R News

The Newsletter of the R Project

Volume 2/1, March 2002

Editorial

by Kurt Hornik

Welcome to the fourth issue of *R News*, the newsletter of the R project for statistical computing.

This is the first *special issue* of *R News*, with an emphasis on applying R in medical statistics. It is special in many ways: first, it took longer to prepare than we had originally antipicated (which typically seems to happen for special issues). Second, the newsletter file is rather large as there are many exciting images—in fact, in trying to keep the size reasonable, in some cases images are only included at a lower resolution, with the "real stuff" available from the respective authors' web pages. And finally, articles are fewer but longer: as many of the applications described are based on recent advances in medical technology, we felt that extra space with background information was warranted.

The main focus in this issue is on using R in the analysis of genome data. Here, several exciting initiatives have recently been started, and R has the potential to become a standard tool in this area. The multitude of research questions calls for a flexible computing environment with seemless integration to databases and web content and, of course, state-of-the-art statistical methods: R can do all of that. Robert Gentleman and Vince Carey have started the Bioconductor project, a collaborative effort to provide common infrastructure for the analysis of genome data. The packages developed by this project (about 20 are currently under way) make heavy use of the S4 formal classes and methods (which are now also available in R) introduced in the "Green Book" by John Chambers, and in fact provide the first large-scale project employing the new S object system.

R 1.4.0 was released on Dec 19, 2001, making it the "Lord of the Rings" release of R. Its new features are described in "Changes in R". The number of R packages distributed via CRAN's main section alone now exceeds 150 (!!!)—"Changes on CRAN" briefly presents the 26 most recent ones. The series of introductory articles on each recommended package continues with "Reading Foreign Files".

Developers of R packages should consider submitting their work to the *Journal of Statistical Software* (JSS, http://www.jstatsoft.org) which has become the 'Statistical Software' section of the Journal of Computational and Graphical Statistics (JCGS). JSS provides peer reviews of code and publishes user manuals alongside, and hence ideally complements *R News* as a platform for disseminating news about exciting statistical software.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

Contents of this issue:

Editorial 1
Reading Foreign Files 2
Maximally Selected Rank Statistics in R 3
Quality Control and Early Diagnostics for cDNA Microarrays 6
Bioconductor 11
AnalyzeFMRI: An R package for the exploration and analysis of MRI and fMRI datasets 17
Using R for the Analysis of DNA Microarray Data 24
Changes in R 1.4.0 32
Changes in R 1.4.1 39
Changes on CRAN 40

Reading Foreign Files

by Duncan Murdoch

One of the first tasks in any statistical analysis is getting the data into a usable form. When your client gives you notebooks filled with observations or a pile of surveys and you control the data entry, then you can choose the form to be something convenient. In R, that usually means some form of text file, either comma- or tab-delimited. Through the **RODBC** and various SQL packages R can also import data directly from many databases.

However, sometimes you receive data that has already been entered into some other statistical package and stored in a proprietary format. In that case you have a couple of choices. You might be able to write programs in that package to export the data in an R-friendly format, but often you don't have access or familiarity with it, and it would be best if R could read the foreign file directly. The **foreign** package is designed for exactly this situation. It has code to read files from Minitab, S, S-PLUS, SAS, SPSS and Stata. In this article I'll describe how to import data from all of these packages. I wrote the S and S-PLUS parts, so I'll describe those in most detail first. Other credits for the **foreign** package go to Doug Bates, Saikat DebRoy, and Thomas Lumley.

For more information on data import and export, I recommend reading the fine manual "R Data Import/Export" which comes with the R distribution.

Reading S and S-PLUS files

Both S and S-PLUS usually store objects (which, as most R users know, are very similar to R objects) in individual files in a data directory, as opposed to the single workspace files that R uses. Sometimes the file has the same name as the object, sometimes it has a constructed name, e.g., '__12'. The mapping between object name and file name is complicated and version dependent. In S-PLUS 2000 for Windows, the file '__nonfi' in the data directory contains a list of object names and corresponding filenames. Other versions use different mechanisms. Sometimes (e.g., in library sections) all objects are stored in one big file; at present the **foreign** package cannot read these.

The **foreign** function `read.S()` can be used to read the individual binary files. I have tested it mainly on S-PLUS 2000 for Windows, but it is based on code that worked on earlier versions as well. It may not work at all on later versions. (This is a continuing problem with reading foreign files. The writers of those foreign applications frequently change the formats of their files, and in many cases, the file formats are unpublished. When the format changes, **foreign** stops working until someone works out the new format and modifies the code.)

`read.S()` takes just one argument: the filename to read from. It's up to you to figure out which file contains the data you want. Because S and R are so similar, `read.S` is surprisingly successful. It can generally read simple vectors, data frames, and can occasionally read functions. It almost always has trouble reading formulas, because those are stored differently in S and R.

For example, I have an S-PLUS data frame called `Bars`:

```
> Bars
    type time      y1      y2
  1    0    1 -0.5820 24.7820
  2    0    2  0.5441 23.6559
317    0    7 -1.1925 25.3925
319    0    8  1.4409 22.7591
631    0   13 -3.4194 27.6194
633    0   14  0.7975 23.4025
738    0   19 -0.3439 24.5439
740    0   20  0.6580 23.5420
```

At the beginning of the session, S-PLUS printed this header:

```
Working data will be in
F:\Program files\sp2000\users\murdoch\_Data
```

so that's the place to go looking for `Bars`: but there's no file there by that name. The '__nonfi' file has these lines near the end:

```
"Bars"
"__107"
```

which give me the correct filename, '__107'. Here I read the file into R:

```
> setwd('F:/Program files/sp2000/users')
NULL
> read.S('murdoch/_Data/__107')
    type time      y1      y2
1      0    1 -0.5820 24.7820
2      0    2  0.5441 23.6559
317    0    7 -1.1925 25.3925
319    0    8  1.4409 22.7591
631    0   13 -3.4194 27.6194
633    0   14  0.7975 23.4025
738    0   19 -0.3439 24.5439
740    0   20  0.6580 23.5420
```

(The only reason to use `setwd` here is to fit into these narrow columns!) We see that `read.S` was successful; only the printing format is different.

Of course, since I have access to S-PLUS to do this, it would have been much easier to export the data frame using `dump()` and read it into R using `source()`. I could also have exported it using `data.dump` and restored it using **foreign**'s `data.restore`, but that has the same limitations as `read.S`.

Reading SAS files

SAS stores its files in a number of different formats, depending on the platform (PC, Unix, etc.), version, etc. The `read.xport` function can read the SAS transport (XPORT) format. How you produce one of these files depends on which version of SAS you're using. In older versions, there was a `PROC XPORT`. Newer versions use a "library engine" called `sasv5xpt` to create them. From the SAS technical note referenced in the `?read.xport` help page, a line like

```
libname xxx sasv5xpt 'xxx.dat';
```

creates a library named `xxx` which lives in the physical file 'xxx.dat'. New datasets created in this library, e.g. with

```
data xxx.abc;
  set otherdata;
```

will be saved in the XPORT format.

To read them, use the `read.xport` function. For example, `read.xport('xxx.dat')` will return a list of data frames, one per dataset in the exported library. If there is only a single dataset, it will be returned directly, not in a list. The `lookup.xport` function gives information about the contents of the library.

Reading Minitab files

Minitab also stores its data files in several different formats. The **foreign** package can only read the "portable" format, the MTP files. In its current incarnation, only numeric data types are supported; in particular, dataset descriptions cannot be read.

The `read.mtp` function is used to read the files. It puts each column, matrix or constant into a separate component of a list.

Reading SPSS and Stata files

The **foreign** functions `read.spss` and `read.dta` can read the binary file formats of the SPSS and Stata packages respectively. The former reads data into a list, while the latter reads it into a data frame. The `write.dta` function is unique in the **foreign** package in being able to *export* data in the Stata binary format.

Caveats and conclusions

It's likely that all of the functions in the **foreign** package have limitations, but only some of them are documented. It's best to be very careful when transferring data from one package to another. If you can, use two different methods of transfer and compare the results; calculate summary statistics before and after the transfer; do anything you can to ensure that the data that arrives in R is the data that left the other package. If it's not, you'll be analyzing programming bugs instead of the data that you want to see.

But this is true of any data entry exercise: errors introduced in data entry aren't of much interest to your client!

Duncan Murdoch
University of Western Ontario
murdoch@stats.uwo.ca

Maximally Selected Rank Statistics in R

by Torsten Hothorn and Berthold Lausen

Introduction

The determination of two groups of observations with respect to a simple cutpoint of a predictor is a common problem in medical statistics. For example, the distinction of a low and high risk group of patients is of special interest. The selection of a cutpoint in the predictor leads to a multiple testing problem, cf. Figure 1. This has to be taken into account when the effect of the selected cutpoint is evaluated. Maximally selected rank statistics can be used for estimation as well as evaluation of a simple cutpoint model. We show how this problems can be treated with the **maxstat** package and illustrate the usage of the package by gene expression profiling data.

Maximally selected rank statistics

The functional relationship between a quantitative or ordered predictor X and a quantitative, ordered or censored response Y is unknown. As a simple model one can assume that an unknown cutpoint μ in X determines two groups of observations regarding the response Y: the first group with X-values less or equal μ and the second group with X-values greater μ. A measure of the difference between two groups with respect to μ is the absolute value of an appropriate standardized two-sample linear rank statistic of the responses. We give a short overview and follow the notation in Lausen and Schumacher (1992).

The hypothesis of independence of X and Y can be formulated as

$$H_0 : P(Y \leq y|X \leq \mu) = P(Y \leq y|X > \mu)$$

for all y and $\mu \in \mathbb{R}$. This hypothesis can be tested as follows. For every reasonable cutpoint μ in X (e.g., cutpoints that provide a reasonable sample size in both groups), the absolute value of the standardized two-sample linear rank statistic $|S_\mu|$ is computed. The maximum of the standardized statistics

$$M = \max_\mu |S_\mu|$$

of all possible cutpoints is used as a test statistic for the hypothesis of independence above. The cutpoint in X that provides the best separation of the responses into two groups, i.e., where the standardized statistics take their maximum, is used as an estimate of the unknown cutpoint.

Several approximations for the distribution of the maximum of the standardized statistics S_μ have been suggested. Lausen and Schumacher (1992) show that the limiting distribution is the distribution of the supremum of the absolute value of a standardized Brownian bridge and consequently the approximation of Miller and Siegmund (1982) can be used. An approximation based on an improved Bonferroni inequality is given by Lausen et al. (1994). For small sample sizes, Hothorn and Lausen (2001) derive an lower bound on the distribution function based on the exact distribution of simple linear rank statistics. The algorithm by Streitberg and Röhmel (1986) is used for the computations. The exact distribution of a maximally selected Gauß statistic can be computed using the algorithms by Genz (1992). Because simple linear rank statistics are asymptotically normal, the results can be applied to approximate the distribution of maximally selected rank statistics (see Hothorn and Lausen, 2001).

The maxstat package

The package **maxstat** implements both cutpoint estimation and the test procedure above with several P-value approximations as well as plotting of the empirical process of the standardized statistics. It depends on the packages **exactRankTests** for the computation of the distribution of linear rank statistics (Hothorn, 2001) and **mvtnorm** for the computation of the multivariate normal distribution (Hothorn et al., 2001). All packages are available at CRAN. The generic method `maxstat.test` provides a formula interface for the specification of predictor and response. An object of class `"maxtest"` is returned. The methods `print.maxtest` and `plot.maxtest` are available for inspectation of the results.

Gene expression profiling

The distinction of two types of diffuse large B-cell lymphoma by gene expression profiling is studied by Alizadeh et al. (2000). Hothorn and Lausen (2001) suggest the mean gene expression (MGE) as quantitative factor for the discrimination between two groups of patients with respect to overall survival time. The dataset `DLBCL` is included in the package. The maximally selected log-rank statistic for cutpoints between the 10% and 90% quantile of MGE using the upper bound of the P-value by Hothorn and Lausen (2001) can be computed by

```
> data(DLBCL)
> maxstat.test(Surv(time, cens) ~ MGE,
               data=DLBCL, smethod="LogRank",
               pmethod="HL")

        LogRank using HL

data:  Surv(time, cens) by MGE
M = 3.171, p-value = 0.02220
sample estimates:
estimated cutpoint
         0.1860526
```

For censored responses, the formula interface is similar to the one used in package **survival**: `time` specifies the time until an event and `cens` is the status indicator (`dead=1`). For quantitative responses y, the formula is of the form '`y ~ x`'. Currently it is not possible to specify more than one predictor `x`. `smethod` allows the selection of the statistics to be used: `Gauss`, `Wilcoxon`, `Median`, `NormalQuantil` and `LogRank` are available. `pmethod` defines which kind of P-value approximation is computed: `Lau92` means the limiting distribution, `Lau94` the approximation based on the improved Bonferroni inequality, `exactGauss` the distribution of a maximally selected Gauß statistic and `HL` is the upper bound of the P-value by Hothorn and Lausen (2001). All implemented approximations are known to be conservative and therefore their minimum P-value is available by choosing `pmethod="min"`.

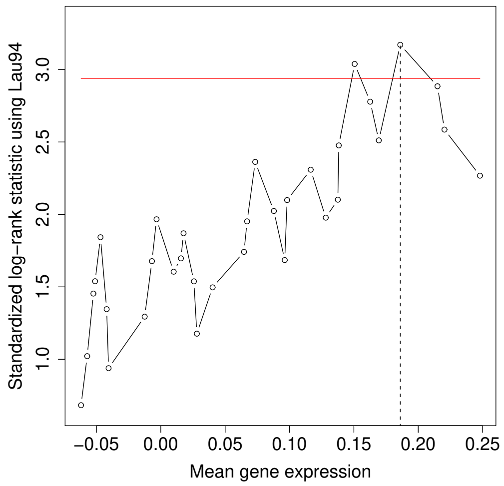

Figure 1: Absolute standardized log-rank statistics and significance bound based on the improved Bonferroni inequality.

For the overall survival time, the estimated cutpoint is 0.186 mean gene expression, the maximum of the log-rank statistics is $M = 3.171$. The probability that, under the null hypothesis, the maximally selected log-rank statistic is greater $M = 3.171$ is less then than 0.022. The empirical process of the standardized statistics together with the α-quantile of the null distribution can be plotted using `plot.maxtest`.

```
> data(DLBCL)
> mod <-
    maxstat.test(Surv(time, cens) ~ MGE,
                 data=DLBCL, smethod="LogRank",
                 pmethod="Lau94", alpha=0.05)
> plot(mod, xlab="Mean gene expression")
```

If the significance level `alpha` is specified, the corresponding quantile is computed and drawn as a horizonal red line. The estimated cutpoint is plotted as vertical dashed line, see Figure 1.

The difference in overall survival time between the two groups determined by a cutpoint of 0.186 mean gene expression is plotted in Figure 2. No event was observed for patients with mean gene expression greater 0.186.

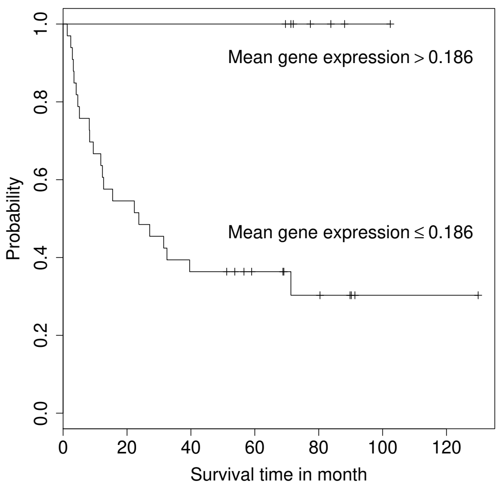

Figure 2: Kaplan-Meier curves of two groups of DLBCL patients separated by the cutpoint 0.186 mean gene expression.

Summary

The package **maxstat** provides a user-friendly interface and implements standard methods as well as recent suggestions for the approximation of the null distribution of maximally selected rank statistics.

Acknowledgement. Support from Deutsche Forschungsgemeinschaft SFB 539-A4 is gratefully acknowledged.

Bibliography

Ash A. Alizadeh, Michael B. Eisen, and Davis, R. E. et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–511, 2000. 4

Alan Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1:141–149, 1992. 4

Torsten Hothorn. On exact rank tests in R. *R News*, 1 (1):11–12, 2001. 4

Torsten Hothorn, Frank Bretz, and Alan Genz. On multivariate t and Gauss probabilities in R. *R News*, 1(2):27–29, 2001. 4

Torsten Hothorn and Berthold Lausen. On the exact distribution of maximally selected rank statistics. *Preprint, Universität Erlangen-Nürnberg, submitted*, 2001. 4

Berthold Lausen, Wilhelm Sauerbrei, and Martin Schumacher. Classification and regression trees (CART) used for the exploration of prognostic factors measured on different scales. In P. Dirschedl and R. Ostermann, editors, *Computational Statistics*, pages 483–496, Heidelberg, 1994. Physica-Verlag. 4

Berthold Lausen and Martin Schumacher. Maximally selected rank statistics. *Biometrics*, 48:73–85, 1992. 3, 4

Rupert Miller and David Siegmund. Maximally selected chi square statistics. *Biometrics*, 38:1011–1016, 1982. 4

Bernd Streitberg and Joachim Röhmel. Exact distributions for permutations and rank tests: An introduction to some recently published algorithms. *Statistical Software Newsletters*, 12(1):10–17, 1986. 4

Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Medizininformatik, Biometrie und Epidemiologie, Waldstraße 6, D-91054 Erlangen
`Torsten.Hothorn@rzmail.uni-erlangen.de`
`Berthold.Lausen@rzmail.uni-erlangen.de`

Quality Control and Early Diagnostics for cDNA Microarrays

by Günther Sawitzki

We present a case study of a simple application of R to analyze microarray data. As is often the case, most of the tools are nearly available in R, with only minor adjustments needed. Some more sophisticated steps are needed in the finer issues, but these details can only be mentioned in passing.

Quality control for microarrays

Microarrays are a recent technology in biological and biochemical research. The basic idea is: you want to assess the abundance of some component of a *sample*. You prepare a more or less well defined *probe* which binds chemically to this specific component. You bring probe and sample into contact, measure the amount of bound component and infer the amount in the original sample.

So far, this is a classical technique in science. But technology has advanced and allows high throughput. Instead of applying samples to probes one by one, we can spot many probes on a chip (a microscope slide or some other carrier) and apply the sample to all of them simultaneously and under identical conditions. Depending on the mechanics and adhesion conditions, today we can spot 10,000–50,000 probes on a chip in well-separated positions. So one sample (or a mixture of some samples) can give information about the response for many probes. These are not i.i.d. observations, but the probes are selected and placed on the chip by our choice, and response as well as errors usually are correlated. The data have to be considered as high dimensional response vectors.

Various approaches how to analyze this kind of data are discussed in the literature. We will concentrate on base level data analysis. Of course any serious analysis does specify diagnostic checks for validating its underlying assumptions. Or, to put it the other way, an approach without diagnostics for its underlying assumptions may be hardly considered serious. The relevant assumptions to be checked depend on the statistical methods applied, and we face the usual problem that any residual diagnostic is influenced by the choice of methods which is used to define fit and hence residuals. However there are some elementary assumptions to check which are relevant to all methods. If you prefer, you can name it quality control.

We use R for quality control of microarrays. We simplify and specialize the problem to keep the presentation short. So for now the aim is to implement an early diagnostic to support quality control for microarrays in a routine laboratory environment - something like a scatterplot matrix, but adapted to the problem at hand.

Some background on DNA hybridization

Proteins are chemical components of interest in biochemical research. But these are very delicate to handle, and specific probes have to be developed individually protein by protein. As a substitute, the genes controlling the protein production can be studied. The genes are encoded in DNA, and in the synthesis process these are transcribed to RNA which then enters to the protein production. DNA and transcribed RNA come in complementary pairs, and the complementary pairs bind specifically (*hybridization*). So for each specific DNA segment the complementary chain gives a convenient probe.

In old Lackmus paper, the probe on the paper provides colour as visible indicator. In hybridization experiments, we have to add an indicator. In DNA/RNA hybridization experiments the DNA is used as a probe. The indicator now is associated with the sample: the RNA sample is transcribed to cDNA (which is more stable) and marked, using dyes or a radioactive marker. For simplicity, we will concentrate on colour markers. After probe and sample have been hybridized in contact and unbound material has been washed off, the amount of observable marker will reflect the amount of bound sample.

Data is collected by scanning. The raw data are intensities by scan pixel, either for an isolated scan channel (frequency band) or for a series of channels. Conventionally these data are pre-processed by image analysis software. Spots are identified by segmentation, and an intensity level is reported for each spot. Additionally, estimates for the local background intensities are reported.

In our specific case, our partner is the Molecular Genome Analysis Department of the German cancer research center (DKFZ). Cancer research has various aspects. We simplify strongly in this presentation and consider the restricted question as to which genes are significantly more (or significantly less) active in cancer cell, in comparison to non-cancerous "normal" cells. Which genes are relevant in cancer? Thinking in terms of classical experiments leads to taking normal and tumor cell samples from each individual and applying a paired comparison. Detecting *differentially expressed* genes is a very different challenge from classical comparison. Gene expression analysis is a search problem in a high dimensional structured data set, while classical paired comparison deals with independent pairs of samples in a univariate context. But in the core gene expression analysis benefits from scores which may be motivated by a classical paired comparison.

Statistical routine immediately asks for factors and variance components. Of course there will be a major variation between persons. But there will be also a major contribution from the chip and hybridizations: the amount of sample provided, the sample preparations, hybridization conditions (e.g., temperature) and the peculiarities of the scanning process which is needed to determine the marker intensity by spot, among others are prone to vary from chip to chip. To control these factors or variance components, sometimes a paired design may be used by applying tumor and normal tissue samples on the same chip, using different markers (e.g., green and red dye). This is the basic design in the fundamental experiments.

The challenge is to find the genes that are relevant. The typical experiment has a large number of probes with a high variation in response between samples from different individuals. But only a small number of genes is expected to be differentially expressed ("*many genes few differentials*").

To get an idea of how to evaluate the data, we can think of the structure we would use in a linear gaussian model to check for differences in the foreground (fg). We would implement the background (bg) as a nuisance covariable and in principle use a statistics based upon something like

$$\Delta = (Y_{tumor} - Y_{normal})$$

where e.g.,

$$\begin{aligned} Y_{tumor} &= \ln(Y_{fg\ tumor} - Y_{bg\ tumor}) \\ Y_{normal} &= \ln(Y_{fg\ normal} - Y_{bg\ normal}) \end{aligned}$$

given or taken some transformation and scaling. Using log intensities is a crude first approach. Finding the appropriate transformations and scaling as a topic of its own, see e.g. Huber et al. (2002). The next non-trivial step is combining spot by spot information to gene information, taking into account background and unknown transformations. Details give an even more complex picture: background correction and transformation may be partly integrated in image processing, or may use information from other spots. But there is an idea of the general structure, and together with statistical folklore it suggests how to do an early diagnostics.

We hope that up to choice of scale Δ covers the essential information of the experiment. We hope we can ignore all other possible covariates and for each spot we can concentrate on $Y_{fg\ tumor}$, $Y_{bg\ tumor}$, $Y_{fg\ normal}$, $Y_{bg\ normal}$ as source of information. Up to choice of scale, $(Y_{fg\ tumor} - Y_{fg\ normal})$ represents the raw "effect" and a scatter plot of $Y_{fg\ tumor}$ vs $Y_{fg\ normal}$ is the obvious raw graphic tool.

Unfortunately this tool is stale. Since tumor and normal sample may have different markers (or may be placed on different slides in single dye experiments) there may be unknown transformations involved going from our target, the amount of bound probe specific sample, to the reported intensities Y. Since it is the most direct raw plot of the effect, the scatterplot is worth being inspected. But more considerations may be necessary.

Diagnostics may be sharper if we know what we are looking for. We want diagnostics that highlight the effect, and we want diagnostics that warn against covariates or violation of critical assumptions. For now, we pick out one example: spatial variation. As with all data collected from a physical carrier, position on the carrier may be an important covariate. We do hope that this is not the case, so that we can omit it from our model. But before running into artifacts, we should check for spatial homogeneity.

Diagnostic for spatial effects

For illustration, we use a paired comparison between tumor and normal samples. After segmentation and image pre-processing we have some intensity information $Y_{fg\ tumor}$, $Y_{bg\ tumor}$, $Y_{fg\ normal}$, $Y_{bg\ normal}$ per spot. A cheap first step is to visualize these for each component by position (row and column) on the chip. With R, the immediate idea is to organize each of these vectors as a matrix with dimensions corresponding to the physical chip layout and to use `image()`.

As usual, some fine tuning is needed. As has been discussed many times, in S and R different concepts of coordinate orientation are used for matrices and plots and hence the image appears rotated. Second, we want an aspect ratio corresponding to the geometric while image follows the layout of R's graphics regions. A wrapper `imagem()` around `image()` is used as a convenient solution for these details (and offers some additional general services which may be helpful when representing a matrix graphically).

Using red and green color palettes for a paired comparison experiment gives a graphical representation which separates the four information channels and is directly comparable to the raw scanned image. An example of these images is in `http://www.statlab.uni-hd.de/projects/genex/`.

The next step is to enhance accessibility of the information. The measured marker intensity is only an indirect indicator of the gene activity. The relation is influenced by many factors, such as sample amount, variation between chips, scanner settings. Internal scanner calibration in the scan machine (which may automatically adjust between scans) and settings of the image processing software are notorious sources of (possibly nonlinear) distortions. So we want to make a paired comparison, but each of both sides undergoes an unknown transformation. Here specific details of the experiment come to help. We cannot assume that both samples in a pair undergo the same transformation. But while we do not know the details of the transformations, we hope that at least each is guaranteed to be monotonous.

At least for experiments with "many genes - few differentials", this gives one of the rare situations where we see a bless of high dimensions, not a curse. The many spots scanned under comparable conditions provide ranks for the measured intensity which are a stable indicator for the rank of the original activity of each. So we apply `imagem()` to the ranks, where the ranks are taken separately for the four components and within each scan run.

The rest is psychology. Green and red are commonly used as dyes in these experiments or in presentations, but these are not the best choice for visualization. If it comes to judging quantitative differences, both colour scales are full of pitfalls. Instead we use a colour palette going from blue to yellow, with more lightness in the middle value range.

To add some sugar, background is compared between the channels representing tumor and normal. If we want a paired comparison, background may be ignorable if it is of the same order for both because it balances in differences. But if we have spots for which the background values differ drastically, background correction may be critical. If these points come in spatial clusters, a lot more work needs to be done in the analysis. To draw attention to this, spots with extremely high values compared to their counterpart are highlighted. This highlighting is implemented as an overlay. Since `image()` has the facility to leave undefined values as background, it is enough to apply `imagem()` again with the uncritical points marked as `NA`, using a fixed colour.

Wrapping it up

Rank transformation and image generation are wrapped up in a single procedure `showchip()`. Since the ranks cover all order information, but lose the original scale, marginal scatterplots are provided as well, on a fixed common logarithmic scale.

Misadjustment in the scanning is a known notorious problem. Estimated densities over all spots within one scan run are provided for the four information items, together with the gamma correction exponent which would be needed to align the medians.

If all conditions were fixed or only one data set were used, this would be sufficient. The target environment however is the laboratory front end where the chip scanning is done as the chips come in. Experimental setup (including chip layout) and sampling protocols are prone to vary. Passing the details as single parameters is error prone, and passing the data repeatedly is forbidding for efficiency reasons due to the size of the data per case.

The relevant information is bundled instead. Borrowing ideas from relational data bases, for each series of experiments one master list is kept which keeps references to tables or lists which describe details of the experiment. These details go from description of the geometry, over a representation of the actual experimental design to various lists which describe the association between spots and corresponding probes. S always had facilities for a limited form of object oriented programming, since functions are first class members in S. Besides data slots, the master list has list elements which are functions. Using enclosure techniques as described in Gentleman and Ihaka (2000), it can extract and cache information from the details. The proper data are kept in separate tables, and methods of the master list can invoke `showchip()` and other procedures with a minimum of parameter passing, while guaranteeing consistency which would endangered if global variables were used.

From the user perspective, a typical session may introduce a new line of experiments. The structure of the session is

```
curex <- NewGenex("<new project name>")
## create a new descriptor object from
## default.  if an additional parameter is given,
## it is a model to be cloned.

curex$nsetspotdesc("<some descriptor name>")
## we access other lists by name.
## This is risky, but for the present purpose
## it does the job ...
## Possibly many more experimental details

curex$save()
## The current experiment descriptor saves
## itself as <new project name>.RData
```

Once an experimental layout has been specified, it is attached to some specific data set as

```
curex$nsetdata("<some data name>")
## associates the data with the
## current experimental layout
```

Of course it would be preferable to set up a reference from the data to the experiment descriptor. But we have to take into account that the data may be used in other evaluation software; so we should not add elements to the low level data if we can avoid it.

When the association has been set up, application is done as

```
curex$showchip(<some chip identification>)
## names or indices of chip/chips to show.
```

This is a poor man's version of object oriented programming in R. For a full grown model of object oriented programming in R, see Chambers and Lang (2001).

Looking at the output

We cannot expect a spatially uniform distribution of the signals over the spots. The probes do not fall at random on the chip, but they are placed, and the placement may reflect some strategy or some tradition. In the sample output (Figure 1), we see a gradient from top to bottom. If we look closer, there may be a separation between upper and lower part. In fact these mirror the source of the genes spotted here. This particular chip is designed for investigations on kidney cancer. About half of the genes come from a "clone library" of kidney related genes, the other from genes generally expected to be cancer related. This feature is apparent in all chips from this special data series. The immediate warning to note is: it might be appropriate to use a model which takes into account this as a factor. On some lines, the spots did not carry an active probe. These unused spots provide an excellent possibility to study the null distribution. The vertical band structure corresponds to the plates in which the probes are provides; the square blocks come from characteristics of the print head. These design factors are well known, and need to taken into account in a formal analysis.

Of course these known features can be taken into account in an adapted version of showchip, and all this information is accessible in the lists mentioned above. In a true application this would be used to specialize `showchip()`. For now let us have another look at the unadapted plot shown in Figure 1.

Background is more diffuse than foreground—good, since we expect some smooth spatial variation of the background while we expect a grainy structure in the foreground reflecting the spot probes. High background spots have a left-right antisymmetry. This is a possible problem in this sample pair if it were unnoticed (it is is an isolated problem on this one chip specimen).

As in the foreground, there is some top/bottom gradient. This had good reason for the foreground signal. But for the background, there is no special asymmetry. As this feature runs through all data of this series of experiments, it seems to indicate a systematic effect (possibly a definition of "background" in the image processing software which picks up too much foreground signal).

Conclusion

`showchip()` is used for the first steps in data processing. It works on minimal assumptions and is used to give some first pilot information. For any evaluation strategy, it can be adapted easily if the critical statistics can be identified on spot level. For a fixed evaluation strategy, it then can be fine tuned to include information on the spatial distribution of the fit and residual contributions of each spot. This is where the real application lies, once proper models are fixed, or while discussing model alternatives for array data analysis.

An outline of how `showchip()` is integrated in a general rank based analysis strategy is given in Sawitzki (2001).

P.S. And of course there is one spot out.

Acknowledgements

This work has been done in cooperation with Molecular Genome Analysis Group of the German cancer research center (DKFZ) http://www.dkfz.de/abt0840/.

Thanks to Wolfgang Huber and Holger Sültmann for background information and fruitful discussions.

A video showing the experiments step by step is accessible as http://www.uni-hd.de/media/mathematik/microarrays.rm and available as DVD on CD from the DKFZ address.

Bibliography

J. M. Chambers and D. T. Lang. Objectoriented programming in R. *R News*, 3(1):17–19, September 2001. 9

R. Gentleman and R. Ihaka. Lexical scope and statistical computing. *Journal of Computational and Graphical Statistics*, 3(9):491–508, September 2000. 8

W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. (Submitted to Bioinformatics), January 2002. 7

G. Sawitzki. Microarrays: Two short reviews and some perspectives. Technical report, StatLab Heidelberg, 2001. URL http://www.statlab.uni-hd.de/projects/genex/. (Göttingen Lectures) Göttingen/Heidelberg 2001. 9

Günther Sawitzki
StatLab Heidelberg
gs@statlab.uni-heidelberg.de

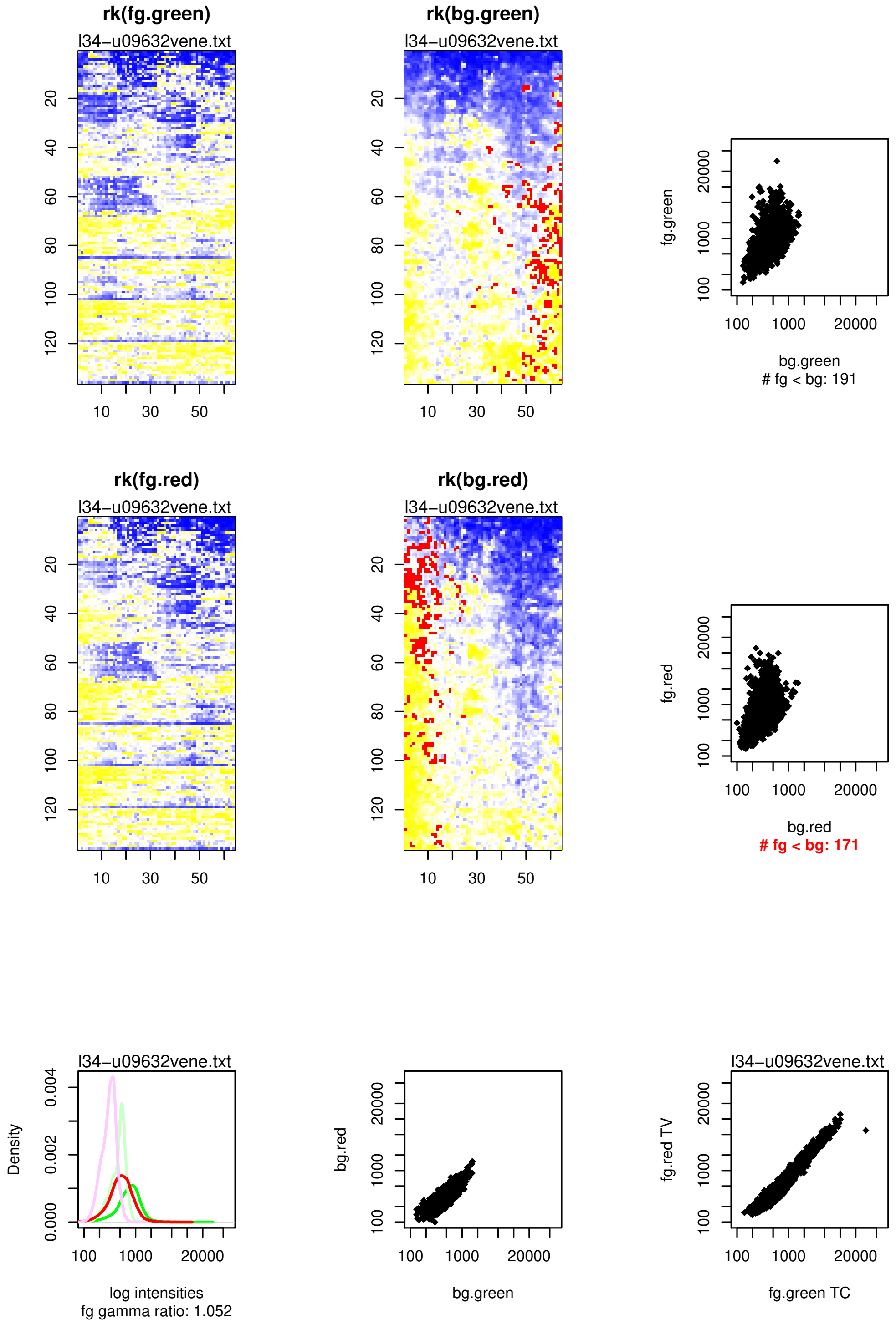

Figure 1: Output of `qcshowhip()`. This example compares tumor center material with tumor progression front.

Bioconductor

Open source bioinformatics using R

by Robert Gentleman and Vincent Carey

Introduction

One of the interesting statistical challenges of the early 21st century is the analysis of genomic data. This field is already producing large complex data sets that hold the promise of answering important questions about how different organisms function. Of particular interest is the ability to study diseases at a new level of resolution.

While the rewards appear substantial there are large hurdles to overcome. The size and complexity of the data may prevent many statisticians from getting involved. The need to understand a reasonable (or unreasonable) amount of biology in order to be effective will also be a barrier to entry for some.

We have recently begun a new initiative to develop software tools to make the analysis of these data sets easier and we hope to substantially lower the barrier to entry for statisticians. The project is called Bioconductor, `www.bioconductor.org`. We chose the name to be suggestive of cooperation and we very much hope that it will be a collaborative project developed in much the same spirit as R.

In this article we will consider three of the Bioconductor packages and the problems that they were designed to address. We will concentrate on the analysis of DNA microarray data. This has been the focus of our initial efforts. We hope to expand our offerings to cover a much wider set of problems and data in the future. The problems that we will consider are:

- data complexity: the basic strategy here is to combine object oriented programming with appropriate data structures. The package is **Biobase**.
- utilizing biological meta data: the basic strategy here is to use databases and hash tables to link symbols. The package is **annotate**.
- gene selection: given a set of expression data (generally thousands of genes) how do we select a small subset that might be of more interest? This is equivalent to ranking genes or groups of genes in some order. The package is **genefilter**.
- documentation: as we build a collection of interdependent but conceptually distinct packages, how do we ensure durable interoperation? We describe a *vignette* documentation concept, implemented using `Sweave` (a new function in R contributed by F. Leisch).

An overview of this area, provided by the European Bioinformatics Institute, may be found at `http://industry.ebi.ac.uk/~brazma/Biointro/biology.html` Also, a recent issue of the Journal of the American Medical Association (November 14, 2001) has a useful series of pedagogic and interpretive papers.

Handling data complexity

Most microarray experiments consists of some number of samples, usually less than 100, on which expression of messenger RNA (mRNA) has been measured. While there are substantial interesting statistical problems involving the design and processing of the raw data we will concentrate on the data that have been processed to the point that we have comparable expression data for a set of genes for each of our samples. For simplicity of exposition we will consider a clinical setting where the samples correspond to patients.

Our primary genomic data resource then consists of an expression array which has several thousand rows (representing genes) and columns representing patients. In addition we have data on the patients, such as age, gender, disease status, duration of symptoms and so on. These data are conceptually and often administratively distinct from the expression data (which may be managed in a completely different database). We will use the term *phenotypic* data to refer to this resource, acknowledging that it may include information on demographics or therapeutic conditions that are not really about phenotype.

What are efficient approaches for coordinating access to genomic and phenotypic data as described here? It is possible to couple the genomic and phenotypic data for each patient as a row in a single data frame, but this seems unnatural. We recognize that the general content and structure of expression data and phenotype data may evolve in different ways as biological technology and research directions also change, and that the corresponding types of information will seldom be treated in the symmetric fashion implied by storing them in a common data frame. Our approach to data structure design for this problem preserves the conceptual independence of these distinct information types and exploits the *formal methods and classes* in the package **methods** which was developed by J. M. Chambers.

Class `exprSet` and its methods

The **methods** package provides a substantial basis for object-oriented programming in R. Object-oriented programming has long been a tool for deal-ing with complexity. If we can find a suitable representation for our data then we can think of it as an object. This object is then passed to various routines that are designed to deal with the special structure in the objects.

An object has *slots* that represent its different components. To coordinate access to the genomic and phenotypic data generated in a typical microarray experiment, we defined a class of objects called `exprSet`. An instance of this class has the following slots:

exprs An array that contains the estimated expression values, with columns representing samples and rows representing genes.

se.exprs An array of the same size as `exprs` containing estimated standard errors. This may be `NULL`.

phenoData An instance of the `phenoData` class which contains the sample level variables for this experiment. This class is described subsequently.

description A character variable describing the experiment. This will probably change to some form of documentation object when and if we adopt a more standard mechanism for documentation.

annotation This is a character string indicating the name of the annotation data that can be used for this `exprSet`.

notes A character string for notes regarding the analysis or for any other purpose.

Once we have defined the class we may create instances of it. A particular data set stored in this format would be called an instance of the class. While we should technically always say something like: *x is an instance of the* `exprSet` *class*, we will often simply say that *x is an* `exprSet`.

We have implemented a number of methods for the `exprSet` and `phenoData` classes. Most of these are preliminary and some will evolve as our understanding and the data analytic processing involved mature. The reader is directed to the documentation in our packages for definitive descriptions.

We would like to say a few words here about the subsetting methods. By implementing special subsetting methods we are able to ensure that the data remain correctly aligned. So, if `x` is an instance of an `exprSet`, we may ask for `x[,1:5]`. The second index in this subscripting expression refers to tissue samples. The value of this expression is an `exprSet` whose contents are restricted to the first five samples in `x`. The new `exprSet`'s `exprs` element contains the first five columns of `x`'s `expr` array. The new `exprSet`'s `se.exprs` array is similarly restricted. But the `phenoData` must also have a subset made and for it we want the first five *rows* of `x`'s phenoData data frame, since it is in the more standard format where columns are variables and rows are cases.

Notice that by representing our data with a formal class we have removed a great deal of the complexity associated with the data. We believe that this representation relieves the data analyst of complex tasks of coordination and checking of diverse data resources and makes working with `exprSets` much simpler than working with the raw data.

We have also defined the `$` operator to work on exprSets. This operator selects variables of the appropriate name from the `phenoData` component of the `exprSet`.

The `phenoData` class was designed to hold the phenotypic (or sample level) data. Instances of this class have the following slots:

pData This is a `data.frame` with samples as the rows and the phenotypic variables as the columns.

varLabels This is a list with one element per phenotypic variable. Names of list elements are the variable names; list element values are character strings with brief textual descriptions of the associated variables.

We find it very helpful to keep descriptions of the variables associated with the data themselves. Such a construct could be helpful for all `data.frames`.

Example

The **golubEsets** package includes `exprSets` embodying the leukemia data of the celebrated Science paper of T. Golub and colleagues (Science 286:531-537, 1999). Upon attaching the `golubTrain` element of the package, we perform the following operations.

Show the data. This gives a concise report.

```
> golubTrain
Expression Set (exprSet) with
 7129 genes
 38 samples
  phenoData object with 11 variables and 38 cases
  varLabels
   Samples: Sample index
   ALL.AML: Factor, indicating ALL or AML
   BM.PB: Factor, sample from marrow or \
     peripheral blood
   T.B.cell: Factor, T cell or B cell leuk.
   FAB: Factor, FAB classification
   Date: Date sample obtained
   Gender: Factor, gender of patient
   pctBlasts: pct of cells that are blasts
   Treatment: response to treatment
   PS: Prediction strength
   Source: Source of sample
```

Subset expression values. This gives a matrix. Note the gene annotation, in Affymetrix ID format.

```
> exprs(golubTrain)[1:4,1:3]
             [,1] [,2] [,3]
AFFX-BioB-5_at -214 -139  -76
AFFX-BioB-M_at -153  -73  -49
AFFX-BioB-3_at  -58   -1 -307
AFFX-BioC-5_at   88  283  309
```

Tabulate stratum membership. This exploits the redefined $ operator.

```
> table(golubTrain$ALL.AML)

ALL AML
 27  11
```

Restrict to the ALL patients. We obtain the dimensions after restriction to patients whose phenoData indicates that they have acute lymphocytic leukemia.

```
> print(dim(exprs(golubTrain[ ,
            golubTrain$ALL.AML=="ALL"])))

[1] 7129   27
```

Other tools for working with `exprSets` are provided, including methods for iterating over genes with function application, and sampling from patients in an `exprSet` (with or without replacement). The latter method returns an `exprSet`.

Annotation

Relating the genes to various biological data resources is essential. While there is much to be learned and many of the biological databases are very incomplete it is never the less essential to start employing these data. Again our approach is quite simple but it has proven effective and we are able to both analyse the data we encounter and to provide resources to others.

Our approach has been to divide the process into two components. One process is the building and collation of annotation data from public databases, and the other process is the extraction and formatting of the collated data for analysis reporting. Data analysts using Bioconductor will typically simply obtain a set of annotation tables for the chip data they are using. Given the appropriate tables they can select genes according to certain conditions such as functional group, or biological process or chromosomal location.

The process of building annotation data sets is carried out by the **AnnBuilder** package. We distribute this but most users will not want to build their own annotation. They will want instead to have access to specific annotation for their chip. **AnnBuilder** relies on several R packages (available from CRAN) including D. Temple Lang's **XML** and T. Keitt's **RPgSQL**.

Typically the annotation data available are a complete set for all genomes or a complete set for a specific genome, such as yeast or humans. Since this is typically much larger than what is currently available on any microarray and in the interest of performance we have found that producing annotation collections at the level of a chip has been quite successful.

The data analyst simply needs to ensure that they have an annotation collection suitable for their chip. A number of these are available from the Bioconductor web page and given the appropriate reference data we can provide custom made collections within a few days.

Currently these collections arrive as a set of comma separated files. The first entry in each row is the identification name or label for the gene. This should correspond to the row names of the `expr` value in the `exprSet` that is being analysed. The annotation filenames should have their first five letters identical to the value in the `annotation` slot of the `exprSet` since this is used to align the two. This package will become more object oriented in the near future and the distribution mechanism will change.

Affymetrix Inc. produces several chips for the human genome. The most popular in current use are the U95v2 chips (with version A being used most often). The probes that are arrayed here have Affymetrix identifiers. We provide functions that map the Affymetrix identifiers to a number of other identifiers such as the Locus Link value, the GenBank value. In addition we have mapped most probes to their Genome Ontology (GO) values http://www.geneontology.org. GO is an attempt to provide standardized descriptions of the biological relevance of genes into the following three categories:

- Biological process
- Cellular component
- Molecular function

Within a category the set of terms forms a directed acyclic graph. Genes typically get a specific value for each of these three categories. We trace each gene to the top (root node) and report the last three nodes in its path to the root of the tree. These top three nodes provide general groupings that may be used to examine the data for related patterns of expression.

Additional data, such as chromosomal location, is also available. As data become available we will be adding it to our offerings. Readers with knowledge of data that we have not included are invited to submit this information to us.

The annotation files are read in to R as environments and used as if they were hash tables. Typically the name they are stored under is the identifier (but that does not need to be the case). The value is can then be accessed using `get`. Since we often want to get or put multiple values (if we have 100 genes we would like to get chromosome location for all of them) there are functions `multiget` and `multiassign`.

Often researchers are interested in finding out more about their genes. For example they would like to look at the different resources at NCBI. We can easily produce HTML pages with active links to the different online resources. `ll.htmlpage` is an example of a function designed to provide links to the Locus Link web page for a list of genes.

It is also possible using connections and the **XML** package to open http connections and read the data from the web sites directly. It could then be processed using other R tools. For example if abstracts or full text searchable articles were available these could be downloaded and searched for relevant terms.

The remainder of this section involves reference to Web sites and R functionality that may change in the future. This portion of the project and the underlying data are in a state of near constant change. An up-to-date version of the material in this section, along with relevant scripts and data images, will be maintained at http://www.bioconductor.org/rnews0102.html. If you have problems with any of the commands described below, please consult this URL.

To illustrate how one can get started analyzing publicly distributed expression data with R, the following commands lead to a matrix `numExprs` containing the Golub training data, and a character vector of Affymetrix expression tags `accTags`:

```
golURL <-
  url(paste("http://www-genome.wi.mit.edu/",
            "mpr/data_set_ALL_AML_train.txt",
            sep = ""),
      "r")
golVec <- scan(golURL, sep = "\t", what = "")
## next command patches up a missing blank
golVec <- c(golVec[1:78], "", golVec[-(1:78)])
golMat <- t(matrix(golVec, nr = 79))
accTags <- golMat[-1, 2]
cExprs <- golMat[-1, seq(3, 78, 2)]
numExprs <- t(apply(cExprs, 1, as.numeric))
```

Let's see what the Affymetrix tags look like:

```
t(t(accTags[c(10,20,30)]))
     [,1]
[1,] "AFFX-BioB-5_st"
[2,] "AFFX-DapX-5_at"
[3,] "AFFX-ThrX-M_at"
```

Using the `annotate` package of Bioconductor, we can map these tags to GenBank identifiers:

```
library(annotate)
library(Biobase)
HGu952genBank() # set up map
multiget(accTags[c(10,20,30)],
         env=HGu95togenBank)
```

The result is a list of GenBank identifiers:

```
$"AFFX-BioB-5_st"
[1] "J04423"

$"AFFX-DapX-5_at"
[1] "L38424"

$"AFFX-ThrX-M_at"
[1] "X04603"
```

Upon submitting the tag "J04423" to GenBank, we find that this gene is from E. coli, and is related to biotin biosynthesis.

A long range objective of the Bioconductor project is to facilitate the production and collation of interpretive biological information of this sort at any stage of the process of filtering and modeling expression data.

Gene filtering

In this section we consider the process of selecting genes that are associated with clinical variables of interest. The package **genefilter** is one approach to this problem. We think of the selection process as the sequential application of filters. If a gene passes all of the filters then we deem it interesting and select it.

For example, genes are potentially interesting if they show some variation across the samples that we have. They are potentially interesting if they have estimated expression levels that are high enough for us to believe that the gene is actually expressed. These are both examples of non-specific filters; we have not made any reference to a covariate. Covariate-based filters specify conditions of magnitude or variation in expression within or between strata defined by covariates that must be satisfied by a gene in order that it be retained for subsequent filtering and analysis.

Closures for non-specific filters

A filter function will usually have some context-specific parameters to define its behavior. To make it easy to associate the correct settings with a filter function we have used R lexical scoping rules. Here is the function `kOverA` that filters genes by requiring that at least k of the samples have expression values larger than A.

```
kOverA <- function(k, A=100, na.rm = TRUE) {
  function(x) {
    if(na.rm)
      x <- x[!is.na(x)]
    sum( x > A ) > k
  }
}
```

To define a filter that requires a minimum of five samples to have expression level at least 150, we simply evaluate the expression

```
myK <- kOverA(5, 150)
```

Now `myK` is a function, since that is what `kOverA` returns. `myK` takes a single argument, `x`, which will be a vector of gene expressions (one expression level for each sample) and evaluates the body of the inner function in `kOverA`. In that body `A`, `k` and `na.rm` are unbound symbols. They will obtain their values as those that we specified when we created the closure (the coupling of the function body with its enclosing environment) `myK`. If `es` is an `exprSet`, the result of `apply(exprs(es),1,myK)` is a logical vector with gth element `TRUE` if gene g (with expression levels stored in row g of the `exprs` slot of `es`) has expression level at least 150 in 5 of the samples, and `FALSE` otherwise.

Another more interesting non-specific filter is the gap filter. It selects genes using a function that has three parameters. The first is the gap, the second is the IQR and the third is a proportion. To apply this filter the (positive) expression levels for the gene are sorted and the proportion indicated are removed from both tails. If in the remainder of the data there is a gap between adjacent values of at least the amount specified by the gap parameter then the gene will pass the filter. Otherwise, if the IQR based on all values is larger than the specified IQR the gene passes the filter. The purpose of this filter is to select genes that have some variation in their expression values and hence might have something interesting to say.

Covariate-dependent filters

An advantage to this method is that any statistical test can be implemented as a filter. Functions implementing covariate-dependent filters are built through closures, have the same appearance as simpler non-specific filters, and are `apply`'d to expression level data in precisely the same way as described above.

Here is a filter-building function that uses the partial likelihood ratio to test for an association between a censored survival time and gene expression levels.

```
coxfilter <- function (surt, cens, p)
{
  autoload("coxph", "survival")
  function(x) {
    srvd <- try(coxph(Surv(surt, cens) ~ x))
    if (inherits(srvd, "try-error"))
      return(FALSE)
    ltest <- -2 * (srvd$loglik[1] -
                     srvd$loglik[2])
    pv <- 1 - pchisq(ltest, 1)
    if (pv < p)
      return(TRUE)
    return(FALSE)
  }
}
```

To use this method, we must create the closure by establishing bindings for the survival time, censoring indicator, and significance level for declaring an association. Ordinarily, the survival data will be captured in the `phenoData` slot of an expression set, say `es`, so we will see something like

```
myCox <- coxfilter( es$stime, es$event, 0.05 )
```

Now `apply(exprs(es), 1, myCox)` is a logical vector with element `TRUE` at index g whenever gene g is associated with survival, and `FALSE` otherwise.

We have provided other simple filters such as a t-test filter and an ANOVA filter. One can simply set up the test and require that the p-value of the test when applied to the appropriate covariate (from the phenoData slot) is smaller than some prespecified value.

In one designed experiment we used testing within gene-specific non-linear mixed effects models to measure the association between gene expression and a continuous outcome. The amount of programming required was minimal. Filtering in this simple way provides us with a very natural paradigm and tool in which to carry out gene selection. Note however, that in some situations we will need to deal with more than one gene at a time.

Equipped with the lexical scoping rules, we have separated the processes of generic filter specification (filter building routines supplied in the **genefilter** package), selection of detailed filter parameters (binding of parameters when the closure is obtained from the filter builder), and application of filters over large numbers of genes. For the latter process we have given examples of "manual" use of `apply`, but more efficient tools for sequential filtering are supplied through the `filterfun` and `genefilter` functions in the package.

A word of warning is in order for those who have not worked extensively with function closures. It is easy to forget that the function is a closure and has some variables captured. Seemingly correct results can obtain if you are not careful. The alternative is to specify all parameters and strata at filtering time. But we find that makes the filtering code much more complicated and less intuitive.

Documentation

We are embarked on the production of packages and protocols to facilitate greater involvement of statisticians in bioinformatics, and to support smoother collaboration between statisticians, biologists and programmers when working with genomic data. Documentation of data structures, methods, and analysis procedures must be correct, thorough, and accessible if this project is to succeed.

R has a number of unique and highly effective protocols for the creation (`prompt`) and use of documents at the function and package level that facilitate simple interactive demonstration of function behaviors (the `example` function, which applies to help topics) and unit testing of functions and packages at build time (the testing carried out by `R CMD check`).

We have introduced two new documentation protocols as we grow the Bioconductor project.

`promptClass`

A revised `promptClass` function has been developed which produces a template that illustrates instantiation of classes (using the `new` function) and searches through the searchlist to obtain information on formal methods that apply to the class being documented. Thus the creator of a new class is motivated to describe not only the structure of the class being created, but also the family of methods that may be used to work with this class.

The *vignette* protocol

The intent of the `CMD check` unit testing protocols of R is to establish that the elements of a package function in accordance with the description in the manual pages. The Bioconductor project has two additional unit testing requirements. First, packages must work together in well-defined ways: examples involving multiple packages and diverse data structures must interoperate correctly and continuously while evolving. Second, users will require 'higher-level' views of programming processes than are typically afforded by package man pages. Users will need to be taken through the steps of data structure creation, searchlist extension, and then through the details of any particular data processing or inference procedure, with considerable narrative guidance.

Sweave allows integrated session narration, shell monitoring, and graphics incorporation in documents that are specified in LaTeX and are renderable using PDF. We propose that Sweave documents (with suffix '.Rnw') and PDF compilations of them be kept in the 'inst/doc' subdirectory of packages related to the Bioconductor project.

Conclusions

Analytical computing for bioinformatics has been characterized to date by the proliferation of separated binaries or scripts that implement various analysis methods in very easy to use formats (e.g., MicroSoft Excel modules, Java applets). This paradigm has a number of drawbacks, including frequent reliance upon idiosyncratic data input and output formats, high costs of establishing interoperability, difficulties of incorporating new methodological insights into existing programs, minimal infrastructure to verify procedure portability, and non-standard documentation.

The Bioconductor project is based on the award-winning interactive programming language S, as deployed in the open-source statistical computing environment R. An interactive programming language provides an ideal platform for prototyping implementations of new ideas and comparing the new approaches to well-established older approaches. The language basis confers well-defined paths to extension and interoperation of any routines. Procedures that are too intensive to work routinely in the interactive environment may be coded in any higher-level language and loaded dynamically in R for higher performance with no change to the user interface. The object-oriented programming facilities provide access to state-of-the-art methods in data structure and software design that help to control code complexity and reduce obstacles to smooth interoperation of diverse procedures. The intersystems interfacing initiative at `www.omegahat.org` provides tools to permit smooth interoperation of Bioconductor routines with completely foreign systems such as relational databases, browsers, or other virtual machines.

The role of R in bioinformatics research has been substantial, with contributions already present on CRAN (**sma**, **GeneSOM**, etc.) and its use in many projects for functional genomics. It is our hope that the Bioconductor project will aid in the federation of programming efforts to support progress in many different areas of biology and clinical research. Those who are interested in this project should visit our web site and consider subscribing to the bioconductor mailing list.

Acknowledgment

Vincent Carey's work on Bioconductor is supported in part by US National Heart Lung and Blood Institute Grant HL66795, Innate Immunity in Heart Lung and Blood Disease.

Robert Gentlemans's work is supported in part by NIH/NCI Grant 2P30 CA06516-38.

Robert Gentleman
DFCI
`rgentlem@jimmy.harvard.edu`

Vincent Carey
Channing Lab
`stvjc@channing.harvard.edu`

AnalyzeFMRI: An R package for the exploration and analysis of MRI and fMRI datasets

by Jonathan Marchini

AnalyzeFMRI is a developing package for the exploration and analysis of large MRI and fMRI datasets. In this article we give a short introduction to MRI and fMRI and describe how we have used R when working with these large datasets. We describe the current version of the package using examples, describe our own approach for fMRI analysis and outline our plans for future functionality in the package.

MRI and fMRI

Magnetic Resonance Imaging (MRI) is a non-invasive medical imaging technique that provides images that represent slices of (brain) tissue. Essentially, measurements occur within each slice on a grid of cube-like *vol*ume *el*ements (*voxels*). These measurements reflect the chemical and magnetic properties of the tissue in each small voxel and result in *structural* MRI images that show detailed contrast between different tissue types (see http://www.stats.ox.ac.uk/~marchini/pictures/high.res/gif). Variations of the imaging parameters sensitize the images to different chemical and physical properties of interest.

The relationship between *functional* MRI (fMRI) and *structural* MRI is analogous to the relationship between still photography and movies. In simple terms fMRI is fast, repeated structural MRI and can be carried out in such a way so as to be sensitive to local changes in levels of brain activity. Increases in local brain activity increase the local levels of blood oxygen. This in turn causes the measured signal to increase. Thus the images produced are said to be *Blood Oxygenation Level Dependent* (BOLD). Through the BOLD mechanism we can use fMRI datasets to localize specific areas or networks of the brain that are responsible for cognitive functions of interest. In this fashion fMRI has revolutionized the area of neuroscience over the last 10 years. An excellent introduction to this area is given by Matthews et al. (2001)

In a typical fMRI experiment the subject is repeatedly imaged whilst performing a task(s) or receiving certain stimuli designed to elicit the cognitive function of interest. Traditionally, tasks/stimuli are applied in alternating blocks of 20–30 seconds. More recently, 'event-related' designs in which the stimulus is applied for short bursts in a stochastic manner have become popular.

In those areas of the brain that are activated by the tasks/stimuli we will observe that the voxel time series are correlated with the design of the experiment (see Figure 1). Focus usually centers on delineating those areas (clusters of voxels) of the brain that exhibit a significant response. Most often we will wish to make inferences using a group of subjects.

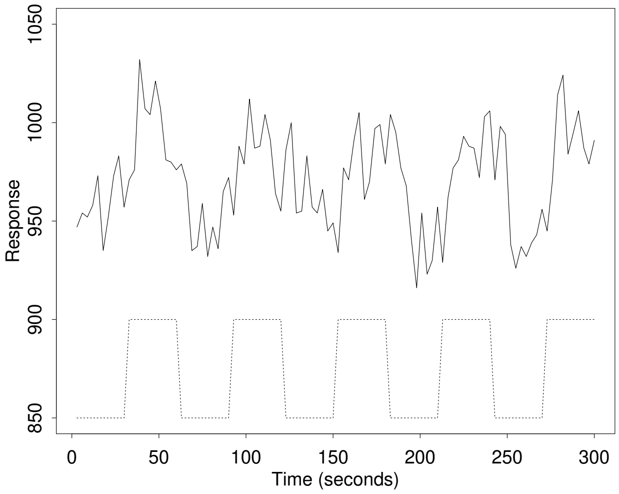

Figure 1: A real voxel time series (solid line) in an area of the brain activated by an alternating 'OFF/ON' visual stimulus (dotted line).

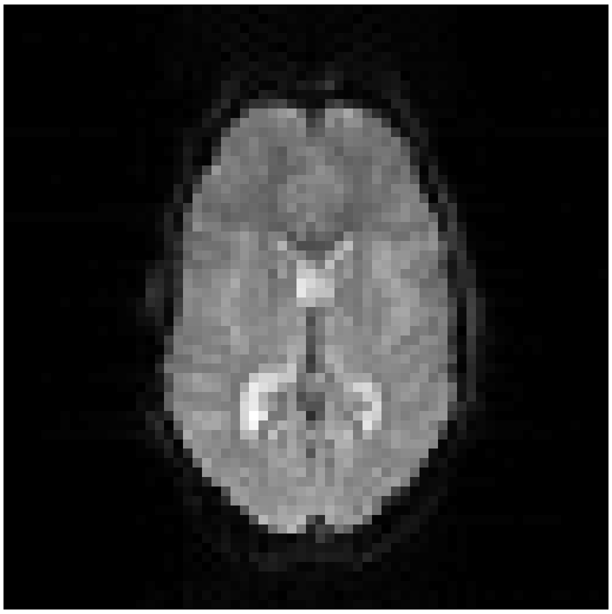

Figure 2: A functional MRI image

The resolution of fMRI datasets is very good. Typically datasets obtained using this technique consist of whole brain scans (≈ 20 images) repeated every 2-3 seconds. Usually each image consists of 64×64 voxels of size $(4\text{mm})^3$ (see Figure 2).

In summary, an fMRI dataset consists of a 3D grid of voxels, each containing a time series of measurements that reflect brain activity. Out of roughly 15,000 voxels that lie inside the brain we wish to identify those that were activated.

Using R for fMRI datasets

In order to analyze fMRI experiments we need to be able to manipulate and visualize the large amounts of data in a relatively easy fashion. This in turn allows us to implement and evaluate existing approaches and develop new methods of analysis. The analysis will often involve several computationally intensive steps and so it is also important to keep an eye on the efficiency of the code.

The high level environment that R provides has allowed us to implement and test our methods quickly and reliably. During this process we have used profiling facilities to identify areas of our code that can be speeded up (Venables, 2001) and where necessary we have written C and Fortran code to achieve this (Venables and Ripley, 2000). We have often found that the memory overheads of carrying out all calculation within R can be high. For this reason we have often used R simply to pass file names and analysis parameters to C code. All file I/O and computations are carried out in C before writing the results to a new data file. In addition we have found it useful to write simple Graphical User Interface's (GUI's) using the **tcltk** package. This allows analyses to be set up and run without recourse to command line functions.

The ANALYZE format and I/O

The ANALYZE image format is a general medical image format developed by the Mayo Clinic[1] that is commonly used within the brain imaging community. The format consists of an image file ('foo.img') together with a header file ('foo.hdr'). The header file details the binary storage type of the image file, dimensions of the dataset and certain imaging parameters used during acquisition. In addition the endianness of the image and header files can be detected by reading the first 4 bytes of the header file, as they contain the constant header file size (348 bytes). The size of a typical fMRI dataset stored in this format is 30Mb.

The package provides read and write capabilities for the ANALYZE format. For example, a simple summary of the dataset is available.

```
> f.analyze.file.summary("./ex.img")
       File name: ./ex.img
  Data dimension: 4-D
     X dimension: 64
     Y dimension: 64
     Z dimension: 21
  Time dimension: 1 time points
Voxel dimensions: 4 mm x 4 mm x 6 mm
       Data type: signed short
                  (16 bits per voxel)
```

[1] http://www.mayo.edu/bir/PDF/ANALYZE75.pdf

Functions of the form `f.read.analyze.*` allow whole or parts of the dataset to be read into R, i.e.,

```
> a <- f.read.analyze.volume("./ex.img")
> dim(a)
[1] 64 64 21 1
```

The function `f.write.analyze` allows an array to be written into a new ANALYZE format file, i.e.,

```
> a <- matrix(rnorm(1000),dim=rep(10,3))
> f.write.analyze(a, file="./ex2", size="float")
```

In addition `f.basic.hdr.list.create` and `f.write.list.to.hdr` allow the user to store custom information in the header files.

Exploratory Data Analysis

fMRI datasets are large and can contain significant noise structure and imaging artifacts. The quality of datasets can vary widely from scanner to scanner and even from day to day on the same scanner. For these reasons it is important to regularly 'eyeball' the datasets to examine their quality. The package provides two methods of examining the structure present in each fMRI dataset.

The first is a simple spectral summary of the dataset produced using `f.spectral.summary`. This function calculates the periodogram of each voxel time series normalized by the median periodogram ordinate. A plot is produced that shows the quantiles of the normalized periodogram ordinates at each frequency. This provides a fast look at a fMRI dataset to identify any artifacts that reside at single frequencies. Figure 3 shows the results of applying this function to a real fMRI dataset. The spike at frequency 18 is consistent with the frequency of the periodic stimulus of the experiment. The spike at frequency 60 highlights the existence of a Nyquist ghost image artifact that can occur in fMRI datasets.

```
> f.spectral.summary(file="ex3.img",
                     mask.file="ex3.m.img",
                     ret.flag=FALSE)
Processing slices... [1] [2] [3] [4] [5] [6] [7]
[8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[18] [19] [20] [21]
```

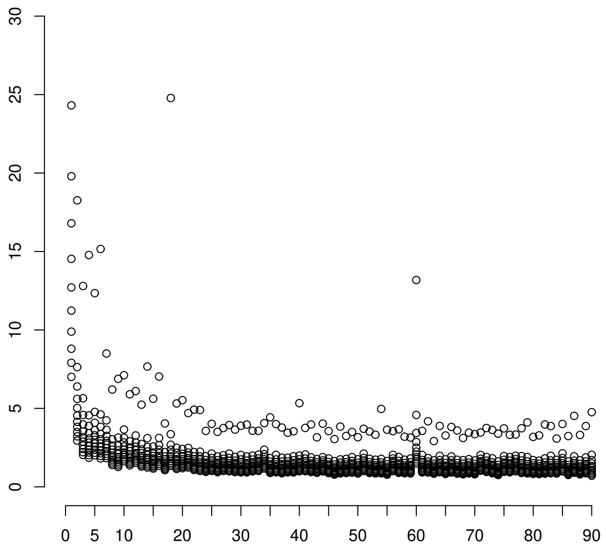

Figure 3: Spectral summary plot of an fMRI dataset.

We provide a GUI that allows visualization of individual voxel time-series, images of specific slices and functional volumes, movies through time of specific slices and spectral summary plots. This GUI is invoke through the function `f.analyzeFMRI.gui`.

Secondly, we have applied Independent Component Analysis (ICA) to fMRI datasets and found that the extracted components are extremely useful in uncovering otherwise hidden structure. In ICA the data matrix X is considered to be a linear combination of non-Gaussian (independent) components, i.e., $X = SA$ where columns of S contain the independent components and A is a linear mixing matrix. In short ICA attempts to 'un-mix' the data by estimating an un-mixing matrix W where $XW = S$.

Under this generative model the measured 'signals' in X will tend to be 'more Gaussian' than the source components (in S) due to the Central Limit Theorem. Thus, in order to extract the independent components/sources we search for an un-mixing matrix W that maximizes the non-Gaussianity of the sources. It is useful to note that in the absence of a generative model for the data the algorithm used is equivalent to Projection Pursuit. We use the Fast ICA algorithm (Hyvarinen, 1999) implemented in package **fastICA** to estimate the sources (see that package for more details).

For fMRI data we concatenate the dataset into a data matrix so that each row of X represents one voxel time series. Using this formulation allows us to estimate spatially independent sources that occur in the dataset. The function `f.ica.fmri` calls C code that reads the data directly from the ANALYZE image file, carries out the ICA decomposition and returns the results into R. We also provide a GUI written using **tcltk** that allows a fast way of visualizing the results of the ICA decomposition. This GUI is invoked through the function `f.ica.fmri.gui` (see Figure 4).

Figure 5 shows one of the estimated components (columns of S) plotted in its spatial configuration together with its associated weighting through time (row of A). This particular component shows the high frequency Nyquist ghosting artifact suggested by the spectral summary plot.

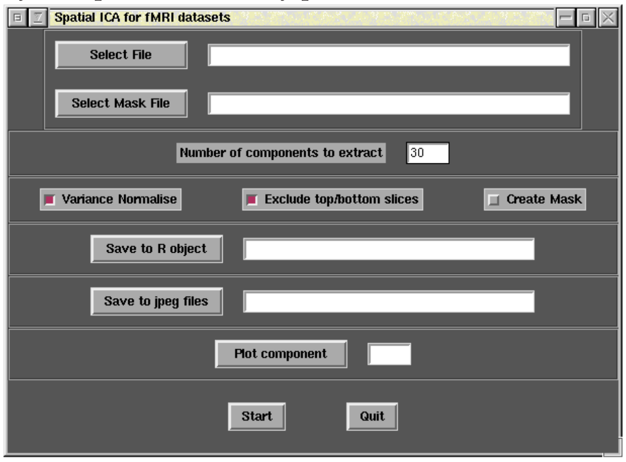

Figure 4: The GUI for spatial ICA for fMRI datasets.

Analysis

The most widely used strategy for the analysis of fMRI experiments is carried out using a two-stage approach. In the first stage a linear model with correlated errors is used for each individual voxel time-series Y. That is,

$$Y = X\beta + \epsilon \quad \epsilon \sim N(0, \sigma^2 V) \tag{1}$$

where the design matrix X contains terms that model the BOLD response to the stimuli and non-linear trends that are often observed in fMRI voxel time-series.

Once this model has been fitted at each voxel, summary statistics (typically t and F statistics) are calculated that reflect the components of the response under study. For example, we might calculate an F statistic that reflects the variance component attributable to the BOLD response. In the context of a group study the statistic at each voxel is calculated from the model fits of all the subjects. These statistic values can then be plotted spatially as a statistic image (see figure 8). The second stage then focuses on the analysis of the statistic image in order to identify those areas of the brain that were activated by the stimuli.

Time-series modelling

In general, the linear model (1) used at each voxel will contain three main components. These being (i) non-linear trends, (ii) the BOLD response to the stimulus, and (iii) auto-correlated noise.

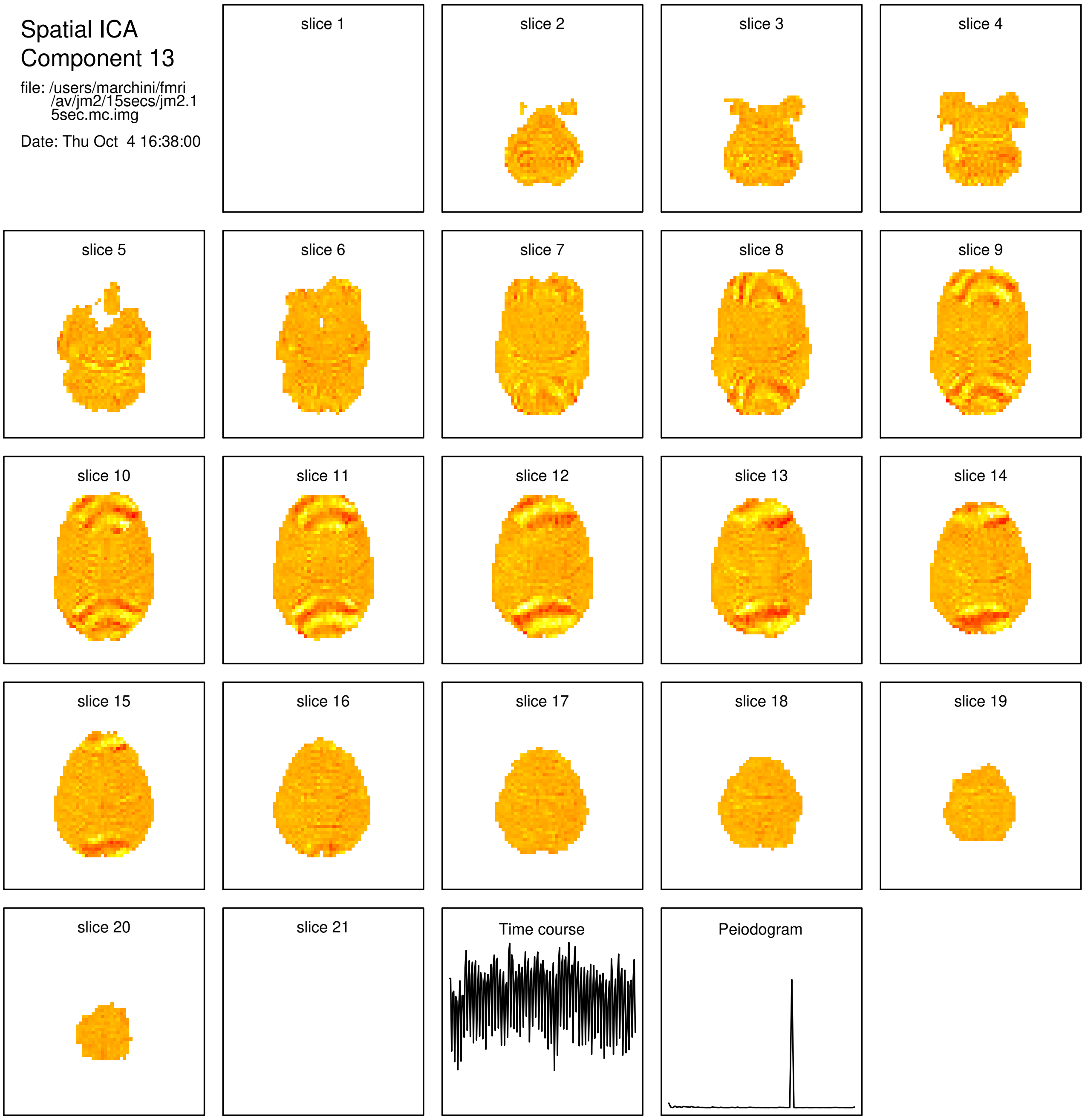

Figure 5: An extracted spatial ICA component showing a high frequency Nyquist ghosting artifact.

Trends are commonly modelled using polynomial, cosine or spline basis terms. Alternatively the trends can be removed from the data before analysis by applying an appropriate filter/smoother. We have found a Gaussian weighted running lines smoother reliably removes the non-linear trends (Friedman, 1984). For efficiency we avoid multiple calls to smoothing functions (such as `supsmu`) by initially calculating a smoothing matrix S and applying it at each voxel.

The nature of the BOLD response implies that in areas of activation we will observe a delayed and blurred version of the stimulus design. This is commonly modelled through the convolution of the stimulus design $x(t)$[2] with a *Haemodyanmic Response Function* (HRF), $h(t)$. That is,

$$BOLD(t) = \sum_s h(t-s)x(s) = h \otimes x(t) \qquad (2)$$

Suggested forms for the HRF include discretized Poisson, Gamma and Gaussian density functions. To allow for spatial variation in the HRF at each voxel a small set of basis HRF's can be used that vary in the amount they blur and delay the stimulus design

[2] normally a vector of 1's and 0's that specify the times when the stimulus is 'ON' and 'OFF' respectively.

(Friston et al., 1995). Each HRF is convolved with the design to make one column of the design matrix X.

The final component in the linear model is the auto-correlated noise term. The auto-correlation structure varies considerably throughout the brain and the chosen model should be chosen to reflect this fact. Suggested models include AR(p) (Bullmore et al., 1996), ARMA models (Locascio et al., 1997) and non-parametric spectral density estimates (Lange and Zeger, 1997). Alternatively, Worsley and Friston (1995) suggest imposing a known correlation structure on the model (using a low-pass filter) before using OLS to fit the model at each voxel. The authors point out that this scheme works for periodic designs[3] but is inefficient for more interesting event-related designs.

In general the chosen noise model should be flexible enough to capture the spatially varying levels of correlation throughout the brain. The model should provide well-calibrated levels of significance for estimated responses and be resistant to commonly occurring image artifacts.

A spectral domain approach

For periodic designs the specification of the HRF and noise model at each voxel time-series is greatly simplified by working in the spectral domain (Marchini and Ripley, 2000). The simplest example of a periodic design involves c repeats of a block which consists of b volumes scanned during a baseline stimulation, followed by b volumes scanned during a stimulus/task of interest. For such designs the majority of the power of the BOLD response will lie at just one frequency $\omega_c = 2\pi c/n$ and will result in a spike at that frequency in the periodogram of the voxel time-series, i.e. the spike at the 18th frequency in Figure 6.

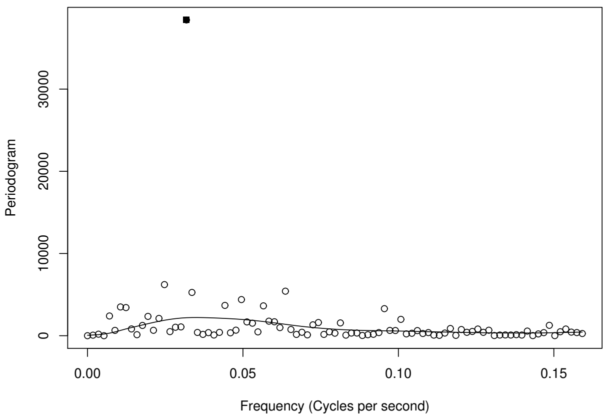

Figure 6: Periodogram and spectral density estimate.

The asymptotic distribution of the periodogram ordinate $I(\omega_c)$ is given by

$$I(\omega_c) \sim \frac{1}{2} f(\omega_c).\chi^2_2, \tag{3}$$

where $f_(\omega)$ represents the non-normalized spectral density of the (second order stationary) noise process. This suggests that a test for an unusually large component at frequency ω_c can be constructed through the use of the statistic R_c where

$$R_c = \frac{I(\omega_c)}{\widehat{f(\omega_c)}}, \tag{4}$$

where $\widehat{f(\omega_c)}$ is an estimate of the non-normalized spectral density. If we can obtain a sufficiently accurate estimate of $f(\omega_c)$ then R_c will have a standard exponential distribution.

Spectral density estimation

Lange and Zeger (1997) used a rectangular smoothing kernel in both space and frequency to estimate $f(\omega)$. This approach tends to produce biased estimates at boundaries between obviously different spectral densities.

We use a smoothing spline to estimate the spectral density from the log-periodogram of the voxel time-series (Wahba, 1980). One advantage of working on log-scale is that large high frequency artifacts are down weighted and thus avoids the biased estimates exhibited by other methods.

To improve the estimation we use a spatially anisotropic filter that only smooths spectral density estimates that are similar. Specifically we use a multivariate normal approximation to the difference between two independent[4] spectral density estimates. This allows us to decide when two estimates are close enough that they can be smoothed. We have also extended this approach to the case of event-related designs by working with the Fourier transform of the time-series as opposed to just the periodogram (Marchini, 2001).

Calibration

One nice property of this method is that we can calculate R_j at all the other frequencies at which we know there to be no response. Under the null model of no activation all these statistics should have the same distribution and can be used to self-calibrate the method.

Figure 7 illustrates the results of our calibration experiments. We applied the method to 6 null datasets[5] with (blue lines) and without (red lines) spatial smoothing. We compared the distribution of

[3] when the columns of the design matrix are close to being eigenvectors of the low-pass filter.
[4] the approximation works well even with the spatial correlation that exists between estimates
[5] datasets in which there is no activation because no stimulus was applied to the subject.

the statistic values at both the design frequency (dotted lines) and at a range of other frequencies (solid lines) to the theoretical exponential distribution. The statistic values are shown on $-\log_{10}(p\text{–value})$ scale to focus on the tails of the distribution. The line at 45^o represents exact agreement with the theoretical distribution. This plot shows how spatial smoothing provides a more accurate estimate of $f(\omega)$. This method of calibration also allows us to choose the smoothing spline parameter in an objective fashion.

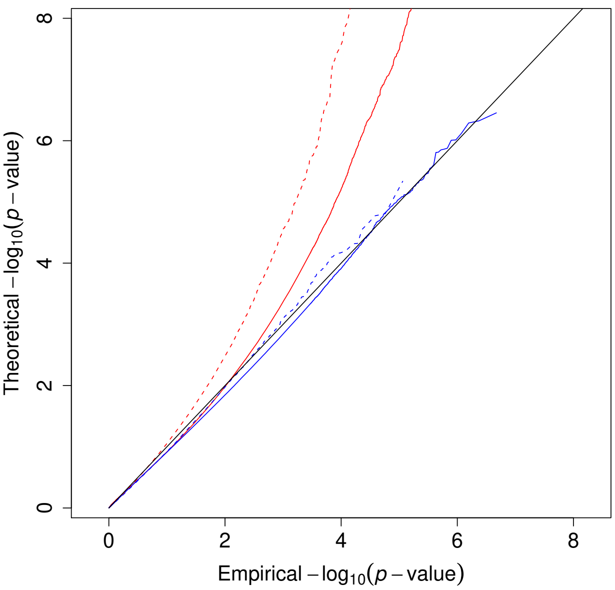

Figure 7: Calibration results

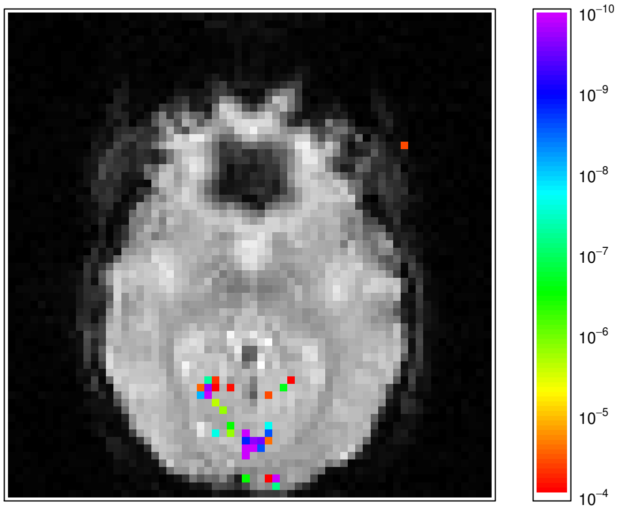

Figure 8: p-value image using spline smoothing spectral density estimation.

Figures 8 and 9 show statistic images produced by using a smoothing spline and an AR(1) model (Bullmore et al., 1996) for the spectral density at each voxel. The images are shown on p-value scale, thresholded to show only values below 10^{-4} and overlaid onto an image of the slice. The spline smoothing approach shows activation just in the visual cortex. The AR(1) approach identifies the same areas but with additional amounts of false positive activation indicating that this approach is poorly calibrated.

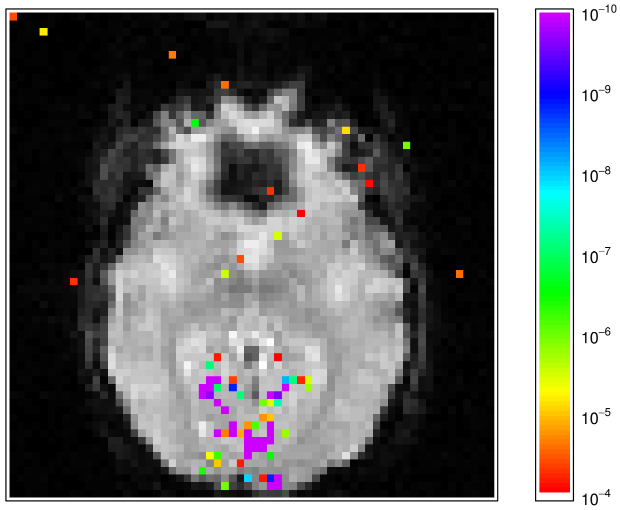

Figure 9: p-value image using AR(1) spectral density estimation.

We have implemented our approach using a mixture of R, C and FORTRAN code. We are currently working on incorporating this method into the **AnalyzeFMRI** package.

Identifying areas of activation statistic

Many different methods are available for the analysis of fMRI statistic images. The most widely used approaches are based on thresholding the statistic map (Worsley and Friston, 1995). The level of the threshold is chosen to protect against false positive activation under the null hypothesis of no activation anywhere in the image. Results from Random Field theory are used to set the appropriate threshold.

This approach tends to attract alot of criticism. In some experiments we know the effect will occur somewhere and we want to estimate the location of the response rather than use a hypothesis test to see if it's there. Also, to validate assumptions required by the Random Field theory results the datasets are significantly spatially smoothed. This tends to blurs the good resolution fMRI affords.

Despite these criticisms this approach has been very successful in answering the questions that neuroscientists want to ask. In addition, for group studies, the resolution is limited by structural variability between individuals. Thus using a small amount of smoothing doesn't matter and can in fact help detection of a response.

More attractive approaches are based on mixture models for levels of activation (Everitt and Bullmore, 1999; Hartvig and Jensen, 2000). It has been suggested that such an approach provides a less arbitrary way of delineating areas of activation although (so far) they have only been used with a 0-1 loss function. It will be interesting and important to investigate the effect of varying the loss function.

More recently it has been suggested that thresholds be defined by controlling the *False Discovery Rate* (FDR) (Genovese et al., 2001). This approach is more focussed on modelling the areas of activation as it is based on controlling the amount of false positive tests compared to all positive tests. In this way the method shares a property of mixture model approaches in that the threshold adapts to the properties of the statistic image.

We are currently in the process of implementing these approaches into the **AnalyzeFMRI** package.

Future plans

The long term goal of the **AnalyzeFMRI** package is to provide a comprehensive software package for the analysis of fMRI and MRI images. We aim to take full advantage of the existing (and future) graphics facilities of R by providing fast interfaces to widely used medical image formats and several graphical exploratory tools for fMRI and MRI datasets. In this way we hope to provide a valuable resource for statisticians working in this field. Hopefully this will negate the need for replication of coding effort and allow researchers new to the field to quickly familiarize themselves with many of the current methods of analysis.

Bibliography

E. Bullmore, M. Brammer, S. C. R. Williams, S. Rabe-Hesketh, N. Janot, A. David, J. Mellers, R. Howard, and P. Sham. Statistical methods of estimation and inference for functional MR image analysis. *Magnetic Resonance in Medicine*, 35:261–277, 1996. 21, 22

B. S. Everitt and E. D. Bullmore. Mixture model mapping of brain activation in functional magnetic resonance images. *Human Brain Mapping*, 7:1–14, 1999. 22

J. H. Friedman. A variable span scatterplot smoother. Technical Report 5, Laboratory for Computational Statistics, Stanford University, 1984. 20

K. Friston, C. Frith, R. Frackowiak, and R. Turner. Characterising evoked hemodynamics with fMRI. *NeuroImage*, 2:157–165, 1995. 21

C. R. Genovese, N. A. Lazar, and T. Nichols. Thresholding of statistical maps in functional neroimgaging using the false discovery rate. Technical report, Department of Statistics, Carnegie Mellon University, 2001. 23

N. V. Hartvig and J. L. Jensen. Spatial mixture modelling of fMRI data. *Human Brain Mapping*, 11:233–248, 2000. 22

A. Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10:626–634, 1999. 19

N. Lange and S. L. Zeger. Non-linear time series analysis for human brain mapping by functional magnetic resonance imaging. *Applied Statistics*, 46:1–29, 1997. 21

J. L. Locascio, P. L. Jennings, C. I. Moore, and S. Corkin. Time series analysis in the time domain and resampling methods for studies of functional magnetic resonance brain imaging. *Human Brain Mapping*, 5:168–193, 1997. 21

J. L. Marchini. *Statistical Issues in the Analysis of MRI Brain Images*. PhD thesis, Department of Statistics, Oxford University, UK, 2001. 21

J. L. Marchini and B. D. Ripley. A new statistical approach to detecting activation in functional MRI. *NeuroImage*, 12:366–380, 2000. 21

P. M. Matthews, P. Jezzard, and S. M. Smith, editors. *Functional Magnetic Resonance Imaging of the Brain: Methods for Neuroscience*. Oxford University Press, 2001. 17

W. N. Venables. Programmers's niche. *R News*, 1(1): 27–30, 2001. 18

W. N. Venables and B. D. Ripley. *S Programming*. Springer, New York, 2000. 18

G. Wahba. Automatic smoothing of the log periodogram. *Journal of the American Statistical Association*, 75:122–132, 1980. 21

K. Worsley and K. Friston. Analysis of fMRI time series revisited – again. *NeuroImage*, 2:173–181, 1995. 21, 22

Jonathan L. Marchini
University of Oxford, UK
marchini@stats.ox.ac.uk

Using R for the Analysis of DNA Microarray Data

by Sandrine Dudoit, Yee Hwa Yang, and Ben Bolstad

Overview

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, portable, and distributed statistical computing environment is a related and equally critical aspect of the analysis of gene expression data. As part of the Bioconductor project, we are working on the development of R packages implementing statistical methods for the design and analysis of DNA microarray experiments. An early effort is found in the **sma** (Statistics for Microarray Analysis) package which we wrote in the Fall of 2000. This small package was initially built to allow the sharing of software with our collaborators, both statisticians and biologists, for pre-processing tasks such as normalization and diagnostic plots. We are in the process of expanding the scope of this package by implementing new methods for microarray experimental design, normalization, estimation, multiple testing, cluster analysis, and discriminant analysis.

This short article begins with a brief introduction to the biology and technology of DNA microarray experiments. Next, we illustrate some of the main steps in the analysis of microarray gene expression data, using the apo AI experiment as a case study. Additional details can be found in Dudoit et al. (2002b). Although we focus primarily on two-color cDNA microarrays, a number of the proposed methods and implementations extend to other platforms as well (e.g., Affymetrix oligonucleotide chips, nylon membrane arrays). Finally, we discuss ongoing revisions and extensions to **sma** as part of the Bioconductor project. This project aims more generally to produce an open source and open development computing environment for biologists and statisticians working on the analysis of genomic data. More information on Bioconductor can be found at `http://www.bioconductor.org`.

Background on DNA microarrays

The ever-increasing rate at which genomes are being sequenced has opened a new area of genome research, *functional genomics*, which is concerned with assigning biological function to DNA sequences. With the availability of the DNA sequences of many genomes (e.g., the yeast *Saccharomyces cerevisae*, the round worm *Caenorhabditis elegans*, the fruit fly *Drosophila melanogaster*, and numerous bacteria) and the recent release of the first draft of the human genome, an essential and formidable task is to define the role of each gene and elucidate interactions between sets of genes. Innovative approaches, such as the cDNA and oligonucleotide microarray technologies, have been developed to exploit DNA sequence data and yield information about gene expression levels for entire genomes.

Different aspects of gene expression can be studied using microarrays, such as expression at the transcription or translation level, subcellular localization of gene products, and protein binding sites on DNA. To date, attention has focused primarily on expression at the transcription stage, i.e., on mRNA or transcript levels. There are several types of microarray systems, including the two-color cDNA microarrays developed in the Brown and Botstein labs at Stanford and the high-density oligonucleotide chips from the Affymetrix company; the brief description below focuses on the former.

cDNA 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) for a more detailed introduction to the biology and technology of cDNA microarrays.

DNA microarray experiments raise numerous statistical questions, in fields as diverse as experimental design, image analysis, multiple testing, cluster analysis, and discriminant analysis. The main steps in the statistical design and analysis of a microarray experiment are summarized in Figure 1. Each step in this process depends critically on the availability of an efficient, portable, and distributed statistical computing environment.

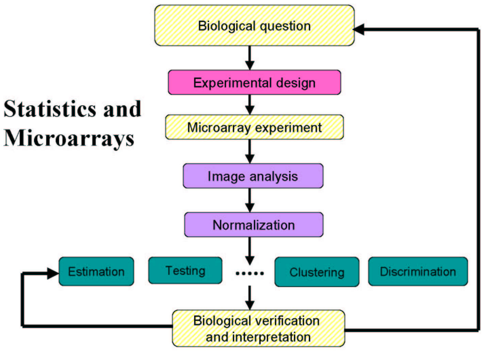

Figure 1: Main steps in the statistical design and analysis of a DNA microarray experiment (solid boxes).

Sample microarray dataset

We will illustrate some of the main steps in the analysis of a microarray experiment using gene expression data from the apo AI experiment described in detail in Callow et al. (2000). Apolipoprotein AI (apo AI) is a gene known to play a pivotal role in HDL metabolism, and mice with the apo AI gene knocked-out have very low HDL cholesterol levels. The goal of this experiment was to identify genes with altered expression in the livers of apo AI knock-out mice compared to inbred control mice. The treatment group consisted of eight mice with the apo AI gene knocked-out and the control group consisted of eight control C57Bl/6 mice. For each of these sixteen mice, target cDNA was obtained from mRNA by reverse transcription and labeled using a red-fluorescent dye, Cy5. The reference sample used in all hybridizations was prepared by pooling cDNA from the eight control mice and was labeled with a green-fluorescent dye, Cy3. Target cDNA was hybridized to microarrays containing 6,384 DNA probes, including 257 related to lipid metabolism. Microarrays were printed using 4×4 print-tips and are thus partitioned into a 4×4 grid matrix (the terms grid, sector, and print-tip group are used interchangeably in the microarray literature). Each grid consists of a 19×21 spot matrix that was printed with a single print-tip.

Raw images and background corrected Cy3 and Cy5 fluorescence intensities for all sixteen hybridizations are available at `http://www.stat.berkeley.edu/users/terry/zarray/Html/apodata.html`. Fluorescence intensity data from processed images for six of the sixteen hybridizations (three knock-out mice and three control mice) were also included in the **sma** package and can be accessed with the command `data(MouseArray)`. Figure 2 displays an RGB overlay of the Cy3 and Cy5 images for one of the sixteen arrays (files 1230ko1G.tif.marray and 1230ko1R.tif.marray corresponding to knock-out mouse 1, i.e., `mouse4` array in the dataset `MouseArray`). Analyses described below were performed using **sma** version 0.5.7 and R version 1.3.1.

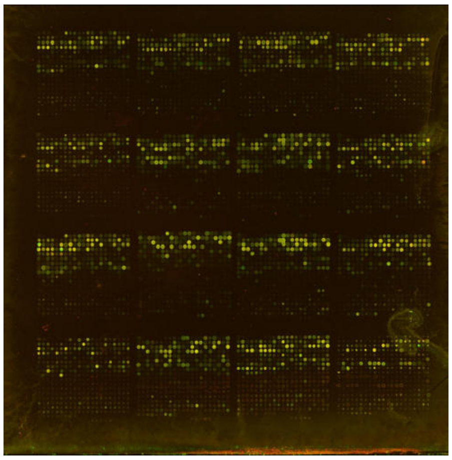

Figure 2: RGB overlay of the Cy3 and Cy5 images for knock-out mouse 1, i.e., `mouse4` array in the dataset `MouseArray`.

Pre-processing

Image analysis

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 (the images for the apo AI experiment are about 2MB in size; more recent images produced from higher resolution scans are around 10 to 20 MB). Image analysis is required to extract foreground and background fluorescence intensity measurements for each spotted DNA sequence. Widely used commercial image processing software packages include **GenePix** for the Axon scanner and **ImaGene** from BioDiscovery; freely available software includes **ScanAlyze**. Image processing is beyond the scope of this article, and the reader is referred to Yang et al. (2002a) for a detailed discussion of microarray image analysis and additional references.

For the apo AI experiment, each of the sixteen hybridizations produced a pair of 16-bit images. These images were processed using the software package **Spot** (Buckley, 2000). **Spot** is built on R and is a specialized version of another R package called **VOIR**, which is being developed by the CSIRO Image Analysis Group (`http://www.cmis.csiro.au/iap/spot.htm`). The segmentation component, based on a seeded region growing algorithm, places no restriction on the size or shape of the spots. The background adjustment method relies on morphological opening to generate an image of the estimated background intensity for the entire slide. The results of an analysis by **Spot** of microarray images are returned as a data frame and can thus immediately be displayed, manipulated, and analyzed in a number of ways within R. The rows of this data frame correspond to the spots on the array and the columns to different spot statistics: e.g., grid row and column coordinates, spot row and column coordinates, red and green foreground intensities, red and green background intensities for different background adjustment methods, spot area, perimeter. The six data frames `mouse1`, ..., `mouse6`, in `MouseArray` contain **Spot** output for three control mice and three knockout mice, respectively. In what follows, we use R and G to denote the background corrected red and green fluorescence intensities of a particular spot, and M to denote the corresponding base-2 log-ratio, $\log_2 R/G$.

Reading in data

In general, microarray image processing results are stored in ASCII files and can be loaded into R using specialized functions like `read.spot` and `read.genepix` for **Spot** and **GenePix** output, respectively. These functions are simply *wrapper* functions around `read.table`, which take into account the specific file format for output from different image processing software packages.

We have written a number of initialization functions to manipulate the image analysis output for batches of arrays, i.e., collections of slides with the same spot layout. The function `init.grid` is an interactive function for specifying the dimensions of the spot and grid matrices. These parameters are determined by the printing layout of the array, and are used for the spatial representation of spot statistics in the function `plot.spatial` and the within-print-tip-group normalization procedure implemented in `stat.ma`. The function `init.data` is an interactive function which creates a data structure for multi-slide microarray experiments. The data structure is a list of four matrices, storing raw red and green foreground intensities, and red and green background intensities. The rows of the matrices correspond to spotted DNA sequences and the columns to hybridizations (i.e., arrays or slides). The function also allows the user to add hybridization data to an existing list. The following commands may be used to extract red and green fluorescence intensities from the image analysis output for the six arrays in `MouseArray`, and compute intensity log-ratios and average log-intensities.

```
data(MouseArray)
mouse.setup <- init.grid()
mouse.data <- init.data()
mouse.MA0 <- stat.ma(mouse.data, mouse.setup,
                     norm="n")
```

Eventually, we would like to retrieve the image analysis results directly from a laboratory database. In addition, to allow more general and systematic representation and manipulation of microarray data, we are working on the definition of new classes of R objects for microarrays using the class/method mechanism in the **methods** package (cf. section on ongoing projects).

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.

Spatial plots of spot statistics. The **sma** function `plot.spatial` creates an image of shades of gray or colors that represents the values of a statistic for each spot on the array. This function can be used to explore whether there are any spatial effects in the data, for example, print-tip or cover-slip effects. The commands

```
M <- stat.ma(mouse.data, mouse.setup,
             norm="n")$M[,4]
plot.spatial(M, mouse.setup, crit1=1)
```

produce an image of the pre-normalization log-ratios M for the array `mouse4`. To display only the spots having the highest and lowest 5% pre-normalization log-ratios M, set the argument `crit1` to 0.05. Figure 3 clearly indicates that the lowest row of grids has high red intensities compared to green, and thus suggests the existence of spatially-dependent dye biases.

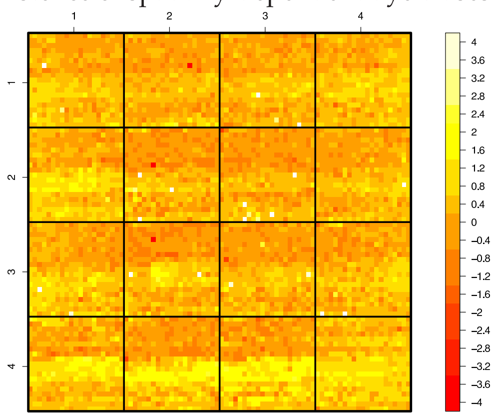

Figure 3: Image of the pre-normalization intensity log-ratios for the `mouse4` array using the `heat.colors` palette (`plot.spatial` output).

Boxplots. Boxplots of spot statistics by plate, sector, or slide can also be useful to identify spot or hybridization artifacts. The boxplots in Figure 4 again suggest the existence of spatially-dependent dye biases.

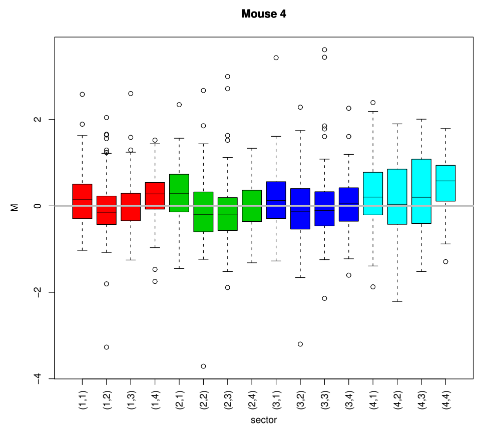

Figure 4: Boxplots by sector of the pre-normalization intensity log-ratios for the `mouse4` array. The pairs (i, j), $i, \ j = 1, \ldots, 4$, refer to sector or grid coordinates in the 4×4 grid matrix shown in Figure 2.

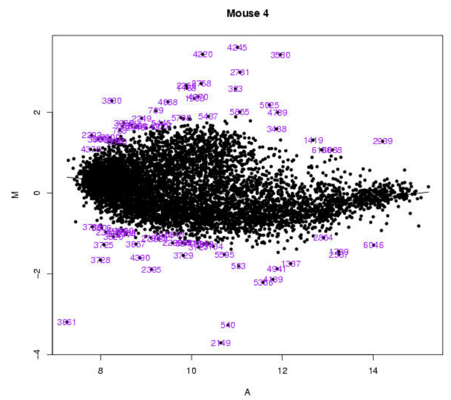

Figure 5: Pre-normalization MA-plot for the `mouse4` array, highlighting spots with the highest and lowest 0.5% log-ratios (`plot.mva` output).

***MA*-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 $M = \log_2 R/G$ vs. the mean log intensity $A = \log_2 \sqrt{RG}$. 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., 2002b; Yang et al., 2001, 2002b). For the apo AI experiment,

```
plot.mva(mouse.data,mouse.setup,norm="l",
   image.id=4,extra.type="tci",plot.type="r",
   crit1=0.005,col.ex="purple",pch=20)
title("Mouse 4")
```

produces an MA-plot for the `mouse4` array, highlighting spots with the highest and lowest 0.5% log-ratios using the corresponding row number in `mouse.data` (Figure 5).

Normalization

The purpose of normalization is to identify and remove sources 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 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., clustering, multiple testing). 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., 2002b). 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, location on the array, plate origin, and possibly other variables. Figure 6 displays the pre-normalization MA-plot for the `mouse4` 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 plot was produced using the function `plot.print.tip.lowess`

```
col <- sort(rep(2:5,4))
lty <- rep(1:4,4)
labs <- paste("(",sort(rep(1:4,4)),",",
   rep(1:4,4),")",sep="")
plot.print.tip.lowess(mouse.data,mouse.setup,
   norm="n",image.id=4,pch=20,lwd=3,
   palette=col,lty.palette=lty,cex=0.6)
legend(10.5,-2.5,legend=labs[order(lty,col)],
   col=col[order(lty,col)],
   lty=lty[order(lty,col)], ncol=4,lwd=2)
title("Mouse 4")
```

Figure 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, four lowess curves clearly stand out from the remaining twelve curves, suggesting strong print-tip or spatial effects. The four curves correspond to the last row of grids in the 4×4 grid matrix.

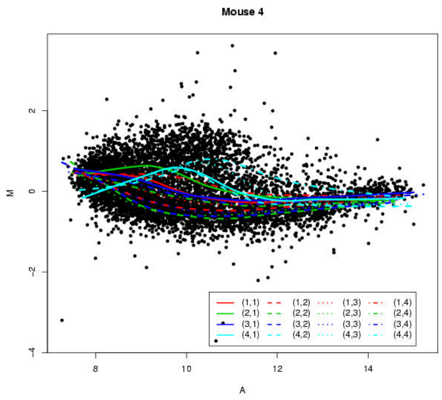

Figure 6: Pre-normalization MA-plot for the `mouse4` array, with the lowess fits for individual print-tips or sectors. Different colors are used to represent lowess curves for print-tips from different rows, and different line types are used to represent lowess curves for print-tips from different columns (`plot.print.tip.lowess` output).

We have developed location and scale normalization methods which correct for intensity and spatial dye biases using *robust local regression* (Yang et al., 2001, 2002b). These procedures are implemented in the **sma** function `stat.ma`, which uses the R function `lowess` to perform robust local regression of the log-ratio M on spot intensity A. Five methods are currently available for normalizing the log-ratios; the method is specified using the `norm` argument of the function `stat.ma`. These five options are: `"n"` for no normalization between the two channels; `"m"` for global median normalization, which sets the median of the intensity log-ratios M to zero; `"l"` for global lowess normalization, i.e., regression of M on A for all spots; `"p"` for within-print-tip-group lowess normalization; and `"s"` for scaled within-print-tip-group normalization. For the `"s"` option, scaling is done by the median absolute deviation (MAD) of the lowess location normalized log-ratios for each print-tip-group. The code

```
mouse.lratio <-
  stat.ma(mouse.data, mouse.setup, norm="p")
```

performs within-print-tip-group lowess location normalization for the six slides in `MouseArray`, and returns normalized log-ratios M and average log-intensities A. For the apo AI experiment, only a small proportion of the spots were expected to vary in intensity between the two channels; normalization was thus performed using all $6,384$ probes. The function returns a list with two components: `M`, a matrix of normalized log-ratios, and `A`, a matrix of average log-intensities. The rows of these matrices correspond to spotted DNA sequences and the columns to the different slides.

Image analysis and normalization are the two main pre-processing steps required in any microarray experiment. For our purpose, 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 these basic statistics, we are planning on deriving spot and slide quality statistics and incorporating these in further analyses (cf. section on ongoing projects).

Main statistical analysis

The phrase "main statistical analysis" refers to the application of statistical methodology to normalized intensity data in order to answer the biological question for which the microarray experiment was designed. Appropriate statistical methods depend largely on the question of interest to the investigator and in general can span the entire field of Statistics. For model organisms such as mice or yeast, biological questions of interest might include the identification of differentially expressed genes in factorial experiments studying the simultaneous gene expression response to factors such as drug treatment, cell type, spatial location of tissues, and time. In human cancer microarray studies, one may be interested in the discovery of new tumor subclasses, or in the identification of genes that are good predictors of clinical outcomes such as survival or response to treatment. There is clearly no general "next step" in the analysis of microarray data, and the implementation of statistical methodology will require the development of question specific R packages (cf. section on ongoing projects). Next, we discuss differential expression, an important and common question in microarray experiments.

Differential expression

Differentially expressed genes are genes whose expression levels are associated with a response or covariate of interest. In general, the covariates could be either polytomous (e.g., treatment/control status, cell type, drug type) or continuous (e.g., dose of a drug, time). Similarly, the responses could be either polytomous or continuous, for example, tumor type, response to treatment, or censored survival time in clinical applications of microarrays. An approach to the identification of differentially expressed genes is to compute, for each gene, a statistic assessing the strength of the association between the expression levels and responses or covariates. In such one gene at a time approaches, genes are ranked based on these statistics and a subset is typically selected for further biological verification. This gene subset may be chosen based on either the number of genes that can be conveniently followed-up (and subject matter knowledge), or statistical significance as described in the next section. The package **sma** contains a number of functions for computing and plotting frequentist and Bayesian statistics for each gene in a microarray experiment.

Single-slide methods. Single-slide experiments aim to compare transcript abundance in two mRNA samples, the red and green labeled mRNA samples hybridized to the same slide. The **sma** functions `stat.Chen`, `stat.Newton`, and `stat.ChurSap` implement, respectively, the single-slide methods of Chen et al. (1997), Newton et al. (2001), and Sapir and Churchill (2000). These methods are based on assumed parametric models for the (R, G) intensity pairs (e.g., Gamma or Gaussian models). The resulting model-dependent rules amount to drawing two curves in the (R, G)-plane and calling a gene differentially expressed if its (R, G) measured intensities fall outside the region between the two curves. The above three functions can be applied individually, with options for producing *MA*-plots showing the model cut-offs. Alternatively, the three methods can be compared graphically as in Figure 7, by calling `plot.single.slide(mouse.data, mouse.setup, norm="p", image.id=4)`. This produces an *MA*-plot, print-tip-group lowess location normalized, for the `mouse4` array with cut-offs from each of the three single-slide methods. Black curves correspond to 1:1, 1:10, 1:100 log-odds ratios for the Newton et al. method; dashed lines are the 95% and 99% confidence limits from the Chen et al. method; dotted lines are the 95% and 99% posterior probability cut-offs from the Sapir and Churchill method.

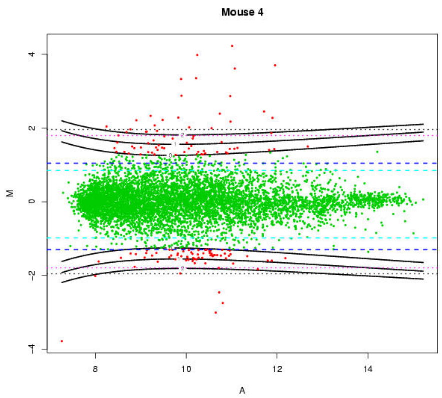

Figure 7: *MA*-plot with cut-offs for three single-slide methods applied to the `mouse4` array (`plot.single.slide` output).

Multiple-slide methods. Multiple-slide experiments typically involve the comparison of transcript abundance in two or more types of mRNA samples hybridized to different slides. **sma** functions for multiple-slide analyses include: `stat.bayesian`, which computes odds of differential expression under a Bayesian model for gene expression; `stat.t2`, which computes two-sample t-statistics for comparing the expression response in two groups of mRNA samples. The multiple testing package **multtest** contains additional functions for computing test statistics (paired t-statistics, F-statistics, etc.) and associated permutation adjusted p-values for each gene on the array (see next section). For the apo AI data, the following commands implement two possible approaches for comparing expression levels in the livers of knock-out and control mice.

```
## Two-sample Welch t-statistics
## Comparison of the expression levels of the
## three control and three knock-out mice
## (data from mouse1, ..., mouse6)
cl <- c(rep(1,3), rep(2,3))
mouse.t2 <- stat.t2(mouse.lratio, cl)
plot.t2(mouse.t2, "Apo AI experiment, six mice")

## Bayesian odds ratio
## Comparison of the expression levels of the
## three knock-out mice to the polled controls
## (data from mouse4, mouse5, mouse6)
mouse.bayesian <-
  stat.bayesian(M=mouse.lratio$M[,4:6])
plot(mouse.bayesian$Xprep$Mbar,
     mouse.bayesian$lods)
```

Multiple testing

The biological question of differential expression can be restated as a problem in *multiple hypothesis testing*: the simultaneous test for each gene of the null hypothesis of no association between the expression levels and the responses or covariates. As a typical microarray experiment measures expression levels for thousands of genes simultaneously, we are faced with an extreme form of multiple testing when assessing the statistical significance of the results.

The **multtest** package implements a number of resampling-based multiple testing procedures. It includes procedures for controlling the family-wise Type I error rate (FWER): Bonferroni, Hochberg, Holm, Šidák, Westfall and Young minP and maxT procedures. The Westfall and Young procedures take into account the joint distribution of the test statistics and are thus in general less conservative than the other four procedures. The **multtest** package also includes procedures for controlling the false discovery rate (FDR): Benjamini and Hochberg and Benjamini and Yekutieli step-up procedures (see Dudoit et al. (2002a) for a review of multiple testing procedures and complete references). These procedures are implemented for tests based on t-statistics, F-statistics, paired t-statistics, block F-statistics, Wilcoxon statistics. The results of the procedures are summarized using *adjusted p-values*, which reflect for each gene the overall experiment Type I error rate when genes with a smaller p-value are declared differentially expressed. The p-values may be obtained either from the nominal distribution of the test statistics or by permutation.

For the apo AI experiment, differentially expressed genes between the eight knock-out and eight control mice were identified by computing two-sample Welch's t-statistics for each gene. In order to assess the statistical significance of the results, four multiple testing procedures (Holm, Westfall and Young maxT, Benjamini and Hochberg FDR, Benjamini and Yekutieli FDR) were considered, and unadjusted and adjusted p-values were estimated based on all possible $\binom{16}{8} = 12,870$ permutations of the knock-out/control labels. Figure 8 displays plots of the ordered adjusted p-values for these four multiple testing procedures. The functions `mt.maxT`, `mt.rawp2adjp`, and `mt.plot` from the **multtest** package were used to compute and display adjusted p-values.

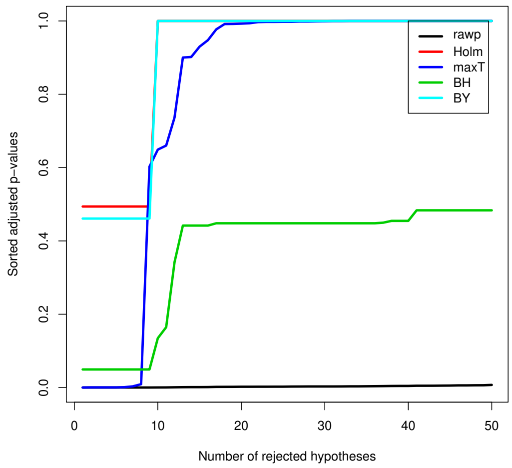

Figure 8: Plot of sorted adjusted p-values for the apo AI experiment (**multtest** package, `mt.plot` output).

Biological verification

In the apo AI experiment, eight spotted DNA sequences clearly stood out from the remaining sequences and had maxT adjusted p-values less than 0.01. These eight sequences correspond to only four distinct genes: apo AI (3 copies), apo CIII (2 copies), sterol C5 desaturase (2 copies), and a novel EST (1 copy). All changes were confirmed by real-time quantitative PCR (RT-PCR) as described in Callow et al. (2000). The presence of apo AI among the differentially expressed genes is to be expected, as this is the gene that was knocked out in the treatment mice. The apo CIII gene, also associated with lipoprotein metabolism, is located very close to the apo AI locus; Callow et al. (2000) showed that the down-regulation of apo CIII was actually due to genetic polymorphism rather than lack of apo AI. The presence of apo AI and apo CIII among the differentially expressed genes thus provides a check of the statistical method, if not a biologically interesting finding. Sterol C5 desaturase is an enzyme which catalyzes one of the terminal steps in cholesterol synthesis and the novel EST shares sequence similarity to a family of ATPases. Note that the application of single-slide methods to array data from individual knock-out mice generally produced a large number of false positives, while at the same time missing some of the confirmed genes.

Ongoing projects

The **sma** package represents a preliminary and limited effort toward the goal of providing a statistical computing environment for the analysis of microarray data. Modifications and extensions to **sma** will be done within the Bioconductor project (`http://www.bioconductor.org`). The Bioconductor project aims more generally to produce an open source and open development computing environment for biologists and statisticians working on the analysis of genomic data. Important changes include the use of the class/method mechanism in the R **methods** package to allow more general and systematic representation and manipulation of microarray data (Bioconductor **Biobase** package). In addition, we are in the process of splitting and expanding the **sma** package into the following task specific packages.

Experimental design (MarrayDesign)

This package will include functions for: setting up a design matrix for a given experimental design (this design matrix can be used to efficiently combine data across slides with functions similar to `lm` or `glm`); searching for optimal designs; and creating graphical representations of the design choices.

Spot quality (MarrayQual)

Building upon our previous work on image analysis and normalization, we plan to devise spot and slide quality statistics and incorporate these statistics in further analyses of the fluorescence intensity data, by weighting the contribution of each spot based on its quality.

Normalization (MarrayNorm)

This package contains functions for diagnostic plots and normalization procedures based on robust local regression. Normalization procedures were extended to accommodate a broader class of dye biases and the use of control sequences spotted on the array and possibly spiked in the target cDNA samples.

Cluster analysis (MarrayClust)

R already has an extensive collection of clustering procedures in packages such as **cluster**, **mva**, and **GeneSOM**. Building upon these existing packages, we are implementing new methods which are useful for the clustering of microarray data, genes or mRNA samples. These include resampling based-methods for estimating the number of clusters, for improving the accuracy of a clustering procedure, and for assessing the accuracy of cluster assignments for individual observations (Fridlyand and Dudoit, 2001).

Class prediction (MarrayPred)

This package builds upon existing R packages such as **class** and **rpart** and contains functions for bagging and boosting classifiers. The functions also return estimates of misclassification rates and measures assessing the confidence of predictions for individual observations. Resampling-based procedures for variable selection as described in Breiman (1999) will also be implemented in the package.

Affymetrix chips (affy)

Low-level analysis of Affymetrix data, such as normalization and calculation of expression estimates, is being handled by the **affy** package `http://biosun01.biostat.jhsph.edu/~ririzarr/Raffy/index.html`.

The packages **MarrayNorm**, **affy**, and **MarrayPred** are closest to being released. Required developments to increase the effectiveness of these R packages within a biological context include the following.

Links to biological databases: An important aspect of the analysis of microarray experiments is the seamless access 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`). The Bioconductor **annotate** package developed by Robert Gentleman already provides browser access to LocusLink and GenBank.

GUI: Not all biologist users will feel comfortable with the command line R environment. To accommodate a broad class of users and allow easy access to the statistical methodology, the computing environment should provide a graphical user interface.

Documentation: We envision two main classes of users for these packages: biologists performing microarray experiments, and researchers involved in the development of statistical methodology for microarray experiments. The former, will be primarily users of a standard set of functions from these packages. They will need guidance in deciding which statistical approaches and packages may be appropriate for their experiments, in choosing among the various options provided by the functions, and in correctly interpreting the results of their visualizations and statistical analyses. Extended tutorials, with step-by-step analyses of microarray experiments, will be provided. The **Sweave** package to be included in R 1.5 will be used for automatic report generation mixing text and R code. Researchers in the second group, will likely be interested in writing their own functions and packages, in addition to using existing functions. The Bioconductor project will provide a framework for integrating their efforts.

Thanks to Natalie Thorne, Ingrid Lönnstedt, and Jessica Mar for their contributions to the **sma** package.

Bibliography

L. Breiman. Random forests - random features. Technical Report 567, Department of Statistics, U.C. Berkeley, 1999. 31

P. O. Brown and D. Botstein. Exploring the new world of the genome with DNA microarrays. In *The Chipping Forecast*, volume 21, pages 33–37. Supplement to Nature Genetics, January 1999. 24

M. J. Buckley. *The Spot user's guide*. CSIRO Mathematical and Information Sciences, August 2000. `http://www.cmis.csiro.au/IAP/spot.htm`. 25

M. J. Callow, S. Dudoit, E. L. Gong, T. P. Speed, and E. M. Rubin. Microarray expression profiling identifies genes with altered expression in HDL deficient mice. *Genome Research*, 10(12):2022–2029, 2000. 25, 30

Y. Chen, E. R. Dougherty, and M. L. Bittner. Ratio-based decisions and the quantitative analysis of cDNA microarray images. *Journal of Biomedical Optics*, 2:364–374, 1997. 29

S. Dudoit, J. P. Shaffer, and J. C. Boldrick. Multiple hypothesis testing in microarray experiments. Submitted, 2002a. 29

S. Dudoit, Y. H. Yang, M. J. Callow, and T. P. Speed. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica Sinica*, 12(1), 2002b. 24, 27

J. Fridlyand and S. Dudoit. Applications of resampling methods to estimate the number of clusters and to improve the accuracy of a clustering method. Technical Report 600, Department of Statistics, U.C. Berkeley, September 2001. 31

M. A. Newton, C. M. Kendziorski, C. S. Richmond, F. R . Blattner, and K. W. Tsui. On differential variability of expression ratios: Improving statistical inference about gene expression changes from microarray data. *Journal of Computational Biology*, 8: 37–52, 2001. 29

M. Sapir and G. A. Churchill. *Estimating the posterior probability of differential gene expression from microarray data*. Poster, The Jackson Laboratory, 2000. `http://www.jax.org/research/churchill/`. 29

Y. H. Yang, M. J. Buckley, S. Dudoit, and T. P. Speed. Comparison of methods for image analysis on cDNA microarray data. *Journal of Computational and Graphical Statistic*, 11(1), 2002a. 25

Y. H. Yang, S. Dudoit, P. Luu, D. M. Lin, V. Peng, J. Ngai, and T. P. Speed. Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research*, 30(4), 2002b. 27, 28

Y. H. Yang, S. Dudoit, P. Luu, and T. P. Speed. Normalization for cDNA microarray data. In Michael L. Bittner, Yidong Chen, Andreas N. Dorsel, and Edward R. Dougherty, editors, *Microarrays: Optical Technologies and Informatics*, volume 4266 of *Proceedings of SPIE*, May 2001. 27, 28

Sandrine Dudoit
University of California, Berkeley
http://www.stat.berkeley.edu/~sandrine
sandrine@stat.berkeley.edu

Yee Hwa (Jean) Yang
University of California, Berkeley
yeehwa@stat.berkeley.edu

Ben Bolstad
University of California, Berkeley
bolstad@stat.berkeley.edu

Changes in R 1.4.0

by the R Core Team

The ‘NEWS’ file of the R sources and hence this article have been reorganized into several sections (user-visible changes, new features, deprecated & defunct, documentation changes, utilities, C-level facilities, bug fixes) in order to provide a better overview of all the changes in R 1.4.0.

User-visible changes

This is a new section to highlight changes in behaviour, which may be given in more detail in the following sections. Many bug fixes are also user-visible changes.

- The default save format has been changed, so saved workspaces and objects cannot (by default) be read in earlier versions of R.
- The number of bins selected by default in a histogram uses the correct version of Sturges’ formula and will usually be one larger.
- `data.frame()` no longer converts logical arguments to factors (following S4 rather than S3).
- `read.table()` has new arguments ‘nrows’ and ‘colClasses’. If the latter is `NA` (the default), conversion is attempted to logical, integer, numeric or complex, not just to numeric.
- `model.matrix()` treats logical variables as a factors with levels `c(FALSE, TRUE)` (rather than 0-1 valued numerical variables). This makes R compatible with all S versions.
- Transparency is now supported on most graphics devices. This means that using `par("bg")`, for example in `legend()`, will by default give a transparent rather than opaque background.
- [dpqr]gamma now has third argument ‘rate’ for S-compatibility (and for compatibility with exponentials). Calls which use positional matching may need to be altered.
- The meaning of `spar = 0` in `smooth.spline()` has changed.
- `substring()` and `substring<-()` do nothing silently on a character vector of length 0, rather than generating an error. This is consistent with other functions and with S.
- For compatibility with S4, any arithmetic operation using a zero-length vector has a zero-length result. (This was already true for logical operations, which were compatible with S4 rather than S3.)
- `undoc()` and `codoc()` have been moved to the new package **tools**.
- The name of the site profile now defaults to ‘R_HOME/etc/Rprofile.site’.
- The startup process for setting environment variables now first searches for a site environment file (given by the environment variable `R_ENVIRON` if set or ‘R_HOME/etc/Renviron.site’

if not), *and* then for a user '.Renviron' file in the current or the user's home directory.

- Former `stars(*, colors = v)` must now be `stars(*, col.segments = v)`.
- The default methods for `La.svd` and `La.eigen` have changed and so there may be sign changes in singular/eigen vectors, including in `cancor`, `cmdscale`, `factanal`, `princomp`, and `varimax`.

New features

- Transparency is now supported on most graphics devices. Internally colors include an alpha channel for opacity, but at present there is only visible support for transparent/opaque. The new color '"transparent"' (or 'NA' or '"NA"') is transparent, and is the default background color for most devices. Those devices (postscript, XFig, PDF, Windows metafile and printer) that previously treated 'bg = "white"' as transparent now have '"transparent"' as the default and will actually print '"white"'. (NB: you may have 'bg = "white"' saved in `.Postscript.options` in your workspace.)
- A package **methods** has been added, containing formal classes and methods ("S4" methods), implementing the description in the book "Programming with Data". See `?Methods` and the references there for more information.
  - In support of this, the '`@`' operator has been added to the grammar.
  - Method dispatch for formal methods (the `standardGeneric` function), is now a primitive. Aside from efficiency issues, this allows S3-style generics to also have formal methods (not really recommended in the long run, but it should at least work). The C-level dispatch is now implemented for primitives that use either `DispatchGroup` or `DispatchOrEval` internally.
  - A version of the function `plot` in the methods package has arguments `x` and `y`, to allow methods for either or both. See `?setMethod` for examples of such methods.
  - The **methods** package now uses C-level code (from within `DispatchOrEval`) to dispatch any methods defined for primitive functions. As with S3-style methods, methods can only be defined if the first argument satisfies `is.object(x)` (not strictly required for formal methods, but imposed for now for simplicity and efficiency).
- Changes to the **tcltk** package:
  - New interface for accessing Tcl variables, effectively making the R representations lexically scoped. The old form is being deprecated.
  - Callbacks can now be expressions, with slightly unorthodox semantics. In particular this allows bindings to contain 'break' expressions (this is necessary to bind code to e.g. Alt-x without having the key combination also insert an 'x' in a text widget.)
  - A bunch of file handling and dialog functions (previously only available via `tkcmd`) have been added.
- The '?' operator is now an actual function. It can be used (as always) as a unary operator ('?plot') and the grammar now allows it as a binary operator, planned to allow differentiating documentation on the same name but different type ('class?matrix,' for example). So far, no such documentation exists.
- New methods `AIC.default()` and `logLik.glm()`, also fixing the `AIC` method for glm objects.
- `axis.POSIXct()` allows the label date/times to be specified via the new 'at' argument.
- `arrows()` now allows 'length = 0' (and draws no arrowheads).
- Modifications to the access functions for more consistency with S: arguments 'name', 'pos' and 'where' are more flexible in `assign()`, `exists()`, `get()`, `ls()`, `objects()`, `remove()` and `rm()`.
- Three new primitive functions have been added to base: `dataClass()`, `objWithClass()`, and `as.environment()`. The first two are support routines for `class()` and `class<-()` in package **methods**. The third replaces `pos.to.env()` in the functions `get()`, `exists()`, and friends.
- `barplot()` now respects an inline 'cex.axis' argument and has a separate 'cex.names' argument so names and the numeric axis labels can be scaled separately. Also, graphics parameters intended for `axis()` such as 'las' can now be used.
- Shading by lines added to functions `barplot()`, `hist()`, `legend()`, `piechart()`, `polygon()` and `rect()`.

- `bxp()` has a show.names argument allowing labels on a single boxplot; it and hence `boxplot()` now makes use of 'pch', 'cex', and 'bg' for outlier `points()`.

  `bxp()` and `boxplot()` also have an argument 'outline' to suppress outlier drawing S-Plus compatibly.

- New `capabilities()` options '"GNOME"' and '"IEEE754"'.

- New function `casefold()`, a wrapper for `tolower/toupper` provided for compatibility with S-Plus.

- `contour()` is now generic.

- `cor.test()` in package **ctest** now also gives an asymptotic confidence interval for the Pearson product moment correlation coefficient.

- `data()`, `demo()` and `library()` now also return the information about available data sets, demos or packages. Similarly, `help.search()` returns its results.

- `density()` allows 'bw' or 'width' to specify a rule to choose the bandwidth, and rules '"nrd0"' (the previous default), '"nrd"', '"ucv"', '"bcv"', '"SJ-ste"' and '"SJ-dpi"' are supplied (based on functions in package **MASS**).

- `df.residual()` now has a default method, used for classes '"lm"' and '"glm"'.

- New argument 'cacheOK' to `download.file()` to request cache flushing.

  All methods for `download.file()` do tilde-expansion on the path name.

  The internal `download.file()` etc now allow URLs of the form 'ftp://user@foo.bar/' and 'ftp://user:pass@foo.bar/'

- `duplicated()` and `unique()` are now generic functions with methods for data frames (as well as atomic vectors).

- `factanal()` and `princomp()` use `napredict()` on their scores, so 'na.action = na.exclude' is supported.

- Function `getNativeSymbolInfo()` returns details about a native routine, potentially including its address, the library in which it is located, the interface by which it can be called and the number of parameters.

- Functions such as `help()` which perform library or package index searches now use NULL as default for their 'lib.loc' argument so that missingness can be propagated more easily. The default corresponds to all currently known libraries as before.

- Added function `file.rename()`.

- `hist.default()` allows 'breaks' to specify a rule to choose the number of classes, and rules '"Sturges"' (the previous default), '"Scott"' and '"FD"' (Freedman-Diaconis) are supplied (based on package **MASS**).

- Function `identical()`, a fast and reliable way to test for exact equality of two objects.

- New generic function `is.na<-()`, from S4. This is by default equivalent to 'x[value] <- NA' but may differ, e.g., for factors where '"NA"' is a level.

- `is.`*xxx* reached through `do_is` are now generic.

- `La.eigen()` and `La.svd()` have new default methods to use later (and often much faster) LAPACK routines. The difference is most noticeable on systems with optimized BLAS libraries.

- `length()` is now generic.

- New function `.libPaths()` for getting or setting the paths to the library trees R knows about. This is still stored in '.lib.loc', which however should no longer be accessed directly.

- Using lm/glm/... with 'data' a matrix rather than a data frame now gives a specific error message.

- `loess()`, `lqs()`, `nls()` and `ppr()` use the standard NA-handling and so support 'na.action = na.exclude'.

- `mahalanobis()` now has a 'tol' argument to be passed to `solve()`.

- `mean()` has a 'data.frame' method applying mean column-by-column. When applied to non-numeric data `mean()` now returns 'NA' rather than a confusing error message (for compatibility with S4). Logicals are still coerced to numeric.

- The formula interface to `mosaicplot()` now allows a contingency table as data argument.

- `new.env()` is now internal and allows you to set hashing. Also, `parent.env()` and `parent.env<-()` are included to provide direct access to setting and retrieving environments.

- Function `nsl()` to look up IP addresses of hosts: intended as a way to test for internet connectivity.

- `Ops()`, `cbind()`, `diff()` and `na.omit()` methods for time series objects moved from package **ts** to package **base**.
- New option 'download.file.method' can be used to set the default method for `download.file()` and functions which use it such as `update.packages()`.
- `order()` and `sort.list()` now implement 'na.last = FALSE' and 'na.last = NA'.
- Started work on new package management system: `packageStatus()` and friends.
- `page()` has a new 'method' argument allowing 'method = print'.
- `png()`, `jpeg()` and `bmp()` devices now have a 'bg' argument to set the background color: useful to set '"transparent"' on `png()`.
- Changes to the `postscript()` device:
  - The symbol font can now be set on a `postscript()` device, and support has been added for using Computer Modern type-1 fonts (including for symbols). (Contributed by Brian D'Urso.)
  - There is now support for URW font families: this will give access to more characters and more appropriate metrics on PostScript devices using URW fonts (such as ghostscript).
  - %%IncludeResource comments have been added to the output. (Contributed by Brian D'Urso.)
- `predict.ppr()` now predicts on 'newdata' containing NAs.
- `princomp()` now has a formula interface.
- `readChar()` now returns what is available if fewer characters than requested are on the file.
- `readline()` allows up to 256 chars for the prompt.
- `read.table()`, `scan()` and `count.fields()` have a new argument 'comment.char', default '#', that can be used to start comments on a line.
- New function `reg.finalizer()` to provide R interface to finalization.
- `reshape()` extends `reshapeLong` and `reshapeWide`, which are deprecated.
- `rle()` now returns a classed object, has a print method and an inverse.
- Changes to `save()` and friends:
  - `save()` now takes an 'envir' argument for specifying where items to be saved are to be found.
  - A new default format for saved workspaces has been introduced. This format provides support for some new internal data types, produces smaller save files when saving code, and provides a basis for a more flexible serialization mechanism.
  - Modified 'save' internals to improve performance when saving large collections of code.
  - `save()` and `save.image()` now take a 'version' argument to specify the workspace file-format version to use. The version used from R 0.99.0 to 1.3.1 is version 1. The new default format is version 2. `load()` can read a version 2 saved workspace if it is compressed.
  - `save()` and `save.image()` now take a 'compress' argument to specify that the saved image should be written using the zlib compression facilities.
  - `save.image()` now takes an argument 'ascii'.
  - `save.image()` now takes an argument 'safe'. If 'TRUE', the default, a temporary file is used for creating the saved workspace. The temporary file is renamed if the save succeeds. This preserves an existing workspace if the save fails, but at the cost of using extra disk space during the save.
  - `save.image()` default arguments can be specified in the 'save.image.defaults' option. These specifications are used when `save.image()` is called from `q()` or GUI analogs.
- `scan()` allows unlimited (by R) lengths of input lines, instead of a limit of 8190 chars.
- `smooth.spline()` has a new 'control.spar' argument and returns 'lambda' besides 'spar'. 'spar' ≤ 0 is now valid and allows to go more closely towards interpolation (lambda $\rightarrow 0$) than before. This also fixes `smooth.spline()` behavior for 'df $\approx$ n - 2'. Better error messages in several situations.

  Note that spar = 0 is no longer the default and no longer entails cross-validation.
- `stars()` has been enhanced; new 'mar' argument uses smaller `mar(gins)` by default; further 'nrow' and 'ncol' as S-Plus, 'frame.plot', 'flip.labels', 'lty' and explicit 'main', 'sub', 'xlab' and 'ylab'. Note that 'colors' has been

replaced by 'col.segments' and there's a new 'col.stars'.

`stars()` now returns the locations invisibly.

- `step()` is now closer to `stepAIC()` and so handles a wider range of objects (but stepAIC [in **MASS**] is still more powerful).

- `symbols()` now has automatic xlab and ylab and a main argument which eliminates an incorrect warning. It better checks wrongly scaled arguments.

- `Sys.setlocale()` now issues a warning if it fails.

- An enhanced function `type.convert()` is now a documented function, rather than just internal to `read.table()`.

- `warning()` allows multiple arguments, following S4's style.

- New function `with()` for evaluating expressions in environments constructed from data.

- Unix `x11()` devices can now have a canvas color set, which can help to distinguish plotting '"white"' from plotting '"transparent"'.

- On Unix, `X11()`, `png()` and `jpeg()` now give informative warnings if they fail to open the device.

- The startup processing now interprets escapes in the values of environment variables set in 'R_HOME/etc/Renviron' in a similar way to most shells.

- The operator '=' is now allowed as an assignment operator in the grammar, for consistency with other languages, including recent versions of S-Plus. Assignments with '=' are basically allowed only at top-level and in braced or parenthesized expressions, to make famous errors such as `if(x=0) 1 else 2` illegal in the grammar. (There is a plan to gradually eliminate the underscore as an assignment in future versions of R.)

- Finalizers can be registered to be run on system exit for both reachable and unreachable objects.

- integer addition, subtraction, and multiplication now return NA's on overflow and issue a warning.

- Printing factors with both level '"NA"' and missing values uses '<NA>' for the missing values to distinguish them.

- Added an experimental interface for locking environments and individual bindings. Also added support for "active bindings" that link a variable to a function (useful for example for linking an R variable to an internal C global).

- GNOME interface now has separate colours for input and output text (like the windows GUI). These can be modified via the properties dialogue.

- Output from the GNOME console is block buffered for increased speed

- The GNOME console inherits standard emacs-style keyboard shortcuts from the GtkText widget for cursor motion, editing and selection. These have been modified to allow for the prompt at the beginning of the command line.

- One can register R functions and C routines to be called at the end of the successful evaluation of each top-level expression, for example to perform auto-saves, update displays, etc. See `addTaskCallback()` and `taskCallbackManager()`. See http://developer.r-project.org/TaskHandlers.pdf.

Deprecated & defunct

- `.Alias` has been removed from all R sources and deprecated.

- `reshapeLong()` and `reshapeWide()` are deprecated in favour of `reshape()`.

- Previously deprecated functions `httpclient()`, `parse.dcf()`, `read.table.url()`, `scan.url()`, and `source.url()` are defunct. Method '"socket"' for `download.file()` no longer exists.

Documentation changes

- *Writing R Extensions* has a new chapter on generic/method functions.

Utilities

- New package **tools** for package development and administration tools, containing the QA tools `checkFF()`, `codoc()` and `undoc()` previously in package **base**, as well as the following new ones:

  - `checkAssignFuns()` for checking whether the final argument of assignment functions in a package is named 'value'.

  - `checkDocArgs()` for checking whether all arguments shown in `\usage` of Rd files are documented in the corresponding `\arguments`.
  - `checkMethods()` for checking whether all methods defined in a package have all arguments of their generic.
  - `checkTnF()` for finding expressions containing the symbols 'T' and 'F'.

- `R CMD Rd2dvi` has more convenient defaults for its output file.

- `R CMD check` now also fully checks the 'Depends' field in the package 'DESCRIPTION' file. It also tests for syntax errors in the R code, whether all methods in the code have all arguments of the corresponding generic, for arguments shown in `\usage` but not documented in `\arguments`, and whether assignment functions have their final argument named 'value'.

C-level facilities

- `arraySubscript` and `vectorSubscript` are now available to package users. All "array-like" packages can use a standard method for calculating subscripts.

- The C routine `type2symbol`, similar to `type2str`, returns a symbol corresponding to the type supplied as an argument.

- The macro `SHLIB_EXT` now includes '.', e.g. '".so"' or '".dll"', since the Mac uses "Lib" without a '.'.

- New FORTRAN entry points `rwarn()` and `rexit()` for warnings and error exits from compiled Fortran code.

- A new serialization mechanism is available that can be used to serialize R objects to connections or to strings. This mechanism is used for the version 2 save format. For now, only an internal C interface is available.

- `R_tryEval()` added for evaluating expressions from C code with errors handled but guaranteed to return to the calling C routine. This is used in embedding R in other applications and languages.

- Support for `attach()`'ing user-defined tables of variables is available and accessed via the **RObjectTables** package currently at http://www.omegahat.org/RObjectTables.

Bug fixes

- Fixed 'share/perl/massage-examples.pl' to detect instances of `par()` at the very start of a line.

- Fixed Pearson residuals for glms with non-canonical `link.(PR#1123)`. Fixed them again for weights (PR#1175).

- Fixed an inconsistency in the evaluation context for on.exit expressions between explicit calls to 'return' and falling off the end returns.

- The code in `model.matrix.default()` handling contrasts was assuming a response was present, and so without a response was failing to record the contrasts for the first variable if it was a factor.

- `diffinv()` could get the time base wrong in some cases.

- `file.append()` was opening all files in text mode: mattered on Windows and classic Macintosh. (PR#1085)

- `f[] <- g` now works for factor `f`.

- `substr<-()` was misbehaving if the replacement was too short.

- The version of 'packages.html' generated when building R or installing packages had an incorrect link to the style sheet. The version used by `help.start()` was correct. (PR#1090)

- `rowsum()` now gives character (not factor codes) as rownames. (PR#1092)

- `plot.POSIXct` and `plot.POSIXlt` now respect the 'xaxt' parameter.

- It is now possible to predict from an intercept-only model: previously `model.matrix.default()` objected to a 0-column model frame.

- `c.POSIXct` was not setting the right classes in 1.3.x.

- `cor(*, use = "all.obs")` ≤ 1 is now guaranteed which ensures that `sqrt(1 - r^2)` is always ok in `cor.test()`. (PR#1099)

- `anova.glm()` had a missing 'drop=FALSE' and so failed for some intercept-less models.

- `predict.arima0()` now accepts vector as well as matrix 'newxreg' arguments.

- `cbind(d,f)` now works for 0-column dataframes. This fixes PR#1102.

- `plot(ts(0:99), log = "y")` now works.

- method '"gnudoit"' of `bug.report()` was incorrectly documented as '"gnuclient"' (PR#1108)

- saving with 'ascii=TRUE' mangled backslashes. (PR#1115)
- `frac(,)` and others now adds a gap appropriately. (PR#1101)
- `logLik.lm()` now uses the correct '"df"' (**nlme** legacy code).
- `closeAllConnections()` works again, and closes all `sink()` diversions.
- `sink(type="message")` works again.
- `sink.number` was (accidentally) returning the result invisibly.
- `as.POSIXct("NA")` (or ..lt) now work; hence, `merge(*, all=TRUE)` now works with dataframes containing POSIXt date columns.
- `integer(2^30+1)` and similar ones do not segfault anymore but duly report allocation errors.
- `seq(0, 0, 1)` now works (PR#1133).
- `reshapeWide()` got it wrong if the '"i"' factor was not sorted (the function is now deprecated since `reshape()` is there, but the bug still needed fixing ...)
- PR#757 was fixed incorrectly, causing improper subsetting of 'pch' etc. in `plot.formula()`.
- `library()` no longer removes environments of functions that are not defined in the top-level package scope. Also, packages loaded by `require()` when sourcing package code are now visible in the remaining source evaluations.
- `names(d) <- v` now works (again) for '"dist"' objects d. (PR#1129)
- Workarounds for problems with incompletely specified date-times in `strptime()` which were seen only on glibc-based systems (PR#1155).
- `promax()` was returning the wrong rotation matrix. (PR#1146)
- The [pqr]signrank and [pqr]wilcox functions failed to check that memory has been allocated (PR#1149), and had (often large) memory leaks if interrupted. They now can be interrupted on Windows and MacOS and don't leak memory.
- `range(numeric(0))` is now `c(NA, NA)` not `NA`.
- `round(x, digits)` for `digits` ≤ 0 always gives an integral answer. Previously it might not due to rounding errors in fround. (PR#1138/9)
- Several memory leaks on interrupting functions have been circumvented. Functions `lqs()` and `mve()` can now be interrupted on Windows and MacOS.
- `image()` was finding incorrect breakpoints from irregularly-spaced midpoints. (PR#1160)
- Use fuzz in the 2-sample Kolmogorov-Smirnov test in package **ctest** to avoid rounding errors (PR#1004, follow-up).
- Use exact Hodges-Lehmann estimators for the Wilcoxon tests in package **ctest** (PR#1150).
- Arithmetic which coerced types could lose the class information, for example 'table - real' had a class attribute but was not treated as a classed object.
- Internal ftp client could crash R under error conditions such as failing to parse the URL.
- Internal clipping code for circles could attempt to allocate a vector of length -1 (related to PR#1174)
- The hash function used internally in `match()`, `unique()` and `duplicated()` was very inefficient for integers stored as numeric, on little-endian chips. It was failing to hash the imaginary part of complex numbers.
- `fifo()` no longer tries to truncate on opening in modes including '"w"'. (Caused the fifo example to fail on HP-UX.)
- Output over 1024 characters was discarded from the GNOME console.
- `rug()` now correctly warns about clipped values also for logarithmic axes and has a 'quiet' argument for suppressing these (PR#1188).
- `model.matrix.default` was not handling correctly 'contrasts.arg' which did not supply a full set of contrasts (PR#1187).
- The 'width' argument of `density()` was only compatible with S for a Gaussian kernel: now it is compatible in all cases.
- The `rbinom()` C code had a transcription error from the original Fortran which led to a small deviation from the intended distribution. (PR#1190)
- `pt(t, , ncp=0)` was wrong if `t` was +/-Inf.
- Subsetting grouping factors gave incorrect degrees of freedom for some tests in package **ctest**. (PR#1124)
- `writeBin()` had a memory leak.
- `qbeta(0.25, 0.143891, 0.05)` was (incorrectly) 3e-308. (PR#1201)

- Fixed alignment problem in 'ppr.f' on Irix. (PR#1002, 1026)
- `glm()` failed on null binomial models. (PR#1216)
- `La.svd()` with 'nu' = 0 or 'nv' = 0 could fail as the matrix passed to DGESVD was not of dimension at least one (it was a vector).
- Rownames in 'xcoef' and 'ycoef' of `cancor()` were wrong if 'x' or 'y' was rank-deficient.
- `lqs()` could give warnings if there was an exact fit. (PR#1184)
- `aov()` didn't find free-floating variables for `Error()` terms when called from inside another function.
- `write.table()` failed if asked to quote a numerical matrix with no row names. (PR#1219)
- `rlnorm( *, *, sd=0)` now returns the mean, `rnbinom(*, *, prob=1)` gives 0, (PR#1218).

Changes in R 1.4.1

by the R Core Team

Bug fixes

- `scan(multi.line = FALSE)` now always gives an immediate error message if a line is incomplete. (As requested in PR#1210)
- `read.table()` is no longer very slow in processing comments: moved to C code and fewer lines checked.
- `type.convert()` could give stack imbalance warnings if used with 'as.is = TRUE'.
- predict.mlm ignored newdata (PR#1226) and also offsets.
- `demo(tkttest)` was inadvertently changed in 1.4.0 so that it would evaluate the requested test, but not display the result.
- `stars(scale = TRUE)` (the default) now works as documented (and as S does). Previously it only scaled the maximum to 1. (PR#1230)
- `d0 <- data.frame(a = 0);data.matrix(d0[0, 0])` and `data.matrix(d0[, 0])` now work.
- `plot(`*multiple time series*`, plot.type = "single")` was computing 'ylim' from the first series only.
- `plot.acf()` has a new 'xpd = `par("xpd")`' argument which by default *does* clipping (of the horizontal lines) as desired ('xpd' = NA was used before, erroneously in most cases).
- `predict(smooth.spline(.), deriv = 1)` now works.
- `identify()` failed when `x` is a structure/matrix. (PR#1238)
- `getMethod()` returns NULL when 'optional=TRUE' as promised in the documentation.
- `setMethod()` allows `...` to be one of the arguments omitted in the method definition (but so far no check for `...` being missing)
- Allow `round()` to work again on very large numbers (introduced in fixing PR#1138). (PR#1254)
- 'Rinternals.h' is now accepted by a C++ compiler.
- `type.convert()` was failing to detect integer overflow.
- `piechart()` was defaulting to foreground colour (black) fills rather than background (as used in 1.3.1 and earlier). Now background is used, but be aware that as from 1.4.0 this may be transparent.
- `La.eigen(*, only.values=TRUE)` does not segfault anymore in one branch (PR#1262).
- `cut()` now produces correct default labels even when 'include.lowest = TRUE' (PR#1263).
- `reformulate()` works properly with a response.
- `cmdscale(*, k = 1)` now works properly.
- Options 'by = "month"' and 'by = "year"' to `seq.POSIXt()` will always take account of changes to/from daylight savings time: this was not working on some platforms.
- `glm.fit.null()` now accepts all the arguments of `glm.fit()` (it could be called from `glm.fit` with arguments it did not accept), and is now documented.
- `cov.wt(cbind(1), cor = TRUE)` now works.
- `predict(glm.object, se.fit = TRUE)` was failing if the fit involved an offset.

- `detach()` on '"package:base"' would crash R. (PR#1271)
- `print` or `summary` on a `manova()` object with no terms, no names on the response and 'intercept = FALSE' (which is not sensible) would give an error.
- `seek()` on file connections was ignoring the 'origin' argument.
- Fixed new environment handling in `library()` to avoid forcing promises created by `delay()`.
- `arima0()` could leak memory: now released via `on.exit()`.
- `qr.coef(qr,*)` now keeps the names of `qr$qr`.
- `read.00Index()` no longer fails on data indexes not generated by `Rdindex` (PR#1274).

Changes on CRAN

by Kurt Hornik and Friedrich Leisch

CRAN packages

The following extension packages from 'src/contrib' were added since the last newsletter.

Bhat Functions for MLE, MCMC, CIs (originally in Fortran). By E. Georg Luebeck.

CircStats Circular Statistics, from 'Topics in circular Statistics' by S. Rao Jammalamadaka and A. SenGupta, World Scientific (2001). S original by Ulric Lund, R port by Claudio Agostinelli.

ROracle Oracle Database Interface driver for R. Uses the ProC/C++ embedded SQL. By David A. James and Jake Luciani.

RQuantLib The RQuantLib packages provides access to (some) of the QuantLib functions from within R. It is currently limited to some Option pricing and analysis functions. The QuantLib project aims to provide a comprehensive software framework for quantitative finance. The goal is to provide a standard free/open source library to quantitative analysts and developers for modeling, trading, and risk management of financial assets. By Dirk Eddelbuettel for the R interface, and the QuantLib group for QuantLib (http://www.quantlib.org/html/group.html).

RSQLite Database Interface R driver for SQLite. Embeds the SQLite database engine in R. By David A. James.

RadioSonde RadioSonde is a collection of programs for reading and plotting SKEW-T,log p diagrams and wind profiles for data collected by radiosondes (the typical weather balloon-borne instrument). By Tim Hoar, Eric Gilleland, and Doug Nychka.

agce Contains some simple functions for the analysis of growth curve experiments. By Raphael Gottardo.

aws Contains R functions to perform the adaptive weights smoothing (AWS) procedure described in Polzehl und Spokoiny (2000), Adaptive weights smoothing with applications to image restoration, *Journal of the Royal Statistical Society*, Ser. B, 62, 2, 335–354. By Joerg Polzehl.

combinat Routines for combinatorics. By Scott Chasalow.

deldir Calculates the Delaunay triangulation and the Dirichlet or Voronoi tesselation (with respect to the entire plane) of a planar point set. By Rolf Turner.

dr Functions, methods, and datasets for fitting dimension reduction regression, including pHd and inverse regression methods SIR and SAVE. These methods are described, for example, in R. D. Cook (1998), Regression Graphics, Wiley, New York. Also included is code for computing permutation tests of dimension. By Sanford Weisberg.

emplik empirical likelihood ratio for means, quantiles, and hazards from possibly right censored data. By Mai Zhou and Art Owen.

evd Extends simulation, distribution, quantile and density functions to univariate, bivariate and (for simulation) multivariate parametric extreme value distributions, and provides fitting functions which calculate maximum likelihood estimates for univariate and bivariate models. By Alec Stephenson.

g.data Create and maintain delayed-data packages (DDP's). Data stored in a DDP are available on demand, but do not take up memory until requested. You attach a DDP with `g.data.attach()`, then read from it and assign to it in a manner similar to S-Plus, except that you must run `g.data.save()` to actually commit to disk. By David Brahm.

geoRglm Functions for inference in generalised linear spatial models. By Ole F. Christensen and Paulo J. Ribeiro Jr.

grid A rewrite of the graphics layout capabilities, plus some support for interaction. By Paul Murrell.

hdf5 Interface to the NCSA HDF5 library. By Marcus G. Daniels.

ifs Iterated Function Systems distribution function estimator. By S. M. Iacus.

lasso2 Routines and documentation for solving regression problems while imposing an L1 constraint on the estimates, based on the algorithm of Osborne et al. (1998). By Justin Lokhorst, Bill Venables and Berwin Turlach; first port to R by Martin Maechler.

lattice Implementation of Trellis Graphics. By Deepayan Sarkar.

moc Fits a variety of mixtures models for multivariate observations with user-defined distributions and curves. By Bernard Boulerice.

pastecs Regulation, decomposition and analysis of space-time series. By Frederic Ibanez, Philippe Grosjean & Michele Etienne.

pear Package for estimating periodic autoregressive models. Also includes methods for plotting periodic time series data. S original by A. I. McLeod, R port by Mehmet Balcilar.

qtl Analysis of experimental crosses to identify genes (called quantitative trait loci, QTLs) contributing to variation in quantitative traits. By Karl W Broman, with ideas from Gary Churchill and Saunak Sen and contributions from Hao Wu.

spatstat Data analysis and modelling of two-dimensional point patterns, including multi-type points and spatial covariates. By Adrian Baddeley and Rolf Turner.

spsarlm Functions for estimating spatial simultaneous autoregressive (SAR) models. By Roger Bivand.

New country mirrors

We now also have CRAN country mirrors in Brazil (thanks to Paulo Justiniano Ribeiro Jr `p.ribeiro@lancaster.ac.uk`) and in Germany.

New submission email

The email address for submissions to CRAN now is `cran@r-project.org` (the old address no longer works). Uploads still go to `ftp://cran.r-project.org/incoming/`.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
`Kurt.Hornik@R-project.org`

Friedrich Leisch
Technische Universität Wien, Austria
`Friedrich.Leisch@ci.tuwien.ac.at`