

The `com.braju.sma` package - A microarray analysis packages based on an object-oriented design and reference variables

Henrik Bengtsson

Mathematical Statistics, Centre for Mathematical Sciences,
Lund University, Box 118, SE-221 00 Lund, Sweden.
hb@maths.lth.se

Extended abstract

We have implemented a cDNA microarray analysis package in R called *com.braju.sma*. It was initially an object-oriented extension to the Statistics for Microarray Analysis package, a.k.a. the *sma* package [4], but it is now a stand-alone package. The package is open source and written in 100% R, which means that it runs on all platforms and that installation is straightforward. This paper presents some of the most commonly used features of the package, but it contains much more. The overall purpose of this paper is not to give a detailed explanation about this package, but to demonstrate how sophisticated analysis packages such as microarray packages can benefit from a well object-oriented design and from using reference variables.

The package has been designed in a true object-oriented manner with an implementation making use of references variables. As a result of trying to fulfill such design and implementation, another package named *R.oo* [3] has been developed. It is part of the *R.classes* bundle and provides the core functionality for an object-oriented design with reference variables and it is based solely on the S3/UseMethod programming paradigm [9]. The *R.oo* package includes functions for defining classes and their methods (generic functions are safely created automatically whenever needed). Furthermore, it provides a transparent way of using reference variables. Instances of classes that are derived from the root class `Object` will automatically be accessed by reference variables. Passing objects to methods as pass-by-reference makes it possible for the methods to change the content of the objects directly without having to return a modified copy, which is the normal approach in R. Passing objects by reference is useful if the objects are huge and creating copies are expensive, which is the case with microarray data. It also makes it possible to create a more

user-friendly method interface, which we have taken advantage of in our package. Furthermore, there are several advantages for separating the object-oriented core from the rest of the microarray package. The major advantage is that the *R.oo* package is likely to be used in other projects and by other developers and end users. A larger user base means more CPU test time, which increases the likelihood of finding bugs that will not be found by standard test procedures. Hence, a separate core package will improve the overall quality of the microarray package.

The package provides methods for reading and writing microarray data in several different file formats generated by some of the most well-known cDNA microarray image analysis applications such as GenePix, ImaGene, QuantArray, ScanAlyze and Spot. Most file versions of these programs are automatically recognized and if not, the package tries to interpret the file format as well as possible. Files can be read by giving either a vector of filenames or a filename pattern, e.g. `MicroarrayData$read(pattern="*.gpr")`, which will read the files in lexical (alphabetic) order. It is also possible to write data back to files in any of the formats above plus the commonly used tab or comma delimited text file formats.

In cDNA microarray analysis, one of the most helpful techniques in exploratory data analysis (EDA) is to plot the data in different ways. Specially adapted plot methods are available for scatter plots between the red and the green channels (foreground or background signals), between the log ratios and the log intensities, or between any other signals or measurements. In addition to the scatter plots, there are various other plot styles, e.g. spatial plots and print order plots [1]. By default, the plot methods color the data points automatically according to what type of data they represent. For instance, gene expression levels are colored in a red-to-green scale with different brightness levels depicting the log ratios and log intensities of the spots. Such color schemas can be helpful in finding artifacts in the measurements. Independent of plot type, it is possible to highlight genes of interest by specifying their spot indices. For instance `highlight(ma, 340:700)` will highlight spots 340 to 700, whether the last plot was a scatter plot, a spatial plot or some other type of plot. See figure 1 for another example. Adding text labels to certain data points is done in a similar way. Furthermore, it is easy to create box plots of data from one or several slides grouped by criteria such as print-tip, slide, slide row etc, where the boxes are (by default) colored to depict the median log ratio/intensity within each group.

The package does not provide a graphical user interface (GUI), but we believe that the object-oriented design, together with the use of reference variables, overcomes the direct short-coming of not having a GUI. Using references makes it possible to reduce the number of arguments needed to be passed by the end user. The highlight method is an example of this. We have developed a programming style guide called the R Coding Convention (RCC) [2] in order to provide a well defined application protocol interface (API) simple enough to be used by anyone. The RCC, which could be seen as an open suggestion to the R community, contains naming conventions similar to those found in Java, where classes, variables and methods can easily be differentiated by only looking at their names. Following the RCC, we hope that the API will be consistent and intuitive for both end users and developers. For these reasons, we do not believe that there is an urgent need for a GUI. However, simple Tcl/Tk test examples have shown that adding a GUI on top of the current

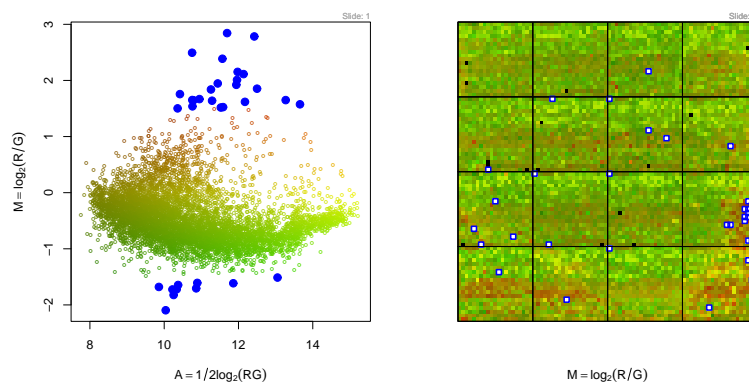


Figure 1: Left: Scatter plot of log ratios vs. log intensities where genes that have more than 3-folded gene expressions are highlighted. Right: Spatial plot as the genes are positioned on the microarray glass. The same extreme genes have been highlighted in this plot.

implementation is straightforward. It is straightforward because the microarray data can be modified directly through reference variables and there is no need to maintain a special lookup table containing names of microarray data variables.

Furthermore, using the Reporter classes `TextReporter`, `HtmlReporter` and `LaTeXReporter` defined in the *R.io* package (part of the *R.classes* bundle), it is possible to generate rich reports in plain text (no graphics), in HTML and in LaTeX, respectively. Because all Reporter classes implement the same interface, the generation of figures (of appropriate format), tables, lists, code examples, table of contents etc. is done in exactly the same way regardless of the output format. Also, using an instance of a `MultiReporter`, it is possible to generate text, HTML and LaTeX reports in one run.

Normalizing cDNA microarray data, i.e. removing artifacts or systematic variation, is most important and is often required to make gene expression levels comparable between genes and between repeated slides. Most of the commonly used normalization methods such as intensity dependent normalization of the log ratios [10] are supported, but also less well-known such as print order normalizations [1]. Identification of the differentially expressed genes can be done by performing a classical t-test or an empirical Bayes adjusted t-test [7].

The classes that store the microarray data are directly compatible with the structures in the *sma* package and converting the *sma* data structures into *com.braju.sma* objects is also straightforward. For this reason, the two packages can be used in parallel and all the functions of the *sma* package can be called using *com.braju.sma* objects. Most of the functionality in *sma* is however reimplemented in our package. The package has been shown to be compatible with another *sma* clone called *smawehi* [5] as well. Related to the compatibility with other packages is the idea of virtual fields. In cDNA microarray analysis the data are often represented by the signals of the red and green channels, which we refer to R and G, respectively. How-

ever, most analysis is done on the log ratios $M = \log_2(R/G)$ and the log intensities $A = \frac{1}{2} \log_2(R \cdot G)$, which basically represent a rotation of the (R,G) plane [10]. For efficiency, data of *com.braju.sma* is often stored in objects of class `MADData`, which contains the fields `M` and `A`. The user can then obtain the values of the red and the green channels by using the `getR(ma)` and `getG(ma)` methods. However, with the concept of virtual fields the user can access the signals as they were fields, i.e. `ma$R` and `ma$G`. The assignment of virtual fields is done by corresponding set methods. This means that it is possible to emulate the concept of private fields, and the end user does not have to (should not) know what the internal representation is. This is commonly referred to as *data hiding* or *encapsulation*. The use of virtual fields minimizes data redundancy, which saves memory and reduces the risk for bugs. It makes it possible to change the internal representation of the microarray objects and at the same time remain compatible with old packages, but also with new packages such as those developed under the Bioconductor project [6, 8]. Virtual fields are implemented by the Object class defined in the *R.oo* package.

Both the *com.braju.sma* package and the *R.classes* bundle can be downloaded from <http://www.maths.lth.se/help/R/> or installed directly using

```
install.packages(c("R.classes", "com.braju.sma"),
  contriburl="http://www.maths.lth.se/help/R")
```

Keywords: cDNA microarray analysis; object-oriented programming; reference variables; coding conventions; data hiding; encapsulation; virtual fields; S3; UseMethod.

References

- [1] Henrik Bengtsson. Identification and normalization of plate effects in cDNA microarray data. Preprints in Mathematical Sciences 2002:28 LUTFMS-5027-2002, Division for Mathematical Statistics, Centre for Mathematical Sciences, Lund University, Sweden, December 2002.
- [2] Henrik Bengtsson. R Coding Conventions (draft). <http://www.maths.lth.se/help/R/>, 2002.
- [3] Henrik Bengtsson. The R.oo package - object-oriented programming with references using S3/UseMethod. Manuscript submitted to the DSC-2003 conference, 2002.
- [4] Ben Bolstad, Sandrine Dudoit, Ingrid Lonnstedt, Natalie Roberts, and Jean Yee Hwa Yang. R package: Statistics for Microarray Analysis (sma). <http://www.stat.berkeley.edu/users/terry/zarray/Software/smacode.html>, 2001. Statistics Department, University of California at Berkeley.
- [5] Gordon Smyth et al. SMAWEHI: An R library for Statistical Microarray Analysis. <http://bioinf.wehi.edu.au/smawehi/>, 2002.
- [6] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- [7] Ingrid Lonnstedt and Terence P. Speed. Replicated microarray data. *Statistical Sinica*, 12(1), 2002.

- [8] Bioconductor Core Team. Bioconductor - open source software for bioinformatics. <http://www.bioconductor.org/>, 2002.
- [9] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S-PLUS*. Springer, 3rd edition, 1999.
- [10] Yee Hwa Yang, Sandrine Dudoit, Percy Luu, and Terry Speed. Normalization for cDNA microarray data (manuscript in preparation).