sequences (spots) or only the second array in the batch `swirl` use

```
R> swirl[1:10, ]
R> swirl[, 2]
```

7. Accessing slots of microarray objects. There are a few ways to access slots of a microarray object. For examples, to obtain the Layout information of the array:

```
R> swirl@maLayout
R> slot(swirl, "maLayout")
R> maLayout(swirl)
```

8. We recommend using the last command as these are simple methods defined by our packages to access slots of the microarray classes. Using such methods is more general than using the `slot` function or `@` operator. In addition, various methods were defined to compute basic statistics from microarray object slots. For instance, for memory management reasons, objects of class `marrayLayout` do not store the spot coordinates of each probe. Rather, these can be obtained from the dimensions of the grid and spot matrices by applying methods such as `maSpotRow` or `maGridCol` to objects of class `marrayLayout`. The commands below may be used to display the number of spots on the array and the dimensions of the grid matrix.

```
R> maNspots(swirl)
R> maNgr(swirl)
R> maNgc(swirl)
```

9. [OPTIONAL] Reading microarray data into R. The user can enter their data in R at many different stages, raw data output from image analysis software, pre- or post-process data from databases (e.g. Gene Traffic, Stanford Microarray Database (SMD) and many others).

```
R> datadir <- system.file("data", package = "marrayInput")
R> dir(datadir)
R> swirl.layout <- read.marrayLayout(fname = file.path(datadir,
+ "fish.gal"), ngr = 4, ngc = 4, nsr = 22, nsc = 24, ctl.col = 4,
+ quote = "", skip = 21)
R> swirl.samples <- read.marrayInfo(file.path(datadir, "SwirlSample.txt"),
+ quote = "")
R> swirl.gnames <- read.marrayInfo(file.path(datadir, "fish.gal"),
+ info.id = 4:5, labels = 5, skip = 21, quote = "")
R> fnames <- dir(path = datadir, pattern = paste("*", "spot", sep = "."))
R> swirl <- read.marrayRaw(fnames, path = datadir, name.Gf = "Gmean",
+ name.Gb = "morphG", name.Rf = "Rmean", name.Rb = "morphR",
+ layout = swirl.layout, gnames = swirl.gnames, samples = swirl.samples)
```

10. [OPTIONAL] For users who prefer command line input for a specific class of image processing output files, we have defined three additional functions. The functions `read.Spot`, `read.Genepix`, and `read.SMD` automate the creation of `marrayRaw` objects from `Spot` and `GenePix` image analysis files, and from the Stanford Microarray Database (SMD) raw data files (`.xls`). The main arguments to these functions are a list of files and the directory path of the files. The following commands read two specific files from the `datadir` directory.

```
R> fnames <- dir(path = datadir, pattern = paste("*", "spot", sep = "."))[1:2]
R> swirl <- read.Spot(fnames, path = datadir, layout = swirl.layout,
+ gnames = swirl.gnames, samples = swirl.samples)
```

11. [OPTIONAL] Alternatively, without specifying any arguments, the functions `read.spot` and `read.GenePix` by default will read in all `Spot` or `GenePix` files within a current working directory. One has the option of setting the layout, probe, and target information manually at a later stage.

```
R> setwd <- datadir
R> swirl <- read.Spot()
R> test.raw <- read.GenePix()
R> slot(swirl, "maLayout") <- swirl.layout
R> slot(swirl, "maGnames") <- swirl.gnames
```

Part II - Diagnostic plots

12. Spatial plots of spot statistics - `maImage`

The generic function `maImage` and associated methods create *images* of shades of gray or colors that correspond to the values of a statistic for each spot on the array. The statistic can be the intensity log-ratio M , a spot quality measure (e.g. spot size or shape), or a test statistic.

```
R> maImage(swirl[, 3], x = "maGb", bar = TRUE)
R> maImage(swirl[, 1], x = "maRb")
R> maImage(swirl[, 2], x = "maM", bar = TRUE)
R> par(mfrow = c(2, 2))
R> for (i in 1:4) maImage(swirl[, i], x = "maRb", bar = FALSE)
R> RGcol <- maPalette(low = "green", mid = "white", high = "red",
+   k = 50)
R> maImage(swirl[, 3], x = "maM")
R> maImage(swirl[, 3], x = "maM", subset = maTop(maM(swirl[, 3]),
+   h = 0.1, l = 0.1), col = RGcol, bar = FALSE, main = "Swirl")
```

13. Boxplots of spot statistics - `maBoxplot`

Boxplots of spot statistics by plate, print-tip-group, or slide can also be useful to identify spot or hybridization artifacts

```
R> maBoxplot(swirl[, 3])
R> maBoxplot(swirl[, 3], x = "maPrintTip", y = "maM", main = "Swirl: pre-normalization")
R> maBoxplot(swirl[, 3], x = "maGridCol", y = "maGb")
```

The function `maBoxplot` may also be used to produce boxplots of spot statistics for all arrays in a batch. Such plots are useful when assessing the need for between array normalization.

```
R> maBoxplot(swirl, y = "maM", main = "Swirl: pre-normalization")
```

14. Scatter-plots of spot statistics - `maPlot`

a) Function for plotting the legend:

```
R> defs <- maDefaultPar(swirl[, 3], x = "maA", y = "maM", z = "maPrintTip")
R> legend.func <- do.call("maLegendLines", defs$def.legend)
```

b) Function for performing and plotting lowess fits

```
R> lines.func <- do.call("maLowessLines", c(list(TRUE, f = 0.3),
+     defs$def.lines))
```

c) Function for performing *MA*-plot.

```
R> maPlot(swirl[, 3], x = "maA", y = "maM", z = "maPrintTip", lines.func,
+     text.func = maText(), legend.func, main = "Swirl: pre-normalization MA-plot")
```

The same plot can be obtain by the command

```
R> maPlot(swirl[, 3])
```

d) Using the default arguments of the function. To highlight, say, the spots with the highest 5% log-ratios using purple symbols "O", set

```
R> maPlot(swirl[, 3], text.func = maText(subset = maTop(maM(swirl[,
+     3]), h = 0.01, l = 0.01), labels = "O", col = "purple"))
```

Part III - Normalization of *swirl* data

15. Normalizes all four arrays in the Swirl experiment simultaneously using different normalization methods.

a) No normalization

```
R> swirl.norm <- maNorm(swirl, norm = "n")
```

b) Median normalization

```
R> swirl.norm <- maNorm(swirl, norm = "m")
```

c) Print-tip-loess normalization

```
R> swirl.norm <- maNorm(swirl, norm = "p")
R> swirl.norm
```

16. Repeat some diagnostic plots to check that the systematic featuers are removed. Plots:

```
R> maPlot(swirl.norm[, 3])
R> maBoxplot(swirl.norm)
R> maImage(swirl.norm[, 3], x = "maM", col = heat.colors(20), bar = TRUE)
```