limma: Linear Models for Microarray Data

User's Guide

Gordon Smyth and Natalie Thorne
The Walter and Eliza Hall Institute of Medical Research
9 September 2003

Table of Contents

1. Introduction

Limma is a package for the analysis of gene expression microarray data, especially the use of linear models for analysing designed experiments and the assessment of differential expression. Limma provides the ability to analyse comparisons between many RNA targets simultaneously. The normalization and data analysis functions are for two-colour spotted microarrays. The linear model and differential expression functions apply to all microarrays including Affymetrix and other multi-array oligonucleotide experiments.

The Bioconductor packages marrayClasses, marrayInput and marrayNorm provide alternative functions for reading and normalizing spotted microarray data. If you are using limma in conjunction with these packages, see Section 12. The package affy provides functions for reading and normalizing Affymetrix microarray data. If you are using the affy package, see Section 13.

This tutorial was prepared using R Version 1.7.1 for Windows and limma Version 1.1.18. The latest version of limma is always available from http://www.bioconductor.org under "Developmental Packages". If you are using Windows, you can install the last official release version of limma from the drop-down menu in R, simply select `Packages` then `Install package(s) from Bioconductor...`. Download is also available from http://bioinf.wehi.edu.au/limma/. The data sets

used in the examples can be downloaded from http://bioinf.wehi.edu.au/marray/genstat2002/. Help with limma is available by sending questions or problems to bioconductor@stat.math.ethz.ch.

2. A Few Preliminaries on R

R is a program for statistical computing. It is a command-driven language meaning that you have to type commands into it rather than pointing and clicking. A good way to get started is to type

```
help.start()
```

at the R prompt or, if you're using Windows, to follow the drop-down menu [Help > Html help]. Following the links [Packages > limma] from the html help page will lead you to the contents page of help topics for commands in limma.

Before you can use any limma commands you have to load the package by typing

```
library(limma)
```

at the R prompt. You can get help on any function in any loaded package by typing ? and the function name at the R prompt, for example

```
?read.maimages
```

for detailed help on the `read.maimages` function. Anything that you create in R is an "object". Objects might include data sets, variables, functions, anything at all. For example

```
x <- 2
```

will create a variable `x` and will assign it the value 2. At any stage of your R session you can type

```
objects()
```

to get a list of all the objects you have created. You see show the contents of any object by typing the name of the object at the prompt, for example either of the following commands will print out the contents of `x`:

```
show(x)
x
```

We hope that you can use limma without having to spend a lot of time learning about the R language itself but a little knowledge in this direction will be very helpful, especially when you want to do something not explicitly provided for in limma or in the other Bioconductor packages. For more details about the R language see *An Introduction to R* which is available from the online help.

3. Reading in Intensity Data

We assume that an experiment has been conducted with one or more microarrays, all printed with the same library of probes. Each array has been scanned to produce a TIFF image. The TIFF images have then been processed using an image analysis program such a ArrayVision, ImageGene, GenePix, QuantArray or SPOT to acquire the red and green foreground and background intensities for each spot. The spot intensities have then been exported from the image analysis program into a series of text files. There should be one file for each array or, in the case of Imagene, two files for each array.

Let `files` be a character vector containing the names of the image analysis output files. The foreground and background intensities can be read into an `RGList` object using a command of the form

```
RG <- read.maimages(files,source="<imageanalysisprogram>",path="<directory>")
```

where `<imageanalysisprogram>` is the name of the image analysis program and `<directory>` is the full path of the directory containing the files. If the files are in the current R working directory then the argument `path` can be omitted; see the help entry for `setwd` for how to set the current working directory. For example, if the files are SPOT output and have common extension "spot" then they can be read using

```
files <- dir(pattern="*\\.spot")
RG <- read.maimages(files, source="spot")
```

The object `files` is then a character vector containing all the spot file names in alphabetical order. If the files are GenePix output files and have extension "gpr" then they can be read using

```
files <- dir(pattern="*\\.gpr")
RG <- read.maimages(files, source="genepix")
```

Consult the help entry for `read.maimages` to see which other image analysis programs are supported. Files are assumed by default to be tab-delimited. If the files use a different separator this may be specified using the `sep=` argument. For example if the Genepix files were comma-separated (csv) then the read command would be

```
RG <- read.maimages(files, source="genepix", sep=",")
```

What should you do if your image analysis program is not currently supported by limma? If your output files are of a standard format, you can supply the column names corresponding to the intensities yourself. For example,

```
RG <- read.maimages(files, columns=list(Rf="F635 Mean",Gf="F532 Mean",Rb="B635
Median",Gb="B532 Median"))
```

is exactly equivalent to the earlier command with `source="genepix"`. "Standard format" means here that there is a unique column name identifying each column of interest and that there are no lines in the file following the last line of data. Header information at the start of the file is ok.

It is a good idea to look at your data to check that it has been read in correctly. Type

```
show(RG)
```

to see a print out the first few lines of data. Also try

```
summary(RG$R)
```

to see a five-number summary of the red intensities for each array, and so on.

It is possible to read the data in several steps. If `RG1` and `RG2` are two data sets corresponding to different sets of arrays then

```
RG <- cbind(RG1, RG2)
```

will combine them into one large data set. Data sets can also be subsetted. For example `RG[,1]` is the data for the first array while `RG[1:100,]` is the data on the first 100 genes.

4. Reading in Gene List Data

If the arrays have been scanned with an Axon scanner, then the gene names will be available in a GenePix Array List (GAL) file. If the GAL file has extension "gal" and is in the current working directory, then it may be read into a data.frame by

```
> gal <- readGAL()
```

The print layout of the arrays can be extracted from the GAL by

```
> layout <- getLayout(gal)
```

5. Spot Quality Weights

It is desirable to use the image analysis to compute a weight for each spot between 0 and 1 which indicates the reliability of the acquired intensities at that spot. For example, if the SPOT image analysis program is used and the size of an ideal perfectly circular spot is known to be 100 pixels, then one might use

```
> RG <- read.maimages(files,source="spot",wt.fun=wtarea(100))
```

The function `wtarea(100)` gives full weight to spots with area 100 pixels and down-weights smaller and larger spots. Spots which have zero area or are more than twice the ideal size are given zero weight. This will create a component called `weights` in the RG list. The weights will be used automatically by functions such as `normalizeWithinArrays` which operate on the RG-list.

With GenePix data

```
> RG <- read.maimages(files,source="genepix",wt.fun=wtflags(0.1))
```

will give weight 0.1 to any spot which receives a negative flag from the GenePix program.

Computing quality weights depends on the image analysis program. Consult the help entry `QualityWeights` to see what quality weight functions are available.

6. Linear Models

The package limma uses an approach called *linear models* to analyse designed microarray experiments. This approach allows very general experiments to be analysed just as easily as a simple replicated experiment. The approach is explained in Smyth (2003) and Yang and Speed (2002). The approach requires one or two matrices to be specified. The first is the *design matrix* which indicates in effect which RNA samples have been applied to each array. The second is the *contrast matrix* which specifies which comparisons you would like to make between the RNA samples. For very simple experiments, you may not need to specify the contrast matrix.

If you have data from Affymetrix experiments, from single-channel spotted microarrays or from spotted microarrays using a common reference, then linear modeling is the same a ordinary analysis of variance or multiple regression except that a model is fitted for every gene. With data of this type you can create design matrices as one would do for ordinary modeling with univariate data. If you have have data from spotted microarrays using a direct design, i.e., a connected design with no common reference, then the linear modeling approach is very powerful but the creation of the design matrix may require more statistical knowledge.

7. Quick Start

For those who want to see very quickly what a limma analysis might look like for cDNA data, here is a quick analysis of four replicate arrays (including two dye-swaps). The data has been scanned using an Axon scanner, producing a Gene Allocation List (GAL) file, and then the intensities have been captured from the images using SPOT software. The GAL file and the image analysis files are in the current working directory of R. For more detail about the data see the Swirl Data example below.

```
> files <- dir(pattern="*.spot")            # Get the names of the files
containing the intensity data
> RG <- read.maimages(files, source="spot") # Read in the data
> RG$genes <- readGAL()                     # Read in GAL file containing gene
names
> RG$printer <- getLayout(RG$genes)         # Set printer layout information
> MA <- normalizeWithinArrays(RG)           # Print-tip group loess
normalization
> MA <- normalizeBetweenArrays(MA)          # Scale normalization between
arrays, optional
> fit <- lmFit(MA, design=c(-1,1,-1,1))     # Estimate all the fold changes by
fitting a linear model.
                                            # The design matrix indicates which

arrays are dye-swaps
> fit <- eBayes(fit)                        # Apply Bayesian smoothing to the
standard errors (very important!)
> options(digits=3)
```

```
> topTable(fit, n=30, adjust="fdr")         # Show the top 30 genes, control
false discovery rate
     Block Row Column      ID   Name     M     t  P.Value    B
3721     8   2      1 control   BMP2 -2.21 -21.1 0.000357 7.96
1609     4   2      1 control   BMP2 -2.30 -20.3 0.000357 7.78
3723     8   2      3 control   Dlx3 -2.18 -20.0 0.000357 7.71
1611     4   2      3 control   Dlx3 -2.18 -19.6 0.000357 7.62
8295    16  16     15 fb94h06 20-L12  1.27  14.1 0.002067 5.78
7036    14   8      4 fb40h07  7-D14  1.35  13.5 0.002067 5.54
515      1  22     11 fc22a09 27-E17  1.27  13.4 0.002067 5.48
5075    10  14     11 fb85f09 18-G18  1.28  13.4 0.002067 5.48
7307    14  19     11 fc10h09 24-H18  1.20  13.2 0.002067 5.40
319      1  14      7 fb85a01  18-E1 -1.29 -13.1 0.002067 5.32
2961     6  14      9 fb85d05 18-F10 -2.69 -13.0 0.002067 5.29
4032     8  14     24 fb87d12 18-N24  1.27  12.8 0.002067 5.22
6903    14   2     15 control    Vox -1.26 -12.8 0.002067 5.20
4546     9  14     10 fb85e07 18-G13  1.23  12.8 0.002067 5.18
683      2   7     11 fb37b09  6-E18  1.31  12.4 0.002182 5.02
1697     4   5     17 fb26b10  3-I20  1.09  12.4 0.002182 4.97
7491    15   5      3 fb24g06  3-D11  1.33  12.3 0.002182 4.96
4188     8  21     12 fc18d12 26-F24 -1.25 -12.2 0.002209 4.89
4380     9   7     12 fb37e11  6-G21  1.23  12.0 0.002216 4.80
3726     8   2      6 control  fli-1 -1.32 -11.9 0.002216 4.76
2679     6   2     15 control    Vox -1.25 -11.9 0.002216 4.71
5931    12   6      3 fb32f06  5-C12 -1.10 -11.7 0.002216 4.63
7602    15   9     18 fb50g12  9-L23  1.16  11.7 0.002216 4.63
2151     5   2     15 control   vent -1.40 -11.7 0.002216 4.62
3790     8   4     22 fb23d08  2-N16  1.16  11.6 0.002221 4.58
7542    15   7      6 fb36g12  6-D23  1.12  11.0 0.003000 4.27
4263     9   2     15 control   vent -1.41 -10.8 0.003326 4.13
6375    13   2     15 control   vent -1.37 -10.5 0.004026 3.91
1146     3   4     18 fb22a12  2-I23  1.05  10.2 0.004242 3.76
157      1   7     13 fb38a01   6-I1 -1.82 -10.2 0.004242 3.75
```

8. One-Sample Experiments

In this section we consider a case study in which two RNA sources are compared directly on a set of replicate or dye-swap arrays. The case study includes reading in the data, data display and exploration, as well as normalization and differential expression analysis. The analysis of differential expression is analogous to a classical one-sample test of location for each gene.

Example. Swirl Zebrafish Data

In this example we assume that the data is provided as a GAL file called "fish.gal" and raw SPOT output files.

Background. The experiment was carried out using zebrafish as a model organism to study the early development in vertebrates. Swirl is a point mutant in the BMP2 gene that affects the dorsal/ventral body axis. The main goal of the Swirl experiment is to identify genes with altered expression in the Swirl mutant compared to wild-type zebrafish.

The arrays. The microarrays used in this experiment were printed with 8448 probes (spots), including 768 control spots. The array printer uses a print head with a 4x4 arrangement of print-tips

and so the microarrays are partitioned into a 4x4 grid of tip groups. Each grid consists of 22x24 spots that were printed with a single print-tip. The gene name associated with each spot is recorded in a GenePix array list (GAL) file:

```
> gal <- readGAL("fish.gal")
> gal[1:30,]
   Block Row Column      ID     Name
1      1   1      1 control    geno1
2      1   1      2 control    geno2
3      1   1      3 control    geno3
4      1   1      4 control    3XSSC
5      1   1      5 control    3XSSC
6      1   1      6 control     EST1
7      1   1      7 control    geno1
8      1   1      8 control    geno2
9      1   1      9 control    geno3
10     1   1     10 control    3XSSC
11     1   1     11 control    3XSSC
12     1   1     12 control    3XSSC
13     1   1     13 control     EST2
14     1   1     14 control     EST3
15     1   1     15 control     EST4
16     1   1     16 control    3XSSC
17     1   1     17 control    Actin
18     1   1     18 control    Actin
19     1   1     19 control    3XSSC
20     1   1     20 control    3XSSC
21     1   1     21 control    3XSSC
22     1   1     22 control    3XSSC
23     1   1     23 control    Actin
24     1   1     24 control    Actin
25     1   2      1 control     ath1
26     1   2      2 control    Cad-1
27     1   2      3 control   DeltaB
28     1   2      4 control     Dlx4
29     1   2      5 control ephrinA4
30     1   2      6 control     FGF8
```

The hybridizations. Two sets of dye-swap experiments were performed making a total of four replicate hybridizations. Each of the arrays compares RNA from swirl fish with RNA from normal ("wild type") fish. The experimenters have prepared a tab-delimited file called "SwirlSamples.txt" which describes the four hybridizations:

```
> SwirlSample <- read.table("SwirlSample.txt",header=TRUE,sep="\t",as.is=TRUE)
> SwirlSample
  SlideNumber     FileName       Cy3       Cy5      Date
1          81 swirl.1.spot     swirl wild type 2001/9/20
2          82 swirl.2.spot wild type     swirl 2001/9/20
3          93 swirl.3.spot     swirl wild type 2001/11/8
4          94 swirl.4.spot wild type     swirl 2001/11/8
```

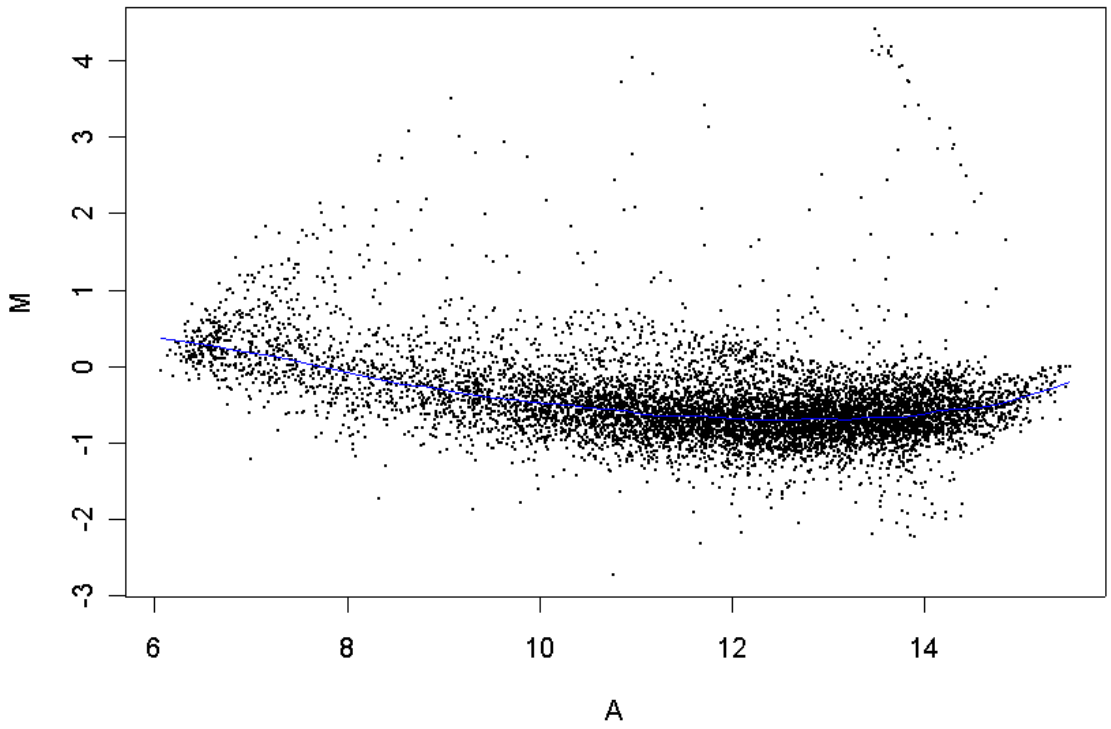
We see that slide numbers 81, 82, 93 and 94 were used to make the arrays. On slides 81 and 93, swirl RNA was labelled with green (Cy3) dye and wild type RNA was labelled with red (Cy5) dye. On slides 82 and 94, the labelling was the other way around.

Reading the first array. Each of the four hybridized arrays was scanned on an Axon scanner to produce a TIFF image, which was then processed using the image analysis software SPOT. The data from the arrays are stored in the four output files listed above. Normally we read in data from all the arrays into R at the same time, but for this tutorial we will start off by reading in the first array manually and doing some exploration:

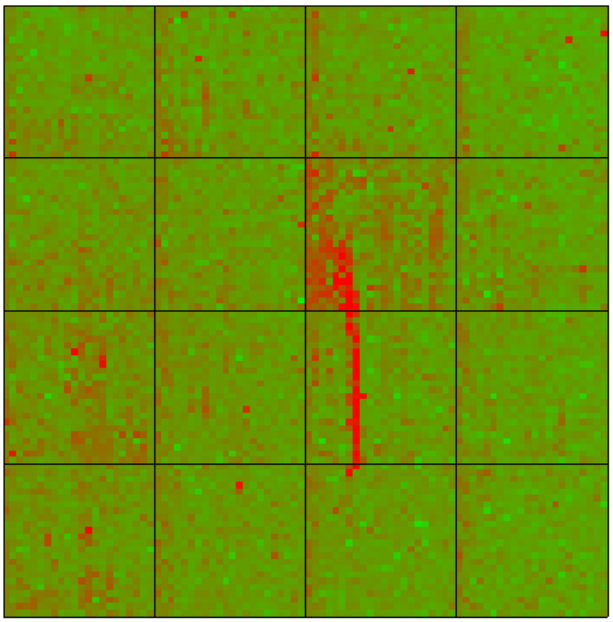
```
> swirl.1.spot <- read.table("swirl.1.spot",header=TRUE)
> names(swirl.1.spot)
 [1] "indexs"            "grid.r"            "grid.c"            "spot.r"
 [5] "spot.c"            "area"              "Gmean"             "Gmedian"
 [9] "GIQR"              "Rmean"             "Rmedian"           "RIQR"
[13] "bgGmean"           "bgGmed"            "bgGSD"             "bgRmean"
[17] "bgRmed"            "bgRSD"             "valleyG"           "valleyR"
[21] "morphG"            "morphG.erode"      "morphG.close.open" "morphR"
[25] "morphR.erode"      "morphR.close.open" "logratio"          "perimeter"
[29] "circularity"       "badspot"
```

We will use `Rmean` and `Gmean` as foreground intensities and `morphR` and `morphG` as background intensities. We can extract M and A-values from the array and do some initial data exploration.

```
> M <- m.spot(swirl.1.spot)
> A <- a.spot(swirl.1.spot)
> plot(A,M,pch=16,cex=0.1)
> lines(lowess(A,M),col="blue")
```



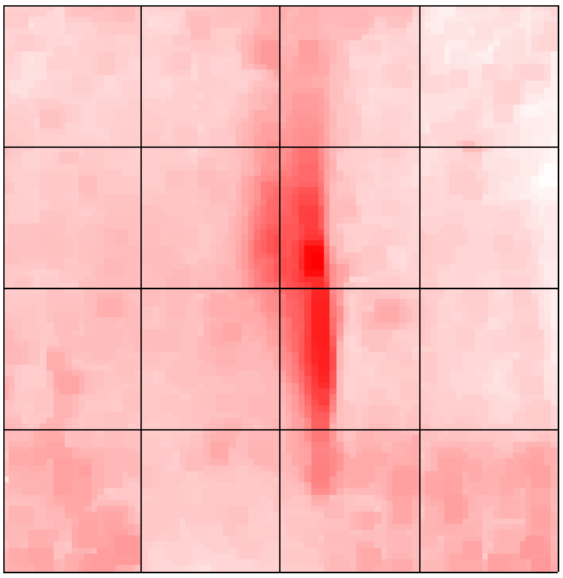
```
> layout <- getLayout(gal)
> layout
$ngrid.r
[1] 12

$ngrid.c
[1] 4

$nspot.r
[1] 22

$nspot.c
[1] 24

> imageplot(M,layout,zlim=c(-3,3))
```

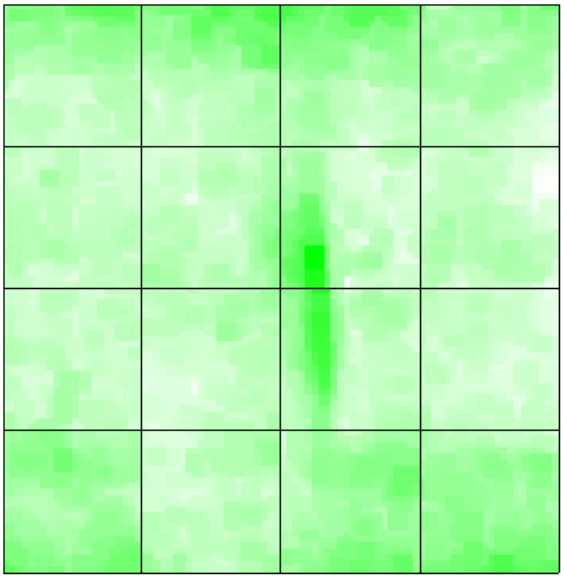


The `imageplot` function lies the slide on its side, so the first print-tip group is bottom left in this plot. We can see a red streak across the middle two grids of the 3rd row. Spots which are affected by this artefact will have suspect M-values. These spots are also visible as a cluster in the top right of the MA-plot.

It is also interesting to look at the variation of background values over the array:

```
> Rb <- swirl.1.spot$morphR
> Gb <- swirl.1.spot$morphG
> imageplot(log(Rb,2),layout,low="white",high="red")
> imageplot(log(Gb,2),layout,low="white",high="green")
```





Reading all the data. Normally we read in all the arrays at once rather than one at a time as above.

```
> slides <- SwirlSample$FileName
```

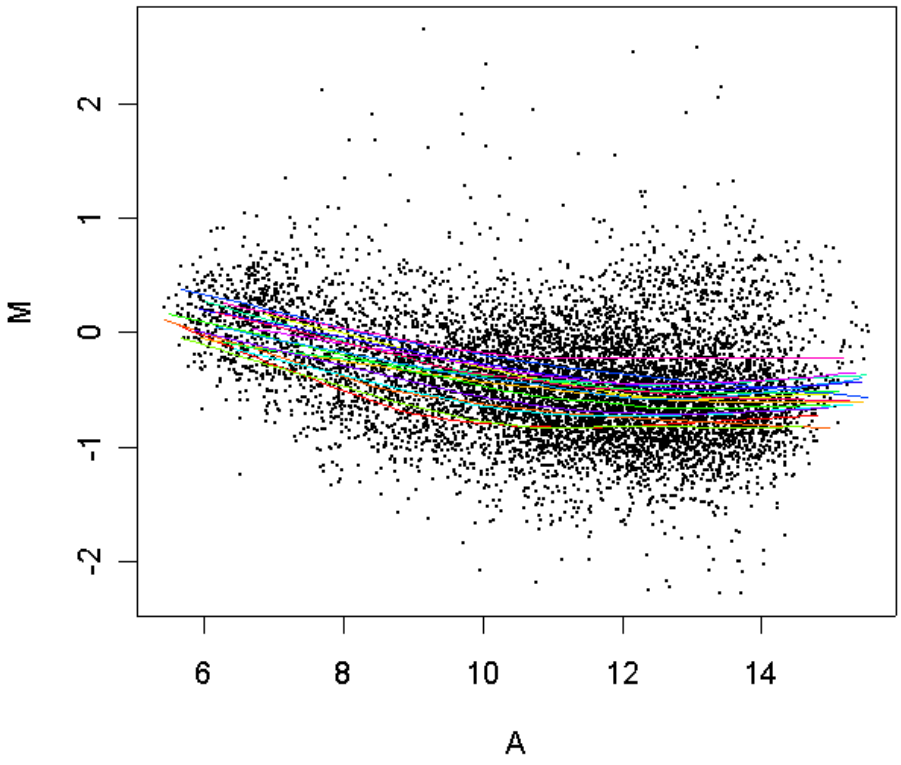
Now we will read the foreground and background intensities from all the arrays into one list. The function `rg.series.spot` assumes by default that the actual file names include have the extension `.spot`.

```
> RG <- read.maimages(slides,source="spot")
Read swirl.1.spot
Read swirl.2.spot
Read swirl.3.spot
```

```
Read swirl.4.spot
> names(RG)
[1] "R"  "G"  "Rb" "Gb"
```

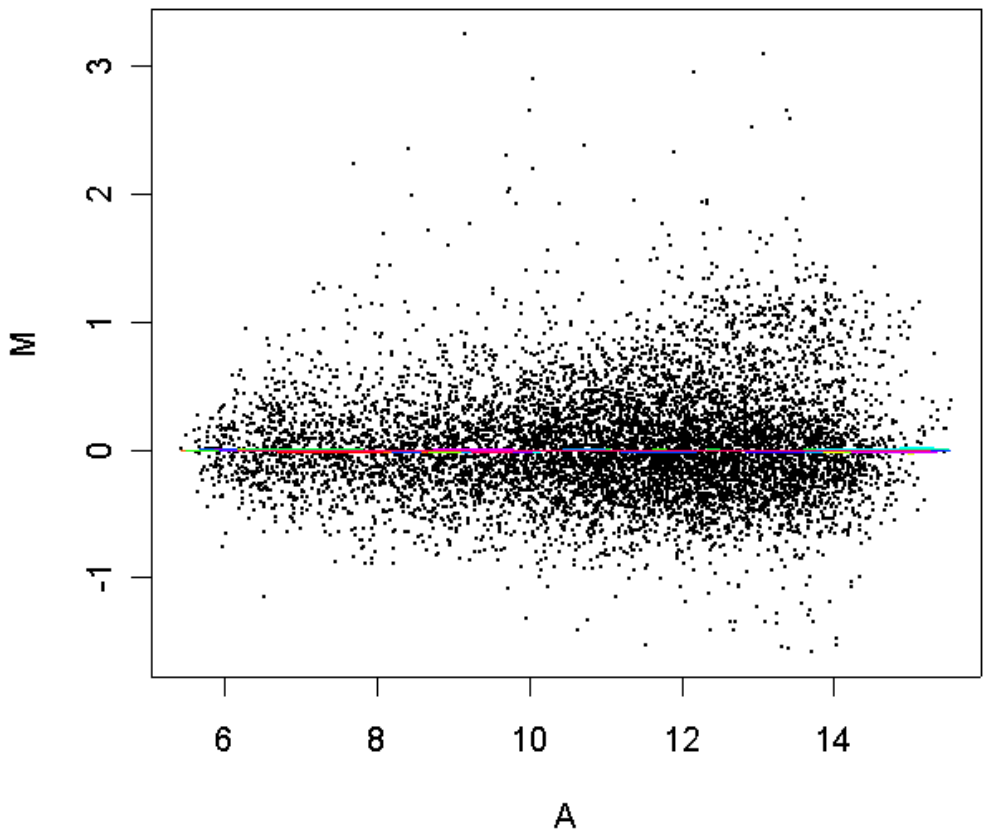
The following plot displays the individual print-tip loess curves for the third array (using a command from the SMA library):

```
> plot.print.tip.lowess(RG,layout,pch=16,cex=0.1,image=3)
```



Now the same curves after normalization:

```
> plot.print.tip.lowess(RG,layout,pch=16,cex=0.1,image=3,norm="p")
```

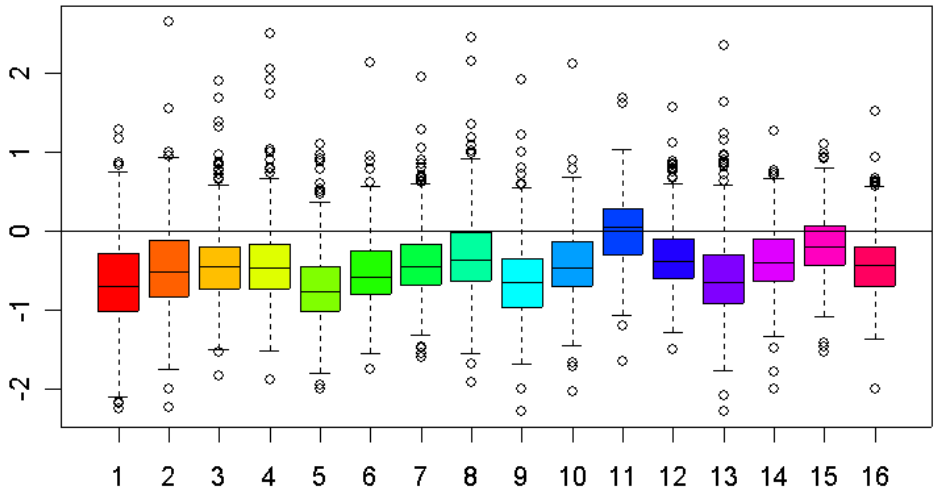


Now store the M and A-values without normalizing:

```
> MAraw <- MA.RG(RG)
> names(MAraw)
[1] "M"  "A"
```

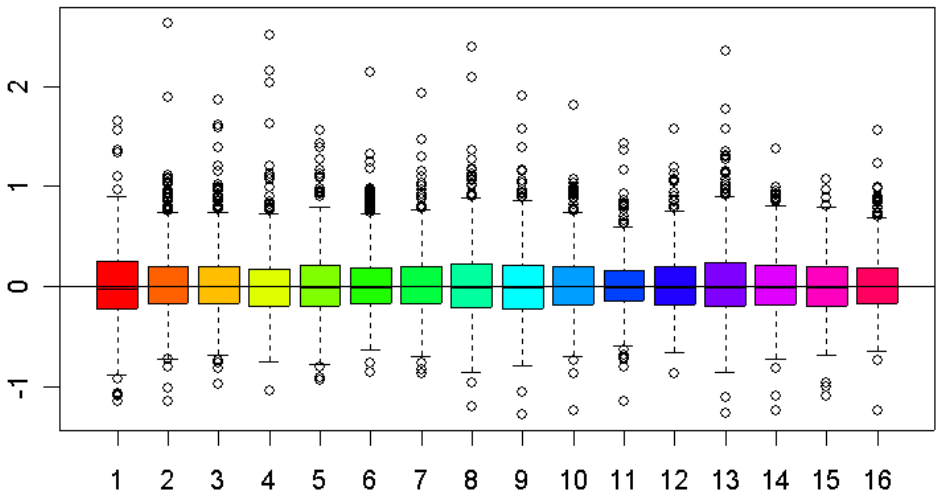
Boxplots can further display differences between the M-values in each print-tip group on the third array. The colouring used for the boxes is the same as that used for the loess curves in the previous two scatterplots.

```
> plot.scale.box(MAraw$M[,3],layout,col=rainbow(layout$ngrid.r*layout$ngrid.c))
> abline(0,0)
```



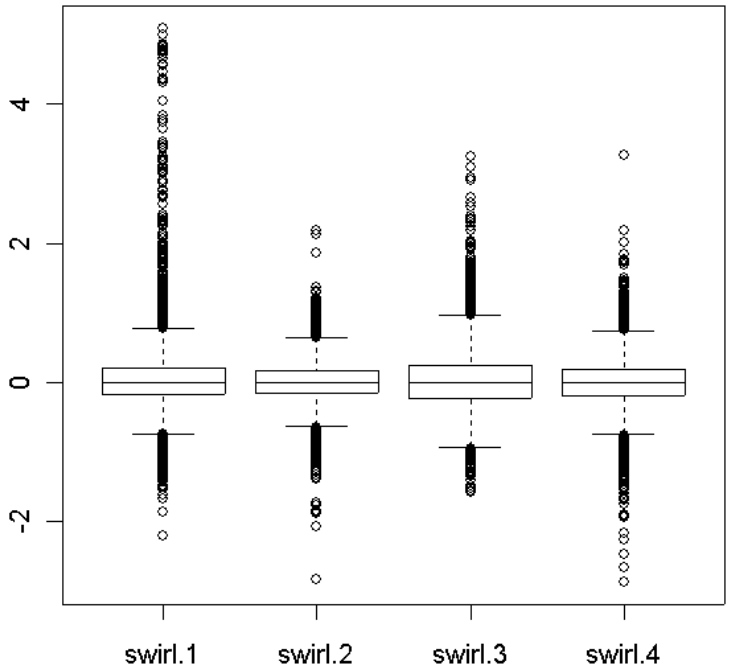
Now the same thing but after print-tip loess normalization. (Note that `normalizeWithinArrays` does print-tip loess normalization by default.)

```
> MA <- normalizeWithinArrays(RG,layout)
> plot.scale.box(MA$M[,3],layout,col=rainbow(layout$ngrid.r*layout$ngrid.c))
> abline(0,0)
```



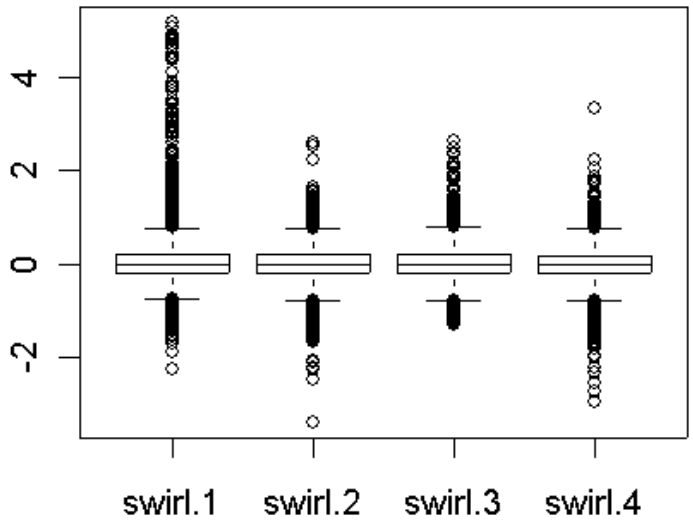
We have normalized the M-values with each array. A further question is whether normalization is required between the arrays. The following plot shows overall boxplots of the M-values for the four arrays.

```
> boxplot(MA$M~col(MA$M),names=slides)
```



There is some evidence that the different arrays have different spreads of M-values, so we will scale normalize between the arrays.

```
> MA <- normalizeBetweenArrays(MA)
> boxplot(MA$M~col(MA$M),names=slides)
```

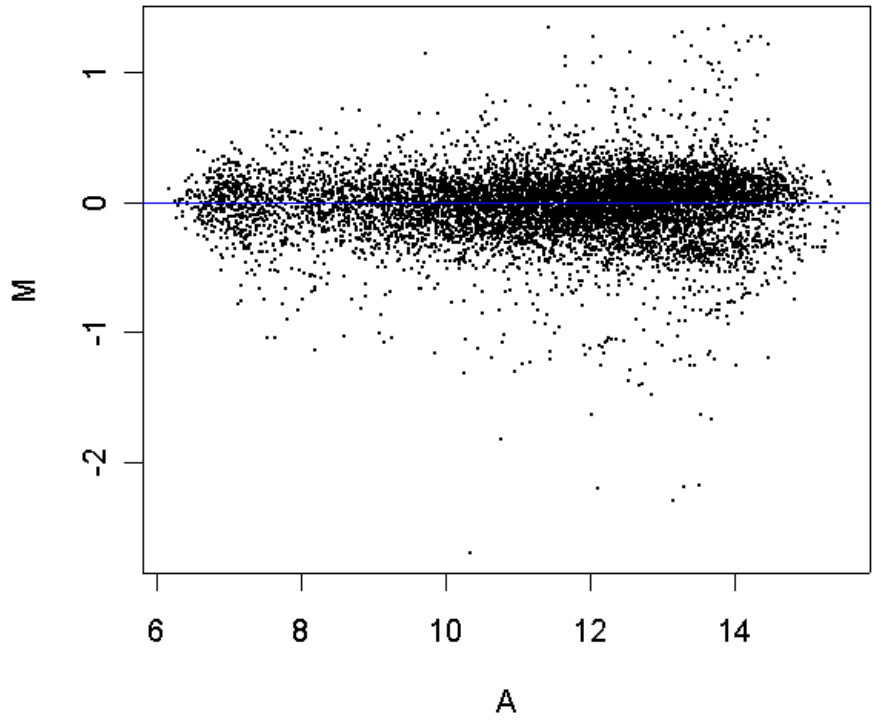


Linear model. Now estimate the average M-value for each gene. We do this by fitting a simple linear model for each gene. The negative numbers in the design matrix indicate the dye-swaps.

```
> design <- c(-1,1,-1,1)
> fit <- lm.series(MA$M,design)
> names(fit)
[1] "coefficients" "stdev.unscaled" "sigma" "df.residual"
```

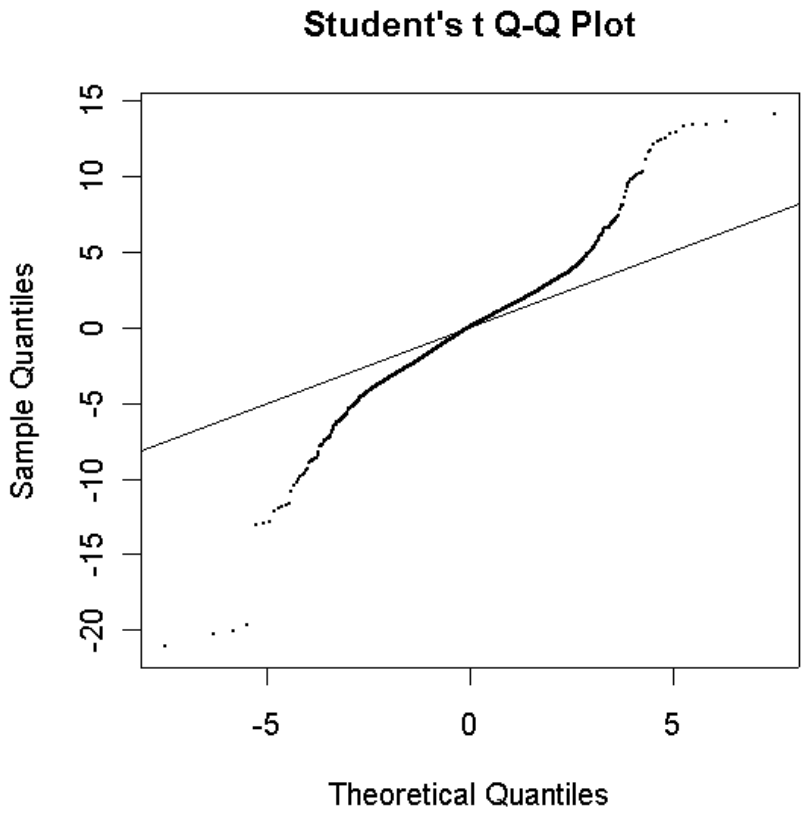
Now create an MA-plot of the average M and A-values for each gene.

```
> M <- fit$coef
> A <- apply(MA$A,1,mean)
> plot(A,M,pch=16,cex=0.1)
> abline(0,0,col="blue")
```



Empirical Bayes analysis. We will now go on and compute empirical Bayes statistics for differential expression. The moderated t-statistics use sample standard deviations which have been shrunk towards a pooled standard deviation value.

```
> eb <- ebayes(fit)
> qqt(eb$t,df=fit$df+eb$df,pch=16,cex=0.1)
> abline(0,1)
```



Visually there seems to be plenty of genes which are differentially expressed. We will obtain a summary table of some key statistics for the top genes.

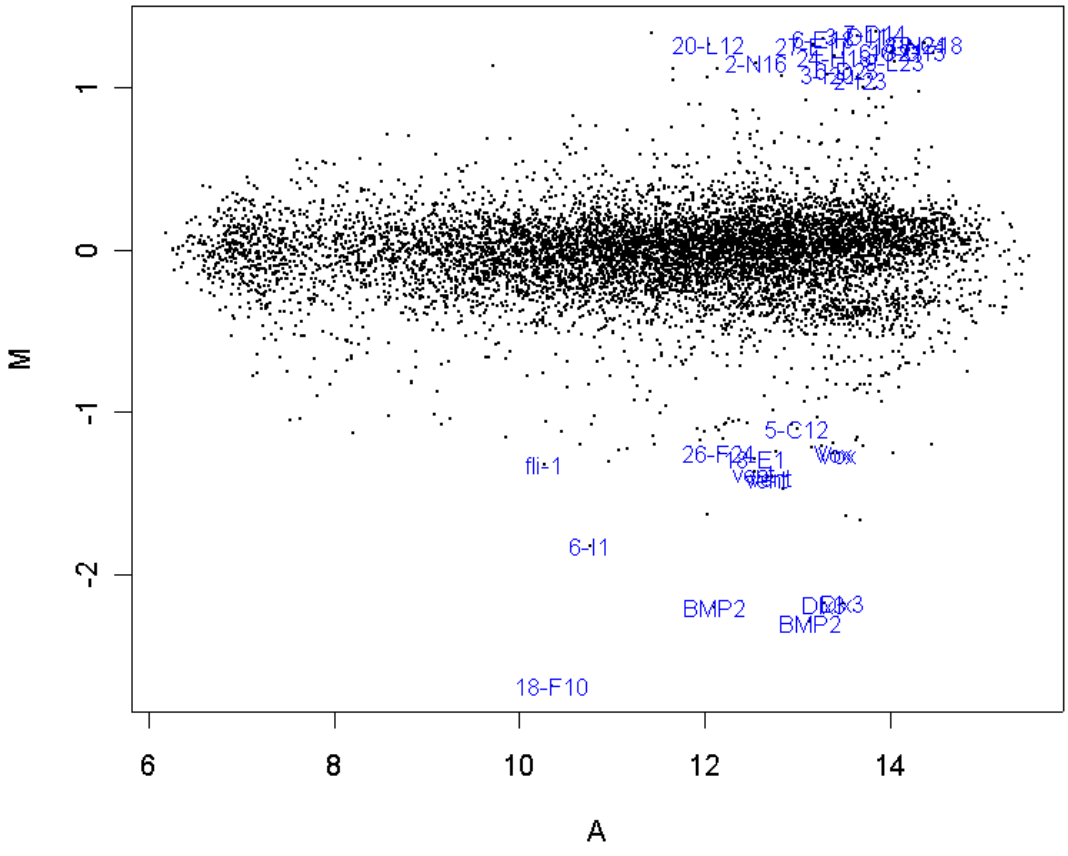
```
> options(digits=3)
> toptable(number=30,genelist=gal,fit=fit,eb=eb,adjust="fdr")
     Block Row Column      ID   Name     M     t  P.Value    B
3721     8   2      1 control   BMP2 -2.21 -21.1 0.000357 7.96
1609     4   2      1 control   BMP2 -2.30 -20.3 0.000357 7.78
3723     8   2      3 control   Dlx3 -2.18 -20.0 0.000357 7.71
1611     4   2      3 control   Dlx3 -2.18 -19.6 0.000357 7.62
8295    16  16     15 fb94h06 20-L12  1.27  14.1 0.002067 5.78
7036    14   8      4 fb40h07  7-D14  1.35  13.5 0.002067 5.54
515      1  22     11 fc22a09 27-E17  1.27  13.4 0.002067 5.48
5075    10  14     11 fb85f09 18-G18  1.28  13.4 0.002067 5.48
7307    14  19     11 fc10h09 24-H18  1.20  13.2 0.002067 5.40
319      1  14      7 fb85a01  18-E1 -1.29 -13.1 0.002067 5.32
2961     6  14      9 fb85d05 18-F10 -2.69 -13.0 0.002067 5.29
4032     8  14     24 fb87d12 18-N24  1.27  12.8 0.002067 5.22
6903    14   2     15 control    Vox -1.26 -12.8 0.002067 5.20
4546     9  14     10 fb85e07 18-G13  1.23  12.8 0.002067 5.18
683      2   7     11 fb37b09  6-E18  1.31  12.4 0.002182 5.02
1697     4   5     17 fb26b10  3-I20  1.09  12.4 0.002182 4.97
7491    15   5      3 fb24g06  3-D11  1.33  12.3 0.002182 4.96
4188     8  21     12 fc18d12 26-F24 -1.25 -12.2 0.002209 4.89
4380     9   7     12 fb37e11  6-G21  1.23  12.0 0.002216 4.80
3726     8   2      6 control  fli-1 -1.32 -11.9 0.002216 4.76
```

```
2679     6   2     15 control    Vox -1.25 -11.9 0.002216 4.71
5931    12   6      3 fb32f06  5-C12 -1.10 -11.7 0.002216 4.63
7602    15   9     18 fb50g12  9-L23  1.16  11.7 0.002216 4.63
2151     5   2     15 control   vent -1.40 -11.7 0.002216 4.62
3790     8   4     22 fb23d08  2-N16  1.16  11.6 0.002221 4.58
7542    15   7      6 fb36g12  6-D23  1.12  11.0 0.003000 4.27
4263     9   2     15 control   vent -1.41 -10.8 0.003326 4.13
6375    13   2     15 control   vent -1.37 -10.5 0.004026 3.91
1146     3   4     18 fb22a12  2-I23  1.05  10.2 0.004242 3.76
157      1   7     13 fb38a01   6-I1 -1.82 -10.2 0.004242 3.75
```

The top gene is BMP2 which is significantly down-regulated in the Swirl zebrafish, as it should be because the Swirl fish are mutant in this gene. Other positive controls also appear in the top 50 genes in terms.

In the table, t is the empirical Bayes moderated t-statistic, the corresponding P-values have been adjusted to control the false discovery rate and B is the empirical Bayes log odds of differential expression. Beware that the Benjamini and Hochberg method used to control the false discovery rate assumes independent statistics which we do not have here (see `help(p.adjust)`).

```
> ord <- order(eb$lods,decreasing=TRUE)
> top30 <- ord[1:30]
> plot(A,M,pch=16,cex=0.1)
> text(A[top30],M[top30],labels=gal[top30,"Name"],cex=0.8,col="blue")
```



9. Two-Sample Experiments

In this section we consider a case study where two RNA sources are compared through a common reference RNA. The analysis of the log-ratios involves a two-sample comparison of means for each gene.

Example. ApoAI Knockout Data

In this example we assume that the data is available as an RG list in the data file ApoAI.RData.

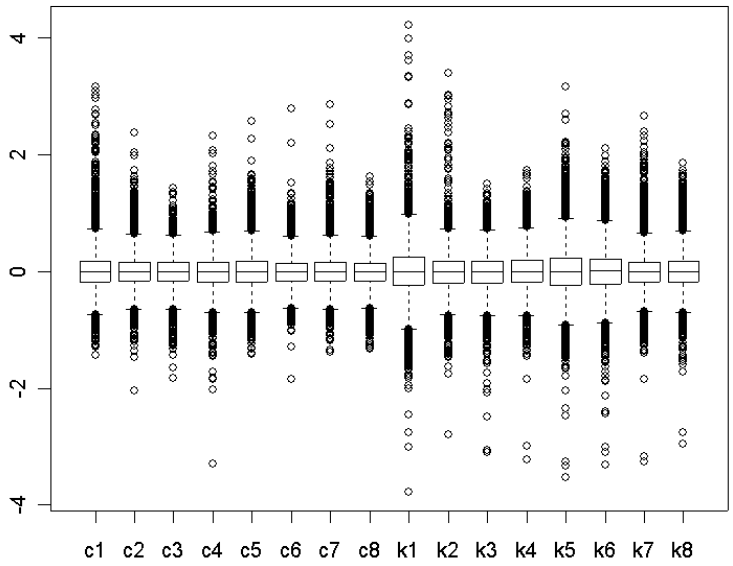
Background. The data is from a study of lipid metabolism by Callow et al (2000). The apolipoprotein AI (ApoAI) gene is known to play a pivotal role in high density lipoprotein (HDL) metabolism. Mouse which have the ApoAI gene knocked out have very low HDL cholesterol levels. The purpose of this experiment is to determine how ApoAI deficiency affects the action of other genes in the liver, with the idea that this will help determine the molecular pathways through which ApoAI operates.

Hybridizations. The experiment compared 8 ApoAI knockout mice with 8 normal C57BL/6 ("black six") mice, the control mice. For each of these 16 mice, target mRNA was obtained from liver tissue and labelled using a Cy5 dye. The RNA from each mouse was hybridized to a separate microarray. Common reference RNA was labelled with Cy3 dye and used for all the arrays. The reference RNA was obtained by pooling RNA extracted from the 8 control mice.

Number of arrays	Red	Green
8	Normal "black six" mice	Pooled reference
8	ApoAI knockout	Pooled reference

This is an example of a single comparison experiment using a common reference. The fact that the comparison is made by way of a common reference rather than directly as for the swirl experiment makes this, for each gene, a two-sample rather than a single-sample setup.

```
> load("ApoAI.RData")
> objects()
[1] "design"   "genelist" "layout"   "RG"
> RG$R[1:4,]
       c1      c2      c3      c4      c5      c6      c7     c8      k1      k2      k3
1 2765.58 1768.22 1440.54  763.06 2027.94  864.05  958.68 644.58  747.11 1388.79 1588.76
2 2868.43 2277.18 1599.92 1238.33 1513.43 1079.33 1228.66 757.33 1930.25 2093.00 1369.81
3 1236.32 1546.84 2639.45  999.48 3689.67 1505.20  785.10 994.86  753.52 1300.00 1301.61
4  383.62  532.50  323.55  585.14  250.74  566.58  409.18 417.79  829.82  402.84  513.91
       k4      k5      k6      k7      k8
1 1280.17 1881.72 1733.53 1170.84 1512.45
2 1071.17 3218.58 2451.04 1605.00 1700.82
3 3292.26 1149.23 3424.30 1901.06 2200.82
4  459.69  391.09  601.00  438.03  507.25
> MA <- normalizeWithinArrays(RG,layout)
> boxplot(MA$M~col(MA$M),names=colnames(RG$R))
```



The differences in scale are moderate, so we won't scale normalize between arrays.

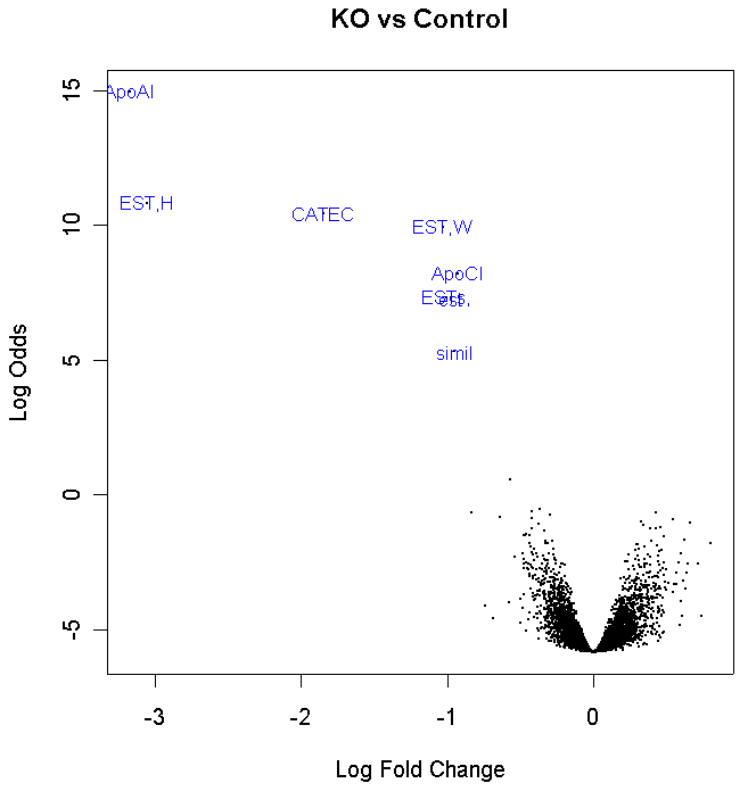
Now we can go on to estimate the fold change between the two groups. In this case the design matrix has two columns. The coefficient for the second column estimates the parameter of interest, the log-ratio between knockout and control mice.

```
> design
   Control-Ref KO-Control
c1           1          0
c2           1          0
c3           1          0
c4           1          0
c5           1          0
c6           1          0
c7           1          0
c8           1          0
k1           1          1
k2           1          1
k3           1          1
k4           1          1
k5           1          1
k6           1          1
k7           1          1
k8           1          1
> fit <- lm.series(MA$M,design)
> fit$coef[1:5,]
     Control-Ref KO-Control
[1,]     -0.6595     0.6393
[2,]      0.2294     0.6552
[3,]     -0.2518     0.3342
[4,]     -0.0517     0.0405
[5,]     -0.2501     0.2230
> eb <- ebayes(fit)
> options(digits=3)
> toptable(coef=2,number=15,genelist=genelist[,1:6],fit=fit,eb=eb,adjust="fdr")
```

```
     GridROW GridCOL ROW COL                 NAME TYPE      M      t  P.Value      B
2149       2       2   8   7      ApoAI,lipid-Img cDNA -3.166 -23.98 3.05e-11 14.927
540        1       2   7  15 EST,HighlysimilartoA cDNA -3.049 -12.96 5.02e-07 10.813
5356       4       2   9   1 CATECHOLO-METHYLTRAN cDNA -1.848 -12.44 6.51e-07 10.448
4139       3       3   8   2 EST,WeaklysimilartoC cDNA -1.027 -11.76 1.21e-06  9.929
1739       2       1   7  17    ApoCIII,lipid-Img cDNA -0.933  -9.84 1.56e-05  8.192
2537       2       3   7  17 ESTs,Highlysimilarto cDNA -1.010  -9.02 4.22e-05  7.305
1496       1       4  15   5                 est cDNA -0.977  -9.00 4.22e-05  7.290
4941       4       1   8   6 similartoyeaststerol cDNA -0.955  -7.44 5.62e-04  5.311
947        1       3   8   2 EST,WeaklysimilartoF cDNA -0.571  -4.55 1.77e-01  0.563
5604       4       3   1  18                     cDNA -0.366  -3.96 5.29e-01 -0.553
4140       3       3   8   3         APXL2,5q-Img cDNA -0.420  -3.93 5.29e-01 -0.619
6073       4       4   5   4          estrogenrec cDNA  0.421   3.91 5.29e-01 -0.652
1337       1       4   7  14 psoriasis-associated cDNA -0.838  -3.89 5.29e-01 -0.687
954        1       3   8   9   Caspase7,heart-Img cDNA -0.302  -3.86 5.30e-01 -0.757
563        1       2   8  17 FATTYACID-BINDINGPRO cDNA -0.637  -3.81 5.30e-01 -0.839
```

Notice that the top gene is ApoAI itself which is heavily down-regulated. Theoretically the M-value should be minus infinity for ApoAI because it is the knockout gene. Several of the other genes are closely related. The top eight genes here were confirmed by independent assay subsequent to the microarray experiment to be differentially expressed in the knockout versus the control line.

```
> plot(fit$coef[,2],eb$lods[,2],pch=16,cex=0.1,
xlab="Log Fold Change",ylab="Log Odds",main="KO vs Control")
> ord <- order(eb$lods[,2],decreasing=TRUE)
> top8 <- ord[1:8]
> text(fit$coef[top8,2],eb$lods[top8,2],
labels=substring(genelist[top8,"NAME"],1,5),cex=0.8,col="blue")
```



10. Factorial Experiments

This case study considers a more involved analysis in which the sources of RNA have a factorial structure.

Example. Weaver Mutant Data

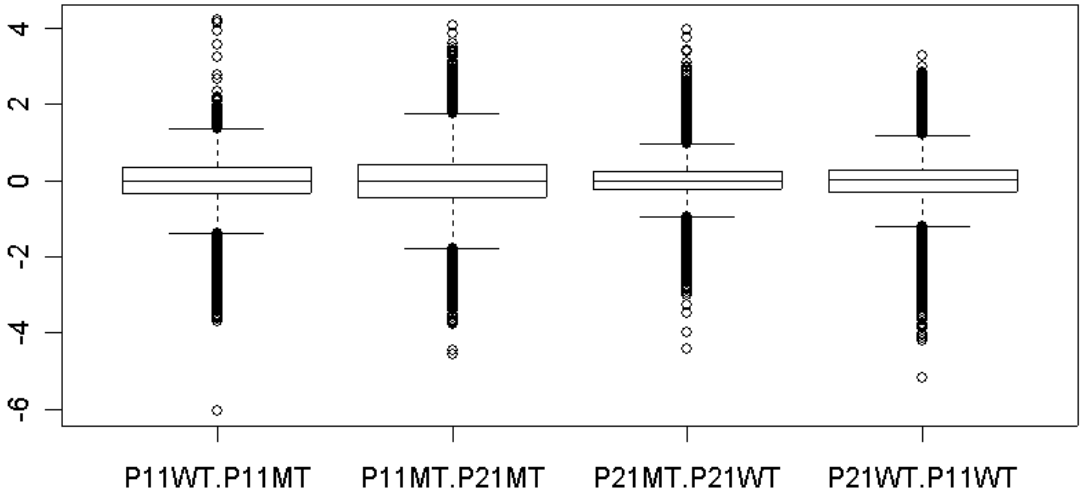
In this example we assume that data is available as an RG list.

Background. This is a case study examining the development of certain neurons in wild-type and weaver mutant mice from Diaz et al (2002). The weaver mutant affects cerebellar granule neurons, the most numerous cell-type in the central nervous system. Weaver mutant mice are characterized by a weaving gait. Granule cells are generated in the first postnatal week in the external granule layer of the cerebellum. In normal mice, the terminally differentiated granule cells migrate to the internal granule layer but in mutant mice the cells die before doing so, meaning that the mutant mice have strongly reduced numbers of cells in the internal granule layer. The expression level of any gene which is specific to mature granule cells, or is expressed in response to granule cell derived signals, is greatly reduced in the mutant mice.

Tissue dissection and RNA preparation. At each time point (P11 = 11 days postnatal and P21 = 21 days postnatal) cerebella were isolated from two wild-type and two mutant littermates and pooled for RNA isolation. RNA was then divided into aliquotes and labelled before hybridizing to the arrays. (This means that different hybridizations are biologically related through using RNA from the same mice, although we will ignore this here. See Yang and Speed (2002) for a detailed discussion of this issue in the context of this experiment.)

Hybridizations. We have just four arrays each comparing two out of the four treatment combinations of time (11 days or 21 days) by genotype (wild-type or mutant). This has the structure of a 2x2 factorial experiment.

```
> objects()
[1] "designIA" "designMt" "gal" "layout" "RG" "Targets"
> Targets
   FileName        Name   Cy5   Cy3
1 cb.1.spot P11WT.P11MT P11WT P11MT
2 cb.2.spot P11MT.P21MT P11MT P21MT
3 cb.3.spot P21MT.P21WT P21MT P21WT
4 cb.4.spot P21WT.P11WT P21WT P11WT
> MA <- normalizeWithinArrays(RG,layout)
> boxplot(MA$M~col(MA$M),names=Targets$Name)
```



First we consider a classical interaction parametrization.

```
> designIA
            TimeWt Mutant11 I/A
P11WT.P11MT      0       -1   0
P11MT.P21MT     -1        0  -1
P21MT.P21WT      0        1   1
P21WT.P11WT      1        0   0
```

TimeWt is late vs early time for the wild-type mice. Mutant11 is mutant vs wild-type at the early time. The third column estimates the interaction between time and genotype.

```
> fitIA <- lm.series(MA$M,designIA)
> ebIA <- ebayes(fitIA)
> options(digits=3)
> toptable(coef="I/A",n=10,genelist=gal,fit=fitIA,eb=ebIA,adjust="fdr")
          ID            Name    M     t P.Value     B
7737    RIKEN           Z6801 6.49 12.95   0.886 -4.03
780     RIKEN            Z636 6.57 12.67   0.886 -4.03
4063    RIKEN           Z3559 6.41 12.37   0.886 -4.03
3627  Control              L1 6.08 11.89   0.886 -4.03
3084    RIKEN           Z2652 4.88  9.38   1.000 -4.04
16230 Control  T7/SP6 7- Vrg2 6.00  9.12   1.000 -4.05
12537   RIKEN          Z11025 5.03  9.03   1.000 -4.05
2866    RIKEN           Z2506 4.19  8.46   1.000 -4.05
11430 Control T7/SP6 5- msx 1 3.31  6.40   1.000 -4.08
15590   RIKEN          Z13718 3.17  5.88   1.000 -4.10
```

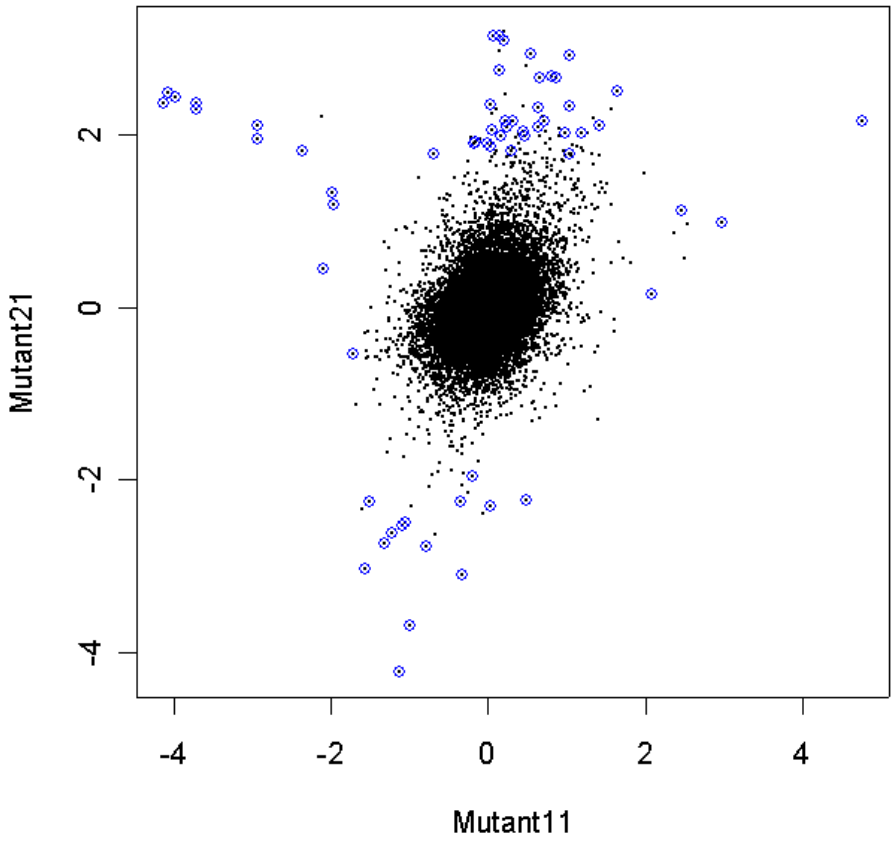
With only four arrays there is only one residual df for the linear model, so even large M-values and t-statistics are not significant after adjusting for multiple testing. There are differentially expressed genes here, although it is difficult to confirm it from the four arrays that we are using for this exercise.

Consider another parametrization.

```
> designMt
            Mutant11 Mutant21 TimeMt
P11WT.P11MT       -1        0      0
P11MT.P21MT        0        0     -1
P21MT.P21WT        0        1      0
P21WT.P11WT        1       -1      1
```

Here Mutant21 is mutant vs wild-type at the later time and TimeMt is late vs early time for the mutant mice.

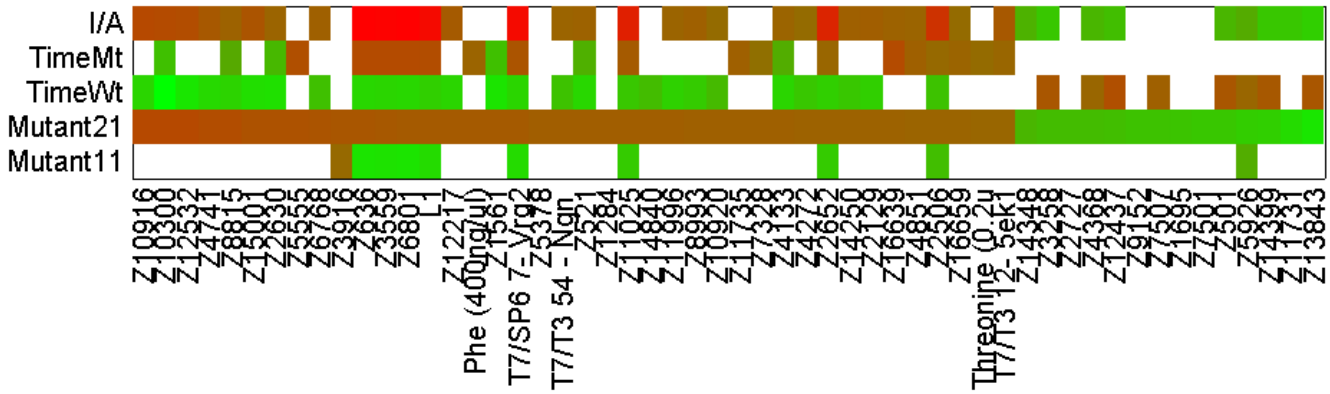
```
> fitMt <- lm.series(MA$M,designMt)
> ebMt <- ebayes(fitMt)
>
plot(fitMt$coef[,"Mutant11"],fitMt$coef[,"Mutant21"],pch=16,cex=0.1,xlab="Mutant
11",ylab="Mutant21")
> sel <- abs(ebMt$t[,"Mutant11"])>4 | abs(ebMt$t[,"Mutant21"])>4
> points(fitMt$coef[sel,"Mutant11"],fitMt$coef[sel,"Mutant21"],col="blue")
```



This scatterplot allows the genes to be visually clustered according to whether they are differentially expressed in the mutant at the two times.

We will now collate the results of the two fits.

```
> fit <- fitIA
> fit$coefficients <- cbind(fitMt$coef,fitIA$coef)
> fit$coefficients <- fit$coef[,c(1,2,4,3,6)]
> fit$coef[1:5,]
     Mutant11 Mutant21  TimeWt TimeMt    I/A
[1,]  -0.5396   0.1670  1.3362  2.043 0.7066
[2,]   0.2481   0.8601 -0.9112 -0.299 0.6120
[3,]  -1.1368  -0.5642 -0.0119  0.561 0.5726
[4,]  -1.0166  -0.5837  0.0837  0.517 0.4329
[5,]   0.0135   0.0614  0.3701  0.418 0.0479
> fit$stdev.unscaled <- cbind(fitMt$std,fitIA$std)
> fit$stdev.unscaled <- fit$std[,c(1,2,4,3,6)]
> fit$std[1:5,]
     Mutant11 Mutant21 TimeWt TimeMt I/A
[1,]    0.866    0.866  0.866  0.866   1
[2,]    0.866    0.866  0.866  0.866   1
[3,]    0.866    0.866  0.866  0.866   1
[4,]    0.866    0.866  0.866  0.866   1
[5,]    0.866    0.866  0.866  0.866   1
> eb <- ebayes(fit)
> heatdiagram(abs(eb$t),fit$coef,"Mutant21",names=gal$Name)
```



This heat diagram shows the expression profiles for all genes judged to be differentially expressed (|t| > 4) with respect to Mutant21. The genes are sorted from left to right in terms of their coefficients for Mutant21, with red meaning up-regulation and green meaning down-regulation. It is especially interesting to see that genes which are up-regulated (red) in the mutant at 21 days are those which have decreasing expression in the wild-type over time, and those which are down-regulated (green) in the mutant are those which increase over time in the wild-type. The mutant is not participating in normal development between 11 and 21 days in respect of these genes.

11. Within-Array Replicate Spots

In this section we consider a case study in which all genes (ESTs and controls) are printed more than once on the array. This means that there is both within-array and between-array replication for each gene. The structure of the experiment is therefore essentially a randomized block experiment for each gene. The approach taken here is to estimate a common correlation for all the genes for between within-array duplicates.

Example. Bob Mutant Data

In this example we assume that the data is available as an RG list.

Background. This data is from a study of transcription factors critical to B cell maturation by Lynn Corcoran and Wendy Dietrich at the WEHI. Mice which have a targeted mutation in the Bob (OBF-1) transcription factor display a number of abnormalities in the B lymphocyte compartment of the immune system. Immature B cells that have emigrated from the bone marrow fail to differentiate into full fledged B cells, resulting in a notable deficit of mature B cells.

Arrays. Arrays were printed with expressed sequence tags (ESTs) from the National Institute of Aging 15k mouse clone library, plus a range of positive, negative and calibration controls. The arrays were printed using a 48 tip print head and 26x26 spots in each tip group. Data from 24 of the tip groups are given here. Every gene (ESTs and controls) was printed twice on each array.

Hybridizations. A retrovirus was used to add Bob back to a Bob deficient cell line. Two RNA sources were compared using 2 dye-swap pairs of microarrays. One RNA source was obtained from the Bob deficient cell line after the retrovirus was used to add GFP ("green fluorescent protein", a neutral protein). The other RNA source was obtained after adding both GFP and Bob protein. RNA from Bob+GFP was labelled with Cy5 in arrays 2 and 4, and with Cy3 in arrays 1 and 4.

```
> objects()
[1] "design" "gal"    "layout" "RG"
> design
[1] -1  1 -1  1
> gal[1:40,]
   Library            Name
1  Control       cDNA1.500
2  Control       cDNA1.500
3  Control Printing.buffer
4  Control Printing.buffer
5  Control Printing.buffer
6  Control Printing.buffer
7  Control Printing.buffer
8  Control Printing.buffer
9  Control       cDNA1.500
10 Control       cDNA1.500
11 Control Printing.buffer
12 Control Printing.buffer
13 Control Printing.buffer
14 Control Printing.buffer
15 Control Printing.buffer
16 Control Printing.buffer
17 Control       cDNA1.500
18 Control       cDNA1.500
19 Control Printing.buffer
20 Control Printing.buffer
21 Control Printing.buffer
22 Control Printing.buffer
23 Control Printing.buffer
24 Control Printing.buffer
25 Control       cDNA1.500
26 Control       cDNA1.500
```

```
27  NIA15k             H31
28  NIA15k             H31
29  NIA15k             H32
30  NIA15k             H32
31  NIA15k             H33
32  NIA15k             H33
33  NIA15k             H34
34  NIA15k             H34
35  NIA15k             H35
36  NIA15k             H35
37  NIA15k             H36
38  NIA15k             H36
39  NIA15k             H37
40  NIA15k             H37
```

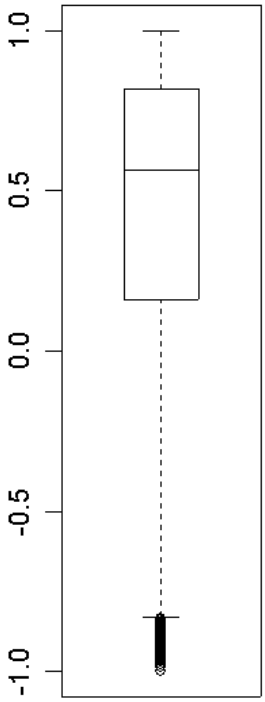
Although there are only four arrays, we have a total of eight spots for each gene, and more for the controls. Naturally the two M-values obtained from duplicate spots on the same array are highly correlated. The problem is how to make use of the duplicate spots in the best way. The approach taken here is to estimate the spatial correlation between the adjacent spots using REML and then to conduct the usual analysis of the arrays using generalized least squares.

First normalize the data using print-tip loess regression.

```
> MA <- normalizeWithinArrays(RG,layout)
```

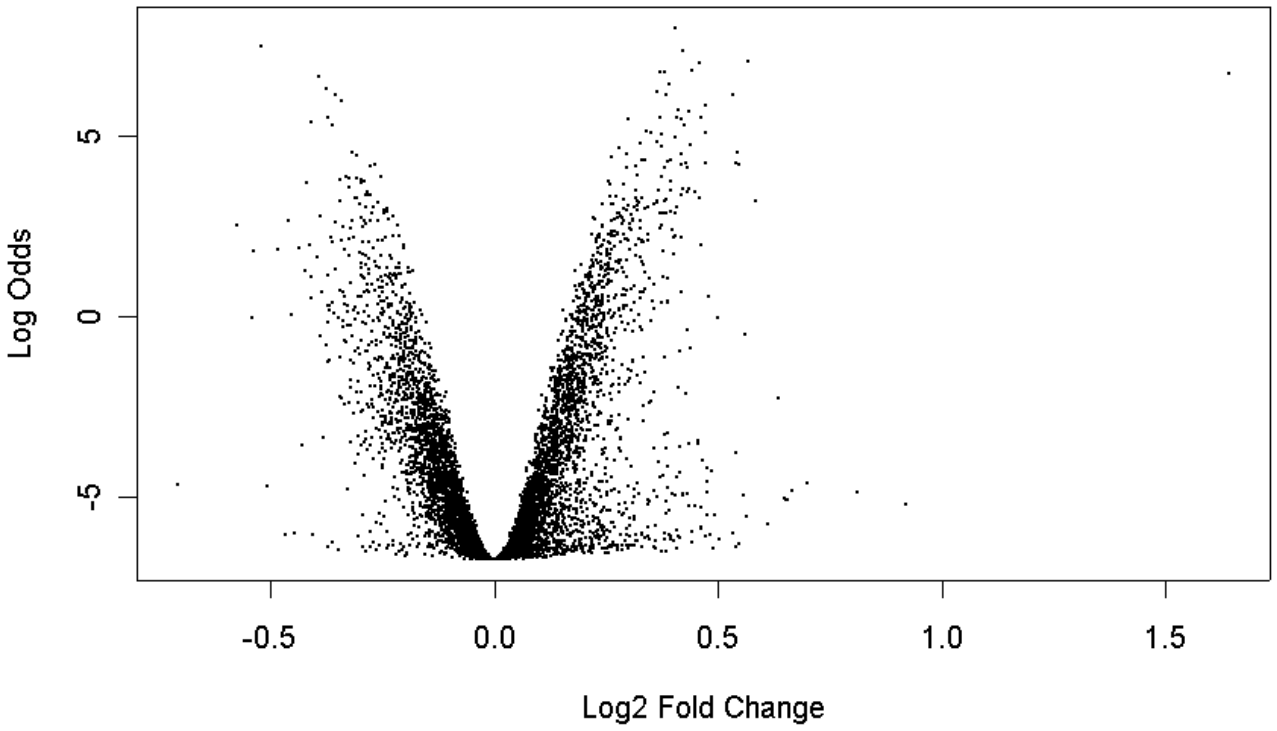
Now estimate the spatial correlation. We estimate a correlation term by REML for each gene, and then take a trimmed mean on the atanh scale to estimate the overall correlation. This command takes a lot of time, perhaps as much as an hour for a series of arrays.

```
> cor <- dupcor.series(MA$M,design,ndups=2) # This is a very slow computation!
> cor$cor
[1] 0.571377
> boxplot(cor$cor.genes)
```




```
> fit <- gls.series(MA$M,design,ndups=2,correlation=0.571377)
> eb <- ebayes(fit)
> genenames <- uniquegenelist(gal[,"Name"],ndups=2)
> toptable(number=40,genelist=genenames,fit=fit,eb=eb,adjust="fdr")
     Name          M          t      P.Value        B
1  H34599  0.4035865  13.053838 0.0004860773 7.995550
2  H31324 -0.5196599 -12.302094 0.0004860773 7.499712
3  H33309  0.4203320  12.089742 0.0004860773 7.352862
4   H3440  0.5678168  11.664229 0.0004860773 7.049065
5  H36795  0.4600335  11.608550 0.0004860773 7.008343
6   H3121  0.4408640  11.362917 0.0004860773 6.825927
7  H36999  0.3806754  11.276571 0.0004860773 6.760715
8   H3132  0.3699805  11.270201 0.0004860773 6.755881
9  H32838  1.6404839  11.213454 0.0004860773 6.712681
10 H36207 -0.3930972 -11.139510 0.0004860773 6.656013
11 H37168  0.3909476  10.839880 0.0005405097 6.421932
12 H31831 -0.3738452 -10.706775 0.0005405097 6.315602
13 H32014  0.3630416  10.574797 0.0005405097 6.208714
14 H34471 -0.3532587 -10.496483 0.0005405097 6.144590
15 H37558  0.5319192  10.493157 0.0005405097 6.141856
16  H3126  0.3849980  10.467091 0.0005405097 6.120389
17 H34360 -0.3409371 -10.308779 0.0005852911 5.988745
18 H36794  0.4716704  10.145670 0.0006399135 5.850807
19  H3329  0.4125222  10.009042 0.0006660758 5.733424
20 H35017  0.4337911   9.935639 0.0006660758 5.669656
21 H32367  0.4092668   9.765338 0.0006660758 5.519781
22 H32678  0.4608290   9.763809 0.0006660758 5.518423
23 H31232 -0.3717084  -9.758581 0.0006660758 5.513778
24  H3111  0.3693533   9.745794 0.0006660758 5.502407
25 H34258  0.2991668   9.722656 0.0006660758 5.481790
```

```
26 H32159  0.4183633   9.702614 0.0006660758 5.463892
27 H33192 -0.4095032  -9.590227 0.0007130533 5.362809
28 H35961 -0.3624470  -9.508868 0.0007205823 5.288871
29 H36025  0.4265827   9.503974 0.0007205823 5.284403
30  H3416  0.3401763   9.316136 0.0008096722 5.111117
31 H33016  0.3519567   9.309343 0.0008096722 5.104784
32 H31404  0.4736604   9.273895 0.0008096722 5.071663
33 H34292  0.3742577   9.249119 0.0008096722 5.048437
34  H3909  0.4035563   9.133708 0.0008799721 4.939425
35  H3418  0.3637057   9.013667 0.0009627391 4.824582
36 H36826  0.3278114   8.981420 0.0009665789 4.793475
37  H3585  0.4387621   8.945039 0.0009753033 4.758249
38 H35907  0.3744189   8.902819 0.0009907401 4.717193
39 H36186  0.2781371   8.830103 0.0010388096 4.646038
40 H34801 -0.3189120  -8.704827 0.0011000262 4.522116
> plot(fit$coef,eb$lods,xlab="Log2 Fold Change",ylab="Log Odds",pch=16,cex=0.1)
```



12. Using limma with the marray Packages

The packages marrayClasses, marrayInput, marrayNorm and marrayTools are designed to read and normalize cDNA data. Normalization will produce an object of class `marrayNorm`. The linear model commands from limma may be used after extracting the M-value matrix and the spot weights from the object. If the normalized data object is called `N`, then a linear model may be fitted using

```
fit <- lm.series(maM(N),design,weights=maW(N))
```

after which one proceeds exactly as in previous sections. The functions `gls.series` or `rlm.series` may be substituted for `lm.series` as appropriate. The design matrix is chosen as in previous sections. Alternatively you can use the function `lmFit` which will accept a marrayNorm object directly, e.g..,

```
fit <- lmFit(N,design)
```

13. Affymetrix and Single-Color Arrays

Suppose for example that there are three arrays hybridized with wt RNA and two arrays hybridized with mutant RNA. The design matrix might be

```
> design
    Wt  Mutant
1    1       0
2    1       0
3    1       0
4    1       1
5    1       1
```

Normalization of Affymetrix data using functions in the package `affy` will produce an object of class `exprSet` or of `AffyBatch` which inherits from `exprSet`. Let `E` be the `exprSet` object. A linear model may be fitted using

```
fit <- lm.series(exprs(E),design)
eb <- ebayes(fit)
```

Then one may proceed as for two-sample two color experiments. The second coefficient in the linear model measures the difference between mutant and wt expression. The function `rlm.series` may be substituted for `lm.series` if a robust fit is desired.

As a second example, suppose that there are three RNA sources to be compared. Suppose that the first three arrays are hybridized with RNA1, the next two with RNA2 and the next three with RNA3. Suppose that all pair-wise comparisons between the RNA sources are of interest. We assume that the data has been normalized and stored in an `exprSet` object, for example by

```
> data <- ReadAffy()
> data2<- rma(data)
```

An appropriate design matrix can be created and a linear model fitted using

```
> design <- model.matrix(~ -1+factor(c(1,1,1,2,2,3,3,3)))
> colnames(design) <- c("group1", "group2", "group3")
> fit <- lm.series(exprs(data2), design)
```

and To make all pairwise comparisons between the three groups the appropriate contrasts can be created by

```
> contrast.matrix <- cbind(group2vs1=c(-1,1,0), group3vs2=c(0,-1,1),
group3vs1=c(-1,0,1))
```

```
> fit2 <- contrasts.fit(fit, contrast.matrix)
> eb <- ebayes(fit2)
```

A list of top genes differential expressed in group2 versus group1 can be obtained from

```
> toptable(coef="group2vs1", fit=fit2, eb=eb, genelist=geneNames(data2),
adjust="fdr")
```

You may classify each gene according to the three pair-wise comparisons using

```
> clas <- classifyTests(eb$t, design=design, contrasts=contrast.matrix)
```

14. Single-Channel Normalization for Two-Color Arrays

We provide a short background on the topic of single-channel normalization for two color arrays. Throughout this section the ApoAI data set will be used to demonstrate single-channel normalization.

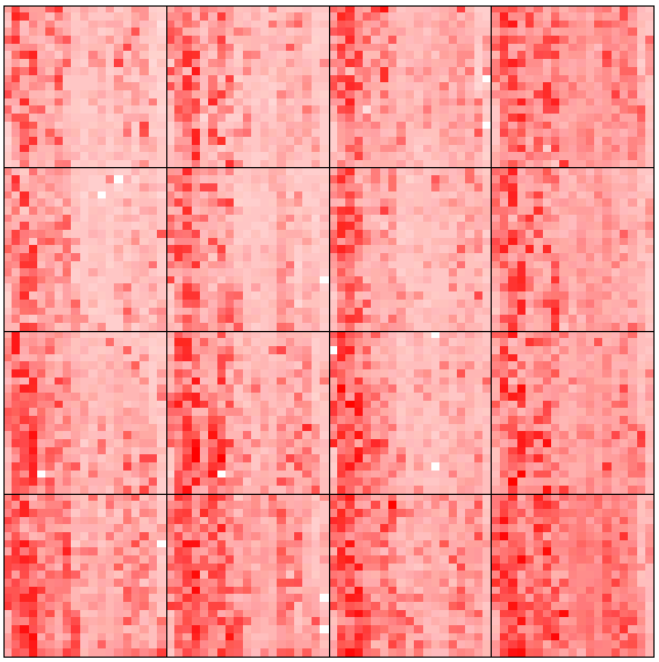
Load the ApoAI data and perform background correction on the `RGlist` data object.

```
> load("ApoAI.RData")
> RG.b <-backgroundCorrect(RG,method="minimum")
```

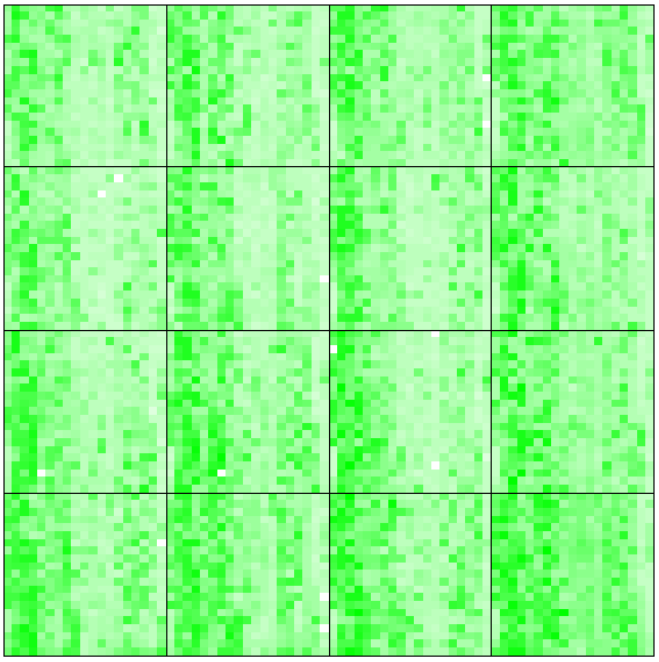
cDNA (or oligo) microarrays compare the gene expression between two different sources of RNA for thousands of genes simultaneously. In general, the log-ratio of spot intensities for the red and green channels form the primary data used for downstream analysis. Thus traditional normalization methods, which remove systematic variation in microarray data, focus on adjusting the log-ratios within each slide. However sometimes it is desirable to work with single-channel (log-intensity) data rather than the log-ratios and so new techniques for normalizing such single-channel data have been investigated. In the current literature there has been limited attention given to single-channel normalization despite many groups basing their entire analyses on single channel data. Single-channel data display a higher level of systematic variation than that observed in log-ratio data.

For example below are `imageplots` of the log-intensity single-channels and the log-ratio for a single array from the ApoAI data set. (The `imageplots` below are based on non-normalised background corrected data). Clearly some of the systematic spatial variation is cancelled out by forming the log-ratio. This is just a simple demonstration of how M-values are less noisy than single-channels.

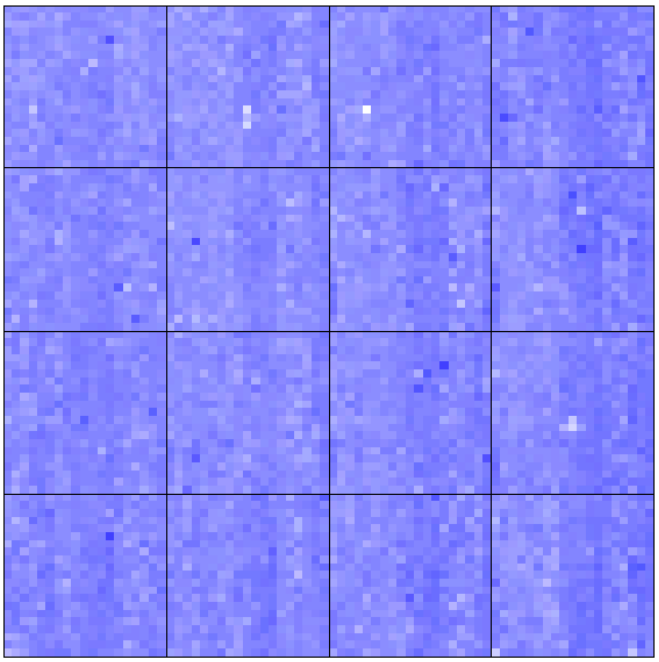
```
> imageplot(log(RG.b$R[,4],2), layout, low="white", high="red")
```



```
> imageplot(log(RG.b$G[,4],2), layout, low="white", high="green")
```



```
> imageplot(log(MA.n$M[,4],2), layout, low="white", high="blue")
```



It should be noted that analysing log-ratios corresponds to doing all analysis on the basis of within-array contrasts while the single-channel approach gives the possibility of recovering information from the between-array variation. This should only be considered after careful single- channel normalization to remove uncontrolled systematic effects at the array level. Yang and Thorne (2003) provides an outline of the motivations for performing single-channel (log-intensity) analysis. We currently perform single-channel normalization using a quantile method based on Bolstad *et al.*'s quantile normalization of high density oligonucleotide data). In the following we demonstrate within-slide and between-slide single-channel normalization routines. We use the ApoAI data set to illustrate the methods.

We perform the normalization of single-channel data using methods in the `normalizeWithinArrays` and `normalizeBetweenArrays` functions.

Note that `RG.b` contains unlogged single-channel intensities and `normalizeWithinArrays` expects its input `RGlist` to be unlogged. There is an argument `log.transform=F` which needs to be implemented if the `RGlist` supplied is already logged. The following command creates an `MAList` containing non-normalized background corrected values.

```
> MA.n <-normalizeWithinArrays(RG.b,layout,method="n")
```

Next we normalize the Mvalues via the default within array normalization of `printtiploess` (we could have use the method `loess` instead, but we find that `printtiploess` is often a good choice since it acts as a proxy for spatial normalization of the Mvalues.

```
> MA.p <-normalizeWithinArrays(RG.b,layout)
```

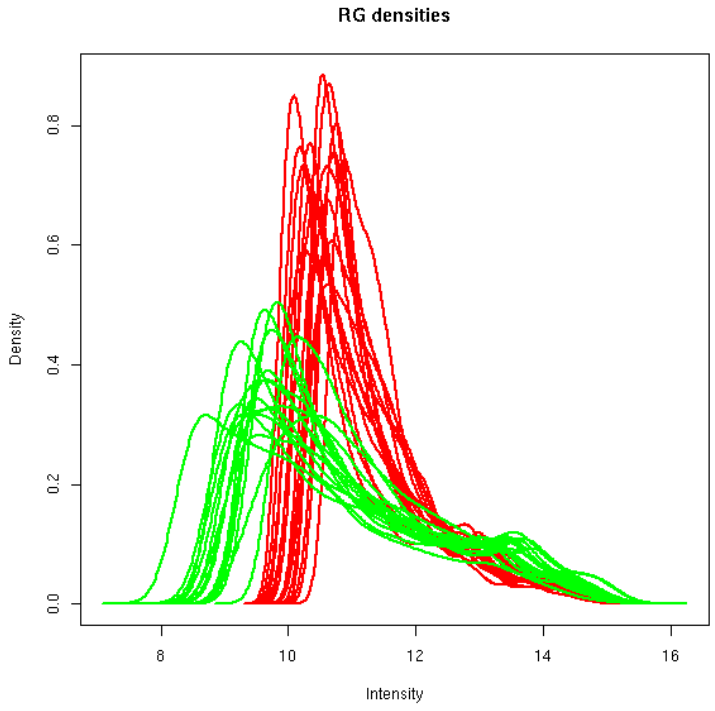
At any stage we can recover the `RGList` of normalized single-channels using `RG.MA`. `RG.MA(MA.p)` would give us within-array only normalized single-channels. Next we perform between array

normalization of the single-channels. We use the function `normalizeBetweenArrays` which takes and returns an `MAList`. `normalizeBetweenArrays` forms an RG matrix when implementing the `quantile` normalization method on the single-channels; and although it returns an `MAList` the single-channel normalised values can be obtained by using the function `RG.MA`. We show how to implement the following between array normalization methods respectively, quantile normalization between all single-channels only (**q**); quantile normalization after printtiploess normalization within arrays (**pq**); quantile normalization between the arrays on the `Aq` values which is then combined with the within array printtiploess normalization `Mp` to give `MpAq` . Notice that for `MpAq` we have mixed and matched different within and between array normlizations to create a *simultaneous within and between array* single-channel normalization method.

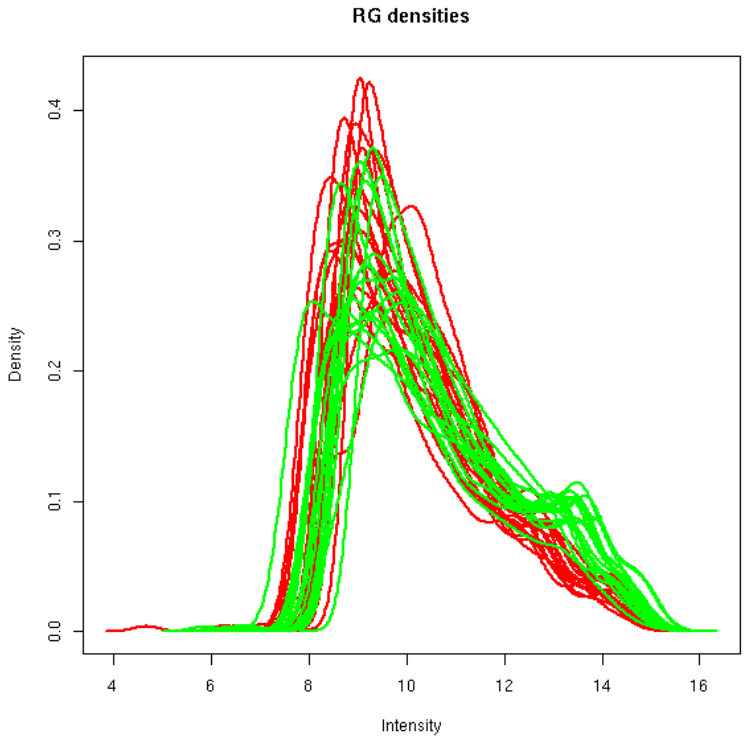
```
> MA.q <- normalizeBetweenArrays(MA.n, method="quantile")
> MA.pq <- normalizeBetweenArrays(MA.p, method="quantile")
> MA.Aq <- normalizeBetweenArrays(MA.n, method="Aquantile")
> MA.MpAq <- new("MAList", list(M=MA.p$M, A=MA.Aq$A))
```

We find that **pq** and **MpAq** work quite well. Next we show some plots of the single-channel log-intensity densities which illustrate the results of the different single-channel normalization methods. We use the function `plotDensities` which will take either an `RGList` or an `MAList`. The form al the call is: `plotDensities(object, log.transform = FALSE, arrays = NULL, singlechannels = NULL, groups = NULL, col = NULL)`. The default usage of plotDensities results in red/green coloring of the densities. Without any background correction there is a significant difference between the red and green single-channel intensity distributions:

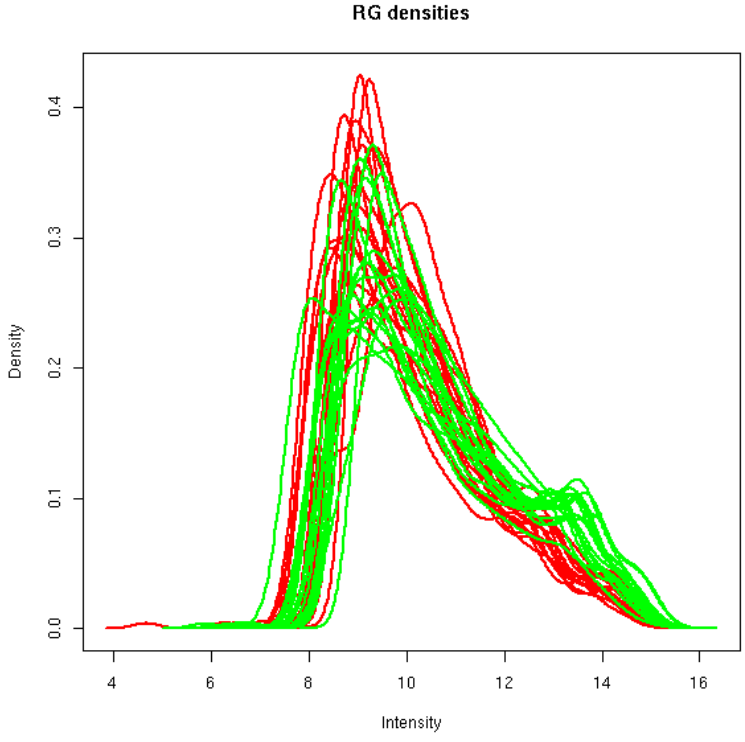
```
> plotDensities(RG, log.transform=TRUE)
```




```
> plotDensities(RG.b, log.transform=TRUE)
```

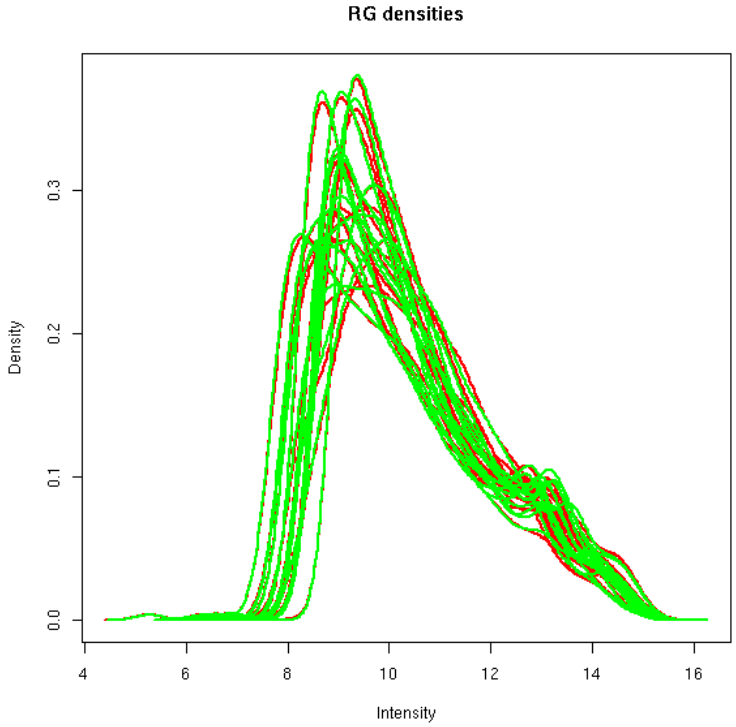



```
> plotDensities(MA.n)
```



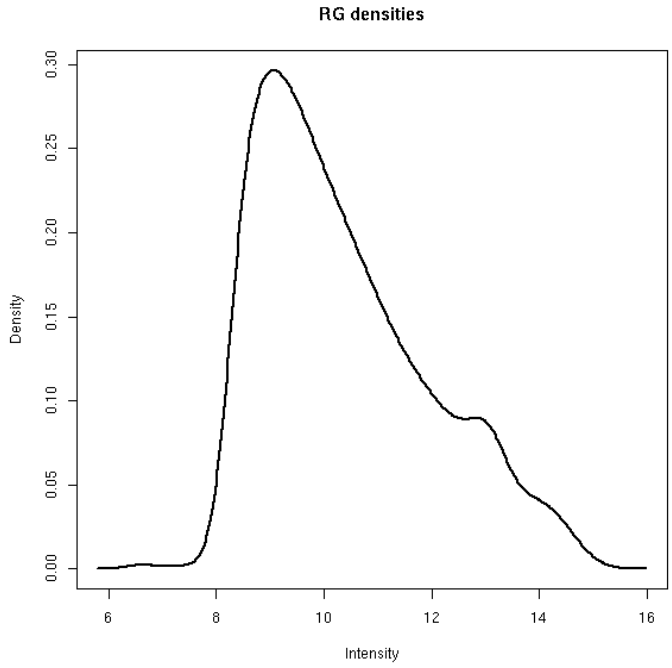
`Printtiploess` makes the single-channels within arrays similar:

```
> plotDensities(MA.p)
```

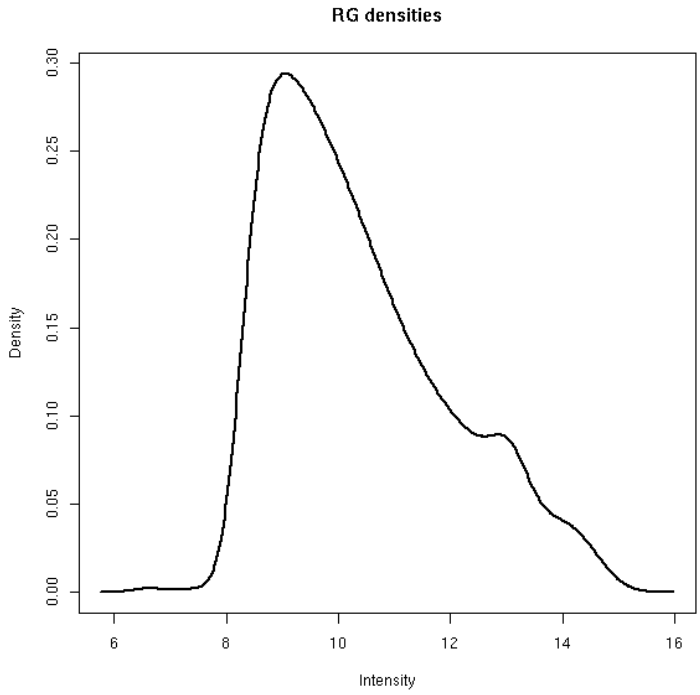


All the single-channels have the same distribution.

```
> plotDensities(MA.q, col="black")
```

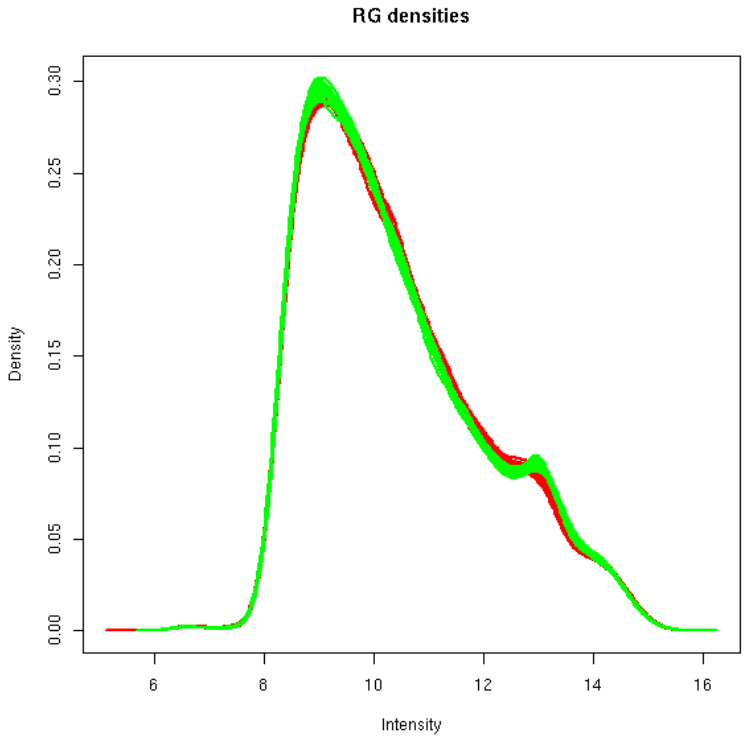



```
> plotDensities(MA.pq, col="black")
```



MpAq gives very similar results as **pq**.

```
> plotDensities(MA.MpAq)
```



Acknowledgements

Thanks to Yee Hwa Yang and Sandrine Dudoit for the first three data sets. The Swirl zebrafish data were provided by Katrin Wuennenburg-Stapleton from the Ngai Lab at UC Berkeley. Thanks to Lynn Corcoran for the Bob Mutant data.

References

1. Callow, M. J., Dudoit, S., Gong, E. L., Speed, T. P., and Rubin, E. M. (2000). Microarray expression profiling identifies genes with altered expression in HDL deficient mice. *Genome Research* **10**, 2022-2029. (Full Text)
2. Diaz, E., Ge, Y., Yang, Y. H., Loh, K. C., Serafini, T. A., Okazaki, Y, Hayashizaki, Y, Speed, T. P., Ngai, J., Scheiffele, P. (2002). Molecular analysis of gene expression in the developing pontocerebellar projection system. *Neuron* **36**, 417-434. (Full Text)
3. Smyth, G. K. (2003). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. (PDF)
4. Smyth, G. K., and Speed, T. P. (2003). Normalization of cDNA microarray data. In: *METHODS: Selecting Candidate Genes from DNA Array Screens: Application to Neuroscience*, D. Carter (ed.). To appear. (PDF)
5. Smyth, G. K., Yang, Y.-H., Speed, T. P. (2003). Statistical issues in microarray data analysis. In: *Functional Genomics: Methods and Protocols*, M. J. Brownstein and A. B. Khodursky (eds.), Methods in Molecular Biology Volume 224, Humana Press, Totowa, NJ, pages 111-136. (PDF)
6. Yang, Y. H., and Speed, T. P. (2002). Design and analysis of comparative microarray experiments. In T. P. Speed (ed.), *Statistical Analysis of Gene Expression Microarray Data*. CRC Press.
7. Yang, Y. H., and Speed, T. P. (2003). Design and analysis of comparative microarray experiments. In T. P. Speed (ed.), *Statistical Analysis of Gene Expression Microarray Data*. Chapman & Hall/CRC Press, pages 35-91.
8. Yang, Y. H., and Thorne, N. P. (2003). Normalization for two-color cDNA microarray data. In: D. R. Goldstein (ed.), *Science and Statistics: A Festschrift for Terry Speed*, IMS Lecture Notes - Monograph Series, Volume 40, pp. 403-418.

Appendix: Conventions

The use of periods "." in function names indicate the type of argument that the function takes. For extension ".series" as in `lm.series` or `gls.series`, indicates that the function operates on data from a series of microarrays, usually represented by a matrix of log-ratios in which columns correspond to arrays.

The online documentation uses conventions suggested by Writing .Rd Files For S4 Classes, Generic Functions and Methods.