Introduction to the Bioconductor marray package : Classes structure component

Sandrine Dudoit[1] and Yee Hwa Yang[2]

October 16, 2005

1. Division of Biostatistics, University of California, Berkeley, `http://www.stat.berkeley.edu/~sandrine` 2. Department of Medicine, University of California, San Francisco, `jean@biostat.berkeley.edu`

Contents

1 Overview

This document provides a tutorial on the class structures used in the `marray` package. The `marray` packages contains basic class definitions and associated methods for pre– and post–normalization intensity data for batches of arrays. To load the `marray` package in your R session, type `library(marray)`. As with any R package, detailed information on functions, classes and methods can be obtained in the help files. For instance, to view the help file for the class `marrayRaw` in a

browser, use `help.start()` followed by `? marrayRaw` or alternately the dyadic `class ? marrayRaw`. Furthermore, se demonstrate the functionality of this collection of R packages using gene expression data from the Swirl zebrafish experiment. To load the Swirl dataset, use `data(swirl)`, and to view a description of the experiments and data, type `? swirl`.
Getting started:

2 Object–oriented programming

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. 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 the Bioconductor package `Biobase` for microarray classes and methods for normalized data). The packages rely on the class/method mechanism provided by John Chambers' R `methods` package, which allows object–oriented programming in R. 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 microarray data, by automatically coordinating various sources of information associated with microarray experiments.

3 Microarray classes

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.

Here, we begin our analysis of microarray data with the output files of image processing packages such as `GenePix` or `Spot`. 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 and G, and M denotes the corresponding base–2 log–ratio, $M = \log_2 R/G$.

3.1 `marrayLayout` class

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*, 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. Keeping track of spot 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)

```
> getClassDef("marrayLayout")

Slots:

Name:       maNgr      maNgc      maNsr      maNsc   maNspots      maSub
Class:    numeric    numeric    numeric    numeric    numeric    logical

Name:     maPlate maControls    maNotes
Class:     factor     factor  character

Extends: "ShowLargeObject"
```

`maNgr`: Object of class "numeric", number of rows for the grid matrix.

`maNgc`: Object of class "numeric", number of columns for the grid matrix.

`maNsr`: Object of class "numeric", number of rows for the spot matrices.

`maNsc`: Object of class "numeric", number of columns for the spot matrices.

`maNspots`: Object of class "numeric", total number of spots on the array, equal to $maNgr \times maNgc \times maNsr \times maNsc$.

`maSub`: Object of class "logical", indicating which spots are currently being considered.

`maPlate`: Object of class "factor", recording the plate origin of the spotted probe sequences.

`maControls`: Object of class "factor", recording the control status of the spotted probe sequences.

maNotes: Object of class "character", any notes concerning the microarray layout, e.g., printing conditions.

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 Section 4). No slots were defined for these quantities for memory management reasons.

3.2 `marrayInfo` class

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. matrix of gene names, annotation, notes on printing conditions) are also stored in object of class `marrayInfo`. The `marrayInfo` class is not specific to the microarray context and has the following definition

```
> getClassDef("marrayInfo")

Slots:

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

Extends: "ShowLargeObject"
```

3.3 `marrayRaw` class

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 (`marrayLayout`), description of the probes spotted onto the arrays (`maGnames`) and mRNA target samples hybridized to the arrays (`maTargets`).

```
> getClassDef("marrayRaw")

Slots:

Name:          maRf         maGf         maRb         maGb          maW
Class:       matrix       matrix       matrix       matrix       matrix

Name:      maLayout     maGnames    maTargets      maNotes
Class: marrayLayout   marrayInfo   marrayInfo    character

Extends: "ShowLargeObject"
```

maRf: Object of class "matrix", red foreground intensities, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maGf`: Object of class "matrix", green foreground intensities, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maRb`: Object of class "matrix", red background intensities, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maGb`: Object of class "matrix", green background intensities, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maW`: Object of class "matrix", spot quality weights, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maLayout`: Object of class "marrayLayout", layout parameters for cDNA microarrays.

`maGnames`: Object of class "marrayInfo", description of spotted probe sequences.

`maTargets`: Object of class "marrayInfo", description of target samples hybridized to the arrays.

`maNotes`: Object of class "character", any notes concerning the microarray experiments, e.g. hybridization or scanning conditions.

3.4 `marrayNorm` class

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 the vignette for the `marrayNorm` package.

```
> getClassDef("marrayNorm")

Slots:

Name:           maA          maM       maMloc     maMscale          maW
Class:       matrix       matrix       matrix       matrix       matrix

Name:      maLayout     maGnames    maTargets      maNotes   maNormCall
Class: marrayLayout   marrayInfo   marrayInfo    character         call

Extends: "ShowLargeObject"
```

`maA`: Object of class "matrix", average log–intensities (base 2) A, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maM`: Object of class "matrix", intensity log–ratios (base 2) M, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maMloc`: Object of class "matrix", location normalization values, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maMscale`: Object of class "matrix", scale normalization values, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maW`: Object of class "matrix", spot quality weights, rows correspond to spotted probe sequences, columns to arrays in the batch.

`maLayout`: Object of class "marrayLayout", layout parameters for cDNA microarrays.

`maGnames`: Object of class "marrayInfo", description of spotted probe sequences.

`maTargets`: Object of class "marrayInfo", description of target samples hybridized to the arrays.

`maNotes`: Object of class "character", any notes concerning the microarray experiments, e.g. hybridization or scanning conditions.

`maNormCall`: Object of class "call", function call for normalizing the batch of arrays.

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

3.5 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 object of class `marrayInfo` describing the target samples in the Swirl experiment, one could use the following code

```
> zebra.RG <- as.data.frame(cbind(c("swirl", "WT", "swirl", "WT"),
+     c("WT", "swirl", "WT", "swirl")))
> dimnames(zebra.RG)[[2]] <- c("Cy3", "Cy5")
> zebra.samples <- new("marrayInfo", maLabels = paste("Swirl array ",
+     1:4, sep = ""), maInfo = zebra.RG, maNotes = "Description of targets for Swirl experiment
> zebra.samples

An object of class "marrayInfo"
@maLabels
[1] "Swirl array 1" "Swirl array 2" "Swirl array 3" "Swirl array 4"

@maInfo
    Cy3   Cy5
1 swirl    WT
2    WT swirl
3 swirl    WT
4    WT swirl

@maNotes
[1] "Description of targets for Swirl experiment"
```

Slots which are not specified in `new` are initialized to the prototype for the corresponding class. These are usually "empty", 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. 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`

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

[1] 4
```

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 or on the slots for the object `swirl` use

```
> slotNames("marrayLayout")

[1] "maNgr"      "maNgc"      "maNsr"      "maNsc"      "maNspots"
[6] "maSub"      "maPlate"    "maControls" "maNotes"

> slotNames(swirl)

[1] "maRf"      "maGf"      "maRb"      "maGb"      "maW"       "maLayout"
[7] "maGnames"  "maTargets" "maNotes"
```

3.6 Testing the validity of an object

The function `validObject` from the R package `methods` may be used to test the validity of an object with respect to its class definition. This function has two arguments: `object`, the object to be tested; and `test`. If `test` is TRUE, the function returns a vector of strings describing the problems, if any.

```
> validObject(maLayout(swirl), test = TRUE)

[1] TRUE
```

4 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

```
> showMethods(classes = "marrayLayout")
> showMethods("show", classes = "marrayLayout")
```

4.1 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 ? summary`, or to see all summary methods for the session

```
> showMethods("summary")

Function "summary":
object = "ANY"
object = "marrayLayout"
object = "marrayInfo"
object = "marrayRaw"
object = "marrayNorm"
```

For example, summary statistics for an object of class `marrayRaw`, such as `swirl`, can be obtained by `print(swirl)` or simply `swirl`.

```
> summary(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 8448spots.

Control spots:
There are   2 types of controls :

Control  probes
    768    7680


Notes on layout:
 No Input File

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

```
  maLabels # of slide        Names experiment Cy3 experiment Cy5      date
1       81         81 swirl.1.spot          swirl      wild type 2001/9/20
2       82         82 swirl.2.spot      wild type          swirl 2001/9/20
3       93         93 swirl.3.spot          swirl      wild type 2001/11/8
4       94         94 swirl.4.spot      wild type          swirl 2001/11/8
  comments
1       NA
2       NA
3       NA
4       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:
```

4.2 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

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

An object of class "marrayRaw"
@maRf
     swirl.2.spot swirl.3.spot
[1,]    16138.720    2895.1600
[2,]    17247.670    2976.6230
[3,]    17317.150    2735.6190
[4,]     6794.381     318.9524
[5,]     6043.542     780.6667
95 more rows ...
```

```
@maGf
     swirl.2.spot swirl.3.spot
[1,]    19278.770    2727.5600
[2,]    21438.960    2787.0330
[3,]    20386.470    2419.8810
[4,]     6677.619     383.2381
[5,]     6576.292     901.0000
95 more rows ...

@maRb
     swirl.2.spot swirl.3.spot
[1,]          136           82
[2,]          133           82
[3,]          133           76
[4,]          105           61
[5,]          105           61
95 more rows ...

@maGb
     [,1] [,2]
[1,]  175   86
[2,]  183   86
[3,]  183   86
[4,]  142   71
[5,]  142   71
95 more rows ...

@maW
<0 x 0 matrix>

@maLayout
An object of class "marrayLayout"
@maNgr
[1] 4

@maNgc
[1] 4

@maNsr
[1] 22

@maNsc
[1] 24
```

```
@maNspots
[1] 8448

@maSub
[1] TRUE TRUE TRUE TRUE TRUE
8443 more elements ...

@maPlate
[1] 1 1 1 1 1
Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
95 more elements ...

@maControls
[1] Control Control Control Control Control
Levels: Control probes
95 more elements ...

@maNotes
[1] "No Input File"


@maGnames
An object of class "marrayInfo"
@maLabels
[1] "geno1" "geno2" "geno3" "3XSSC" "3XSSC"
95 more elements ...

@maInfo
     "ID" "Name"
1 control  geno1
2 control  geno2
3 control  geno3
4 control  3XSSC
5 control  3XSSC
95 more rows ...

@maNotes
[1] "C:/GNU/R/rw1041/library/marrayInput/data/fish.gal"


@maTargets
An object of class "marrayInfo"
@maLabels
[1] "82" "93"
```

```
@maInfo
  # of slide        Names experiment Cy3 experiment Cy5      date comments
2         82 swirl.2.spot      wild type          swirl 2001/9/20       NA
3         93 swirl.3.spot          swirl      wild type 2001/11/8       NA

@maNotes
[1] "C:/GNU/R/rw1041/library/marrayInput/data/SwirlSample.txt"


@maNotes
[1] ""
```

4.3 Methods for accessing slots of microarray objects

A number of simple 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. 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` and `marrayNorm`. 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.

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

[1] 8448

> maNspots(swirl.layout)

[1] 8448

> maNgr(swirl)

[1] 4

> maNgc(swirl.layout)

[1] 4

> maPrintTip(swirl[1:10, 3])

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

4.4 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 make use of the `setReplaceMethod` function from the R `methods` package. As with the accessor methods just described, the assignment methods are named after the slots. For example, to replace the `maNotes` slot of `swirl.layout`

```
> maNotes(swirl.layout)

[1] "No Input File"

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

[1] "New value"
```

To initialize slots of an empty `marrayLayout` object

```
> L <- new("marrayLayout")
> L

An object of class "marrayLayout"
@maNgr
numeric(0)

@maNgc
numeric(0)

@maNsr
numeric(0)

@maNsc
numeric(0)

@maNspots
numeric(0)

@maSub
[1] TRUE

@maPlate
factor(0)
Levels:

@maControls
factor(0)
Levels:
```

```
@maNotes
character(0)

> maNgr(L) <- 4
```

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

4.5 Methods for coercing microarray objects

To facilitate navigation between different classes of microarray objects, we have defined methods for coercing microarray objects from one class into another. 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`:

```
> 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), see package `convert` package for more details.

```
> library(convert)
> as(normdata, "exprSet")
```

4.6 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 Swirl arrays 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 `marrayLayout` object use

```
> 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` and `marrayNorm`.