1

Bioconductor R packages for exploratory analysis and normalization of cDNA microarray data

Sandrine Dudoit
Yee Hwa Yang

Abstract

This chapter describes a collection of four R packages for exploratory analysis and normalization of two–color cDNA microarray fluorescence intensity data. R's object–oriented class/method mechanism is exploited to allow efficient and systematic representation and manipulation of large microarray datasets of multiple types. The `marrayClasses` package contains class definitions and associated methods for pre– and post–normalization intensity data for batches of arrays. The `marrayInput` package provides functions and `tcltk` widgets to automate data input and the creation of microarray specific R objects for storing these data. Functions for diagnostic plots of microarray spot statistics, such as boxplots, scatter–plots, and spatial color images are provided in `marrayPlots`. Finally, the `marrayNorm` package implements robust adaptive location and scale normalization procedures, which correct for different types of dye biases (e.g. intensity, spatial, plate biases) and allow the use of control sequences spotted onto the array and possibly spiked into the mRNA samples. The four new packages were developed as part of the Bioconductor project, which aims more generally to produce an open source and open development statistical computing framework for the analysis of genomic data.

1.1 Introduction

1.1.1 Overview of packages

Microarray experiments generate large and complex multivariate datasets. Careful statistical design and analysis are essential to improve the efficiency and reliability of microarray experiments, from the early design and pre–processing stages to higher–level analyses. Access to an efficient and portable statistical computing environment is a related and equally critical aspect of the analysis of gene expression data. This chapter describes a collection of four R (Ihaka and Gentleman, 1996) packages for exploratory analysis and normalization of two–color cDNA microarray fluorescence intensity data. An earlier version of these packages can be found in the `sma` package which was written in the Fall of 2000. The four new packages were developed as part of the Bioconductor project, which aims more generally to produce an open source and open development statistical computing framework for the analysis of genomic data (`http://www.bioconductor.org`). Particular emphasis is placed on facilitating access to the powerful statistical methodology implemented in the R language and on enhancing its effectiveness within a biological context. Like most Bioconductor packages, these four packages rely on R's *object–oriented class/method mechanism* (John Chambers' `methods` package) to allow efficient and systematic representation and manipulation of large microarray datasets of multiple types. Efforts to reduce the barrier of entry into R include providing *widgets*, i.e., small–scale graphical interfaces, for data input and basic analysis procedures. A brief description of the four `marray` packages is given next.

`marrayClasses`. This package contains class definitions and associated methods for pre– and post–normalization intensity data for batches of arrays. Methods are provided for the creation and modification of microarray objects, basic computations, printing, subsetting, and class conversions.

`marrayInput`. This package provides functionality for reading microarray data into R, such as intensity data from image processing output files (e.g. `.spot` and `.gpr` files for the `Spot` and `GenePix` packages, respectively) and textual information on probes and targets (e.g. from gal files and god lists). `tcltk` widgets are supplied to facilitate and automate data input and the creation of microarray specific R objects for storing these data.

`marrayPlots`. This package provides functions for diagnostic plots of microarray spot statistics, such as boxplots, scatter–plots, and spatial color images. Examination of diagnostic plots of intensity data is important in order to identify printing, hybridization, and scanning artifacts which can lead to biased inference concerning gene expression.

`marrayNorm`. This package implements robust adaptive location and scale normalization procedures, which correct for different types of dye biases (e.g. intensity, spatial, plate biases) and allow the use of control sequences spotted onto the array and possibly spiked into the mRNA samples. Normalization is needed to ensure that observed differences in intensities are indeed due to differential expression and not experimental artifacts; fluorescence intensities should therefore be normalized before any analysis which involves comparisons among genes within or between arrays.

Bioconductor packages are distributed under an open source license, such as GPL or LGPL, and may be downloaded from the project website, `http://www.bioconductor.org`, for Linux, Unix, MS Windows, and Mac OS X operating systems. Sources, binaries, and documentation for R and other R packages can be obtained from the "Comprehensive R Archive Network" (CRAN), `http://cran.r-project.org/`. As with any other Bioconductor 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 `help(maNorm)` or `?maNorm`. In addition, each Bioconductor package contains step–by–step tutorials in the `/doc` subdirectory. These tutorials are generated using the `Sweave` function from the R 1.5.0 package `tools` (Leisch, 2002). They form integrated statistical documents intermixing text, R code, and code output (numerical, textual, and graphical). Within this framework, documents can be regenerated automatically whenever data or analyses are modified. The present document was also generated using `Sweave`; the `.Rnw` source file is available at `http://www.stat.berkeley.edu/~sandrine/`. In addition, a demonstration script is provided for the `marrayPlots` package in the `/demo` subdirectory, and can be run using `demo(marrayPlots)`.

The chapter is organized as follows. The remainder of this section gives background on two–color cDNA microarray experiments. Section 1.2 provides an overview of the statistical and computational methodology for exploratory analysis and normalization of cDNA microarray data. Section 1.3 describes the Swirl microarray experiment, which serves as a case study for illustrating the statistical methods and software implementation. Section 1.4 discusses the four R packages, `marrayClasses`, `marrayInput`, `marrayPlots`, and `marrayNorm`, in greater details and provides a demonstration of package functionality using the `swirl` dataset. Finally, Section 1.5 summarizes our findings and outlines ongoing efforts as part of the Bioconductor project.

1.1.2 Two-color cDNA microarray experiments

DNA microarrays consist of thousands of individual DNA sequences printed in a high–density array on a glass microscope slide using a robotic printer or *arrayer*. The *relative abundance* of these spotted DNA sequences in two DNA or RNA samples may be assessed by monitoring the *differential hybridization* of the two samples to the sequences on the array. For mRNA samples, the two samples or *targets* are reverse–transcribed into cDNA, labeled using different fluorescent dyes (usually a red–fluorescent dye, Cyanine 5 or Cy5, and a green–fluorescent dye, Cyanine 3 or Cy3), then mixed in equal proportions and hybridized with the arrayed DNA sequences or *probes* (following the definition of probe and target adopted in "The Chipping Forecast", a January 1999 supplement to Nature Genetics). After this competitive hybridization, the slides are imaged using a *scanner* and fluorescence measurements are made separately for each dye at each spot on the array. The ratio of the red and green fluorescence intensities for each spot is indicative of the relative abundance of the corresponding DNA probe in the two nucleic acid target samples. See Brown and Botstein, 1999 and Schena, 2000, for a more detailed introduction to the biology and technology of cDNA microarrays.

The term *array layout* refers to the layout of DNA probe sequences on the array, as determined by the printing process. In general, probe sequences are spotted on a glass microscope slide using an arrayer which has an $ngr \times ngc$ print–head, that is, a regular array of *ngr* rows and *ngc* columns of print–tips or pins. The resulting microarrays are thus partitioned into an *ngr* $\times$ *ngc grid matrix*. The terms *grid*, *sector*, *pin–group*, and *print–tip–group* are used interchangeably in the microarray literature. Each grid consists of an *nsr* $\times$ *nsc spot matrix* that was printed with a single print–tip. DNA probes are usually printed sequentially from a collection of 384–well plates (or 96–well plates), thus, in some sense, plates are proxies for time of printing. In addition, a number of control probe sequences may be spotted on the array for normalization or other calibration purposes. The term *array batch* is used to refer to a collection of arrays with the same layout.

The *raw data* from a microarray experiment are the *image files* produced by the scanner; these are typically pairs of 16–bit tagged image file format (TIFF) files, one for each fluorescent dye (images usually range in size from a few megabytes (MB) to 10 or 20 MB for high resolution scans). Image analysis is required to extract *foreground* and *background fluorescence intensity* measurements for each spotted DNA sequence. Image processing is beyond the scope of this chapter, and the reader is referred to Yang et al., 2002a, for a detailed discussion of microarray image analysis, a description of the image processing R package `Spot`, and additional references.

1.2 Methods

1.2.1 Standards for microarray data

There is still much debate regarding standards for storing and reporting microarray–based gene expression data. Significant progress toward the definition of such standards is found in the Minimum Information About a Microarray Experiment – MIAME documents produced by the Microarray Gene Expression Database – MGED group (Brazma et al., 2001, `http://www.mged.org/Workgroups/MIAME/miame.html`). The MIAME documents focus on the content and structure of the necessary information, rather than the technical format for representing and storing the data. The standards apply to different microarray platforms, such as cDNA spotted microarrays and Affymetrix oligonucleotide chips.

A description of a microarray experiment should contain information about the genes whose expression has been measured (*gene annotation*) and about the nature and preparation of the target samples hybridized to the arrays (*sample annotation*), in addition to the quantitative gene expression measurements. At least three levels of expression data are relevant: (i) the microarray scanned images, or raw data; (ii) the microarray image quantification data, i.e., output files from image analysis software packages; and (iii) the gene expression matrix of derived expression levels, where rows correspond to spots and columns to target samples. Reliability information and a detailed description of how the expression values were obtained should also be stored (e.g. record of image analysis, normalization, and quality–based filtering procedures, etc.).

Here, we begin our analysis of microarray data with the output files of image processing packages such as `GenePix` or `Spot` (intermediate expression data, level (ii), above). In what follows, red and green background intensities are denoted by R_b and G_b, respectively, and red and green foreground intensities by R_f and G_f, respectively. Background–corrected red and green fluorescence intensities are denoted by $R = (R_f - R_b)$ and $G = (G_f - G_b)$, and M denotes the corresponding base 2 log–ratio, $M = \log_2 R/G$. We use R's object–oriented class/method mechanism for representation and manipulation of microarray data. Microarray specific object classes were defined as described next to keep track of the three main types of microarray data (gene annotation, sample annotation, and expression data) at different stages of the analysis process.

1.2.2 Object–oriented programming: microarray classes and methods

Microarray experiments generate large and complex multivariate datasets, which contain textual information on probe sequences (e.g. gene names, annotation, layout parameters) and mRNA target samples (e.g. description of samples, protocols, hybridization and scanning conditions), in addition to the primary fluorescence intensity data. Efficient and coordinated access to these various types of data is an important aspect of computing with microarray data. The `marray` packages rely on the *class/method mechanism* provided by John Chambers' R `methods` package, which allows *object–oriented programming* in R. To facilitate the management of microarray data at different stages of the analysis process, a collection of microarray specific data structures or *classes* were defined (see also Chapter 2 for a discussion of the Bioconductor packages `Biobase` and `annotate`, which provide basic structures and methods for microarray and annotation data). Broadly speaking, classes reflect how we think of certain objects and what information these objects should contain. Classes are defined in terms of *slots* which contain the relevant data for the application at hand. *Methods* define how a particular function should behave depending on the class of its arguments and allow computations to be adapted to particular classes, that is, data types. For example, a microarray object should contain intensity data as well as information on the probe sequences spotted on the array and the target samples hybridized to it. Useful methods for microarray classes include specializations of printing, subsetting, and plotting functions for the types of data represented by these classes. The use of classes and methods greatly reduces the complexity of handling large and varied datasets associated with microarray experiments, by automatically coordinating different sources of information.

1.2.3 Diagnostic plots

Before proceeding to normalization or any higher–level analysis, it is instructive to look at diagnostic plots of spot statistics, such as red and green foreground and background log–intensities, intensity log–ratio, area, etc. Stratifying spot statistics according to layout parameters such as print–tip or plate is useful for the purpose of identifying printing, hybridization, and scanning artifacts as demonstrated in Section 1.4.3.

2D spatial images. In a *2D spatial image*, shades of gray or colors are used to represent the value of a statistic for each spot on the array. Each rectangle in the grid corresponds to a particular spot and its coordinates reflect the location of the 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. These pseudo images may be used to explore spatial dependencies in the values of a particular statistic due to, for example, print–tip or cover–slip effects.

Boxplots. *Boxplots*, also called *box–and–whisker plots*, were first proposed by John Tukey in 1977 as simple graphical summaries of the distribution of a variable. The summary consists of the median, the upper and lower quartiles, the range, and, possibly, individual extreme values. The central box in the plot represents the *inter–quartile range (IQR)*, which is defined as the difference between the *upper quartile* and *lower quartile*, i.e., the difference between the 75th and 25th percentiles. The line in the middle of the box represents the *median* or 50th percentile; a measure of central location of the data. Extreme values greater than 1.5 IQR above the 75th percentile and less than 1.5 IQR below the 25th percentile are typically plotted individually.

Scatter–plots. Single–slide expression data are typically displayed by plotting the log–intensity $\log_2 R$ in the red channel vs. the log–intensity $\log_2 G$ in the green channel. Such plots tend to give an unrealistic sense of concordance between the red and green intensities and can mask interesting features of the data. We thus prefer to plot the intensity log–ratio $M = \log_2 R/G = \log_2 R - \log_2 G$ vs. the mean log–intensity $A = \log_2 \sqrt{RG} = (\log_2 R + \log_2 G)/2$. An MA*–plot* amounts to a 45^o counterclockwise rotation of the $(\log_2 G, \log_2 R)$– coordinate system, followed by scaling of the coordinates. It is thus another representation of the (R, G) data in terms of the log–ratios M which directly measure differences between the red and green channels and are the quantities of interest to most investigators. We have found MA–plots to be more revealing than their $\log_2 R$ vs. $\log_2 G$ counterparts in terms of identifying spot artifacts and for normalization purposes (Dudoit et al., 2002; Yang et al., 2001; Yang et al., 2002b). Applications of MA–plots are also discussed in Chapters 4, 7, 9, and 14.

1.2.4 Normalization using robust local regression

The purpose of *normalization* is to identify and remove the effects of systematic variation, other than differential expression, in the measured fluorescence intensities (e.g. different labeling efficiencies and scanning properties of the Cy3 and Cy5 dyes; different scanning parameters, such as PMT (photo multiplier tube) settings; print–tip, spatial, or plate effects). It is necessary to normalize the fluorescence intensities before any analysis which involves comparing expression levels within or between slides (e.g. classification, multiple testing), in order to ensure that differences in intensities are indeed due to differential expression and not experimental artifacts. The need for normalization can be seen most clearly in *self–self experiments*, in which two identical mRNA samples are labeled with different dyes and hybridized to the same slide (Dudoit et al., 2002). Although there is no differential expression and one expects the red and green intensities to be equal, the red intensities often tend to be lower than the green intensities. Furthermore, the imbalance in the red and green intensities is usually not constant across the spots within and between arrays, and can vary according to overall spot intensity A, location on the array, plate origin, and possibly other variables.

Location normalization. We have developed *location normalization* methods which correct for intensity, spatial, and other dye biases using *robust locally–weighted regression* (Cleveland, 1979; Cleveland and Devlin, 1988; Yang et al., 2001; Yang et al., 2002b). Local regression is a *smoothing* method for summarizing multivariate data using general curves and surfaces. The smoothing is achieved by fitting a polynomial function of the predictor variables *locally* to the data, in a fashion that is analogous to computing a moving average. *Robust* fitting guards against deviant points distorting the smoothed points. For the *lowess* and *loess* procedures, polynomials are fitted locally using iterated weighted least squares. In the context of microarray experiments, robust local regression allows us to capture the non–linear dependence of the intensity log–ratio $M = \log_2 R/G$ on the overall intensity $A = \log_2 \sqrt{RG}$, while ensuring that the computed normalization values are not driven by a small number of differentially expressed genes with extreme log–ratios. The `marrayNorm` and `marrayPlots` packages rely on the R `loess` (`modreg` package) and `lowess` functions; greater details can be found in the help files for these functions.

Scale normalization. For *scale normalization*, a robust estimate of scale, such as the *median absolute deviation (MAD)*, may be used (Yang et al., 2001; Yang et al., 2002b). For a collection of numbers, $x_1, \ldots, x_n$, the MAD is the median of their absolute deviations from the median $m = \text{median}\{x_1, \ldots, x_n\}$, that is, $MAD = \text{median}\{|x_1 - m|, \ldots, |x_n - m|\}$. The R function for MAD is `mad`.

Location and scale normalized intensity log–ratios M_{norm} are given by

$$M_{norm} = \frac{M - l}{s},$$

where l and s denote the location and scale normalization values, respectively. The location value l can be obtained, for example, by robust local regression of M on A within print–tip–group. The scale value s could be the within–print–tip–group MAD of location normalized log–ratios. Note that related approaches to normalization are implemented in the software package `SNOMAD`, described in Chapter 9 and available at `http://pevsnerlab.kennedykrieger.org/snomadinput.html`.

1.3 Application: Swirl microarray experiment

We demonstrate the functionality of the four `marray` R packages using gene expression data from the Swirl experiment. These data were provided by Katrin Wuennenberg–Stapleton from the Ngai Lab at UC Berkeley. (The swirl embryos for this experiment were provided by David Kimelman and David Raible at the University of Washington.) This experiment was carried out using zebrafish as a model organism to study early development in vertebrates. Swirl is a point mutant in the BMP2 gene that affects the dorsal/ventral body axis. Ventral fates such as blood are reduced, whereas dorsal structures such as somites and notochord are expanded. A goal of the Swirl experiment is to identify genes with altered expression in the swirl mutant compared to wild–type zebrafish. Two sets of dye–swap experiments were performed, for a total of four replicate hybridizations. For each of these hybridizations, target cDNA from the swirl mutant was labeled using one of the Cy3 or Cy5 dyes and wild–type target cDNA was labeled using the other dye. Target cDNA was hybridized to microarrays containing 8,448 probes, including 768 control spots (e.g. negative, positive, and normalization controls). Microarrays were printed using 4×4 print–tips and are thus partitioned into a 4×4 grid matrix. Each grid consists of a 22×24 spot matrix that was printed with a single print–tip. Here, spot row and plate coordinates coincide, as each row of spots corresponds to probe sequences from the same 384–well plate ($384 = 16 \times 24$).

Each of the four hybridizations produced a pair of 16–bit images, which were processed using the image analysis software package `Spot` (Buckley, 2000; Yang et al., 2002a). Raw images of the Cy3 and Cy5 fluorescence intensities for all four hybridizations are available at `http://fgl.lsa.berkeley.edu/Swirl/index.html`. The dataset includes four output files, `swirl.1.spot`, `swirl.2.spot`, `swirl.3.spot`, and `swirl.4.spot`, from the `Spot` package. Each of these files contains 8,448 rows and 30 columns; rows correspond to spots and columns to different statistics from the `Spot` image analysis output. The file `fish.gal`, a Gene Array List file (or `.gal` file) for the `GenePix` package, was generated by the program `GalFileMaker`, version 1.2 (`http://www.microarrays.org/software.html`). It contains information on individual probe sequences, such as gene names, spot IDs, spot coordinates. Hybridization information for the mutant and wild–type target samples is stored in `SwirlSample.txt`. All fluorescence intensity data from processed images are included in the `marrayInput` package. The function `data` is used to load the `swirl` dataset into R. To view a description of the experiments and dataset, type `?swirl`.

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

1.4 Software

1.4.1 Package `marrayClasses` *– Classes and methods for cDNA microarray data*

The following microarray *classes* were defined to represent pre–and post–normalization fluorescence intensity data, and data on probes and targets for batches of arrays. Here, a batch of arrays refers to a set of arrays with the same layout, as described in Section 1.1.2.

Microarray classes

▶ **Class `marrayLayout`.** Keeping track of array layout information is essential for quality assessment of fluorescent intensity data and for normalization purposes. Important layout parameters are 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 useful to keep track of gene names, plate origin of the probes, and information on the spotted control sequences (e.g. probe sequences which should have equal abundance in the two target samples, such as housekeeping genes). The class `marrayLayout` was designed to keep track of these various layout parameters and contains the following slots (the classes of the slots are listed below the slot names)

```
R> getClassDef("marrayLayout")

Slots:

Name:       maNgr      maNgc      maNsr      maNsc
Class:    numeric    numeric    numeric    numeric

Name:    maNspots      maSub    maPlate maControls
Class:    numeric    logical     factor     factor

Name:     maNotes
Class:  character
```

Here, `maNgr` and `maNgc` store the dimensions of the grid matrix, `maNsr` and `maNsc` store the dimensions of the spot matrices, `naNspots` refers to the total number of spots on the array, `maSub` keeps track of the subset of spots currently being considered, `maPlate` is a vector of plate labels, `maControls` is a vector of spot control labels, and `maNotes` can be used to store any character string describing the array. In addition, a number of methods were defined to compute other important layout parameters, such as print–tip, grid matrix and spot matrix coordinates: `maPrintTip`, `maGridRow`, `maGridCol`, `maSpotRow`, and `maSpotCol` (see discussion below). No slots were defined for these quantities for memory management reasons. For details on slots and methods associated with the `marrayLayout` class, type `?marrayLayout`.

▶ **Class `marrayInfo`.** Information on the target mRNA samples co–hybridized to the arrays is stored in objects of 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. Descriptions of the spotted probe sequences (e.g. gene names, annotation, notes on printing conditions) are also stored in objects of class `marrayInfo`. The `marrayInfo` class is not specific to the microarray context and has the following definition

```
R> getClassDef("marrayInfo")

Slots:

Name:    maLabels     maInfo    maNotes
Class:  character data.frame  character
```

▶ **Class `marrayRaw`.** Pre–normalization intensity data for a batch of arrays are stored in objects of class `marrayRaw`, which contain slots for the matrices of Cy3 and Cy5 background and foreground intensities (`maGb`, `maRb`, `maGf`, `maRf`), spot quality weights (`maW`), layout parameters of the arrays (`maLayout`), description of the probes spotted onto the arrays (`maGnames`) and mRNA target samples hybridized to the arrays (`maTargets`).

```
R> getClassDef("marrayRaw")

Slots:

Name:           maRf          maGf          maRb
Class:        matrix        matrix        matrix

Name:           maGb           maW      maLayout
Class:        matrix        matrix  marrayLayout

Name:        maGnames     maTargets       maNotes
Class:    marrayInfo    marrayInfo     character
```

▶ **Class `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`. For more details on the creation of normalized microarray objects, the reader is referred to Section 1.4.4.

```
R> getClassDef("marrayNorm")

Slots:

Name:            maA           maM        maMloc
Class:        matrix        matrix        matrix

Name:       maMscale           maW      maLayout
Class:        matrix        matrix  marrayLayout
```

```
Name:      maGnames    maTargets      maNotes
Class:   marrayInfo   marrayInfo    character

Name:    maNormCall
Class:         call
```

▶ **Class `marraySpots`.** This class is used to store information on the spotted probe sequences for a batch of arrays. The class contains slots for the layout of the arrays, `maLayout`, and a description of the probe sequences spotted onto the arrays, `maGnames`.

```
R> getClassDef("marraySpots")

Slots:

Name:      maGnames     maLayout
Class:   marrayInfo marrayLayout
```

▶ **Class `marrayTwo`.** The `marrayTwo` class can be viewed as a leaner version of the `marrayRaw` and `marrayNorm` classes. It contains slots for only two types of spot statistics (`maX` and `maY`), the layout of the arrays (`maLayout`), and a description of the target samples hybridized to the arrays (`maTargets`). The two spot statistics can be, for example, the unnormalized green and red foreground intensities, or the normalized log–ratios M and average intensities $A = \log_2 \sqrt{RG}$.

```
R> getClassDef("marrayTwo")

Slots:

Name:           maX          maY     maLayout
Class:       matrix       matrix marrayLayout

Name:     maTargets
Class:   marrayInfo
```

Most microarray objects contain an `maNotes` slot which can be used to store any string of characters describing the experiments, for example, notes on the printing, hybridization, or scanning conditions.

Creating and accessing slots of microarray objects

▶ **Creating new objects.** The function `new` from the `methods` package may be used to create new objects from a given class. For example, to create an instance of the class `marrayInfo` describing the target samples in the Swirl experiment, one could use the following code

```
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 = "Description of targets for Swirl experiment")
R> zebra.samples

Object of class marrayInfo.

       maLabels   Cy3   Cy5
1 Swirl array 1 swirl    WT
2 Swirl array 2    WT swirl
3 Swirl array 3 swirl    WT
4 Swirl array 4    WT swirl

Number of labels:  4
Dimensions of maInfo matrix:  4  rows by  2  columns

Notes:
Description of targets for Swirl experiment
```

Slots which are not specified in `new` are initialized to the *prototype* for the corresponding class. This is usually an "empty" object, e.g. `matrix(0,0,0)`. In most cases, microarray objects can be created automatically using the input functions and their corresponding widgets in the `marrayInput` package (Section 1.4.2). These were used to create the object `swirl` of class `marrayRaw`.

► **Accessing slots.** Different components or slots of the microarray objects may be accessed using the operator `@`, or alternately, the function `slot`, which evaluates the slot name. For example, to access the `maLayout` slot in the object `swirl` and the `maNgr` slot in the layout object `L`

```
R> L <- slot(swirl, "maLayout")
R> L@maNgr
```

The function `slotNames` can be used to get information on the slots of a formally defined class or an instance of the class. For example, to get information on the slots for the `marrayLayout` class and on the slots for the object `swirl` use

```
R> slotNames("marrayLayout")
R> slotNames(swirl)
```

Basic microarray methods

The following basic *methods* were defined to facilitate manipulation of microarray data objects. To see all methods available for a particular class, e.g. `marrayLayout`, or just the print methods

```
R> showMethods(classes = "marrayLayout")
R> showMethods("print", classes = "marrayLayout")
```

▶ **Printing methods for microarray objects.** Since there is usually no need to print out fluorescence intensities for thousands of genes, the `print` method was overloaded for microarray classes by simple report generators. For an overview of the available microarray *printing methods*, type `methods?print`. For example, summary statistics for an object of class `marrayRaw`, such as `swirl`, can be obtained by `print(swirl)` or simply `swirl`.

```
R> swirl

Pre-normalization intensity data:   Object of class marrayRaw.

Number of arrays:         4 arrays.

A) Layout of spots on the array:
Array layout:         Object of class marrayLayout.

Total number of spots:             8448
Dimensions of grid matrix:         4 rows by 4 cols
Dimensions of spot matrices:       22 rows by 24 cols

Currently working with a subset of 8448 spots.

Control spots:
There are   2 types of controls :
Control       N
    768    7680


Notes on layout:
No Input File

B) Samples hybridized to the array:
Object of class marrayInfo.

  maLabels # of slide        Names experiment Cy3
1       81          81 swirl.1.spot          swirl
2       82          82 swirl.2.spot      wild type
3       93          93 swirl.3.spot          swirl
4       94          94 swirl.4.spot      wild type
```

```
  experiment Cy5      date comments
1      wild type 2001/9/20       NA
2          swirl 2001/9/20       NA
3      wild type 2001/11/8       NA
4          swirl 2001/11/8       NA

Number of labels:  4
Dimensions of maInfo matrix:  4  rows by  6  columns

Notes:
C:/GNU/R/rw1041/library/marrayInput/data/SwirlSample.txt

C) Summary statistics for log-ratio distribution:
              Min. 1st Qu. Median  Mean 3rd Qu. Max.
swirl.1.spot -2.73   -0.79  -0.58 -0.48   -0.29 4.42
swirl.2.spot -2.72   -0.15   0.03  0.03    0.21 2.35
swirl.3.spot -2.29   -0.75  -0.46 -0.42   -0.12 2.65
swirl.4.spot -3.21   -0.46  -0.26 -0.27   -0.06 2.90

D) Notes on intensity data:
```

▶ **Subsetting methods for microarray objects.** In many instances, one is interested in accessing only a subset of arrays in a batch and/or spots in an array. *Subsetting methods* `"["` were defined for this purpose. For an overview of the available microarray subsetting methods, type `methods?"["` or to see all subsetting methods for the session `showMethods("[")`. When using the `"["` operator, the first index refers to spots and the second to arrays in a batch. Thus, to access the first 100 probe sequences in the second and third arrays in the batch `swirl` use

```
R> swirl[1:100, 2:3]
```

▶ **Methods for accessing slots of microarray objects.** A number of simple *accessor methods* were defined to access slots of the microarray classes. Using such methods is more general than using the `slot` function or `@` operator. In particular, if the class definitions are changed, any function which uses the `@` operator will need to be modified. When using a method to access the data in the slot, only that particular method needs to be modified. Accessor methods are named after the slot, thus, to access the layout information for the array batch `swirl` one may also use `maLayout(swirl)`.

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 `maGridRow`, `maGridCol`, `maSpotRow`, and `maSpotCol` to objects of class `marrayLayout`. Print–tip–group coordinates are given by `maPrintTip`. Similar methods were also defined to operate directly on objects of class `marrayRaw`, `marrayNorm`, `marraySpots`, and `marrayTwo`. The commands below may be used to display the number of spots on the array, the dimensions of the grid matrix, and the print–tip–group coordinates.

```
R> swirl.layout <- maLayout(swirl)
R> maNspots(swirl)

[1] 8448

R> maNspots(swirl.layout)

[1] 8448

R> maNgr(swirl)

[1] 4

R> maNgc(swirl.layout)

[1] 4

R> maPrintTip(swirl[525:534, 3])

 [1] 1 1 1 1 2 2 2 2 2 2
```

▶ **Methods for assigning slots of microarray objects.** A number of methods were defined to replace slots of microarray objects, without explicitly using the `@` operator or `slot` function. These *assignment methods* make use of the `setReplaceMethod` function from the R `methods` package. Like the accessor methods just described, the assignment methods are named after the slots. For example, to replace the `maNotes` slot of `swirl.layout`

```
R> maNotes(swirl.layout)

[1] "No Input File"

R> maNotes(swirl.layout) <- "New value"
R> maNotes(swirl.layout)

[1] "New value"
```

To initialize slots of an empty `marrayLayout` object

```
R> L <- new("marrayLayout")
R> maNgr(L) <- 4
```

Similar methods were defined to operate on objects of class `marrayInfo`, `marrayRaw`, `marrayNorm`, `marraySpots`, and `marrayTwo`.

▶ **Methods for coercing microarray objects.** To facilitate navigation between different classes of microarray objects, we have defined methods for converting microarray objects from one class into another.

These *coercing methods* make use of the `setAs` function from the R `methods` package. A list of such methods can be obtained by `methods?coerce`. For example, to coerce an object of class `marrayRaw` into an object of class `marrayNorm`

```
R> swirl.norm <- as(swirl, "marrayNorm")
```

It is also possible to convert objects of class `marrayRaw` or `marrayNorm` into objects of class `exprSet` (see definition in the `Biobase` package)

```
R> as(swirl, "exprSet")

Expression Set (exprSet) with
        8448 genes
        4 samples
                 phenoData object with 6 variables and 4 cases
         varLabels
                : # of slide
                : Names
                : experiment Cy3
                : experiment Cy5
                : date
                : comments
```

▶ **Functions for computing layout parameters.** In some cases, plate information is not stored in `marrayLayout` objects when the data are first read into R. We have defined a function, `maCompPlate`, which computes plate indices from the dimensions of the grid matrix and number of wells in a plate. For example, the zebrafish arrays used in the Swirl experiment were printed from 384–well plates, but the plate IDs were not stored in the `fish.gal` file. To generate plate IDs (arbitrarily labeled by integers starting with 1) and store these in the `maPlate` slot of the `maLayout` slot for `swirl` use

```
R> maPlate(swirl) <- maCompPlate(swirl, n = 384)
```

Similar functions were defined to generate and manipulate spot coordinates: `maCompCoord`, `maCompInd`, `maCoord2Ind`, `maInd2Coord`. The function `maGeneTable` produces a table of spot coordinates and gene names for objects of class `marrayRaw`, `marrayNorm`, and `marraySpots`.

1.4.2 *Package* `marrayInput` *– Data input for cDNA microarrays*

We begin our analysis of microarray data with the fluorescence intensities produced by image processing of the microarray scanned images. Microarray image quantification data are typically stored in tables whose rows correspond to the spotted probe sequences and columns to different spot statistics, e.g. grid row and column coordinates, spot row and column coordinates, red and green background and foreground intensities for different segmentation and background adjustment methods, spot morphology statistics, etc. For the `GenePix` image processing software, these are the `.gpr` files, and for `Spot`, these are the `.spot` files. We also consider probe and target textual information stored, for example, in `.gal` and `.gdl` (god list) files. The `marrayInput` package provides functionality for reading such microarray data into R. The main functions are `read.marrayLayout`, `read.marrayInfo`, and `read.marrayRaw`, which create objects of classes `marrayLayout`, `marrayInfo`, and `marrayRaw`, respectively. In addition, widgets are provided for each of these functions to facilitate data entry.

Textual information and fluorescence intensity data from processed images for the Swirl experiment were included as part of the `marrayInput` package and can be accessed as follows (here, `datadir` is the name of the R package sub–directory containing the data files)

```
R> datadir <- system.file("data", package = "marrayInput")
R> dir(datadir)
```

```
[1] "00Index"          "fish.gal"
[3] "swirl.1.spot"     "swirl.2.spot"
[5] "swirl.3.spot"     "swirl.4.spot"
[7] "swirl.RData"      "SwirlSample.txt"
```

Main input functions

▶ `read.marrayLayout`. This function may be used to read in and store layout information for a batch of arrays. The following commands store layout parameters for the Swirl experiment in the object `swirl.layout` of class `marrayLayout`. The location of the control spots is extracted from the fourth (`ctl.col=4`) column of the file `fish.gal`.

```
R> swirl.layout <- read.marrayLayout(fname = file.path(datadir,
+      "fish.gal"), ngr = 4, ngc = 4, nsr = 22, nsc = 24,
+      skip = 21, ctl.col = 4)
R> ctl <- rep("Control", maNspots(swirl.layout))
R> ctl[maControls(swirl.layout) != "control"] <- "N"
R> maControls(swirl.layout) <- factor(ctl)
```

▶ `read.marrayInfo`. This function creates objects of class `marrayInfo`, which store, for example, textual information on probe sequences and target samples for a batch of arrays. The following commands create such objects for the Swirl experiment, by reading in text files `SwirlSample.txt` and `fish.gal` supplied by the experimenter.

```
R> swirl.targets <- read.marrayInfo(
+       file.path(datadir, "SwirlSample.txt"))
R> swirl.gnames <- read.marrayInfo(
+       file.path(datadir, "fish.gal"),
+       info.id = 4:5, labels = 5, skip = 21)
```

▶ `read.marrayRaw`. This function creates objects of class `marrayRaw` for a batch of arrays. It takes as its main arguments a list of filenames for the intensity data (e.g. `GenePix` output files, `.gpr`), and the names of already created layout, probe, and target description objects, e.g. `swirl.layout`, `swirl.gnames`, and `swirl.targets` for the Swirl experiment. The following commands read in all the `.spot` files residing in the `datadir` directory. The arguments further specify that the red and green foreground intensities are stored under the headings `Rmean` and `Gmean`, and that the red and green background intensities are stored under the headings `morphR` and `morphG`, respectively.

```
R> fnames <- dir(path = datadir,
+       pattern = paste("*", "spot", sep = "."))
R> swirl.raw <- read.marrayRaw(fnames, path = datadir,
+       name.Gf = "Gmean", name.Gb = "morphG",
+       name.Rf = "Rmean", name.Rb = "morphR",
+       layout = swirl.layout, gnames = swirl.gnames,
+       targets = swirl.targets)
```

Widgets for input functions

To facilitate data input and automate the creation of microarray objects, each of the above three input functions has a corresponding `tcltk` *widget*, which provides a point–and–click graphical interface: `widget.marrayLayout`, `widget.marrayInfo`, and `widget.marrayRaw`. A screen–shot of the `marrayRaw` widget is shown in Figure 1.1; the command to launch the widget and read in `Spot` image output files is

```
R> widget.marrayRaw(path = datadir, ext = "spot")
```

Wrapper input functions

The functions `read.Spot`, `read.GenePix`, and `read.SMD` automate the creation of `marrayRaw` objects from `Spot` (`.spot`) and `GenePix` (`.gpr`) image analysis files, and from the Stanford Microarray Database (SMD) raw data files (`.xls`). The main arguments to these functions are a list of filenames and the directory path for these files. The following commands read in 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,
+      targets = swirl.targets)
```

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

1.4.3 Package `marrayPlots` *– Diagnostic plots for cDNA microarray data*

Three main functions were defined to produce boxplots, scatter–plots, and 2D spatial images of spot statistics for pre– and post–normalization intensity data. The main arguments to these functions are microarray objects of classes `marrayRaw`, `marrayNorm`, or `marrayTwo`, and arguments specifying which spot statistics to display (e.g. Cy3 and Cy5 background intensities, intensity log–ratios M), and which subset of spots to include in the plots. Default graphical parameters are chosen for convenience using the function `maDefaultPar` (e.g. color palette, axis labels, plot title), but the user has the option to overwrite these parameters at any point. Note that by default the plots are done for the first array in a batch, that is, the first array in a microarray object of class `marrayRaw`, `marrayNorm`, or `marrayTwo`. Data stored in such objects were generated from microarrays with the same layout as described in Section 1.1.2. To produce plots for other arrays in a batch, subsetting methods may be used. For example, to produce diagnostic plots for the second array in the batch of zebrafish arrays `swirl`, the argument `swirl[,2]` should be passed to the plot functions.

Spatial plots of spot statistics – `maImage`

The function `maImage` uses the R base function `image` to create *2D spatial images* of shades of gray or colors that correspond to the values of a statistic for each spot on an array. Details on the arguments of the function are given in the help file `?maImage`. In addition to existing color palette functions, such as `rainbow` and `heat.colors`, a new function `maPalette` was defined to generate color palettes from user supplied low, middle, and high color values. To create white–to–green, white–to–red, and green–to–white–to–red palettes for microarray images

```
R> Gcol <- maPalette(low = "white", high = "green", k = 50)
R> Rcol <- maPalette(low = "white", high = "red", k = 50)
R> RGcol <- maPalette(low = "green", high = "red",
+       mid = "white", k = 50)
```

Useful diagnostic plots are spatial images of the Cy3 and Cy5 background intensities; these images may reveal hybridization artifacts such as scratches on the slides and cover–slip effects. The following commands produce spatial images of the Cy3 and Cy5 background intensities for the Swirl 93 array using white–to–green and white–to–red color palettes, respectively (Swirl 93 refers to the third array in the batch `swirl`, with label `"93"`, as given by `maLabels(maTargets(swirl))`).

```
R> tmp <- maImage(swirl[, 3], x = "maGb", subset = TRUE,
+       col = Gcol, contours = FALSE, bar = FALSE)
```

```
R> tmp <- maImage(swirl[, 3], x = "maRb", subset = TRUE,
+       col = Rcol, contours = FALSE, bar = FALSE)
```

Note that the same images can be obtained using the default arguments of the function by the shorter commands

```
R> maImage(swirl[, 3], x = "maGb")
R> maImage(swirl[, 3], x = "maRb")
```

If `bar=TRUE`, a calibration color bar is displayed to the right of the images. Other options include displaying contours, and altering graphical parameters such as axis labels and plot title. The `maImage` function returns the values and corresponding colors used to produce the color bar, as well as a six number summary of the spot statistics.

The 2D spatial images of background intensities for the Swirl 93 array are shown in Figure 1.2. It can be noted that the Cy3 and Cy5 background intensities are not uniform across the slide and are higher in the top right corner, perhaps due to cover slip effects or tilt of the slide during scanning. Such patterns were not as clearly visible in the individual Cy3 and Cy5 TIFF images. Similar displays of the Cy3 and Cy5 foreground intensities do not exhibit such strong spatial patterns. For the Swirl 81 array, background images uncovered the existence of a scratch with very high background in print–tip–groups (3,2) and (3,3). Note that spatial images of background intensities revealed a bug (since then fixed) in the image processing package `Spot`, whereby the morphological opening background estimate was sometimes set to a constant and artificially large value for the last row of spots on the array.

The `maImage` function may also be used to generate an image of the pre–normalization log–ratios M, using a green–to–red color palette. Figure 1.3

panel (b) displays such an image for the Swirl 93 array, highlighting only those spots with the highest and lowest 10% pre–normalization log–ratios M. The image suggests the existence of spatial dye biases in the intensity log–ratio, with higher values in grid (3,3) and lower values in grid column 1 of the array.

```
R> tmp <- maImage(swirl[, 3], x = "maM", bar = FALSE,
+       main = "Swirl 93 array: image of pre-normalization M")
```

```
R> tmp <- maImage(swirl[, 3], x = "maM",
+       subset = maTop(maM(swirl[,3]), h = 0.1, l = 0.1),
+       col = RGcol, contours = FALSE, bar = FALSE,
+       main = "Swirl 93 array: image of pre-normalization
+               M for % 10 tails")
```

Note that the `maImage` function (and the functions `maBoxplot` and `maPlot` to be described next) can be used to plot other statistics than fluorescence intensities, for example, spot quality data or layout parameters such as plate IDs (`maPlate` slot).

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. The function `maBoxplot`, based on the R base function `boxplot`, produces boxplots of microarray spot statistics for the classes `marrayRaw`, `marrayNorm`, and `marrayTwo` (see details in `?maBoxplot`). The function `maBoxplot` has three main arguments

`m`: Microarray object of class `marrayRaw`, `marrayNorm`, or `marrayTwo`.

`x`: Name of accessor method for the spot statistic used to stratify the data, typically a slot name for the microarray layout object such as `maPlate` or a method such as `maPrintTip`. If `x` is `NULL`, the data are not stratified.

`y`: Name of accessor method for the spot statistic of interest, typically a slot name for the microarray object `m`, such as `maM`.

Figure 1.4 panel (a) displays boxplots of pre–normalization log–ratios M for each of the 16 print–tip–groups for the Swirl 93 array. This plot was generated by the following commands

```
R> maBoxplot(swirl[, 3], x = "maPrintTip", y = "maM",
+       main = "Swirl 93 array: pre-normalization")
```

The boxplots clearly reveal the need for normalization, since most log–ratios M are negative in spite of the fact that only a small proportion of genes are expected to be differentially expressed in the mutant and wild–type zebrafish. As is often the case, this corresponds to higher signal in the Cy3 channel than in the Cy5 channel, even in the absence of differential expression. In addition, the boxplots uncover spatial dye biases in the log–ratios. In particular, print–tip–group (3,3) clearly stands out from the remaining ones, as suggested also in the images of Figure 1.3. 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, for example, to deal with scale differences among different arrays. The following command produces a boxplot of the pre–normalization intensity log–ratios M for each array in the batch `swirl`. Figure 1.5 panel (a) suggests that different normalizations may be required for different arrays, including possibly scale normalization.

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

Scatter–plots of spot statistics – `maPlot`

The function `maPlot` produces *scatter–plots* of microarray spot statistics for the classes `marrayRaw`, `marrayNorm`, and `marrayTwo`. It also allows the user to highlight and annotate subsets of points on the plot, and display fitted curves from robust local regression or other smoothing procedures (see details in `?maPlot`). It relies on the R base functions `plot`, `text`, and `legend`. The function `maPlot` has seven main arguments

`m`: Microarray object of class `marrayRaw`, `marrayNorm`, or `marrayTwo`.

`x`: Name of accessor function for the abscissa spot statistic, typically a slot name for the microarray object `m`, such as `maA`.

`y`: Name of accessor function for the ordinate spot statistic, typically a slot name for the microarray object `m`, such as `maM`.

`z`: Name of accessor method for the spot statistic used to stratify the data, typically a slot name for the microarray layout object such as `maPlate` or a method such as `maPrintTip`. If `z` is `NULL`, the data are not stratified.

`lines.func`: Function for computing and plotting smoothed fits of `y` as a function of `x`, separately within values of `z`, e.g. `maLoessLines`. If `lines.func` is `NULL`, no fitting is performed.

`text.func:` Function for highlighting a subset of points, e.g. `maText`. If `text.func` is `NULL`, no points are highlighted.

`legend.func`: Function for adding a legend to the plot, e.g. `maLegend-Lines`. If `legend.func` is `NULL`, there is no legend.

As usual, optional graphical parameters may be supplied and these will overwrite the default parameters set in the plot functions. A number of functions for computing and plotting the fits can be used, such as `maLowessLines` and `maLoessLines` for robust local regression using the R functions `lowess` and `loess`, respectively. Functions are also provided for highlighting points (e.g. `maText`) and adding a legend to the plot (e.g. `maLegendLines`).

Figure 1.6 panel (a) displays the pre–normalization MA–plots for the Swirl 93 array, with the sixteen lowess fits for each of the print–tip–groups (using a smoother span $f = 0.3$ for the `lowess` function). The figure was generated with the following commands

```
R> defs <- maDefaultPar(swirl[, 3],
+      x = "maA", y = "maM", z = "maPrintTip")
R> legend.func <- do.call("maLegendLines", defs$def.legend)
R> lines.func <- do.call("maLowessLines",
+      c(list(TRUE, f = 0.3), defs$def.lines))
R> maPlot(swirl[, 3], x = "maA", y = "maM", z = "maPrintTip",
+      lines.func, text.func = maText(), legend.func,
+      main = "Swirl 93 array: pre-normalization MA-plot")
```

The same plots can be obtain using the default arguments of the function by the command

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

Figure 1.6 illustrates the non–linear dependence of the log–ratio M on the overall spot intensity A and thus suggests that an intensity or A–dependent normalization method is preferable to a global one (e.g. median normalization). Also, the lowess fits vary among print–tip–groups, again revealing the existence of spatial dye biases. To highlight, say, the spots with the highest and lowest 5% log–ratios using purple symbols `"O"`, set

```
text.func=maText(subset=maTop(maM(swirl[,3]),
       h=0.05,l=0.05),labels="O",col="purple")}.
```

Wrapper functions for basic sets of diagnostic plots

Three wrapper functions are provided to automatically generate a standard set of diagnostic plots: `maDiagnPlots1`, `maRawPlots`, and `maNormPlots`. For example, `maDiagnPlots1` produces eight plots of pre– and post–normalization cDNA microarray data: 2D spatial images of Cy3 and Cy5 background intensities, and of pre– and post–normalization log–ratios M; boxplots of pre– and post–normalization log–ratios M by print–tip–group; MA–plots of pre– and post–normalization log–ratios M by print–tip–group. All three functions provide options for saving the figures to a file, in postscript or jpeg format.

```
R> maDiagnPlots1(swirl[,2],
+       title="Swirl 93 array: Diagnostic plots", save=TRUE,
+       fname="swirl93.jpeg", dev="jpeg")
```

1.4.4 *Package* `marrayNorm` – *Location and scale normalization for cDNA microarray data*

General normalization function `maNormMain`

The main function for location and scale normalization of cDNA microarray data is `maNormMain`. It has eight arguments described in details in the help file `?maNormMain`. The main arguments are: `mbatch`, an object of class `marrayRaw` or `marrayNorm`, containing intensity data for the batch of arrays to be normalized; `f.loc` and `f.scale`, lists of location and scale normalization functions; and `a.loc` and `a.scale`, functions for computing the weights used in composite normalization (Yang et al., 2002b). Other arguments mainly deal with controlling output. Normalization is performed simultaneously for each array in the batch using the location and scale normalization procedures specified by the lists of functions `f.loc` and `f.scale`. Typically, only one function is given in each list, otherwise composite normalization is performed using the weights given by `a.loc` and `a.scale`. The `maNormMain` function returns objects of class `marrayNorm`.

The `marrayNorm` package contains functions for median (`maNormMed`), intensity or A–dependent (`maNormLoess`), and 2D spatial (`maNorm2D`) location normalization. The R robust local regression function `loess` is used for intensity dependent and 2D spatial normalization. The package also contains a function for scale normalization using the median absolute deviation or MAD (`maNormMAD`). The functions allow normalization to be done separately within values of a layout parameter, such as plate or print–tip–group, and using different subsets of probe sequences (e.g. dilution series of control probe sequences). Arguments are available for controlling the local regression, when applicable.

Simple normalization function `maNorm`

A simple wrapper function `maNorm` is provided for users interested in applying a standard set of normalization procedures using default parameters. This function returns an object of class `marrayNorm` and has seven arguments described in details in the help file `?maNorm`. The main arguments are: `mbatch`, an object of class `marrayRaw` or `marrayNorm`, containing intensity data for the batch of arrays to be normalized; `norm`, a character string specifying the normalization procedure, e.g. `"p"` or `"printTipLoess"` for within–print–tip–group intensity dependent location normalization using the `loess` function; and `subset`, a logical or numeric vector indicating the subset of points used to compute the normalization values.

Simple scale normalization function `maNormScale`

A simple wrapper function `maNormScale` is provided for users interested in applying a standard set of scale normalization procedures using default parameters. This function returns an object of class `marrayNorm` and has six arguments described in details in the help file `?maNormScale`. The main arguments are: `mbatch`, an object of class `marrayRaw` or `marrayNorm`, containing intensity data for the batch of arrays to be normalized; `norm`, a character string specifying the normalization procedure; and `subset`, a logical or numeric vector indicating the subset of points used to compute the normalization values. This function performs in particular between slide scale normalization using the MAD (`norm = "g"` or `"globalMAD"`).

Normalization for the Swirl experiment

The pre–normalization MA–plot for the Swirl 93 array in Figure 1.6 panel (a) illustrates the non–linear dependence of the log–ratio M on the overall spot intensity A and the existence of spatial dye biases. Only a small proportion of the spots are expected to vary in intensity between the two channels. We thus perform within–print–tip–group loess location normalization using all $8,448$ probes on the array.

▶ **Using main function `maNormMain`.** The following command normalizes all four arrays in the Swirl experiment simultaneously and stores the results in the object `swirl.norm` of class `marrayNorm`. A summary of the normalized data can be viewed using the print method, that is, by typing `print(swirl.norm)` or simply `swirl.norm`.

```
R> swirl.norm <- maNormMain(swirl,
+      f.loc = list(maNormLoess(x = "maA", y = "maM",
+      z = "maPrintTip", w = NULL, subset = TRUE, span = 0.4)),
+      f.scale = NULL, a.loc = maCompNormEq(),
+      a.scale = maCompNormEq())
```

This is the default normalization procedure in `maNormMain`, thus the same results could be obtained by calling

```
R> swirl.norm <- maNormMain(swirl)
```

▶ **Using simple function `maNorm`.** Alternately, the simple wrapper function could be used to perform the same normalization

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

▶ **Using simple function `maNormScale`.** The `maNormScale` function may be used to perform scale normalization separately from location normalization. The following examples do not necessarily represent a recommended analysis but are simply used for demonstrating the software functionality. Within–print–tip–group intensity dependent *location* normalization followed by within–print–tip–group *scale* normalization using the median absolute deviation, could be performed in one step by

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

or sequentially by

```
R> swirl.norm1 <- maNorm(swirl, norm = "p")
R> swirl.norm2 <- maNormScale(swirl.norm1, norm = "p")
```

For between slide scale normalization using MAD scaled by the geometric mean of MAD across slides (Yang et al., 2001; Yang et al., 2002b)

```
R> swirl.normg <- maNormScale(swirl.norm, norm = "g")
```

▶ **Plots for normalized intensity data.** The diagnostic plot functions of Section 1.4.3 may also be applied to objects of class `marrayNorm`, using the same commands as with objects of class `marrayRaw`. For example, the post–normalization boxplots and MA–plots in panels (b) of Figures 1.4, 1.5, and 1.6 were produced by

```
R> maBoxplot(swirl.norm[, 3], x = "maPrintTip",
+       y = "maM", main = "Swirl 93 array: post-normalization")

R> maBoxplot(swirl.norm, y = "maM",
+       main = "Swirl arrays: post-normalization")

R> maPlot(swirl.norm[, 3],
+       main = "Swirl 93 array: post-normalization MA-plot")
```

Note that normalization was performed using the `loess` function, but the fitted lines in the MA–plots were produced using `lowess`. To see the effect of within–print–tip–group location normalization, compare panels (a) and (b) in each of these figures. Normalized log–ratios M are now evenly distributed about about zero across the range of intensities A for each print–tip–group. Furthermore, the non–linear location normalization seems to have eliminated, to some extent, the scale differences among print–tip–groups and arrays.

1.5 Discussion

Access to portable, extensible, and interoperable statistical software is an essential aspect of the analysis of microarray data. We have described four packages for exploratory analysis and normalization of cDNA microarray data, and illustrated their functionality using gene expression data from the Swirl experiment. This case study highlighted the importance of exploratory data analysis using diagnostic plots of various spot statistics. Spatial color images (function `maImage`) of foreground and background intensities and log–ratios of intensities may be used to reveal hybridization (e.g. scratches and cover slip effects) and printing artifacts (e.g. small spots with little probe material). Boxplots may assist in identifying spatial or other types of dye biases in the fluorescence intensities, in addition to scale differences within and between arrays (function `maBoxplot`). Scatter–plots such as MA–plots (function `maPlot`) are useful in revealing intensity dependent biases and experimental artifacts. For instance, examination of the range of intensities A may reveal saturation due to improper scanner settings.

Normalization is a key step in the pre–processing of cDNA microarray data; one that can have a large impact on the results of downstream analyses, such as classification or the identification of differentially expressed genes. Normalization is required to ensure that observed differences in fluorescence intensities are indeed reflecting differential gene expression, and not some printing, hybridization, or scanning artifact. The simplest approach to within–slide location normalization is to subtract a constant from all intensity log–ratios, typically their mean or median. Such global normalization methods are still widely used in spite of the evidence of spatial and intensity dependent dye biases in numerous experiments, including the Swirl experiment. We thus recommend more flexible normalization methods, based on robust locally–weighted regression, which take into account the effects of predictor variables such as spot intensity A, location, and plate origin (e.g. using the lowess or loess procedures). We are currently investigating the application of single channel normalization methods initially developed for the analysis of Affymetrix chip data (see Chapter 4 for a discussion of the Bioconductor package `affy`). The design of microarray classes and methods in Bioconductor packages should facilitate the sharing of code between packages.

Normalized microarray data consist primarily of pairs (M, A) of intensity log–ratios and average log–intensities for each spot in each of several slides, in addition to probe and target textual information, and possibly spot quality weights. We are now in a position to address the main question for which the Swirl microarray experiment was designed, that is, the identification of genes that are differentially expressed between swirl mutant and wild–type zebrafish. The Bioconductor packages `Biobase`, `edd`, `genefilter`, `multtest`, `ROC`, and `sma` may be used to address this question. In general, appropriate methods for the main statistical analysis will depend largely on the question of interest to the investigator and can span the entire discipline of Statistics (e.g. linear modeling, time series analysis, multiple testing, classification). The implementation of suitable statistical methodology will require the development of question specific R packages; a number of such packages are already available on the Bioconductor website.

We envisage two main classes of users for these and other Bioconductor packages. The first class, primarily biologists, will be interested in applying a standard set of procedures from the packages. Researchers in the second group will likely be interested in writing their own functions and packages, in addition to using existing functions. To accommodate these two types of users, the packages were designed at two levels. Regarding the first group, particular emphasis is placed on facilitating access to the statistical methodology implemented in R. The definition of microarray specific classes and methods for storing and manipulating different types of microarray data greatly reduces the complexity associated with handling microarray data (see also `Biobase` and `annotate` for classes and methods for microarray and annotation data). Other steps have been taken to reduce the barrier of entry into R and allow biologists to readily exploit its power and versatility for the analysis of genomic data. New developments include: the design of widgets, or small–scale graphical interfaces for basic input and analysis procedures, and providing seamless access within R to biological information resources such as the National Center for Biotechnology Information (NCBI) Entrez system (`http://www.ncbi.nlm.nih.gov/Entrez/`) or the Gene Ontology (GO) Consortium (`http://www.geneontology.org`). Bioconductor packages `AnnBuilder` and `annotate` already provide browser access to LocusLink, GenBank, and PubMed. Regarding the second class of users, the R language allows the design of functions and packages that are extensible and interoperable.

Acknowledgments. We are grateful to Katrin Wuennenberg–Stapleton from the Ngai Lab at UC Berkeley for providing data from the Swirl microarray experiment. We would also like to thank Robert Gentleman and Vincent Carey at Harvard University for many helpful discussions and feedback on the design of the packages.

Bibliography

Brazma A, Hingamp P, Quackenbush J, Sherlock G, Spellman P, Stoeckert C, Aach J, Ansorge W, Ball CA, Causton HC, Gaasterland T, Glenisson P, Holstege FCP, Kim IF, Markowitz V, Matese JC, Parkinson H, Robinson A, Sarkans U, Schulze-Kremer S, Stewart J, Taylor R, Vilo J, Vingron M (2001). Minimum information about a microarray experiment (MIAME)–toward standards for microarray data. *Nature Genetics* 29:365–371

Brown PO, Botstein D (1999). Exploring the new world of the genome with DNA microarrays. In: *The Chipping Forecast*, volume 21, 33–37. Supplement to Nature Genetics

Buckley MJ (2000). *The Spot user's guide.* CSIRO Mathematical and Information Sciences. `http://www.cmis.csiro.au/IAP/Spot/spotmanual.htm`

Cleveland WS (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74(368):829–836

Cleveland WS, Devlin SJ (1988). Locally-weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association* 83:596–610

Dudoit S, Yang YH, Callow MJ, Speed TP (2002). Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica Sinica* 12(1):111–139

Ihaka R, Gentleman R (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5:299–314

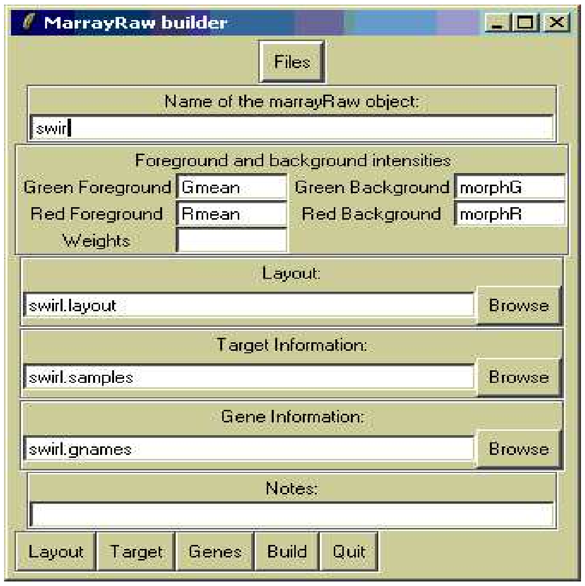
Leisch F (2002). Dynamic generation of statistical reports using literate data analysis. Technical Report 69, SFB Adaptive Information Systems and Modelling in Economics and Management Science, Vienna University of Economics and Business Administration

Schena M (ed.) (2000). *Microarray Biochip Technology.* Eaton

Yang YH, Buckley MJ, Dudoit S, Speed TP (2002a). Comparison of methods for image analysis on cDNA microarray data. *Journal of Computational and Graphical Statistic* 11(1):108–136

Yang YH, Dudoit S, Luu P, Lin DM, Peng V, Ngai J, Speed TP (2002b). Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research* 30(4):e15

Yang YH, Dudoit S, Luu P, Speed TP (2001). Normalization for cDNA microarray data. In: ML Bittner, Y Chen, AN Dorsel, ER Dougherty (eds.), *Microarrays: Optical Technologies and Informatics*, volume 4266 of *Proceedings of SPIE*


MarrayRaw builder

Files

Name of the marrayRaw object:

swirl

Foreground and background intensities

Green Foreground: Gmean
Green Background: morphG
Red Foreground: Rmean
Red Background: morphR
Weights:

Layout:

swirl.layout

Browse

Target Information:

swirl.samples

Browse

Gene Information:

swirl.gnames

Browse

Notes:

Layout | Target | Genes | Build | Quit

Figure 1.1. Screen–shot of the widget for creating objects of class `marrayRaw` from image processing output files.

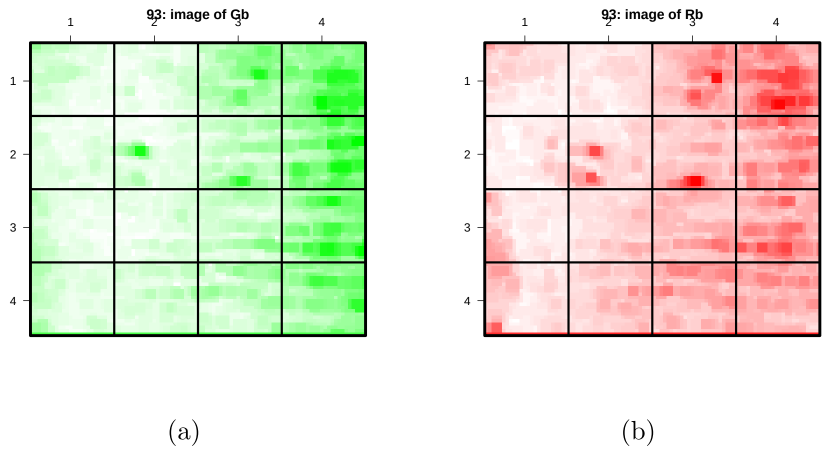


Figure 1.2. 2D spatial images of background intensities for the Swirl 93 array. Panel (a): Cy3 background intensities using white–to–green color palette. Panel (b): Cy5 background intensities using white–to–red color palette.

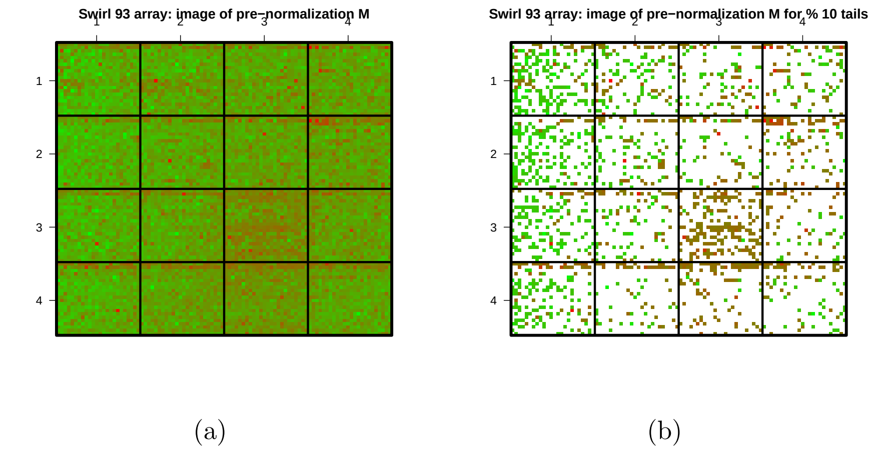


Figure 1.3. 2D spatial images of the pre–normalization intensity log–ratios M for the Swirl 93 array, using a green–to–red color palette. Panel (a): All spots are displayed. Panel (b): Only spots with the highest and lowest 10% log–ratios are highlighted.

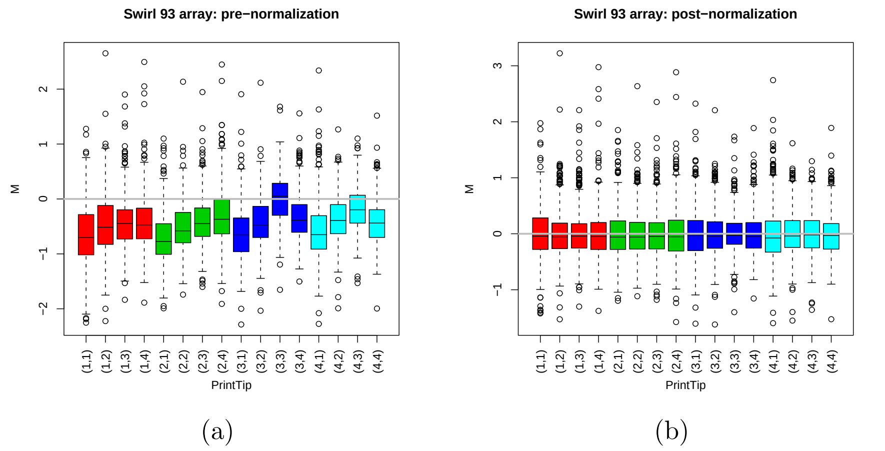


Figure 1.4. Boxplots by print–tip–group of the pre– and post–normalization intensity log–ratios M for the Swirl 93 array.

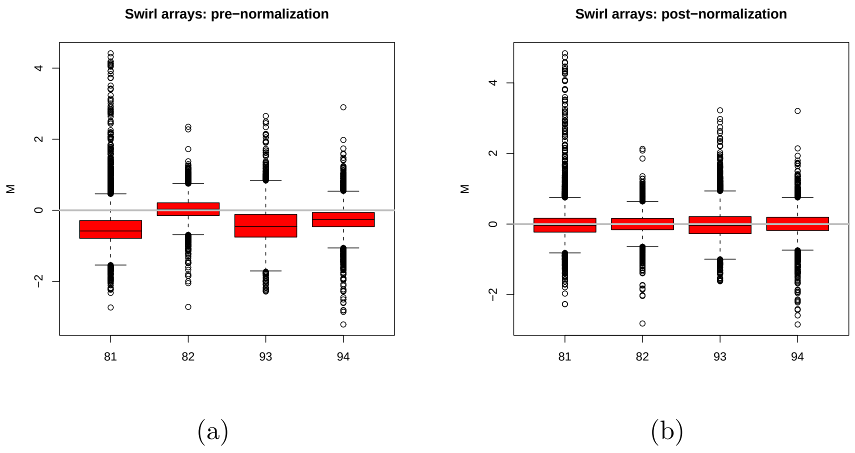


Figure 1.5. Boxplots of the pre–and post–normalization intensity log–ratios M for the four arrays in the Swirl experiment.

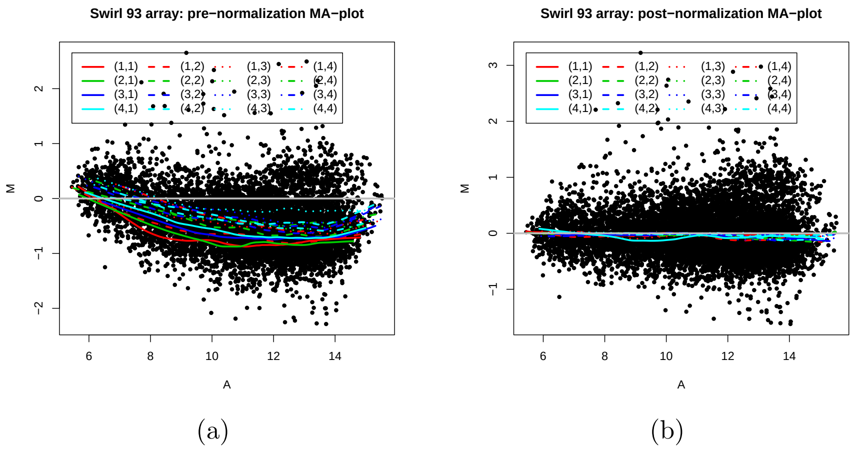


Figure 1.6. Pre– and post–normalization MA–plots for the Swirl 93 array, with the lowess fits for individual print–tip–groups. Different colors are used to represent lowess curves for print–tips from different rows, and different line types are used to represent lowess curves for print–tips from different columns.