limma:
Linear Models for Microarray Data
User's Guide

Gordon K. Smyth, Matthew Ritchie, Natalie Thorne and James Wettenhall
The Walter and Eliza Hall Institute of Medical Research
Melbourne, Australia

20 April 2006

This free open-source software implements academic research by the authors and co-workers. If you use it, please support the project by citing the appropriate journal articles listed in Section 2.1.

Contents

1 Introduction 3

2 Preliminaries 4
2.1 Citing limma 4
2.2 Installation 5
2.3 How to get help 5

3 Quick Start 7
3.1 A brief introduction to R 7
3.2 Sample limma Session 8
3.3 Data Objects 9

4 Reading Two-Color Data 11
4.1 Scope of this Chapter 11
4.2 Recommended Files 11
4.3 The Targets Frame 11
4.4 Reading in Intensity Data 13
4.5 Spot Quality Weights 14
4.6 Reading the Gene List 15
4.7 Printer Layout 16
4.8 The Spot Types File 16

5 Data Exploration 19

6 Pre-Processing Two-Color Data 20
6.1 Background Correction 20
6.2 Within-Array Normalization 22
6.3 Between-Array Normalization 24
6.4 Using Objects from the marray Package 27

7 Linear Models Overview 28
7.1 Introduction 28
7.2 Affymetrix and Other Single-Channel Designs 29
7.3 Common Reference Designs 30

7.4 Direct Two-Color Designs 31

8 Specific Designs **33**
8.1 Simple Comparisons 33
8.1.1 Replicate Arrays 33
8.1.2 Dye Swaps 33
8.2 Technical Replication 34
8.3 Paired Samples 38
8.4 Two Groups: Common Reference 38
8.5 Two Groups: Affymetrix 41
8.6 Several Groups 42
8.7 Factorial Designs 43
8.8 Time Course Experiments 46

9 Separate Channel Analysis of Two-Color Data **48**

10 Statistics for Differential Expression **50**
10.1 Output from eBayes 50
10.2 Adjusting for Multiple Testing 51

11 Case Studies **53**
11.1 Swirl Zebrafish: A Single-Sample Experiment 53
11.2 ApoAI Knockout Data: A Two-Sample Experiment 64
11.3 Ecoli Lrp Data: Affymetrix Data with Two Targets 67
11.4 Estrogen Data: A 2x2 Factorial Experiment with Affymetrix Arrays 70
11.5 Weaver Mutant Data: A 2x2 Factorial Experiment with Two-Color Data . . . 74
11.6 Bob Mutant Data: Within-Array Replicate Spots 79

Chapter 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. It has features which make the analyses stable even for experiments with small number of arrays—this is achieved by borrowing information across genes. The normalization and exploratory data analysis functions are for two-colour spotted microarrays. The linear model and differential expression functions apply to all microarrays including Affymetrix and other single-channel microarray experiments.

This guide gives a tutorial-style introduction to the main limma features but does not describe every feature of the package. A full description of the package is given by the individual function help documents available from the R online help system. To access the online help, type `help(package=limma)` at the R prompt or else start the html help system using `help.start()` or the Windows drop-down help menu.

The Bioconductor package marray provides alternative functions for reading and normalizing spotted microarray data. The marray package provides flexible location and scale normalization routines for log-ratios from two-color arrays. The limma package overlaps with marray in functionality but is based on a more general separation between within-array and between-array normalization. If you are using limma in conjunction with marray, see Section 6.4. The Bioconductor package affy provides functions for reading and normalizing Affymetrix microarray data. Advice on how to use limma with the affy package is given throughout the User's Guide, see for example Section 7.2 and the *E. coli* and estrogen case studies.

This guide describes limma as a command-driven package. Packages limmaGUI and affylmGUI are also available which provides graphical user interfaces to the most commonly used functions in limma [25]. Both packages are available from Bioconductor or from `http://bioinf.wehi.edu.au/limmaGUI`. The package limmaGUI is for use with two-color data while afflmGUI is for Affymetrix data.

This user's guide was prepared using R Version 2.2.0 for Windows and limma version 2.4.1. The limma homepage is `http://bioinf.wehi.edu.au/limma`.

Chapter 2

Preliminaries

2.1 Citing limma

Limma is an implementation of a body of methodological research by the authors and co-workers. To give fair professional credit, please be careful to cite the appropriate methodological papers whenever you use results from the limma software in a publication. Such citations are the main means by which the authors receive professional credit for their work.

If you use limma for differential expression analysis, please cite:

> Smyth, G. K. (2004). Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, Vol. 3, No. 1, Article 3.

This article describes the linear modeling approach implemented by `lmFit` and the empirical Bayes statistics implemented by `eBayes`, `topTable` etc.

If you use the `duplicateCorrection` function to handle duplicate spots or technical replication, please cite

> Smyth, G. K., Michaud, J., and Scott, H. (2005). The use of within-array replicate spots for assessing differential expression in microarray experiments. *Bioinformatics* **21**(9), 2067–2075.

If you use limma for pre-processing or normalization of two-color microarray data, please cite:

> Smyth, G. K., and Speed, T. P. (2003). Normalization of cDNA microarray data. *Methods* **31**, 265–273.

This article describes the functions `read.maimages`, `normalizeWithinArrays`, `normalizeBetweenArrays` etc, including the use of spot quality weights.

The limma software itself can be cited as:

> Smyth, G. K. (2005). Limma: linear models for microarray data. In: *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds.), Springer, New York, pages 397–420.

This article describes the software package in the context of the Bioconductor project and surveys the range of experimental designs for which the package can be used, including spot-specific dye-effects. The pre-processing capabilities of the package are also described but more briefly, with examples of background correction, spot quality weights and filtering with control spots. This article is also the best current reference for the `normexp` background correction method.

Finally, if you are using one of the menu-driven interfaces to the software, please cite the appropriate one of

> Wettenhall, J. M., and Smyth, G. K. (2004). limmaGUI: a graphical user interface for linear modeling of microarray data. *Bioinformatics*, **20**, 3705–3706.
>
> Wettenhall, J. M., Simpson, K. M., Satterley, K., and Smyth, G. K. (2006). affylmGUI: a graphical user interface for linear modeling of single channel microarray data. *Bioinformatics*. Advance online publication 2 February 2006.

2.2 Installation

Limma is a package for the R computing environment and it is assumed that you have already installed R. See the R project at `http://www.r-project.org`.

Installing from CRAN. Limma is available as a contributed package from the R Project CRAN site. If you are using R on a system with a suitable internet connection and with installation privileges on your computer, you should be able to install it via

```
> install.packages("limma")
```

at the R prompt from an internet-connected computer. If you are using Windows, use the drop-down menu Packages ≻ Install package(s) from CRAN ....

Installing from Bioconductor. Limma is available as part of the Bioconductor project at `http://www.bioconductor.org`. Bioconductor works on a 6-monthly official release cycle, lagging each major R release by a few weeks. This means that Bioconductor software is updated only once every six months, unless you are using the developmental version of R. Updates of limma between the Bioconductor official releases can be obtained from CRAN.

Change-log. Limma is updated frequently, often a couple of times a week. Once you have installed limma, the change-log can also be viewed from the R prompt. To see the most recent 20 lines type:

```
> changeLog(n=20)
```

2.3 How to get help

Most questions about limma will hopefully be answered by the documentation or references. If you've run into a question which isn't addressed by the documentation, or you've found a conflict between the documentation and software itself, then there is an active support community which can offer help.

The authors of the package always appreciate receiving reports of bugs in the package functions or in the documentation. The same goes for well-considered suggestions for improvements.

Any other questions or problems concerning limma should be sent to the Bioconductor mailing list `bioconductor@stat.math.ethz.ch`. To subscribe to the mailing list, see `https://stat.ethz.ch/mailman/listinfo/bioconductor`. Please send requests for general assistance and advice to the mailing list rather than to the individual authors. Users posting to the mailing list for the first time should read the helpful posting guide at `http://www.bioconductor.org/doc/postingGuide.html`. Note that each function in limma has it's own online help page, as described in the next section. Mailing list etiquette requires that you read the relevant help page carefully before posting a problem to the list.

Chapter 3

Quick Start

3.1 A brief introduction to 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 using a mouse. In this guide it will be assumed that you have successfully downloaded and installed R from `http://www.r-project.org`. A good way to get started is to type

```
> help.start()
```

at the R prompt or, if you're using R for Windows, to follow the drop-down menu items Help ≻ Html help. Following the links Packages ≻ limma from the html help page will lead you to the contents page of help topics for functions 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
```

or equivalently

```
> help("read.maimages")
```

for detailed help on the `read.maimages` function. The individual function help pages are especially important for listing all the arguments which a function will accept and what values the arguments can take.

A key to understanding R is to appreciate that 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. For more background on using R for statistical analyses see [4].

3.2 Sample limma Session

This is a quick overview of what an analysis might look like. The first example assumes four replicate two-color arrays, the second and fourth of which are dye-swapped. We assume that the images have been analyzed using GenePix to produce a `.gpr` file for each array and that a targets file `targets.txt` has been prepared with a column containing the names of the `.gpr` files.

```
> library(limma)
> targets <- readTargets("targets.txt")
```

Set up a filter so that any spot with a flag of -99 or less gets zero weight.

```
> f <- function(x) as.numeric(x$Flags > -99)
```

Read in the data.

```
> RG <- read.maimages(targets$FileName, source="genepix", wt.fun=f)
```

The following command implements a type of adaptive background correction. This is optional but recommended for GenePix data.

```
> RG <- backgroundCorrect(RG, method="rma")
```

Print-tip loess normalization:

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

Estimate the fold changes and standard errors by fitting a linear model for each gene. The design matrix indicates which arrays are dye-swaps.

```
> fit <- lmFit(MA, design=c(-1,1,-1,1))
```

Apply empirical Bayes smoothing to the standard errors.

```
> fit <- eBayes(fit)
```

Show statistics for the top 10 genes.

```
> topTable(fit)
```

The second example assumes Affymetrix arrays hybridized with either wild-type (wt) or mutant (mt) RNA. There should be three or more arrays in total to ensure some replication. The targets file is now assumed to have another column `Genotype` indicating which RNA source was hybridized on each array.

```
> library(affy)
> library(limma)
> targets <- readTargets("targets.txt")
```

Read and pre-process the Affymetrix CEL file data.

```
> ab <- ReadAffy(filenames=targets$FileName)
> eset <- rma(ab)
```

Form an appropriate design matrix for the two RNA sources and fit linear models. The design matrix has two columns. The first represents log-expression in the wild-type and the second represents the log-ratio between the mutant and wild-type samples. See Section 8.5 for more details on the design matrix.

```
> design <- cbind(WT=1, MUvsWT=targets$Genotype=="mu")
> fit <- lmFit(eset, design)
> fit <- eBayes(fit)
> topTable(fit, coef="MUvsWT")
```

This code fits the linear model, smooths the standard errors and displays the top 10 genes for the mutant versus wild-type comparison.

3.3 Data Objects

There are four main types of data objects created and used in limma:

`RGList`. Red-Green list. A class used to store raw intensities as they are read in from an image analysis output file, usually by `read.maimages()`.

`MAList`. Intensities converted to M-values and A-values, i.e., to within-spot and whole-spot contrasts on the log-scale. Usually created from an `RGList` using `MA.RG()` or `normalizeWithinArrays()`. Objects of this class contain one row for each spot. There may be more than one spot and therefore more than one row for each probe.

`MArrayLM`. Store the result of fitting gene-wise linear models to the normalized intensities or log-ratios. Usually created by `lmFit()`. Objects of this class normally contain one row for each unique probe.

`TestResults`. Store the results of testing a set of contrasts equal to zero for each probe. Usually created by `decideTests()`. Objects of this class normally contain one row for each unique probe.

For those who are familiar with matrices in R, all these objects are designed to obey many analogies with matrices. In the case of `RGList` and `MAList`, rows correspond to spots and columns to arrays. In the case of `MarrayLM`, rows correspond to unique probes and columns to parameters or contrasts. The functions `summary`, `dim`, `length`, `ncol`, `nrow`, `dimnames`, `rownames`, `colnames` have methods for these classes. For example

```
> dim(RG)
```

```
[1] 11088 4
```

shows that the `RGList` object `RG` contains data for 11088 spots and 4 arrays.

```
> colnames(RG)
```

will give the names of the filenames or arrays in the object, while if `fit` is an `MArrayLM` object then

```
> colnames(fit)
```

would give the names of the coefficients in the linear model fit.

Objects of any of these classes may be subsetted, so that `RG[,j]` means the data for array `j` and `RG[i,]` means the data for probes indicated by the index `i`. Multiple data objects may be combined using `cbind`, `rbind` or `merge`. Hence

```
> RG1 <- read.maimages(files[1:2], source="genepix")
> RG2 <- read.maimages(files[3:5], source="genepix")
> RG <- cbind(RG1, RG2)
```

is equivalent to

```
> RG <- read.maimages(files[1:5], source="genepix")
```

Alternatively, if control status has been set in the an `MAList` object then

```
> i <- MA$genes$Status=="Gene"
> MA[i,]
```

might be used to eliminate control spots from the data object prior to fitting a linear model.

Chapter 4

Reading Two-Color Data

4.1 Scope of this Chapter

This chapter is for two-color arrays. If you are using Affymetrix arrays, you should use the affy or affyPLM packages to read and normalize the data. If you have single channel arrays other than Affymetrix, you will need to the read the intensity data into your R session yourself using the basic R read functions such as `read.table`. You will need to create a matrix containing the log-intensities with rows for probes and columns for arrays.

4.2 Recommended Files

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, ImaGene, 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.

You will need to have the image analysis output files. In most cases these files will include the IDs and names of the probes and possibly other annotation information. A few image analysis programs, for example SPOT, do not write the probe IDs into the output files. In this case you will also need a genelist file which describes the probes. It most cases it is also desirable to have a *targets file* which describes which RNA sample was hybridized to each channel of each array. A further optional file is the *spot types file* which identifies special probes such as control spots.

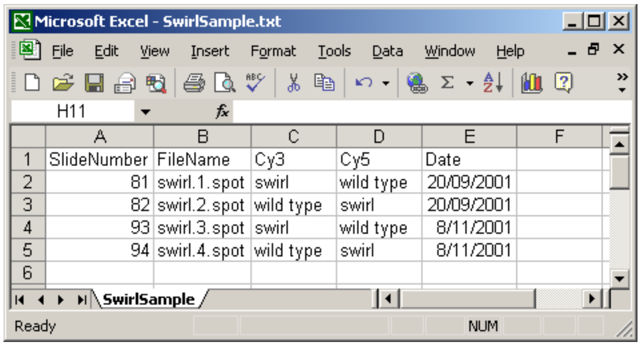
4.3 The Targets Frame

The first step in preparing data for input into limma is usually to create a targets file which lists the RNA target hybridized to each channel of each array. It is normally in tab-delimited text format and should contain a row for each microarray in the experiment. The file can have any name but the default is `Targets.txt`. If it has the default name, it can be read into the R session using

```
> targets <- readTargets()
```

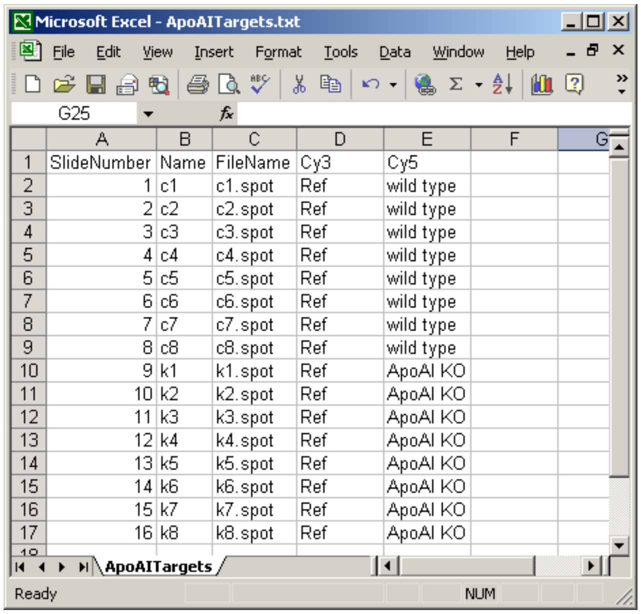
Once read into R, it becomes the *targets frame*.

The targets frame normally contains a `FileName` column, giving the name of the image-analysis output file, a `Cy3` column giving the RNA type labelled with Cy3 dye for that slide and a `Cy5` column giving the RNA type labelled with Cy5 dye for that slide. Other columns are optional. The targets file can be prepared using any text editor but spreadsheet programs such as Microsoft Excel are convenient. The targets file for the Swirl case study includes optional `SideNumber` and `Date` columns:



	A	B	C	D	E	F
1	SlideNumber	FileName	Cy3	Cy5	Date	
2	81	swirl.1.spot	swirl	wild type	20/09/2001	
3	82	swirl.2.spot	wild type	swirl	20/09/2001	
4	93	swirl.3.spot	swirl	wild type	8/11/2001	
5	94	swirl.4.spot	wild type	swirl	8/11/2001	
6						

It is often convenient to create short readable labels to associate with each array for use in output and in plots, especially if the file names are long or non-intuitive. A column containing these labels can be included in the targets file, for example the `Name` column used for the ApoAI case study:



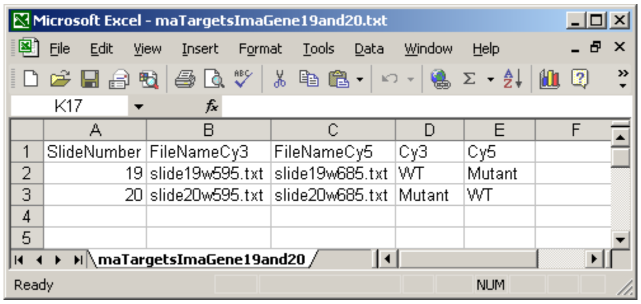
	A	B	C	D	E	F	G
1	SlideNumber	Name	FileName	Cy3	Cy5		
2	1	c1	c1.spot	Ref	wild type		
3	2	c2	c2.spot	Ref	wild type		
4	3	c3	c3.spot	Ref	wild type		
5	4	c4	c4.spot	Ref	wild type		
6	5	c5	c5.spot	Ref	wild type		
7	6	c6	c6.spot	Ref	wild type		
8	7	c7	c7.spot	Ref	wild type		
9	8	c8	c8.spot	Ref	wild type		
10	9	k1	k1.spot	Ref	ApoAI KO		
11	10	k2	k2.spot	Ref	ApoAI KO		
12	11	k3	k3.spot	Ref	ApoAI KO		
13	12	k4	k4.spot	Ref	ApoAI KO		
14	13	k5	k5.spot	Ref	ApoAI KO		
15	14	k6	k6.spot	Ref	ApoAI KO		
16	15	k7	k7.spot	Ref	ApoAI KO		
17	16	k8	k8.spot	Ref	ApoAI KO		

This column can be used to created row names for the targets frame by

```
> targets <- readTargets("targets.txt", row.names="Name")
```

The row names can be propagated to become array names in the data objects when these are read in.

For ImaGene files, the FileName column is split into a `FileNameCy3` column and a `FileNameCy5` because ImaGene stores red and green intensities in separate files. This is a short example:



	A	B	C	D	E	F
1	SlideNumber	FileNameCy3	FileNameCy5	Cy3	Cy5	
2	19	slide19w595.txt	slide19w685.txt	WT	Mutant	
3	20	slide20w595.txt	slide20w685.txt	Mutant	WT	
4						
5						

4.4 Reading in Intensity Data

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. The file names are usually read from the Targets File. For example, the Targets File `Targets.txt` is in the current working directory together with the SPOT output files, then one might use

```
> targets <- readTargets()
> RG <- read.maimages(targets$FileName, source="spot")
```

Alternatively, and even more simply, one may give the targets frame itself in place of the `files` argument as

```
> RG <- read.maimages(targets, source="spot")
```

In this case the software will look for the column `FileName` in the targets frame.

If the files are GenePix output files then they might be read using

```
> RG <- read.maimages(targets, source="genepix")
```

given an appropriate targets file. 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, although other separators can be specified using the `sep=` argument.

Reading data from ImaGene software is a little different to that of other image analysis programs because the red and green intensities are stored in separate files. This means that the targets frame should include two filename columns called, say, `FileNameCy3` and `FileNameCy5`, giving the names of the files containing the green and red intensities respectively. An example is given in Section 4.3. Typical code with ImaGene data might be

```
> targets <- readTargets()
> files <- targets[,c("FileNameCy3","FileNameCy5")]
> RG <- read.maimages(files, source="imagene")
```

For ImaGene data, the `files` argument to `read.maimages()` is expected to be a 2-column matrix of filenames rather than a vector.

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(R="F635 Mean",G="F532 Mean",Rb="B635 Median",Gb="B532 Median"),
+       annotation=c("Block","Row","Column","ID","Name"))
```

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 of 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.

4.5 Spot Quality Weights

It is desirable to use the image analysis output to compute a weight for each spot between 0 and 1 which indicates the reliability of the acquired intensities for 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.

The appropriate way to computing spot quality weights depends on the image analysis program that you are used. Consult the help entry `QualityWeights` to see what quality weight functions are available. The `wt.fun` argument is very flexible and allows you to construct your own weights. The `wt.fun` argument can be any function which takes a data set as argument and computes the desired weights. For example, if you wish to give zero weight to all Genepix flags less than -50 you could use

```
> myfun <- function(x) as.numeric(x$Flags > -50.5)
> RG <- read.maimages(files, source="genepix", wt.fun=myfun)
```

The `wt.fun` facility can be used to compute weights based on any number of columns in the image analysis files. For example, some researchers like to filter out spots if the foreground mean and median from GenePix for a given spot differ by more than a certain threshold, say 50. This could be achieved by

```
> myfun <- function(x, threshold=50) {
+    okred <- abs(x[,"F635 Median"]-x[,"F635 Mean"]) < threshold
+    okgreen <- abs(x[,"F532 Median"]-x[,"F532 Mean"]) < threshold
+    as.numeric(okgreen & okred)
+}
> RG <- read.maimages(files, source="genepix", wt.fun=myfun)
```

Then all the “bad” spots will get weight zero which, in limma, is equivalent to flagging them out. The definition of `myfun` here could be replaced with any other code to compute weights using the columns in the GenePix output files.

4.6 Reading the Gene List

The `RGList` read by `read.maimages()` will almost always contain a component called `genes` containing the IDs and other annotation information associated with the probes. The only exceptions are SPOT data, `source="spot"`, or when reading generic data, `source="generic"`, without setting the annotation argument, `annotation=NULL`. Try

```
> names(RG$genes)
```

to see if the `genes` component has been set.

If the `genes` component is not set, the probe IDs will need to be read from a separate file. If the arrays have been scanned with an Axon scanner, then the probes IDs will be available in a tab-delimited 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

```
> RG$genes <- readGAL()
```

Non-Genepix gene lists can be read into R using the function `read.delim` from R base.

4.7 Printer Layout

The printer layout is the arrangement of spots and blocks of spots on the arrays. The blocks are sometimes called print-tip groups or pin-groups or meta rows and columns. Each block corresponds to a print tip on the print-head used to print the arrays, and the layout of the blocks on the arrays corresponds to the layout of the tips on the print-head. The number of spots in each block is the number of times the print-head was lowered onto the array. Where possible, for example for Agilent, GenePix or ImaGene data, `read.maimages` will set the printer layout information in the component `printer`. Try

```
> names(RG$printer)
```

to see if the printer layout information has been set.

If you've used `readGAL` to set the `genes` component, you may also use `getLayout` to set the `printer` information by

```
> RG$printer <- getLayout(RG$genes)
```

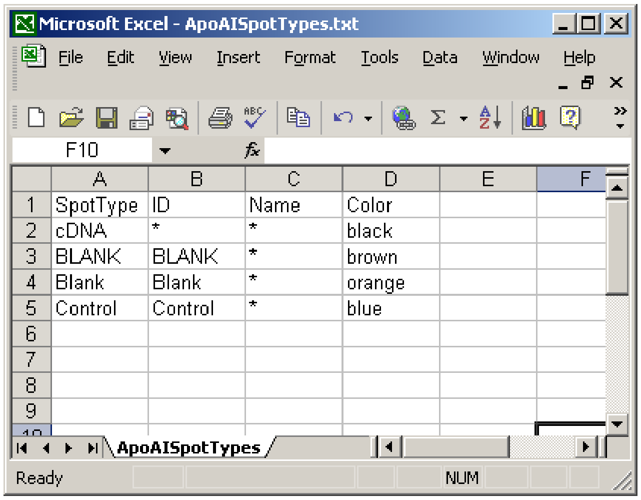
Note this will work only for GenePix GAL files, not for general gene lists.

4.8 The Spot Types File

The Spot Types file (STF) is another optional tab-delimited text file which allows you to identify different types of spots from the gene list. The STF is used to set the control status of each spot on the arrays so that plots may highlight different types of spots in an appropriate way. It is typically used to distinguish control spots from those corresponding to genes of interest and to distinguish positive from negative controls, ratio from calibration controls and so on. The STF should have a `SpotType` column giving the names of the different spot-types. One or more other columns should have the same names as columns in the gene list and should contain patterns or regular expressions sufficient to identify the spot-type. Any other columns are assumed to contain plotting attributes, such as colors or symbols, to be associated with the spot-types. There is one row for each spot-type to be distinguished.

The STF uses simplified regular expressions to match patterns. For example, `AA*` means any string starting with `AA`, `*AA` means any code ending with `AA`, `AA` means exactly these two letters, `*AA*` means any string containing `AA`, `AA.` means `AA` followed by exactly one other character and `AA\.` means exactly `AA` followed by a period and no other characters. For those familiar with regular expressions, any other regular expressions are allowed but the codes ˆ for beginning of string and `$` for end of string should be excluded. Note that the patterns are matched sequentially from first to last, so more general patterns should be included first. The first row should specify the default spot-type and should have pattern `*` for all the pattern-matching columns.

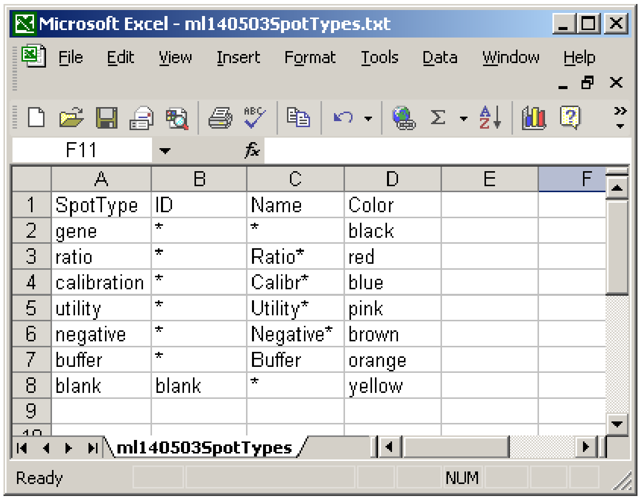
Here is a short STF appropriate for the ApoAI data:



	A	B	C	D	E	F
1	SpotType	ID	Name	Color		
2	cDNA	*	*	black		
3	BLANK	BLANK	*	brown		
4	Blank	Blank	*	orange		
5	Control	Control	*	blue		
6						
7						
8						
9						

In this example, the columns `ID` and `Name` are found in the gene-list and contain patterns to match. The asterisks are wildcards which can represent anything. Be careful to use upper or lower case as appropriate and don't insert any extra spaces. The remaining column gives colors to be associated with the different types of points. This code assumes of that the probe annotation data.frame includes columns `ID` and `Name`. This is usually so if GenePix has been used for the image analysis, but other image analysis software may use other column names.

Here is a STF below appropriate for arrays with Lucidea Universal ScoreCard control spots.



	A	B	C	D	E	F
1	SpotType	ID	Name	Color		
2	gene	*	*	black		
3	ratio	*	Ratio*	red		
4	calibration	*	Calibr*	blue		
5	utility	*	Utility*	pink		
6	negative	*	Negative*	brown		
7	buffer	*	Buffer	orange		
8	blank	blank	*	yellow		
9						

If the STF has default name `SpotTypes.txt` then it can be read using

```
> spottypes <- readSpotTypes()
```

It is typically used as an argument to the `controlStatus()` function to set the status of each spot on the array, for example

```
> RG$genes$Status <- controlStatus(spottypes, RG)
```

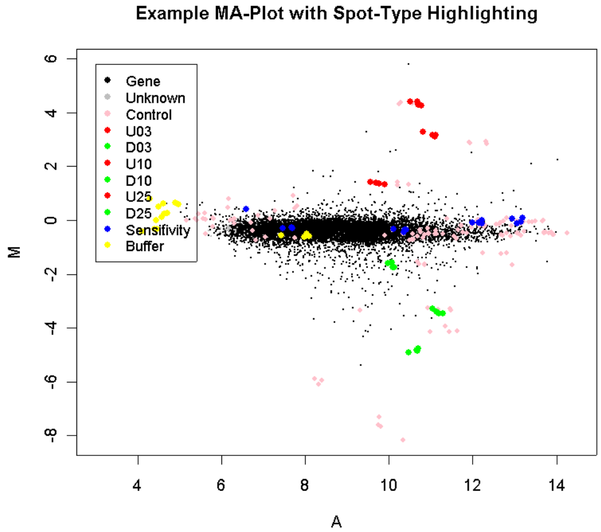
Chapter 5

Data Exploration

It is advisable to display your data in various ways as a quality check and to check for unexpected effects. We recommend an imageplot of the raw log-ratios and an MA-plot of the raw data at least for each array as routine quality assessment displays. See the Swirl case study for some examples. The functions `imageplot3by2` and `plotMA3by2` can be used to automate the production of plots for all arrays in an experiment.

The following is an example MA-Plot for an Incyte array with various spike-in and other controls. (Data courtesy of Rebecca McCracken and Steve Gerondakis, Walter and Eliza Hall Institute of Medical Research.) The plot was produced using

```
> spottypes <- readSpotTypes()
> RG$genes$Status <- controlStatus(spottypes, RG)
> plotMA(RG)
```



The array includes spike-in ratio controls which are 3-fold, 10-fold and 25-fold up and down regulated, as well as non-differentially expressed sensitivity controls and negative controls.

Chapter 6

Pre-Processing Two-Color Data

6.1 Background Correction

The default background correction action is to subtract the background intensity from the foreground intensity for each spot. If the `RGList` object has not already been background corrected, then `normalizeWithinArrays` will do this by default. Hence

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

is equivalent to

```
> RGb <- backgroundCorrect(RG, method="subtract")
> MA <- normalizeWithinArrays(RGb)
```

However there are many other background correction options which may be preferable in certain situations.

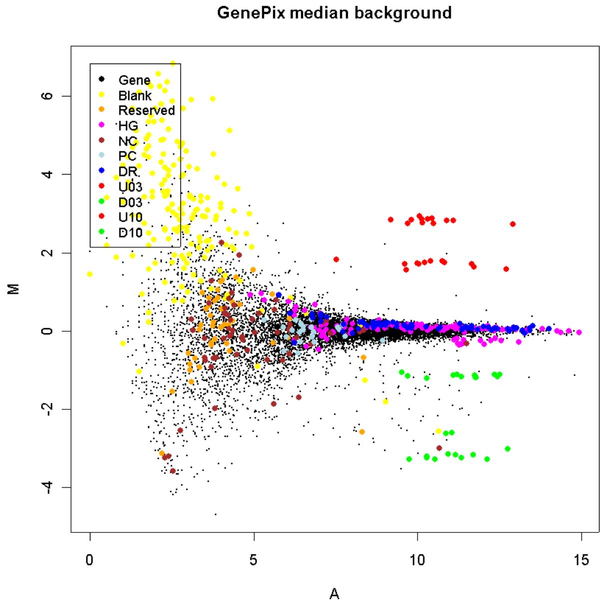
For the purpose of assessing differential expression, we often find

```
> RG <- backgroundCorrect(RG, method="normexp", offset=50)
```

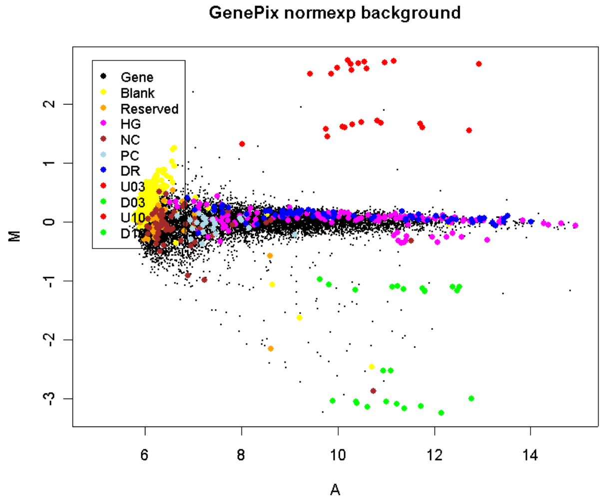
to be preferable to the simple background subtraction when using output from most image analysis programs. This method adjusts the foreground adaptively for the background intensities and results in strictly positive adjusted intensities, i.e., negative or zero corrected intensities are avoided. The use of an offset damps the variation of the log-ratios for very low intensities spots towards zero.

To illustrate some differences between the different background correction methods we consider one cDNA array which was self-self hybridized, i.e., the same RNA source was hybridized to both channels. For this array there is no actual differential expression. The array was printed with a human 10.5k library and hybridized with Jurkatt RNA on both channels. (Data courtesy Andrew Holloway and Dileepa Diyagama, Peter MacCallum Cancer Centre, Melbourne.) The array included a selection of control spots which are highlighted on the plots. Of particular interest are the spike-in ratio controls which should show up and down fold changes of 3 and 10. The first plot displays data acquired with GenePix software and background corrected by subtracting the median local background, which is the default with

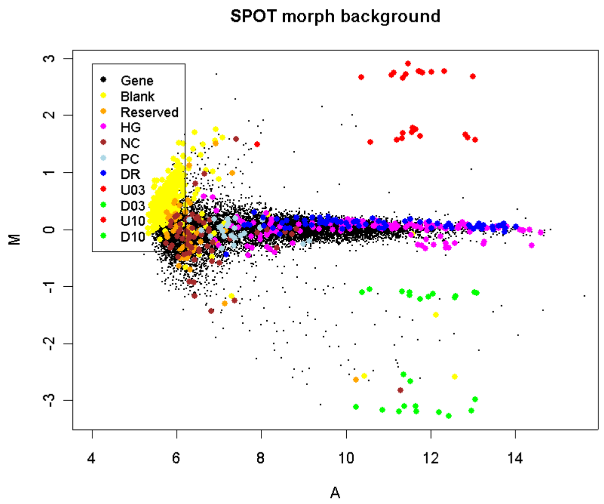
GenePix data. The plot shows the typical wedge shape with fanning of the M-values at low intensities. The range of observed M-values dominates the spike-in ratio controls. The are also 1148 spots not shown on the plot because the background corrected intensities were zero or negative.



The second plot shows the same array background corrected with `method="normexp"` and `offset=50`. The spike-in ratio controls now standout clearly from the range of the M-values. All spots on the array are shown on the plot because there are now no missing M-values.



The third plot shows the same array quantified with SPOT software and with “morph” background subtracted. This background estimator produces a similar effect to that with `normexp`.



The effect of using "morph" background or using `method="normexp"` with an offset is to stabilize the variability of the M-values as a function of intensity. The empirical Bayes methods implemented in the limma package for assessing differential expression will yield most benefit when the variabilities are as homogenous as possible between genes. This can best be achieved by reducing the dependence of variability on intensity as far as possible.

6.2 Within-Array Normalization

Limma implements a range of normalization methods for spotted microarrays. Smyth and Speed [20] describe some of the most commonly used methods. The methods may be broadly classified into methods which normalize the M-values for each array separately (within-array normalization) and methods which normalize intensities or log-ratios to be comparable across arrays (between-array normalization). This section discusses mainly within-array normalization, which all that is usually required for the traditional log-ratio analysis of two-color data. Between-array normalization is discussed further in Section 6.3.

Print-tip loess normalization [29] is the default normalization method and can be performed by

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

There are some notable cases where this is not appropriate. For example, Agilent arrays do not have print-tip groups, so one should use global loess normalization instead:

```
> MA <- normalizeWithinArrays(RG, method="loess")
```

Print-tip loess is also unreliable for small arrays with less than, say, 150 spots per print-tip group. Even larger arrays may have particular print-tip groups which are too small for print-tip loess normalization if the number of spots with non-missing M-values is small for one or more of the print-tip groups. In these cases one should either use global `"loess"` normalization or else use robust spline normalization

```
> MA <- normalizeWithinArrays(RG, method="robustspline")
```

which is an empirical Bayes compromise between print-tip and global loess normalization, with 5-parameter regression splines used in place of the loess curves.

Loess normalization assumes that the bulk of the probes on the array are not differentially expressed. It doesn't assume that that there are equal numbers of up and down regulated genes or that differential expression is symmetric about zero, provided that the loess fit is implemented in a robust fashion, but it is necessary that there be a substantial body of probes which do not change expression levels. This assumption can be suspect for boutique arrays where the total number of unique genes on the array is small, say less than 150, particularly if these genes have been selected for being specifically expressed in one of the RNA sources. In such a situation, the best strategy is to include on the arrays a series of non-differentially expressed control spots, such as a titration series of whole-library-pool spots, and to use the up-weighting method discussed below. A whole-library-pool means that one makes a pool of a library of probes, and prints spots from the pool at various concentrations [28]. The library should be sufficiently large than one can be confident that the average of all the probes is not differentially expressed. The larger the library the better. Good results have been obtained with library pools with as few as 500 clones. In the absence of such control spots, normalization of boutique arrays requires specialist advice.

Any spot quality weights found in `RG` will be used in the normalization by default. This means for example that spots with zero weight (flagged out) will not influence the normalization of other spots. The use of spot quality weights will not however result in any spots being removed from the data object. Even spots with zero weight will be normalized and will appear in the output object, such spots will simply not have any influence on the other spots. If you do not wish the spot quality weights to be used in the normalization, their use can be over-ridden using

```
> MA <- normalizeWithinArrays(RG, weights=NULL)
```

The output object `MA` will still contain any spot quality weights found in `RG`, but these weights are not used in the normalization step.

It is often useful to make use of control spots to assist the normalization process. For example, if the arrays contain a series of spots which are known in advance to be non-differentially expressed, these spots can be given more weight in the normalization process. Spots which are known in advance to be differentially expressed can be down-weighted. Suppose for example that the `controlStatus()` has been used to identify spike-in spots which are differentially expressed and a titration series of whole-library-pool spots which should not be differentially expressed. Then one might use

```
> w <- modifyWeights(RG$weights, RG$genes$Status, c("spikein","titration"), c(0,2))
> MA <- normalizeWithinArrays(RG, weights=w)
```

to give zero weight to the spike-in spots and double weight to the titration spots. The idea of up-weighting the titration spots is in the same spirit as the composite normalization method proposed by [28] but is more flexible and generally applicable. The above code assumes that `RG` already contains spot quality weights. If not, one could use

```
> w <- modifyWeights(array(1,dim(RG)), RG$genes$Status, c("spikein","titration"), c(0,2))
> MA <- normalizeWithinArrays(RG, weights=w)
```

instead.

Limma contains some more sophisticated normalization methods. In particular, some between-array normalization methods are discussed in Section 6.3 of this guide.

6.3 Between-Array Normalization

This section explores some of the methods available for between-array normalization of two-color arrays. A feature which distinguishes most of these methods from within-array normalization is the focus on the individual red and green intensity values rather than merely on the log-ratios. These methods might therefore be called *individual channel* or *separate channel* normalization methods. Individual channel normalization is typically a prerequisite to individual channel analysis methods such as that provided by `lmscFit()`. Further discussion of the issues involved is given by [31]. This section shows how to reproduce some of the results given in [31]. The ApoAI data set from Section 11.2 will be used to illustrate these methods. We assume that the the ApoAI data has been loaded and background corrected as follows:

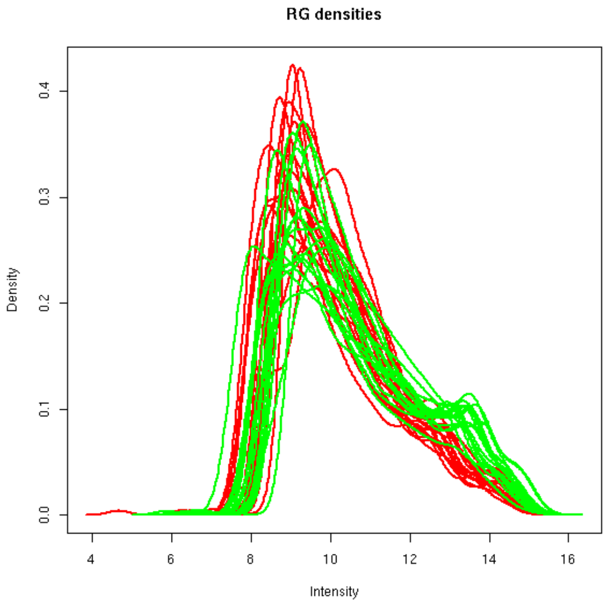
```
> load("ApoAI.RData")
```

An important issue to consider before normalizing between arrays is how background correction has been handled. For between-array normalization to be effective, it is important to avoid missing values in log-ratios which might arise from negative or zero corrected intensities. The function `backgroundCorrect()` gives a number of useful options. For the purposes of this section, the data has been corrected using the `"minimum"` method:

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

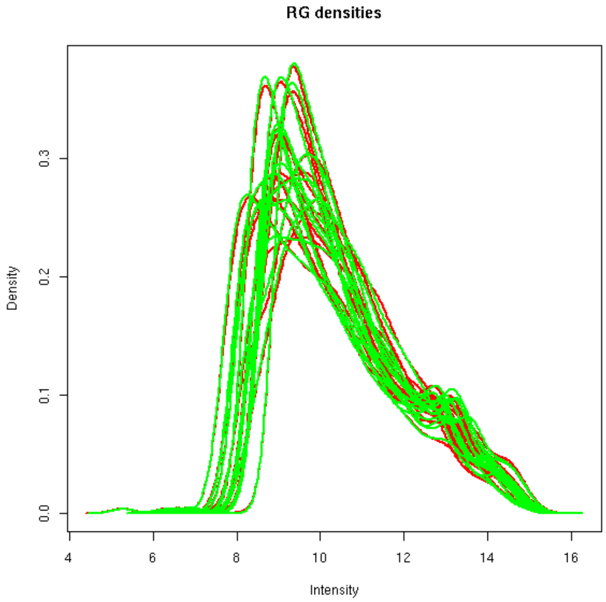
`plotDensities` displays smoothed empirical densities for the individual green and red channels on all the arrays. Without any normalization there is considerable variation between both channels and between arrays:

```
> plotDensities(RG.b)
```



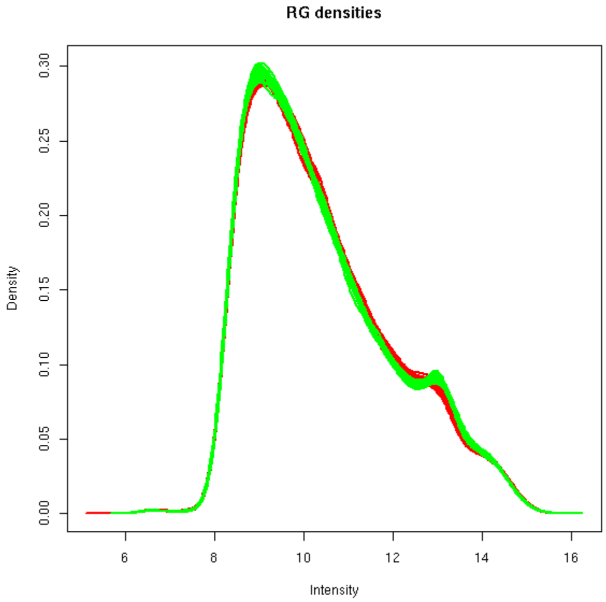
After loess normalization of the M-values for each array the red and green distributions become essentially the same for each array, although there is still considerable variation between arrays:

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



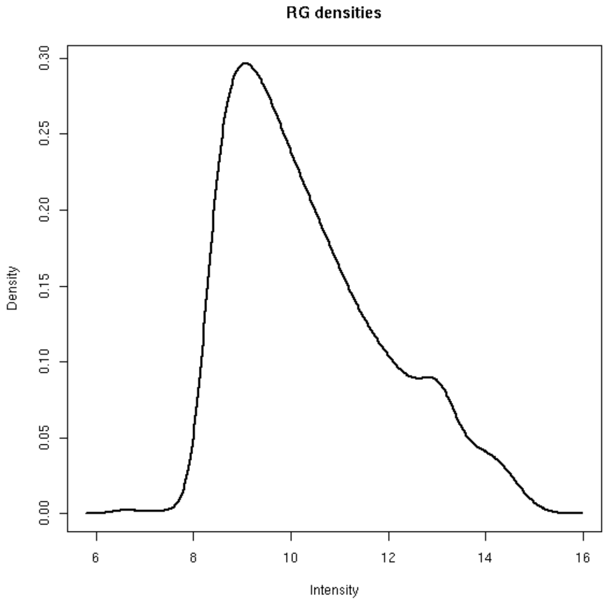
Loess normalization doesn't affect the A-values. Applying quantile normalization to the A-values makes the distributions essentially the same across arrays as well as channels:

```
> MA.pAq <- normalizeBetweenArrays(MA.p, method="Aquantile")
> plotDensities(MA.pAq)
```



Applying quantile normalization directly to the individual red and green intensities produces a similar result but is somewhat noisier:

```
> MA.q <- normalizeBetweenArrays(RG.b, method="quantile")
> plotDensities(MA.q, col="black")

Warning message:
number of groups=2 not equal to number of col in: plotDensities(MA.q, col = "black")
```



There are other between-array normalization methods not explored here. For example `normalizeBetweenArrays` with `method="vsn"` gives an interface to the variance-stabilizing normalization methods of the vsn package.

6.4 Using Objects from the marray Package

The package marray is a well known R package for pre-processing of two-color microarray data. Marray provides functions for reading, normalization and graphical display of data. Marray and limma are both descendants of the earlier and path-breaking SMA package available from `http://www.stat.berkeley.edu/users/terry/zarray/Software/smacode.html` but limma has maintained and built upon the original data structures whereas marray has converted to a fully formal data class representation. For this reason, Limma is backwardly compatible with SMA while marray is not.

Normalization functions in marray focus on a flexible approach to location and scale normalization of M-values, rather than the within and between-array approach of limma. Marray provides some normalization methods which are not in limma including 2-D loess normalization and print-tip-scale normalization. Although there is some overlap between the normalization functions in the two packages, both providing print-tip loess normalization, the two approaches are largely complementary. Marray also provides highly developed functions for graphical display of two-color microarray data.

Read functions in marray produce objects of class `marrayRaw` while normalization produces objects of class `marrayNorm`. Objects of these classes may be converted to and from limma data objects using the convert package. `marrayRaw` objects may be converted to `RGList` objects and `marrayNorm` objects to `MAList` objects using the `as` function. For example, if `Data` is an `marrayNorm` object then

```
> library(convert)
> MA <- as(Data, "MAList")
```

converts to an `MAList` object.

`marrayNorm` objects can also be used directly in limma without conversion, and this is generally recommended. If `Data` is an `marrayNorm` object, then

```
> fit <- lmFit(Data, design)
```

fits a linear model to `Data` as it would to an `MAList` object. One difference however is that the marray read functions tend to populate the `maW` slot of the `marrayNorm` object with qualitative spot quality flags rather than with quantitative non-negative weights, as expected by limma. If this is so then one may need

```
> fit <- lmFit(Data, design, weights=NULL)
```

to turn off use of the spot quality weights.

Chapter 7

Linear Models Overview

7.1 Introduction

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 outlined in [21, 30]. 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.

The philosophy of the approach is as follows. You have to start by fitting a linear model to your data which fully models the systematic part of your data. The model is specified by the design matrix. Each row of the design matrix corresponds to an array in your experiment and each column corresponds to a coefficient which is used to describe the RNA sources in your experiment. With Affymetrix or single-channel data, or with two-color with a common reference, you will need as many coefficients as you have distinct RNA sources, no more and no less. With direct-design two-color data you will need one fewer coefficient than you have distinct RNA sources, unless you wish to estimate a dye-effect for each gene, in which case the number of RNA sources and the number of coefficients will be the same. Any set of independent coefficients will do, providing they describe all your treatments. The main purpose of this step is to estimate the variability in the data, hence the systematic part needs to modelled so it can be distinguished from random variation.

In practice the requirement to have exactly as many coefficients as RNA sources is too restrictive in terms of questions you might want to answer. You might be interested in more or fewer comparisons between the RNA source. Hence the contrasts step is provided so that you can take the initial coefficients and compare them in as many ways as you want to answer any questions you might have, regardless of how many or how few these might be.

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 as 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 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.

For statistical analysis and assessing differential expression, limma uses an empirical Bayes method to moderate the standard errors of the estimated log-fold changes. This results in more stable inference and improved power, especially for experiments with small numbers of arrays [21]. For arrays with within-array replicate spots, limma uses a pooled correlation method to make full use of the duplicate spots [19].

7.2 Affymetrix and Other Single-Channel Designs

Affymetrix data will usually be normalized using the affy package. We will assume here that the data is available as an `exprSet` object called `eset`. Such an object will have an slot containing the log-expression values for each gene on each array which can be extracted using `exprs(eset)`. Affymetrix and other single-channel microarray data may be analysed very much like ordinary linear models or anova models. The difference with microarray data is that it is almost always necessary to extract particular contrasts of interest and so the standard parametrizations provided for factors in R are not usually adequate.

There are many ways to approach the analysis of a complex experiment in limma. A straightforward strategy is to set up the simplest possible design matrix and then to extract from the fit the contrasts of interest.

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()
> eset <- 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 <- lmFit(eset, design)
```

To make all pair-wise comparisons between the three groups the appropriate contrast matrix can be created by

```
> contrast.matrix <- makeContrasts(group2-group1, group3-group2, group3-group1, levels=design)
> fit2 <- contrasts.fit(fit, contrast.matrix)
> fit2 <- eBayes(fit2)
```

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

```
> topTable(fit2, coef=1, adjust="BH")
```

The outcome of each hypothesis test can be assigned using

```
> results <- decideTests(fit2)
```

A Venn diagram showing numbers of genes significant in each comparison can be obtained from

```
> vennDiagram(results)
```

7.3 Common Reference Designs

Now consider two-color microarray experiments in which a common reference has been used on all the arrays. Such experiments can be analysed very similarly to Affymetrix experiments except that allowance must be made for dye-swaps. The simplest method is to setup the design matrix using the `modelMatrix()` function and the targets file. As an example, we consider part of an experiment conducted by Joëlle Michaud, Catherine Carmichael and Dr Hamish Scott at the Walter and Eliza Hall Institute to compare the effects of transcription factors in a human cell line. The targets file is as follows:

```
> targets <- readTargets("runxtargets.txt")
> targets
   SlideNumber       Cy3       Cy5
1         2144      EGFP      AML1
2         2145      EGFP      AML1
3         2146      AML1      EGFP
4         2147      EGFP AML1.CBFb
5         2148      EGFP AML1.CBFb
6         2149 AML1.CBFb      EGFP
7         2158      EGFP      CBFb
8         2159      CBFb      EGFP
9         2160      EGFP AML1.CBFb
10        2161 AML1.CBFb      EGFP
11        2162      EGFP AML1.CBFb
12        2163 AML1.CBFb      EGFP
13        2166      EGFP      CBFb
14        2167      CBFb      EGFP
```

In the experiment, green fluorescent protein (EGFP) has been used as a common reference. An adenovirus system was used to transport various transcription factors into the nuclei of HeLa cells. Here we consider the transcription factors AML1, CBFbeta or both. A simple design matrix was formed and a linear model fit:

```
> design <- modelMatrix(targets,ref="EGFP")
> design
   AML1 AML1.CBFb CBFb
1     1         0    0
2     1         0    0
3    -1         0    0
4     0         1    0
5     0         1    0
6     0        -1    0
```

```
7     0         0   1
8     0         0  -1
9     0         1   0
10    0        -1   0
11    0         1   0
12    0        -1   0
13    0         0   1
14    0         0  -1
> fit <- lmFit(MA, design)
```

It is of interest to compare each of the transcription factors to EGFP and also to compare the combination transcription factor with AML1 and CBFb individually. An appropriate contrast matrix was formed as follows:

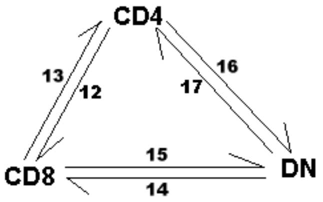
```
> contrast.matrix <- makeContrasts(AML1,CBFb,AML1.CBFb,AML1.CBFb-AML1,AML1.CBFb-CBFb,
+      levels=design)
> contrast.matrix
          AML1 CBFb AML1.CBFb AML1.CBFb - AML1 AML1.CBFb - CBFb
AML1         1    0         0               -1                0
AML1.CBFb    0    0         1                1                1
CBFb         0    1         0                0               -1
```

The linear model fit can now be expanded and empirical Bayes statistics computed:

```
> fit2 <- contrasts.fit(fit, contrasts.matrix)
> fit2 <- eBayes(fit2)
```

7.4 Direct Two-Color Designs

Two-colour designs without a common reference require the most statistical knowledge to choose the appropriate design matrix. A direct design is one in which there is no single RNA source which is hybridized to every array. As an example, we consider an experiment conducted by Dr Mireille Lahoud at the Walter and Eliza Hall Institute to compare gene expression in three different populations of dendritic cells (DC).



Arrow heads represent Cy5, i.e. arrows point in the Cy3 to Cy5 direction.

This experiment involved six cDNA microarrays in three dye-swap pairs, with each pair used to compare two DC types. The design is shown diagrammatically above. The targets file was as follows:

```
> targets
        SlideNumber     FileName Cy3 Cy5
ml12med          12 ml12med.spot CD4 CD8
ml13med          13 ml13med.spot CD8 CD4
ml14med          14 ml14med.spot  DN CD8
ml15med          15 ml15med.spot CD8  DN
ml16med          16 ml16med.spot CD4  DN
ml17med          17 ml17med.spot  DN CD4
```

There are many valid choices for a design matrix for such an experiment and no single correct choice. We chose to setup the design matrix as follows:

```
> design <- modelMatrix(targets, ref="CD4")
Found unique target names:
 CD4 CD8 DN
> design
        CD8 DN
ml12med   1  0
ml13med  -1  0
ml14med   1 -1
ml15med  -1  1
ml16med   0  1
ml17med   0 -1
```

In this design matrix, the CD8 and DN populations have been compared back to the CD4 population. The coefficients estimated by the linear model will correspond to the log-ratios of CD8 vs CD4 (first column) and DN vs CD4 (second column). After appropriate normalization of the expression data, a linear model was fit using

```
> fit <- lmFit(MA, design, ndups=2)
```

The use of `ndups` is to specify that the arrays contained duplicates of each gene, see Section 11.6.

The linear model can now be interrogated to answer any questions of interest. For this experiment it was of interest to make all pairwise comparisons between the three DC populations. This was accomplished using the contrast matrix

```
> contrast.matrix <- cbind("CD8-CD4"=c(1,0),"DN-CD4"=c(0,1),"CD8-DN"=c(1,-1))
> rownames(contrast.matrix) <- colnames(design)
> contrast.matrix
    CD8-CD4 DN-CD4 CD8-DN
CD8       1      0      1
DN        0      1     -1
```

The contrast matrix can be used to expand the linear model fit and then to compute empirical Bayes statistics:

```
> fit2 <- constrast.fit(fit, contrast.matrix)
> fit2 <- eBayes(fit2)
```

Chapter 8

Specific Designs

8.1 Simple Comparisons

8.1.1 Replicate Arrays

The simplest possible microarray experiment is one with a series of replicate two-color arrays all comparing the same two RNA sources. For a three-array experiment comparing wild type (wt) and mutant (mu) RNA, the targets file might contain the following entries:

FileName	Cy3	Cy5
File1	wt	mu
File2	wt	mu
File3	wt	mu

A list of differentially expressed genes might be found for this experiment by

```
> fit <- lmFit(MA)
> fit <- eBayes(fit)
> topTable(fit)
```

where `MA` holds the normalized data. The default design matrix used here is just a single column of ones. The experiment here measures the fold change of mutant over wild type. Genes which have positive M-values are more highly expressed in the mutant RNA while genes with negative M-values are more highly expressed in the wild type. The analysis is analogous to the classical single-sample *t*-test except that we have used empirical Bayes methods to borrow information between genes.

8.1.2 Dye Swaps

A simple modification of the above experiment would be to swap the dyes for one of the arrays. The targets file might now be

FileName	Cy3	Cy5
File1	wt	mu
File2	mu	wt
File3	wt	mu

Now the analysis would be

```
> design <- c(1,-1,1)
> fit <- lmFit(MA, design)
> fit <- eBayes(fit)
> topTable(fit)
```

Alternatively the design matrix could be set, replacing the first of the above code lines, by

```
> design <- modelMatrix(targets, ref="wt")
```

where `targets` is the data frame holding the targets file information.

If there are at least two arrays with each dye-orientation, then it is possible to estimate and adjust for any probe-specific dye effects. The dye-effect is estimated by an intercept term. If the experiment was

FileName	Cy3	Cy5
File1	wt	mu
File2	mu	wt
File3	wt	mu
File4	mu	wt

then we could set

```
> design <- cbind(DyeEffect=1,MUvsWT=c(1,-1,1,-1))
> fit <- lmFit(MA, design)
> fit <- eBayes(fit)
```

The genes which show dye effects can be seen by

```
> topTable(fit, coef="DyeEffect")
```

The genes which are differentially expressed in the mutant are obtained by

```
> topTable(fit, coef="MUvsWT")
```

The fold changes and significant tests in this list are corrected for dye-effects. Including the dye-effect in the model in this way uses up one degree of freedom which might otherwise be used to estimate the residual variability, but it is valuable if many genes show non-negligible dye-effects.

8.2 Technical Replication

In the previous sections we have assumed that all arrays are biological replicates. Now consider an experiment in which two wild-type and two mice from the same mutant strain are compared using two arrays for each pair of mice. The targets might be

FileName	Cy3	Cy5
File1	wt1	mu1
File2	wt1	mu1
File3	wt2	mu2
File4	wt2	mu2

The first and second and third and fourth arrays are *technical replicates*. It would not be correct to treat this experiment as comprising four replicate arrays because the technical replicate pairs are not independent, in fact they are likely to be positively correlated.

One way to analyze these data is the following:

```
> biolrep <- c(1, 1, 2, 2)
> corfit <- duplicateCorrelation(MA, ndups = 1, block = biolrep)
> fit <- lmFit(MA, block = biolrep, cor = corfit$consensus)
> fit <- eBayes(fit)
> topTable(fit, adjust = "BH")
```

The vector `biolrep` indicates the two blocks corresponding to biological replicates. The value `cofit$consensus` estimates the average correlation within the blocks and should be positive. This analysis is analogous to *mixed model* analysis of variance [12, Chapter 18] except that information has been borrowed between genes. Information is borrowed by constraining the within-block correlations to be equal between genes and by using empirical Bayes methods to moderate the standard deviations between genes [19].

If the technical replicates were in dye-swap pairs as

FileName	Cy3	Cy5
File1	wt1	mu1
File2	mu1	wt1
File3	wt2	mu2
File4	mu2	wt2

then one might use

```
> design <- c(1, -1, 1, -1)
> corfit <- duplicateCorrelation(MA, design, ndups = 1, block = biolrep)
> fit <- lmFit(MA, design, block = biolrep, cor = corfit$consensus)
> fit <- eBayes(fit)
> topTable(fit, adjust = "BH")
```

In this case the correlation `corfit$consensus` should be negative because the technical replicates are dye-swaps and should vary in opposite directions.

This method of handling technical replication using `duplicateCorrelation()` is somewhat limited. If for example one techical replicate was dye-swapped and the other not,

FileName	Cy3	Cy5
File1	wt1	mu1
File2	mu1	wt1
File3	wt2	mu2
File4	wt2	mu2

then there is no way to use `duplicateCorrelation()` because the technical replicate correlation will be negative for the first pair but positive for the second. An alternative strategy is to include a coefficient in the design matrix for each of the two biological blocks. This could be accomplished by defining

```
> design <- cbind(MU1vsWT1 = c(1,-1,0,0), MU2vsWT2 = c(0,0,1,1))
> fit <- lmFit(MA, design)
```

This will fit a linear model with two coefficients, one estimating the mutant vs wild-type comparison for the first pair of mice and the other for the second pair of mice. What we want is the average of the two mutant vs wild-type comparisons, and this is extracted by the contrast `(MU1vsWT1+MU2vsWT2)/2`:

```
> cont.matrix <- makeContrasts(MUvsWT = (MU1vsWT1 + MU2vsWT2)/2,
+                              levels = design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust = "BH")
```

The technique of including an effect for each biological replicate is well suited to situations with a lot of technical replication. Here is a larger example from a real experiment. Three mutant mice are to be compared with three wild-type mice. Eighteen two-color arrays were used with each mouse appearing on six different arrays:

```
> targets

     FileName Cy3 Cy5
1391 1391.spot wt1 mu1
1392 1392.spot mu1 wt1
1340 1340.spot wt2 mu1
1341 1341.spot mu1 wt2
1395 1395.spot wt3 mu1
1396 1396.spot mu1 wt3
1393 1393.spot wt1 mu2
1394 1394.spot mu2 wt1
1371 1371.spot wt2 mu2
1372 1372.spot mu2 wt2
1338 1338.spot wt3 mu2
1339 1339.spot mu2 wt3
1387 1387.spot wt1 mu3
1388 1388.spot mu3 wt1
1399 1399.spot wt2 mu3
1390 1390.spot mu3 wt2
1397 1397.spot wt3 mu3
1398 1398.spot mu3 wt3
```

The comparison of interest is the average difference between the mutant and wild-type mice. `duplicateCorrelation()` could not be used here because the arrays do not group neatly into biological replicate groups. In any case, with six arrays on each mouse it is much safer and more conservative to fit an effect for each mouse. We could proceed as

```
> design <- modelMatrix(targets, ref = "wt1")
> design <- cbind(Dye = 1, design)
> colnames(design)

[1] "Dye" "mu1" "mu2" "mu3" "wt2" "wt3"
```

The above code treats the first wild-type mouse as a baseline reference so that columns of the design matrix represent the difference between each of the other mice and wt1. The design matrix also includes an intercept term which represents the dye effect of Cy5 over Cy3 for each gene. If no dye effect is expected then the second line of code can be omitted.

```
> fit <- lmFit(MA, design)
> cont.matrix <- makeContrasts(muvswt = (mu1+mu2+mu3-wt2-wt3)/3,
+                              levels = design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust = "BH")
```

The contrast defined by the function `makeContrasts` represents the average difference between the mutant and wild-type mice, which is the comparison of interest.

This general approach is applicable to many studies involving biological replicates. Here is another example based on a real example conducted by the Scott Lab at the Walter and Eliza Hall Institute (WEHI). RNA is collected from four human subjects from the same family, two affected by a leukemia-inducing mutation and two unaffected. Each of the two affected subjects (A1 and A2) is compared with each of the two unaffected subjects (U1 and U2):

FileName	Cy3	Cy5
File1	U1	A1
File2	A1	U2
File3	U2	A2
File4	A2	U1

Our interest is to find genes which are differentially expressed between the affected and unaffected subjects. Although all four arrays compare an affected with an unaffected subject, the arrays are not independent. We need to take account of the fact that RNA from each subject appears on two different arrays. We do this by fitting a model with a coefficient for each subject and then extracting the contrast between the affected and unaffected subjects:

```
> design <- modelMatrix(targets, ref = "U1")
> fit <- lmFit(MA, design)
> cont.matrix <- makeContrasts(AvsU = (A1+A2-U2)/2, levels = design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust = "BH")
```

8.3 Paired Samples

Paired samples occur when we compare two treatments and each sample given one treatment is naturally paired with a particular sample given the other treatment. This is a special case of blocking with blocks of size two. The classical test associated with this situation is the paired t-test.

Suppose an experiment is conducted with Affymetrix or single-channel arrays to compare a new treatment (T) with a control (C). Six dogs are used from three sib-ships. For each sib-pair, one dog is given the treatment while the other dog is a control. This produces the targets frame:

FileName	SibShip	Treatment
File1	1	C
File2	1	T
File3	2	C
File4	2	T
File5	3	C
File6	3	T

A moderated paired *t*-test can be computed by allowing for sib-pair effects in the linear model:

```
> SibShip <- factor(targets$SibShip)
> Treat <- factor(targets$Treatment, levels=c("C","T"))
> design <- model.matrix(~SibShip+Treat)
> fit <- lmFit(eset, design)
> fit <- eBayes(fit)
> topTable(fit, coef="TreatT")
```

8.4 Two Groups: Common Reference

Suppose now that we wish to compare two wild type (Wt) mice with three mutant (Mu) mice using arrays hybridized with a common reference RNA (Ref):

FileName	Cy3	Cy5
File1	Ref	WT
File2	Ref	WT
File3	Ref	Mu
File4	Ref	Mu
File5	Ref	Mu

The interest here is in the comparison between the mutant and wild type mice. There are two major ways in which this comparison can be made. We can

1. create a design matrix which includes a coefficient for the mutant vs wild type difference, or

2. create a design matrix which includes separate coefficients for wild type and mutant mice and then extract the difference as a contrast.

For the first approach, the design matrix should be as follows

```
> design

       WTvsREF MUvsWT
Array1       1      0
Array2       1      0
Array3       1      1
Array4       1      1
Array5       1      1
```

Here the first coefficient estimates the difference between wild type and the reference for each probe while the second coefficient estimates the difference between mutant and wild type. For those not familiar with model matrices in linear regression, it can be understood in the following way. The matrix indicates which coefficients apply to each array. For the first two arrays the fitted values will be just the `WTvsREF` coefficient, which is correct. For the remaining arrays the fitted values will be `WTvsREF + MUvsWT`, which is equivalent to mutant vs reference, also correct. For reasons that will be apparent later, this is sometimes called the *treatment-contrasts* parametrization. Differentially expressed genes can be found by

```
> fit <- lmFit(MA, design)
> fit <- eBayes(fit)
> topTable(fit, coef="MUvsWT", adjust="BH")
```

There is no need here to use `contrasts.fit()` because the comparison of interest is already built into the fitted model. This analysis is analogous to the classical *pooled two-sample t-test* except that information has been borrowed between genes.

For the second approach, the design matrix should be

```
       WT MU
Array1  1  0
Array2  1  0
Array3  0  1
Array4  0  1
Array5  0  1
```

The first coefficient now represents wild-type vs the reference and the second represents mutant vs the reference. Our comparison of interest is the difference between these two coefficients. We will call this the *group-means* parametrization. Differentially expressed genes can be found by

```
> fit <- lmFit(MA, design)
> cont.matrix <- makeContrasts(MUvsWT=WT-MU, levels=design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust="BH")
```

The results will be exactly the same as for the first approach.

The design matrix can be constructed

1. manually,
2. using the limma function `modelMatrix()`, or
3. using the built-in R function `model.matrix()`.

Let `Group` be the factor defined by

```
> Group <- factor(c("WT","WT","Mu","Mu","Mu"), levels=c("WT","Mu"))
```

For the first approach, the treatment-contrasts parametrization, the design matrix can be computed by

```
> design <- cbind(WTvsRef=1,MUvsWT=c(0,0,1,1,1))
```

or by

```
> param <- cbind(WTvsRef=c(-1,1,0),MUvsWT=c(0,-1,1))
> rownames(param) <- c("Ref","WT","Mu")
> design <- modelMatrix(targets, parameters=param)
```

or by

```
> design <- model.matrix(~Group)
> colnames(design) <- c("WTvsRef","MUvsWT")
```

all of which produce the same result. For the second approach, the group-means parametrization, the design matrix can be computed by

```
> design <- cbind(WT=c(1,1,0,0,0),MU=c(0,0,1,1,1))
```

or by

```
> param <- cbind(WT=c(-1,1,0),MU=c(-1,0,1))
> rownames(param) <- c("Ref","WT","Mu")
> design <- modelMatrix(targets, parameters=param)
```

or by

```
> design <- model.matrix(~0+Group)
> colnames(design) <- c("WT","Mu")
```

all of which again produce the same result.

8.5 Two Groups: Affymetrix

Suppose now that we wish to compare two wild type (Wt) mice with three mutant (Mu) mice using Affymetrix arrays or any other single-channel array technology:

FileName	Target
File1	WT
File2	WT
File3	Mu
File4	Mu
File5	Mu

Everything is exactly as in the previous section, except that the function `modelMatrix()` would not be used. We can either

1. create a design matrix which includes a coefficient for the mutant vs wild type difference, or
2. create a design matrix which includes separate coefficients for wild type and mutant mice and then extract the difference as a contrast.

For the first approach, the treatment-contrasts parametrization, the design matrix should be as follows:

```
> design

       WT MUvsWT
Array1  1      0
Array2  1      0
Array3  1      1
Array4  1      1
Array5  1      1
```

Here the first coefficient estimates the mean log-expression for wild type mice and plays the role of an intercept. The second coefficient estimates the difference between mutant and wild type. Differentially expressed genes can be found by

```
> fit <- lmFit(eset, design)
> fit <- eBayes(fit)
> topTable(fit, coef="MUvsWT", adjust="BH")
```

where `eset` is an `exprSet` or `matrix` object containing the log-expression values. For the second approach, the design matrix should be

```
       WT MU
Array1  1  0
Array2  1  0
Array3  0  1
Array4  0  1
Array5  0  1
```

Differentially expressed genes can be found by

```
> fit <- lmFit(eset, design)
> cont.matrix <- makeContrasts(MUvsWT=WT-MU, levels=design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust="BH")
```

For the first approach, the treatment-contrasts parametrization, the design matrix can be computed by

```
> design <- cbind(WT=1,MUvsWT=c(0,0,1,1,1))
```

or by

```
> design <- model.matrix(~Group)
> colnames(design) <- c("WT","MUvsWT")
```

For the second approach, the group-means parametrization, the design matrix can be computed by

```
> design <- cbind(WT=c(1,1,0,0,0,MU=c(0,0,1,1,1))
```

or by

```
> design <- model.matrix(~0+Group)
> colnames(design) <- c("WT","MU")
```

8.6 Several Groups

The above approaches for two groups extend easily to any number of groups. Suppose that three RNA targets to be compared using Affymetrix^{TM} arrays. Suppose that the three targets are called "RNA1", "RNA2" and "RNA3" and that the column `targets$Target` indicates which one was hybridized to each array. An appropriate design matrix can be created using

```
> f <- factor(targets$Target, levels=c("RNA1","RNA2","RNA3"))
> design <- model.matrix(~0+f)
> colnames(design) <- c("RNA1","RNA2","RNA3")
```

To make all pair-wise comparisons between the three groups one could proceed

```
> fit <- lmFit(eset, design)
> contrast.matrix <- makeContrasts(RNA2-RNA1, RNA3-RNA2, RNA3-RNA1,
+                                  levels=design)
> fit2 <- contrasts.fit(fit, contrast.matrix)
> fit2 <- eBayes(fit2)
```

A list of top genes for RNA2 versus RNA1 can be obtained from

```
> topTable(fit2, coef=1, adjust="BH")
```

The outcome of each hypothesis test can be assigned using

```
> results <- decideTests(fit2)
```

A Venn diagram showing numbers of genes significant in each comparison can be obtained from

```
> vennDiagram(results)
```

The statistic `fit2$F` and the corresponding `fit2$F.p.value` combine the three pair-wise comparisons into one F-test. This is equivalent to a one-way ANOVA for each gene except that the residual mean squares have been moderated between genes. To find genes which vary between the three RNA targets in any way, look for genes with small p-values. To find the top 30 genes:

```
> o <- order(fit2$F.p.value}
> fit2$genes[o[1:30],]
```

Now suppose that the experiment had been conducted using two-color arrays with a common reference instead of Affymetrix$^{\text{TM}}$ arrays. For example the targets frame might be

FileName	Cy3	Cy5
File1	Ref	RNA1
File2	RNA1	Ref
File3	Ref	RNA2
File4	RNA2	Ref
File5	Ref	RNA3

For this experiment the design matrix could be formed by

```
> design <- modelMatrix(targets, ref="Ref")
```

and everything else would be as for the Affymetrix$^{\text{TM}}$ experiment.

8.7 Factorial Designs

Factorial designs are those where more than one experimental dimension is being varied and each combination of treatment conditions is observed. Suppose that cells are extracted from wild type and mutant mice and these cells are either stimulated (S) or unstimulated (U). RNA from the treated cells is then extracted and hybridized to a microarray. We will assume for simplicity that the arrays are single-color arrays such as Affymetrix. Consider the following targets frame:

FileName	Strain	Treatment
File1	WT	U
File2	WT	S
File3	Mu	U
File4	Mu	S
File5	Mu	S

The two experimental dimensions or *factors* here are Strain and Treatment. Strain specifies the genotype of the mouse from which the cells are extracted and Treatment specifies whether the cells are stimulated or not. All four combinations of Strain and Treatment are observed, so this is a factorial design. It will be convenient for us to collect the Strain/Treatment combinations into one vector as follows:

```
> TS <- paste(targets$Strain, targets$Treatment, sep=".")
> TS

[1] "WT.U" "WT.S" "Mu.U" "Mu.S" "Mu.S"
```

It is especially important with a factorial design to decide what are the comparisons of interest. We will assume here that the experimenter is interested in

1. which genes respond to stimulation in wild-type cells,
2. which genes respond to stimulation in mutant cells, and
3. which genes respond differently in mutant compared to wild-type cells.

as these are the questions which are most usually relevant in a molecular biology context. The first of these questions relates to the `WT.S` vs `WT.U` comparison and the second to `Mu.S` vs `Mu.U`. The third relates to the difference of differences, i.e., `(Mu.S-Mu.U)-(WT.S-WT.U)`, which is called the *interaction* term.

We describe first a simple way to analyse this experiment using limma commands in a similar way to that in which two-sample designs were analyzed. Then we will go on to describe the more classical statistical approaches using factorial model formulas. All the approaches considered are equivalent and yield identical bottom-line results. The most basic approach is to fit a model with a coefficient for each of the four factor combinations and then to extract the comparisons of interest as contrasts:

```
> TS <- factor(TS, levels=c("WT.U","WT.S","Mu.U","Mu.S"))
> design <- model.matrix(~0+TS)
> colnames(design) <- levels(TS)
> fit <- lmFit(eset, design)
```

This fits a model with four coefficients corresponding to `WT.U`, `WT.S`, `Mu.U` and `Mu.S` respectively. Our three contrasts of interest can be extracted by

```
> cont.matrix <- makeContrasts(
+     WT.SvsU=WT.S-WT.U,
+     Mu.SvsU=Mu.S-Mu.U,
+     Diff=(Mu.S-Mu.U)-(WT.S-WT.U),
+     levels=design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

We can use `topTable()` to look at lists of differentially expressed genes for each of three contrasts, or else

```
> results <- decideTests(fit2)
> vennDiagram(results)
```

to look at all three contrasts simultaneously.

The analysis of factorial designs has a long history in statistics and a system of factorial *model formulas* has been developed to facilitate the analysis of complex designs. It is important to understand though that the above three molecular biology questions do not correspond to any of the usual parametrizations used in statistics for factorial designs. Suppose for example that we proceed in the usual statistical way,

```
> Strain <- factor(targets$Strain, levels=c("WT","Mu"))
> Treatment <- factor(targets$Treatment, levels=c("U","S"))
> design <- model.matrix(~Strain*Treatment)
```

This creates a design matrix which defines four coefficients with the following interpretations:

Coefficient	Comparison	Interpretation
Intercept	`WT.U`	Baseline level of unstimulated WT
StrainMu	`Mu.U-WT.U`	Difference between unstimulated strains
TreatmentS	`WT.S-WT.U`	Stimulation effect for WT
StrainMu:TreatmentS	`(Mu.S-Mu.U)-(WT.S-WT.U)`	Interaction

This is called the *treatment-contrast* parametrization. Notice that one of our comparisons of interest, `Mu.S-Mu.U`, is not represented and instead the comparison `Mu.U-WT.U`, which might not be of direct interest, is included. We need to use contrasts to extract all the comparisons of interest:

```
> fit <- lmFit(eset, design)
> cont.matrix <- cbind(WT.SvsU=c(0,0,1,0),Mu.SvsU=c(0,0,1,1),Diff=c(0,0,0,1))
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

This extracts the WT stimulation effect as the third coefficient and the interaction as the fourth coefficient. The mutant stimulation effect is extracted as the sum of the third and fourth coefficients of the original model. This analysis yields exactly the same results as the previous analysis.

An even more classical statistical approach to the factorial experiment would be to use the *sum to zero* parametrization. In R this is achieved by

```
> contrasts(Strain) <- contr.sum(2)
> contrasts(Treatment) <- contr.sum(2)
> design <- model.matrix(~Strain*Treatment)
```

This defines four coefficients with the following interpretations:

Coefficient	Comparison	Interpretation
Intercept	`(WT.U+WT.S+Mu.U+Mu.S)/4`	Grand mean
Strain1	`(WT.U+WT.S-Mu.U-Mu.S)/4`	Strain main effect
Treatment1	`(WT.U-WT.S+Mu.U-Mu.S)/4`	Treatment main effect
Strain1:Treatment1	`(WT.U-WT.S-Mu.U+Mu.S)/4`	Interaction

This parametrization has many appealing mathematical properties and is the classical parametrization used for factorial designs in much experimental design theory. However it defines only one coefficient which is directly of interest to us, namely the interaction. Our three contrasts of interest could be extracted using

```
> fit <- lmFit(eset, design)
> cont.matrix <- cbind(WT.SvsU=c(0,0,-2,-2),Mu.SvsU=c(0,0,-2,2),Diff=c(0,0,0,4))
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

The results will be identical to those for the previous two approaches.

The three approaches described here for the 2 × 2 factorial problem are equivalent and differ only in the parametrization chosen for the linear model. The three fitted model objects `fit` will differ only in the `coefficients` and associated components. The residual standard deviations `fit$sigma`, residual degrees of freedom `fit$df.residual` and all components of `fit2` will be identical for the three approaches. Since the three approaches are equivalent, users are free to choose whichever one is most convenient or intuitive.

8.8 Time Course Experiments

Time course experiments are those in which RNA is extracted at several time points after the onset of some treatment or stimulation. Simple time course experiments are similar to experiments with several groups covered in Section 8.6. Here we consider a two-way experiment in which time course profiles are to be compared for two genotypes. Consider the targets frame

FileName	Target
File1	wt.0hr
File2	wt.0hr
File3	wt.6hr
File4	wt.24hr
File5	mu.0hr
File6	mu.0hr
File7	mu.6hr
File8	mu.24hr

The targets are RNA samples collected from wild-type and mutant animals at 0, 6 and 24 hour time points. This can be viewed as a factorial experiment but a simpler approach is to use the group-mean parametrization.

```
> lev <- c("wt.0hr","wt.6hr","wt.24hr","mu.0hr","mu.6hr","mu.24hr")
> f <- factor(targets$Target, levels=lev)
> design <- model.matrix(~0+f)
> colnames(design) <- lev
> fit <- lmFit(eset, design)
```

Which genes respond at either the 6 hour or 24 hour times in the wild-type? We can find these by extracting the contrasts between the wild-type times.

```
> cont.wt <- makeContrasts(
+      "wt.6hr-wt.0hr",
+      "wt.24hr-wt.6hr",
+ levels=design)
> fit2 <- contrasts.fit(fit, cont.wt)
> fit2 <- eBayes(fit2)
```

Choose genes so that the expected false discovery rate is less than 5%.

```
> selected.wt <- p.adjust(fit2$F.p.value, method="BH") < 0.05
```

Any two contrasts between the three times would give the same result. The same gene list would be obtained had `"wt.24hr-wt.0hr"` been used in place of `"wt.24hr-wt.6hr"` for example.

Which genes respond in the mutant?

```
> cont.mu <- makeContrasts(
+      "mu.6hr-mu.0hr",
+      "mu.24hr-wt.6hr",
+ levels=design)
> fit2 <- contrasts.fit(fit, cont.mu)
> fit2 <- eBayes(fit2)
> selected.mu <- p.adjust(fit2$F.p.value, method="BH") < 0.05
```

Which genes respond *differently* in the mutant relative to the wild-type?

```
> cont.dif <- makeContrasts(
+     Dif6hr =(mu.6hr-mu.0hr)-(wt.6hr-wt.0hr),
+     Dif24hr=(mu.24hr-mu.6hr)-(wt.24hr-wt.6hr),
+ levels=design)
> fit2 <- contrasts.fit(fit, cont.dif)
> fit2 <- eBayes(fit2)
> selected.dif <- p.adjust(fit2$F.p.value, method="BH") < 0.05
```

The method of analysis described in this section was used for a six-point time course experiment on histone deacetylase inhibitors [13].

Chapter 9

Separate Channel Analysis of Two-Color Data

Consider an experiment comparing young and old animals for both both wild-type and mutant genotypes.

FileName	Cy3	Cy5
File1	wt.young	wt.old
File2	wt.old	wt.young
File3	mu.young	mu.old
File4	mu.old	mu.young

Each of the arrays in this experiment makes a direct comparison between young and old RNA targets. There are no arrays which compare wild-type and mutant animals. This is an example of an *unconnected* design in that there are no arrays linking the wild-type and mutant targets. It is not possible to make comparisons between wild-type and mutant animals on the basis of log-ratios alone. So to do this it is necessary to analyse the red and green channels intensities separately, i.e., to analyze log-intensities instead of log-ratios. It is possible to do this using a mixed model representation which treats each spot as a randomized block [26, 18]. Limma implements mixed model methods for separate channel analysis which make use of shrinkage methods to ensure stable and reliable inference with small numbers of arrays [18]. Limma also provides between-array normalization to prepare for separate channel analysis, for example

```
> MA <- normalizeBetweenArrays(MA, method="Aquantile")
```

scales the intensities so that A-values have the same distribution across arrays.

The first step in the differential expression analysis is to convert the targets frame to be channel rather than array orientated.

```
> targets2 <- targetsA2C(targets)
> targets2
```

```
        channel.col FileName   Target
File1.1           1    File1 wt.young
```

```
File1.2          2    File1   wt.old
File2.1          1    File2   wt.old
File2.2          2    File2 wt.young
File3.1          1    File3 mu.young
File3.2          2    File3   mu.old
File4.1          1    File4   mu.old
File4.2          2    File4 mu.young
```

The following code produces a design matrix with eight rows and four columns:

```
> u <- unique(targets2$Target)
> f <- factor(targets2$Target, levels=u)
> design <- model.matrix(~0+f)
> colnames(design) <- u
```

Inference proceeds as for within-array replicate spots except that the correlation to be estimated is that between the two channels for the same spot rather than between replicate spots.

```
> corfit <- intraspotCorrelation(MA, design)
> fit <- lmscFit(MA, design, correlation=corfit$consensus)
```

Subsequent steps proceed as for log-ratio analyses. For example if we want to campare wild-type young to mutant young animals, we could extract this contrast by

```
> cont.matrix <- makeContrasts("mu.young-wt.young",levels=design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust="BH")
```

Chapter 10

Statistics for Differential Expression

10.1 Output from eBayes

A number of summary statistics are computed by the `eBayes()` function for each gene and each contrast. The M-value (`M`) is the contrast estimate. For example, if the contrast compares two treatment conditions, the M-value is the $\log_2$-fold-change between those two conditions. The A-value (`A`) is the the mean log-expression level for that gene across all channels and all arrays in the linear model fit. Note that the A-value is the mean log-expression over all arrays in the experiment, even if only a smaller number of arrays are involved in the contrast of interest. The moderated t-statistic (`t`) is the ratio of the M-value to its standard error. This has the same interpretation as an ordinary t-statistic except that the standard errors have been moderated across genes, i.e., shrunk towards a common value [21]. This has the effect of borrowing information from the ensemble of genes to aid with inference about each individual gene. The ordinary t-statistics are not recommended, but they can be recovered by

```
> tstat.ord <- fit$coef / fit$stdev.unscaled / fit$sigma
```

after fitting a linear model. The ordinary t-statistic is on `fit$df.residual` degrees of freedom while the moderated t-statistic is on `fit$df.residual+fit$df.prior` degrees of freedom. The p-value (`p-value`) is obtained from the distribution of the moderated t-statistic.

The B-statistic (`lods` or `B`) is the log-odds that that gene is differentially expressed. Suppose for example that B=1.5. The odds of differential expression is exp(1.5)=4.48, i.e, about four and a half to one. The probability that the gene is differentially expressed is 4.48/(1+4.48)=0.82, i.e., the probability is about 82% that this gene is differentially expressed. A B-statistic of zero corresponds to a 50-50 chance that the gene is differentially expressed. The B-statistic is automatically adjusted for multiple testing by assuming that 1% of the genes, or some other percentage specified by the user, are expected to be differentially expressed. If there are no missing values in your data, then the moderated t and B statistics will rank the genes in exactly the same order. Even you do have spot weights or missing data, the p-values and B-statistics will usually provide a very similar ranking of the genes.

Please keep in mind that the moderated t-statistic p-values and the B-statistic probabilities depend on various sorts of mathematical assumptions which are never exactly true for microarray data. The B-statistics also depend on a prior guess for the proportion of differentially expressed genes. Therefore they are intended to be taken as a guide rather than as a strict measure of the probability of differential expression. Of the three statistics, the moderated-t, the associated p-value and the B-statistics, we usually base our gene selections on the p-value. All three measures are closely related, but the moderated-t and its p-value do not require a prior guess for the number of differentially expressed genes.

The above mentioned statistics are computed for every contrast for each gene. The `eBayes()` function computes one more useful statistic. The moderated F-statistic (F) combines the t-statistics for all the contrasts into an overall test of significance for that gene. The moderated F-statistic tests whether any of the contrasts are non-zero for that gene, i.e., whether that gene is differentially expressed on any contrast. The moderated-F has numerator degrees of freedom equal to the number of contrasts and denominator degrees of freedom the same as the moderated-t. Its p-value is stored as `fit$F.p.value`. It is similar to the ordinary F-statistic from analysis of variance except that the denominator mean squares are moderated across genes.

In a complex experiment with many contrasts, it may be desirable to select genes firstly on the basis of their moderated F-statistics, and subsequently to decide which of the individual contrasts are significant for the selected genes. This cuts down on the number of tests which need to be conducted and therefore on the amount of adjustment for multiple testing. The function `decideTests()` with `method="nestedF"` is able to conduct such tests.

10.2 Adjusting for Multiple Testing

The p-values computed by `eBayes()` are not adjusted for multiple testing. Such adjustment is done by the functions `topTable()` and `decideTests()`. The functions give access to the methods implemented in the R function `p.adjust()`, notably to Holm's step-down Bonferroni method (`adjust="holm"`) and to Benjamini and Hochberg's (1995) step-up method for controlling the false discovery rate (`adjust="BH"`). Benjamini and Hochberg's method is the most popular in the microarray context. The approach adopted by `p.adjust()` and by limma is to implement the adjustment methods through adjusted p-values. The meaning of the adjusted p-value is as follows. If you wish to control the false discovery rate to be less than a certain value q, say 0.05, then select as differentially expressed all those genes which have adjusted p-value less than q. This procedure is equivalent to the procedure of Benjamini and Hochberg (1995), although that procedure is not formulated in terms of adjusted p-values in the original paper.

Note that Benjamini and Hochberg's method assumes that the p-values are independent across probes, whereas they are likely to be somewhat dependent as a result of being based on observations made on the same set of arrays. Reiner et al [14] have argued that Benjamini and Hochberg's method does continue to control the false discovery rate for a range of dependence structures including positive dependence.

It may be found in some cases that none of the adjusted p-values are small even though many of the raw p-values seem highly significant. If none of the raw p-value are less than $1/G$, where G is the number of genes included in the fit, then all of the adjusted p-values will the equal to 1. (if `fit` is an `MArrayLM` object, then G is given by `nrow(fit)`.) This is true for any of the adjustment methods except for `adjust="none"`. Since $1/G$ is about the expected size of the smallest p-values given purely random variation and uniform p-values, this means that there is no overall evidence of differential expression. If G is large this may be true even if many of the raw p-values would be highly significant taken as individual values.

Chapter 11

Case Studies

11.1 Swirl Zebrafish: A Single-Sample Experiment

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.

In this example we assume that the data is provided as a GAL file called `fish.gal` and raw SPOT output files and that these files are in the current working directory. The data used for this case study can be downloaded from `http://bioinf.wehi.edu.au/limmaGUI/DataSets.html`.

```
> dir()
[1] "fish.gal"       "swirl.1.spot"    "swirl.2.spot"    "swirl.3.spot"    "swirl.4.spot"
[6] "SwirlSample.txt"
```

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 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 targets file called `SwirlSamples.txt` which describes the four hybridizations:

```
> library(limma)
> targets <- readTargets("SwirlSample.txt")
> targets
        SlideNumber     FileName       Cy3       Cy5      Date
swirl.1          81 swirl.1.spot     swirl wild type 2001/9/20
swirl.2          82 swirl.2.spot wild type     swirl 2001/9/20
swirl.3          93 swirl.3.spot     swirl wild type 2001/11/8
swirl.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.

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 under `FileName`. Now we read the intensity data into an `RGList` object in R. The default for SPOT output is that `Rmean` and `Gmean` are used as foreground intensities and `morphR` and `morphG` are used as background intensities:

```
> RG <- read.maimages(targets$FileName, source="spot")
Read swirl.1.spot
Read swirl.2.spot
Read swirl.3.spot
Read swirl.4.spot
> RG
An object of class "RGList"
$R
       swirl.1   swirl.2   swirl.3    swirl.4
[1,] 19538.470 16138.720 2895.1600 14054.5400
[2,] 23619.820 17247.670 2976.6230 20112.2600
[3,] 21579.950 17317.150 2735.6190 12945.8500
[4,]  8905.143  6794.381  318.9524   524.0476
[5,]  8676.095  6043.542  780.6667   304.6190
8443 more rows ...

$G
       swirl.1   swirl.2   swirl.3    swirl.4
[1,] 22028.260 19278.770 2727.5600 19930.6500
[2,] 25613.200 21438.960 2787.0330 25426.5800
[3,] 22652.390 20386.470 2419.8810 16225.9500
[4,]  8929.286  6677.619  383.2381   786.9048
[5,]  8746.476  6576.292  901.0000   468.0476
8443 more rows ...

$Rb
     swirl.1 swirl.2 swirl.3 swirl.4
[1,]     174     136      82      48
[2,]     174     133      82      48
[3,]     174     133      76      48
[4,]     163     105      61      48
[5,]     140     105      61      49
8443 more rows ...

$Gb
     swirl.1 swirl.2 swirl.3 swirl.4
[1,]     182     175      86      97
[2,]     171     183      86      85
[3,]     153     183      86      85
[4,]     153     142      71      87
[5,]     153     142      71      87
8443 more rows ...
```

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:

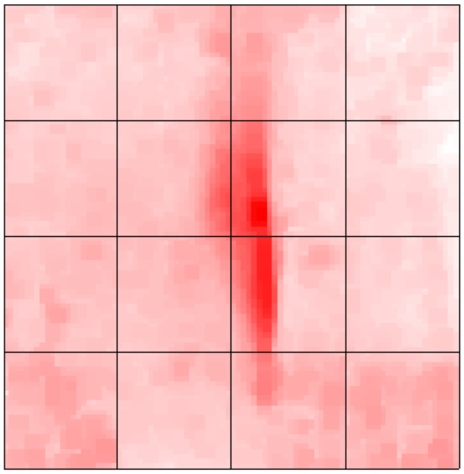
```
> RG$genes <- readGAL("fish.gal")
> RG$genes[1:30,]

   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
```

Because we are using SPOT output, the 4x4x22x24 print layout also needs to be set. The easiest way to do this is to infer it from the GAL file:

```
> RG$printer <- getLayout(RG$genes)
```

Image plots. It is interesting to look at the variation of background values over the array. Consider image plots of the red and green background for the first array:

```
> imageplot(log2(RG$Rb[,1]), RG$printer, low="white", high="red")
> imageplot(log2(RG$Gb[,1]), RG$printer, low="white", high="green")
```



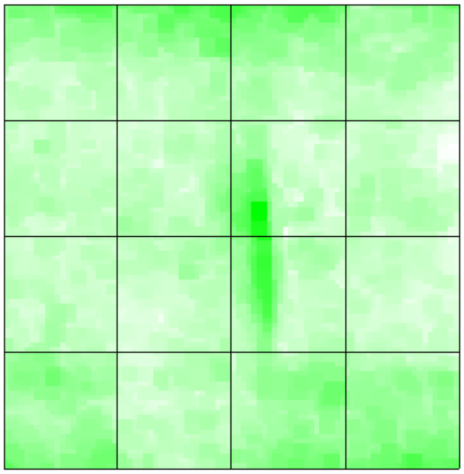
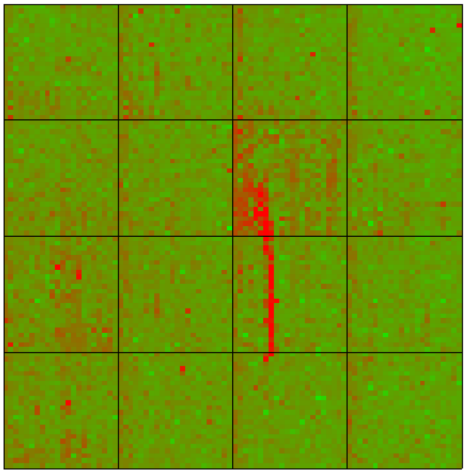


Image plot of the un-normalized log-ratios or M-values for the first array:

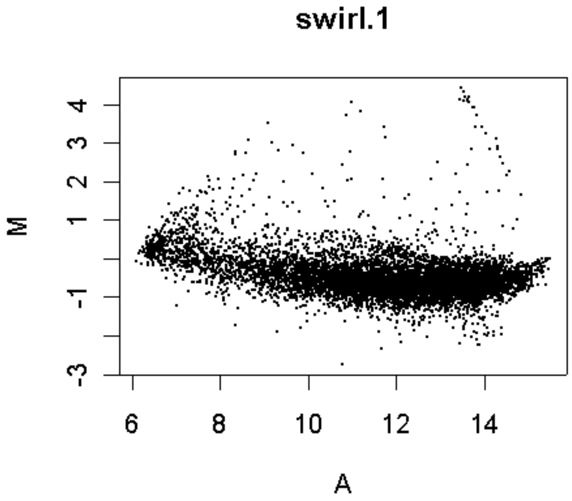
```
> MA <- normalizeWithinArrays(RG, method="none")
> imageplot(MA$M[,1], RG$printer, zlim=c(-3,3))
```



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 caused by a scratch or dust on the array. Spots which are affected by this artefact will have suspect M-values. The streak also shows up as darker regions in the background plots.

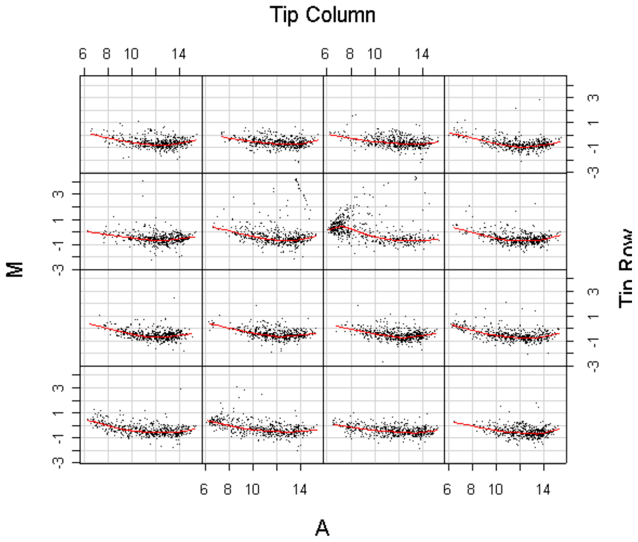
MA-plots. An MA-plot plots the log-ratio of R vs G against the overall intensity of each spot. The log-ratio is represented by the M-value, $M = \log_2(R) - \log_2(G)$, and the overall intensity by the A-value, $A = (\log_2(R) + \log_2(G))/2$. Here is the MA-plot of the un-normalized values for the first array:

```
> plotMA(MA)
```



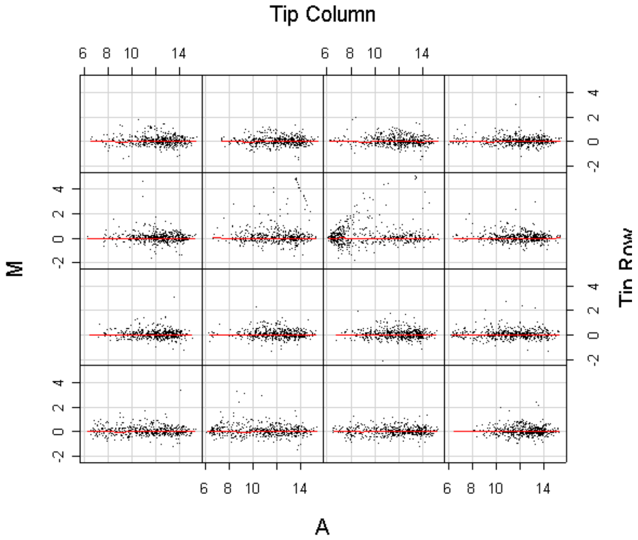
The red streak seen on the image plot can be seen as a line of spots in the upper right of this plot. Now we plot the individual MA-plots for each of the print-tip groups on this array, together with the loess curves which will be used for normalization:

```
> plotPrintTipLoess(MA)
```



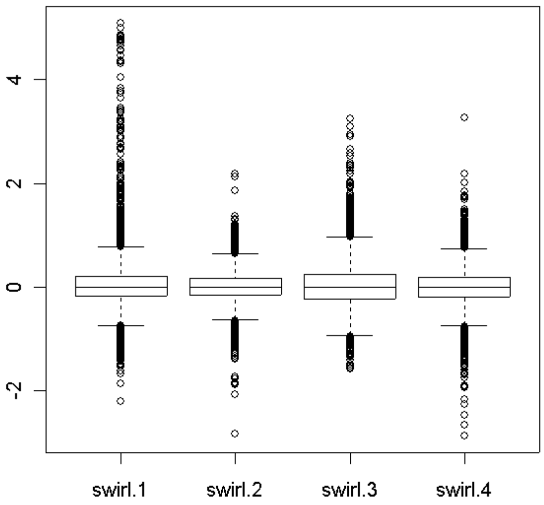
Normalization. Print-tip loess normalization:

```
> MA <- normalizeWithinArrays(RG)
> plotPrintTipLoess(MA)
```



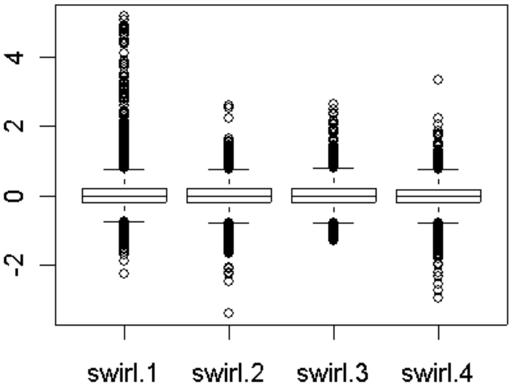
We have normalized the M-values with each array. A further question is whether normalization is required between the arrays. The following plot shows overall boxplots of the M-values for the four arrays.

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



There is evidence that the different arrays have different spreads of M-values, so we will scale normalize between the arrays. (Note this is not done routinely for all two-color data sets. It should only be done when there is good evidence of a scale difference.)

```
> MA <- normalizeBetweenArrays(MA,method="scale")
> boxplot(MA$M~col(MA$M),names=colnames(MA$M))
```



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 <- lmFit(MA,design)
> fit
An object of class "MArrayLM"
$coefficients
[1] -0.3943421 -0.3656843 -0.3912506 -0.2505729 -0.3432590
8443 more rows ...

$rank
[1] 1

$assign
NULL

$qr
$qr
     [,1]
[1,]  2.0
[2,] -0.5
[3,]  0.5
[4,] -0.5

$qraux
[1] 1.5

$pivot
[1] 1

$tol
[1] 1e-07

$rank
[1] 1

attr(,"class")
[1] "qr"

$df.residual
[1] 3 3 3 3 3
8443 more elements ...

$sigma
[1] 0.3805154 0.4047829 0.4672451 0.3206071 0.2838043
8443 more elements ...

$cov.coefficients
     [,1]
[1,] 0.25

$stdev.unscaled
[1] 0.5 0.5 0.5 0.5 0.5
8443 more rows ...
```

```
$pivot
[1] 1

$method
[1] "ls"

$design
     [,1]
[1,]   -1
[2,]    1
[3,]   -1
[4,]    1

$genes
  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
8443 more rows ...

$Amean
[1] 13.51784 13.72765 13.48607 10.89367 10.97615
8443 more elements ...
```

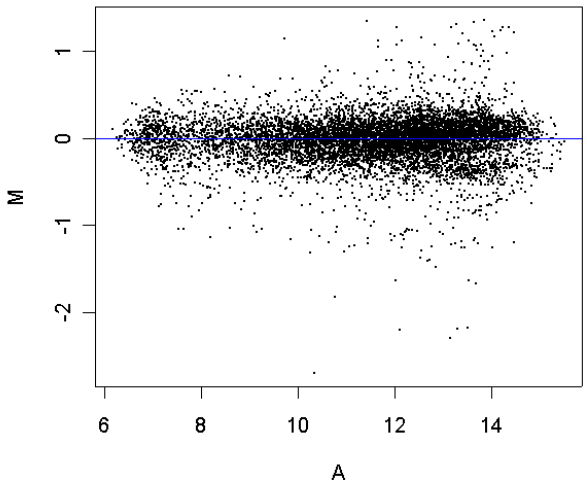
In the above fit object, `coefficients` is the average M-value for each gene and `sigma` is the sample standard deviations for each gene. Ordinary t-statistics for comparing mutant to wt could be computed by

```
> ordinary.t <- fit$coef / fit$stdev.unscaled / fit$sigma
```

We prefer though to use empirical Bayes moderated t-statistics which are computed below.

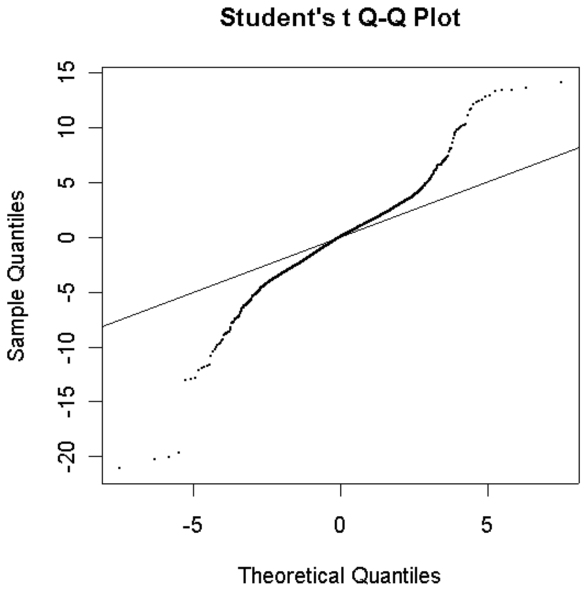
Now create an MA-plot of the average M and A-values for each gene.

```
> plotMA(fit)
> 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.

```
> fit <- eBayes(fit)
> qqt(fit$t,df=fit$df.prior+fit$df.residual,pch=16,cex=0.2)
> abline(0,1)
```



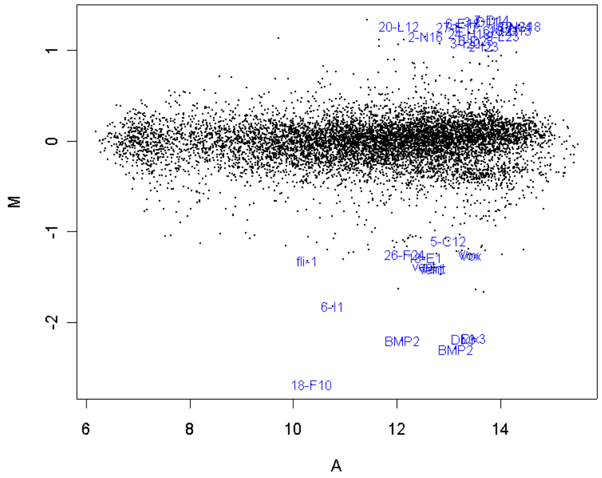
Visually there seems to be plenty of genes which are differentially expressed. We will obtain a summary table of some key statistics for the top genes.

```
> options(digits=3)
> topTable(fit,number=30,adjust="BH")
     Block Row Column      ID   Name     M    A     t  P.Value    B
3721     8   2      1 control   BMP2 -2.21 12.1 -21.0 0.000333 8.02
1609     4   2      1 control   BMP2 -2.30 13.1 -20.3 0.000333 7.84
3723     8   2      3 control   Dlx3 -2.18 13.3 -20.0 0.000333 7.77
1611     4   2      3 control   Dlx3 -2.18 13.5 -19.6 0.000333 7.68
8295    16  16     15 fb94h06 20-L12  1.27 12.0  14.0 0.001993 5.81
7036    14   8      4 fb40h07  7-D14  1.35 13.8  13.5 0.001993 5.57
515      1  22     11 fc22a09 27-E17  1.27 13.2  13.4 0.001993 5.52
5075    10  14     11 fb85f09 18-G18  1.28 14.4  13.3 0.001993 5.51
7307    14  19     11 fc10h09 24-H18  1.20 13.4  13.2 0.001993 5.43
319      1  14      7 fb85a01  18-E1 -1.29 12.5 -13.0 0.001993 5.36
2961     6  14      9 fb85d05 18-F10 -2.69 10.3 -13.0 0.001993 5.35
4032     8  14     24 fb87d12 18-N24  1.27 14.2  12.8 0.001993 5.25
6903    14   2     15 control    Vox -1.26 13.4 -12.8 0.001993 5.23
4546     9  14     10 fb85e07 18-G13  1.23 14.2  12.7 0.001993 5.21
683      2   7     11 fb37b09  6-E18  1.31 13.3  12.4 0.002087 5.06
1697     4   5     17 fb26b10  3-I20  1.09 13.3  12.3 0.002087 4.99
7491    15   5      3 fb24g06  3-D11  1.33 13.6  12.3 0.002087 4.99
4188     8  21     12 fc18d12 26-F24 -1.25 12.1 -12.2 0.002122 4.93
4380     9   7     12 fb37e11  6-G21  1.23 14.0  12.0 0.002133 4.84
3726     8   2      6 control  fli-1 -1.32 10.3 -11.9 0.002133 4.80
2679     6   2     15 control    Vox -1.25 13.4 -11.8 0.002133 4.75
2151     5   2     15 control   vent -1.40 12.7 -11.7 0.002133 4.67
7602    15   9     18 fb50g12  9-L23  1.16 14.0  11.7 0.002133 4.66
5931    12   6      3 fb32f06  5-C12 -1.10 13.0 -11.7 0.002133 4.66
3790     8   4     22 fb23d08  2-N16  1.16 12.5  11.6 0.002144 4.61
7542    15   7      6 fb36g12  6-D23  1.12 13.5  11.0 0.002900 4.29
4263     9   2     15 control   vent -1.41 12.7 -10.8 0.003160 4.18
6375    13   2     15 control   vent -1.37 12.5 -10.5 0.003829 3.96
157      1   7     13 fb38a01   6-I1 -1.82 10.8 -10.2 0.004124 3.81
1146     3   4     18 fb22a12  2-I23  1.05 13.7  10.2 0.004124 3.78
```

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 30 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.

```
> plotMA(fit)
> ord <- order(fit$lods,decreasing=TRUE)
> top30 <- ord[1:30]
> text(fit$Amean[top30],fit$coef[top30],labels=fit$genes[top30,"Name"],cex=0.8,col="blue")
```



11.2 ApoAI Knockout Data: A Two-Sample Experiment

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.

In this example we assume that the data is available as an RGList in the data file `ApoAI.RData`. The data used for this case study can be downloaded from `http://bioinf.wehi.edu.au/limmaGUI/DataSets.html`.

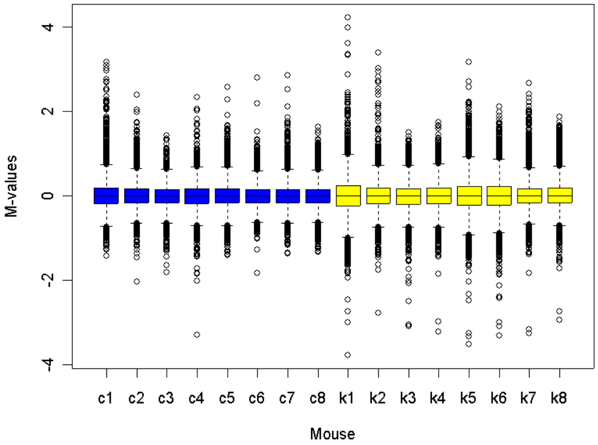
Background. The data is from a study of lipid metabolism by [3]. The apolipoprotein AI (ApoAI) gene is known to play a pivotal role in high density lipoprotein (HDL) metabolism. Mice which have the ApoAI gene knocked out have very low HDL cholesterol levels. The purpose of this experiment is to determine 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] "RG"
> names(RG)
[1] "R" "G" "Rb" "Gb" "printer" "genes" "targets"
> RG$targets
      FileName  Cy3     Cy5
c1 a1koc1.spot Pool C57BL/6
c2 a1koc2.spot Pool C57BL/6
c3 a1koc3.spot Pool C57BL/6
c4 a1koc4.spot Pool C57BL/6
c5 a1koc5.spot Pool C57BL/6
c6 a1koc6.spot Pool C57BL/6
c7 a1koc7.spot Pool C57BL/6
c8 a1koc8.spot Pool C57BL/6
k1 a1kok1.spot Pool ApoAI-/-
k2 a1kok2.spot Pool ApoAI-/-
k3 a1kok3.spot Pool ApoAI-/-
k4 a1kok4.spot Pool ApoAI-/-
k5 a1kok5.spot Pool ApoAI-/-
k6 a1kok6.spot Pool ApoAI-/-
k7 a1kok7.spot Pool ApoAI-/-
k8 a1kok8.spot Pool ApoAI-/-
> MA <- normalizeWithinArrays(RG)
> cols <- MA$targets$Cy5
> cols[cols=="C57BL/6"] <- "blue"
> cols[cols=="ApoAI-/-"] <- "yellow"
> boxplot(MA$M~col(MA$M),names=rownames(MA$targets),col=cols,xlab="Mouse",ylab="M-values")
```



Since the common reference here is a pool of the control mice, we expect to see more differences from the pool for the knock-out mice than for the control mice. In terms of the above plot, this should translate into a wider range of M-values for the knock-out mice arrays than for the control arrays, and we do see this. Since the different arrays are not expected to have the same range of M-values, between-array scale normalization of the M-values is not appropriate here.

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 <- cbind("Control-Ref"=1,"KO-Control"=MA$targets$Cy5=="ApoAI-/-")
> design
      Control-Ref KO-Control
 [1,]           1          0
 [2,]           1          0
 [3,]           1          0
 [4,]           1          0
 [5,]           1          0
 [6,]           1          0
 [7,]           1          0
 [8,]           1          0
 [9,]           1          1
[10,]           1          1
[11,]           1          1
[12,]           1          1
[13,]           1          1
[14,]           1          1
[15,]           1          1
[16,]           1          1
> fit <- lmFit(MA, 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
> fit <- eBayes(fit)
> options(digits=3)
```

Normally at this point one would just type

```
> topTable(fit,coef=2)
```

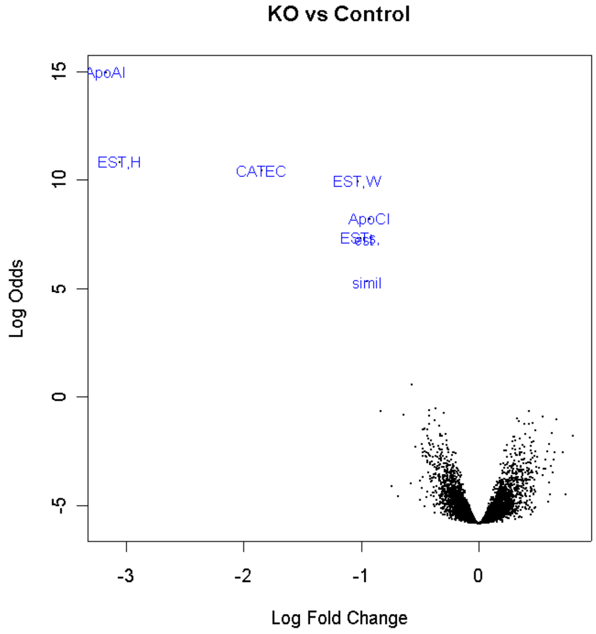
However, the gene annotation is a bit wide for the printed page, so we will tell codetopTable() to show just one column of the annotation information:

```
> topTable(fit,coef=2,number=15,genelist=fit$genes$NAME)
                 ProbeID      M     A      t  P.Value      B
2149      ApoAI,lipid-Img -3.166 12.47 -23.98 2.98e-11 14.939
540  EST,HighlysimilartoA -3.049 12.28 -12.97 4.93e-07 10.825
5356 CATECHOLO-METHYLTRAN -1.848 12.93 -12.44 6.42e-07 10.457
4139 EST,WeaklysimilartoC -1.027 12.61 -11.75 1.21e-06  9.928
```

```
1739    ApoCIII,lipid-Img -0.933 13.74  -9.83 1.57e-05  8.192
2537 ESTs,Highlysimilarto -1.010 13.63  -9.01 4.21e-05  7.307
1496                  est -0.977 12.23  -9.00 4.21e-05  7.292
4941 similartoyeaststerol -0.955 13.29  -7.44 5.59e-04  5.314
947  EST,WeaklysimilartoF -0.571 10.54  -4.55 1.77e-01  0.563
5604                      -0.366 12.71  -3.96 5.27e-01 -0.558
4140          APXL2,5q-Img -0.420  9.79  -3.93 5.27e-01 -0.621
6073           estrogenrec  0.421  9.79   3.91 5.27e-01 -0.654
1337 psoriasis-associated -0.838 11.66  -3.89 5.27e-01 -0.683
954    Caspase7,heart-Img -0.302 12.14  -3.85 5.29e-01 -0.765
563  FATTYACID-BINDINGPRO -0.637 11.62  -3.81 5.29e-01 -0.837
```

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.

```
> volcanoplot(fit,coef=2,highlight=8,names=fit$genes$NAME,main="KO vs Control")
```



11.3 Ecoli Lrp Data: Affymetrix Data with Two Targets

The data are from experiments reported in [8] and are available from the www site `http://visitor.ics.uci.edu/genex/cybert/tutorial/index.html`. The data is also available from the ecoliLeucine data package available from the Bioconductor www site under ”Experimental Data”. Hung et al [8] state that

> The purpose of the work presented here is to identify the network of genes that are differentially regulated by the global *E. coli* regulatory protein, leucine-responsive regulatory protein (Lrp), during steady state growth in a glucose supplemented minimal salts medium. Lrp is a DNA-binding protein that has been reported to affect the expression of approximately 55 genes.

Gene expression in two *E. coli* bacteria strains, labelled lrp+ and lrp-, were compared using eight Affymetrix ecoli chips, four chips each for lrp+ and lrp-.

The following code assumes that the data files for the eight chips are in your current working directory.

```
> dir()
[1] "Ecoli.CDF"           "nolrp_1.CEL"          "nolrp_2.CEL"
[4] "nolrp_3.CEL"         "nolrp_4.CEL"          "wt_1.CEL"
[7] "wt_2.CEL"            "wt_3.CEL"             "wt_4.CEL"
```

The data is read and normalized using the affy package. The package `ecolicdf` must also be installed, otherwise the `rma()` function will attempt to download and install it for you—without giving you to opportunity to veto the download.

```
> library(limma)
> library(affy)
Welcome to Bioconductor
         Vignettes contain introductory material.  To view,
         simply type: openVignette()
         For details on reading vignettes, see
         the openVignette help page.
> Data <- ReadAffy()
> eset <- rma(Data)
Background correcting
Normalizing
Calculating Expression
> pData(eset)
            sample
nolrp_1.CEL      1
nolrp_2.CEL      2
nolrp_3.CEL      3
nolrp_4.CEL      4
wt_1.CEL         5
wt_2.CEL         6
wt_3.CEL         7
wt_4.CEL         8
```

Now we consider differential expression between the lrp+ and lrp- strains.

```
> strain <- c("lrp-","lrp-","lrp-","lrp-","lrp+","lrp+","lrp+","lrp+")
> design <- model.matrix(~factor(strain))
> colnames(design) <- c("lrp-","lrp+vs-")
> design
  lrp- lrp+vs-
1    1       0
```

```
2    1       0
3    1       0
4    1       0
5    1       1
6    1       1
7    1       1
8    1       1
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$"factor(strain)"
[1] "contr.treatment"
```

The first coefficient measures $\log_2$-expression of each gene in the lrp- strain. The second coefficient measures the $\log_2$-fold change of lrp+ over lrp-, i.e., the log-fold change induced by lrp.

```
> fit <- lmFit(eset, design)
> fit <- eBayes(fit)
> options(digits=2)
> topTable(fit, coef=2, n=40, adjust="BH")
```

```
                         ProbeSetID     M    A     t P.Value      B
4282  IG_821_1300838_1300922_fwd_st -3.32 12.4 -23.1 5.3e-05  8.017
5365                  serA_b2913_st  2.78 12.2  15.8 6.0e-04  6.603
1389                  gltD_b3213_st  3.03 10.9  13.3 1.6e-03  5.779
4625                   lrp_b0889_st  2.30  9.3  11.4 4.0e-03  4.911
1388                  gltB_b3212_st  3.24 10.1  11.1 4.0e-03  4.766
4609                  livK_b3458_st  2.35  9.9  10.8 4.0e-03  4.593
4901                  oppB_b1244_st -2.91 10.7 -10.6 4.0e-03  4.504
4903                  oppD_b1246_st -1.94 10.4 -10.5 4.0e-03  4.434
5413                  sodA_b3908_st  1.50 10.3   9.7 6.5e-03  3.958
4900                  oppA_b1243_st -2.98 13.0  -9.1 9.2e-03  3.601
5217                   rmf_b0953_st -2.71 13.6  -9.0 9.3e-03  3.474
7300                  ytfK_b4217_st -2.64 11.1  -8.9 9.3e-03  3.437
5007                  pntA_b1603_st  1.58 10.1   8.3 1.4e-02  3.019
4281  IG_820_1298469_1299205_fwd_st -2.45 10.7  -8.1 1.6e-02  2.843
4491                  ilvI_b0077_st  0.95 10.0   7.4 2.9e-02  2.226
5448                  stpA_b2669_st  1.79 10.0   7.4 2.9e-02  2.210
611                        b2343_st -2.12 10.8  -7.1 3.4e-02  2.028
5930                  ybfA_b0699_st -0.91 10.5  -7.0 3.5e-02  1.932
1435                  grxB_b1064_st -0.91  9.8  -6.9 3.8e-02  1.810
4634                  lysU_b4129_st -3.30  9.3  -6.9 3.9e-02  1.758
4829                   ndk_b2518_st  1.07 11.1   6.7 4.3e-02  1.616
2309 IG_1643_2642304_2642452_rev_st  0.83  9.6   6.7 4.3e-02  1.570
4902                  oppC_b1245_st -2.15 10.7  -6.3 5.9e-02  1.238
4490                  ilvH_b0078_st  1.11  9.9   5.9 8.8e-02  0.820
1178                  fimA_b4314_st  3.40 11.7   5.9 8.8e-02  0.743
6224                  ydgR_b1634_st -2.35  9.8  -5.8 8.8e-02  0.722
4904                  oppF_b1247_st -1.46  9.9  -5.8 8.8e-02  0.720
792                        b3914_st -0.77  9.5  -5.7 1.0e-01  0.565
5008                  pntB_b1602_st  1.47 12.8   5.6 1.0e-01  0.496
```

```
4610                livM_b3456_st  1.04  8.5   5.5 1.1e-01  0.376
5097                ptsG_b1101_st  1.16 12.2   5.5 1.1e-01  0.352
4886                nupC_b2393_st  0.79  9.6   5.5 1.1e-01  0.333
4898                ompT_b0565_st  2.67 10.5   5.4 1.2e-01  0.218
5482                 tdh_b3616_st -1.61 10.5  -5.3 1.3e-01  0.092
1927       IG_13_14080_14167_fwd_st -0.55  8.4  -5.3 1.3e-01  0.076
6320                yeeF_b2014_st  0.88  9.9   5.3 1.3e-01  0.065
196                 atpG_b3733_st  0.60 12.5   5.2 1.4e-01 -0.033
954                 cydB_b0734_st -0.76 11.0  -5.0 1.8e-01 -0.272
1186                fimI_b4315_st  1.15  8.3   5.0 1.8e-01 -0.298
4013     IG_58_107475_107629_fwd_st -0.49 10.4  -4.9 2.0e-01 -0.407
```

The column M gives the $\log_2$-fold change while the column A gives the average $\log_2$-intensity for the probe-set. Positive M-values mean that the gene is up-regulated in lrp+, negative values mean that it is repressed.

It is interesting to compare this table with Tables III and IV in [8]. Note that the top-ranked gene is an intergenic region (IG) tRNA gene. The knock-out gene itself is in position four. Many of the genes in the above table, including the ser, glt, liv, opp, lys, ilv and fim families, are known targets of lrp.

11.4 Estrogen Data: A 2x2 Factorial Experiment with Affymetrix Arrays

This data is from the estrogen package on Bioconductor. A subset of the data is also analysed in the factDesign package vignette. To repeat this case study you will need to have the R packages affy, estrogen and hgu95av2cdf installed.

The data gives results from a 2x2 factorial experiment on MCF7 breast cancer cells using Affymetrix HGU95av2 arrays. The factors in this experiment were estrogen (present or absent) and length of exposure (10 or 48 hours). The aim of the study is the identify genes which respond to estrogen and to classify these into early and late responders. Genes which respond early are putative direct-target genes while those which respond late are probably downstream targets in the molecular pathway.

First load the required packages:

```
> library(limma)
> library(affy)
Welcome to Bioconductor
         Vignettes contain introductory material.  To view,
         simply type: openVignette()
         For details on reading vignettes, see
         the openVignette help page.
> library(hgu95av2cdf)
```

The data files are contained in the `extdata` directory of the estrogen package:

```
> datadir <- file.path(.find.package("estrogen"),"extdata")
> dir(datadir)
```

```
 [1] "00Index"       "bad.cel"       "high10-1.cel"  "high10-2.cel"  "high48-1.cel"
 [6] "high48-2.cel"  "low10-1.cel"   "low10-2.cel"   "low48-1.cel"   "low48-2.cel"
[11] "phenoData.txt"
```

The targets file is called phenoData.txt. We see there are two arrays for each experimental condition, giving a total of 8 arrays.

```
> targets <- readTargets("phenoData.txt",path=datadir,sep="",row.names="filename")
> targets
            filename estrogen time.h
low10-1   low10-1.cel   absent     10
low10-2   low10-2.cel   absent     10
high10-1 high10-1.cel  present     10
high10-2 high10-2.cel  present     10
low48-1   low48-1.cel   absent     48
low48-2   low48-2.cel   absent     48
high48-1 high48-1.cel  present     48
high48-2 high48-2.cel  present     48
```

Now read the cel files into an AffyBatch object and normalize using the `rma()` function from the affy package:

```
> ab <- ReadAffy(filenames=targets$filename, celfile.path=datadir)
> eset <- rma(ab)
Background correcting
Normalizing
Calculating Expression
```

There are many ways to construct a design matrix for this experiment. Given that we are interested in the early and late estrogen responders, we can choose a parametrization which includes these two contrasts.

```
> treatments <- factor(c(1,1,2,2,3,3,4,4),labels=c("e10","E10","e48","E48"))
> contrasts(treatments) <- cbind(Time=c(0,0,1,1),E10=c(0,1,0,0),E48=c(0,0,0,1))
> design <- model.matrix(~treatments)
> colnames(design) <- c("Intercept","Time","E10","E48")
```

The second coefficient picks up the effect of time in the absence of estrogen. The third and fourth coefficients estimate the $\log_2$-fold change for estrogen at 10 hours and 48 hours respectively.

```
> fit <- lmFit(eset,design)
```

We are only interested in the estrogen effects, so we choose a contrast matrix which picks these two coefficients out:

```
> cont.matrix <- cbind(E10=c(0,0,1,0),E48=c(0,0,0,1))
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

We can examine which genes respond to estrogen at either time using the moderated F-statistics on 2 degrees of freedom. The moderated F p-value is stored in the component `fit2$F.p.value`.

What p-value cutoff should be used? One way to decide which changes are significant for each gene would be to use Benjamini and Hochberg's method to control the false discovery rate across all the genes and both tests:

```
> results <- decideTests(fit2, method="global")
```

Another method would be to adjust the F-test p-values rather than the t-test p-values:

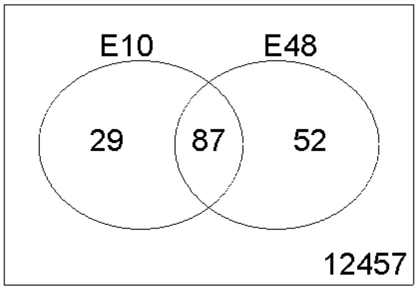
```
> results <- decideTests(fit2, method="nestedF")
```

Here we use a more conservative method which depends far less on distributional assumptions, which is to make use of control and spike-in probe-sets which theoretically should not be differentially-expressed. The smallest p-value amongst these contols turns out to be about 0.00014:

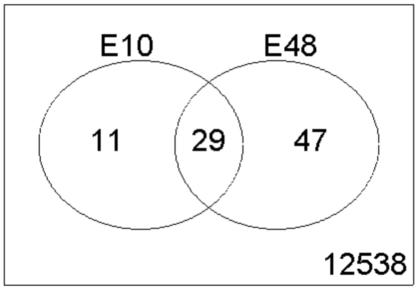
```
> i <- grep("AFFX",geneNames(eset))
> summary(fit2$F.p.value[i])
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0001391 0.1727000 0.3562000 0.4206000 0.6825000 0.9925000
```

So a cutoff p-value of 0.0001, say, would conservatively avoid selecting any of the control probe-sets as differentially expressed:

```
> results <- classifyTestsF(fit2, p.value=0.0001)
> summary(results)
   E10   E48
-1    40    76
0  12469 12410
1    116   139

> table(E10=results[,1],E48=results[,2])
    E48
E10  -1     0     1
  -1    29    11     0
  0     47 12370    52
  1      0    29    87

> vennDiagram(results,include="up")
```



```
> vennDiagram(results,include="down")
```



We see that 87 genes were up regulated at both 10 and 48 hours, 29 only at 10 hours and 52 only at 48 hours. Also, 29 genes were down-regulated throughout, 11 only at 10 hours and 47 only at 48 hours. No genes were up at one time and down at the other.

`topTable` gives a detailed look at individual genes. The leading genes are clearly significant, even using the default p-value adjustment method, which is the highly conservative Holm's method.

```
> options(digits=3)
> topTable(fit2,coef="E10",n=20)
            ID    M     A    t  P.Value    B
9735   39642_at  2.94  7.88  23.7 5.99e-05 9.97
12472    910_at  3.11  9.66  23.6 6.26e-05 9.94
1814   31798_at  2.80 12.12  16.4 1.29e-03 7.98
```

```
11509  41400_at  2.38 10.04  16.2 1.41e-03 7.92
10214  40117_at  2.56  9.68  15.7 1.86e-03 7.70
953     1854_at  2.51  8.53  15.2 2.46e-03 7.49
9848   39755_at  1.68 12.13  15.1 2.59e-03 7.45
922   1824_s_at  1.91  9.24  14.9 2.86e-03 7.37
140   1126_s_at  1.78  6.88  13.8 5.20e-03 6.89
580     1536_at  2.66  5.94  13.3 7.30e-03 6.61
12542    981_at  1.82  7.78  13.1 8.14e-03 6.52
3283   33252_at  1.74  8.00  12.6 1.12e-02 6.25
546     1505_at  2.40  8.76  12.5 1.20e-02 6.19
4405   34363_at -1.75  5.55 -12.2 1.44e-02 6.03
985   1884_s_at  2.80  9.03  12.1 1.59e-02 5.95
6194   36134_at  2.49  8.28  11.8 1.90e-02 5.79
7557   37485_at  1.61  6.67  11.4 2.50e-02 5.55
1244     239_at  1.57 11.25  10.4 5.14e-02 4.90
8195   38116_at  2.32  9.51  10.4 5.16e-02 4.90
10634  40533_at  1.26  8.47  10.4 5.31e-02 4.87
```

```
> topTable(fit2,coef="E48",n=20)
             ID     M     A     t  P.Value     B
12472    910_at  3.86  9.66  29.2 1.04e-05 11.61
1814   31798_at  3.60 12.12  21.1 1.62e-04  9.89
953     1854_at  3.34  8.53  20.2 2.29e-04  9.64
8195   38116_at  3.76  9.51  16.9 1.02e-03  8.48
8143   38065_at  2.99  9.10  16.2 1.42e-03  8.21
9848   39755_at  1.77 12.13  15.8 1.72e-03  8.05
642     1592_at  2.30  8.31  15.8 1.76e-03  8.03
11509  41400_at  2.24 10.04  15.3 2.29e-03  7.81
3766   33730_at -2.04  8.57 -15.1 2.48e-03  7.74
732     1651_at  2.97 10.50  14.8 3.02e-03  7.57
8495   38414_at  2.02  9.46  14.6 3.36e-03  7.48
1049    1943_at  2.19  7.60  14.0 4.69e-03  7.18
10214  40117_at  2.28  9.68  14.0 4.79e-03  7.16
10634  40533_at  1.64  8.47  13.5 6.24e-03  6.93
9735   39642_at  1.61  7.88  13.0 8.46e-03  6.65
4898   34851_at  1.96  9.96  12.8 9.47e-03  6.55
922   1824_s_at  1.64  9.24  12.8 1.00e-02  6.50
6053   35995_at  2.76  8.87  12.7 1.05e-02  6.46
12455    893_at  1.54 10.95  12.7 1.06e-02  6.45
10175  40079_at -2.41  8.23 -12.6 1.09e-02  6.42
```

11.5 Weaver Mutant Data: A 2x2 Factorial Experiment with Two-Color Data

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

Background. This is a case study examining the development of certain neurons in wild-type and weaver mutant mice from [5]. 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 aliquots 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. There are four different treatment combinations, P11wt, P11mt, P21wt and P21mt, which might think of as a 2x2 factorial structure. We consider ten arrays in total. There are six arrays comparing the four different RNA sources to a common reference, which was a pool of RNA from all the time points, and four arrays making direct comparisons between the four treatment combinations.

First read in the data. We assume that the data is an directory called `c:/Weaver`. The data used for this case study can be downloaded from `http://bioinf.wehi.edu.au/limma/data/weaver.zip`. We first read in the targets frame, and then read the intensity data using file names recorded in the targets file. The data was produced using SPOT image analysis software and is stored in the subdirectory `/spot`. Notice that a spot quality weight function as been set. For these arrays the median spot area is just over 50 pixels. The spot quality function has been set so that any spot with an area less than 50 pixels will get reduced weight, so that a hypothetical spot of zero area would get zero weight.

```
> library(limma)
> targets <- readTargets("targets.txt")
> targets

              FileName     Tissue  Mouse   Cy5   Cy3
cbmut.3   cbmut.3.spot Cerebellum Weaver P11wt  Pool
cbmut.4   cbmut.4.spot Cerebellum Weaver P11mt  Pool
cbmut.5   cbmut.5.spot Cerebellum Weaver P21mt  Pool
cbmut.6   cbmut.6.spot Cerebellum Weaver P21wt  Pool
cbmut.15 cbmut.15.spot Cerebellum Weaver P21wt  Pool
cbmut.16 cbmut.16.spot Cerebellum Weaver P21mt  Pool
cb.1         cb.1.spot Cerebellum Weaver P11wt P11mt
cb.2         cb.2.spot Cerebellum Weaver P11mt P21mt
cb.3         cb.3.spot Cerebellum Weaver P21mt P21wt
cb.4         cb.4.spot Cerebellum Weaver P21wt P11wt

> wtfun <- function(x) pmin(x$area/50, 1)
> RG <- read.maimages(targets$FileName, source = "spot", path = "spot", wt.fun = wtfun)
```

```
Read spot/cbmut.3.spot
Read spot/cbmut.4.spot
Read spot/cbmut.5.spot
Read spot/cbmut.6.spot
Read spot/cbmut.15.spot
Read spot/cbmut.16.spot
Read spot/cb.1.spot
Read spot/cb.2.spot
Read spot/cb.3.spot
Read spot/cb.4.spot
```

The SPOT software does not store probe IDs in the output files, so we need to read in the ID and annotation information separately. We also read in a spottypes file and set a range of control spots.

```
> RG$genes <- read.delim("genelist.txt", header = TRUE, as.is = TRUE)
> RG$printer <- list(ngrid.r = 8, ngrid.c = 4, nspot.r = 25, nspot.c = 24)
> spottypes <- readSpotTypes("spottypes.txt")
```

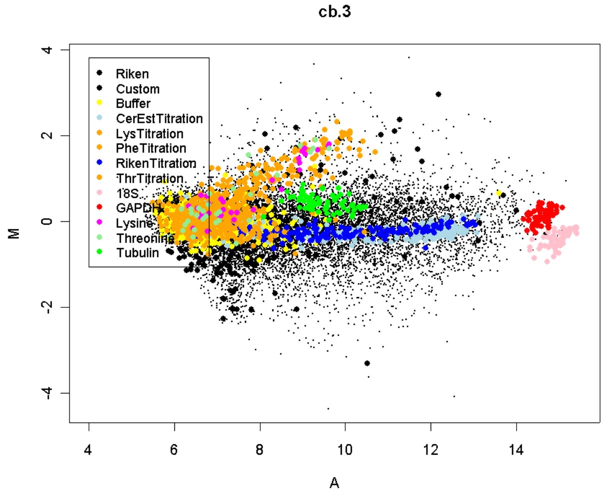
Note for R 2.1.0 or 2.1.1. There is a bug in the `read.table` function of R 2.1 which prevents `readSpotTypes()` from reading in the backslashes in the `Name` column of the file `spottypes.txt` correctly. If you are reproducing this data example in R 2.1, you should insert the commands

```
> spottypes$Name <- gsub("\\(","\\\\(",spottypes$Name)
> spottypes$Name <- gsub("\\)","\\\\)",spottypes$Name)
```

at this point as a work-around.

```
> spottypes

          SpotType      ID                     Name        col cex
1            Riken       *                        *      black 0.2
2           Custom Control                        *      black 1.0
3           Buffer Control                   3x SSC     yellow 1.0
4  CerEstTitration Control             cer est \\(*  lightblue 1.0
5     LysTitration Control                 Lys \\(*     orange 1.0
6     PheTitration Control                 Phe \\(*     orange 1.0
7   RikenTitration Control           Riken est \\(*       blue 1.0
8     ThrTitration Control                 Thr \\(*     orange 1.0
9              18S Control      18S \\(0.15ug/ul\\)       pink 1.0
10           GAPDH Control   GAPDH \\(0.15 ug/ul\\)        red 1.0
11          Lysine Control   Lysine \\(0.2 ug/ul\\)    magenta 1.0
12       Threonine Control Threonine \\(0.2ug/ul\\) lightgreen 1.0
13         Tubulin Control Tubulin \\(0.15 ug/ul\\)      green 1.0

> RG$genes$Status <- controlStatus(spottypes, RG)

Matching patterns for: ID Name
Found 19200 Riken
Found 2304 Custom
```

```
Found 710 Buffer
Found 192 CerEstTitration
Found 224 LysTitration
Found 260 PheTitration
Found 160 RikenTitration
Found 224 ThrTitration
Found 64 18S
Found 64 GAPDH
Found 32 Lysine
Found 32 Threonine
Found 64 Tubulin
Setting attributes: values col cex

> plotMA(RG,array=9,xlim=c(4,15.5))
```



Here Buffer is an obvious negative control while 18S, GAPDH, Lysine, Threonine and Tubulin are single-gene positive controls, sometime called house-keeping genes. RikenTitration is a titration series of a pool of the entire Riken library, and can be reasonably expected to be non-differentially expressed. CerEstTitration is a titration of a pool of a cerebellum EST library. This will show higher expression in later mutant tissues. The Lys, Phe and Thr series are single-gene titration series which were not spike-in in this case and can be treated as negative controls.

Now normalize the data. Because the Riken titration library, being based on a pool of a large number of non-specific genes, should not be differentially expressed, we up-weight these spots in the print-tip normalization step:

```
> w <- modifyWeights(RG$weights, RG$genes$Status, "RikenTitration", 2)
> MA <- normalizeWithinArrays(RG, weights = w)
```

Now fit a linear model to the data. Because of the composite design, with some common reference arrays and some direct comparison arrays, the simplest method is to use a group-mean parametrization with all RNA samples compared back to the Pool.

```
> design <- modelMatrix(targets, ref = "Pool")

Found unique target names:
 P11mt P11wt P21mt P21wt Pool

> design

         P11mt P11wt P21mt P21wt
cbmut.3      0     1     0     0
cbmut.4      1     0     0     0
cbmut.5      0     0     1     0
cbmut.6      0     0     0     1
cbmut.15     0     0     0     1
cbmut.16     0     0     1     0
cb.1        -1     1     0     0
cb.2         1     0    -1     0
cb.3         0     0     1    -1
cb.4         0    -1     0     1
```

All the control spots are removed before fitting the linear model:

```
> isGene <- MA$genes$Status == "Riken"
> fit <- lmFit(MA[isGene, ], design)
```

We now extract all possible comparisons of interest as contrasts. We look for the mutant vs wt comparisons at 11 and 21 days, the time effects for mutant and wt, and the interaction terms:

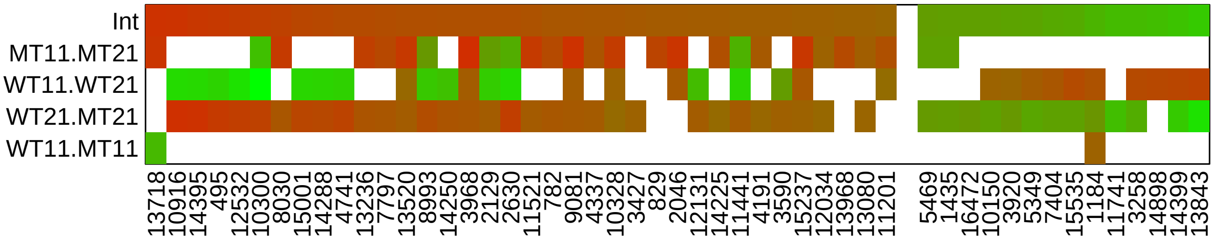
```
> cont.matrix <- makeContrasts(
+     WT11.MT11=P11mt-P11wt,
+     WT21.MT21=P21mt-P21wt,
+     WT11.WT21=P21wt-P11wt,
+     MT11.MT21=P21mt-P11mt,
+     Int=(P21mt-P11mt)-(P21wt-P11wt),
+     levels=design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

Adjustment for multiple testing, with Benjamini and Hochberg's method applied to the F-test p-values across genes with 5% false discovery rate, and nesting F-testing used within contrasts, leads to the following:

```
> results <- decideTests(fit2, method = "nestedF")
> summary(results)

   WT11.MT11 WT21.MT21 WT11.WT21 MT11.MT21   Int
-1         3        29       132       136    14
0      16884     16788     16653     16571 16846
1          9        79       111       189    36
```

There are 187 genes up and 136 genes down in mutant at 21 days vs 11 days. There are 36 genes which respond more up in the mutant than the wt, and 14 genes which respond more down in the mutant than the wt. A heatdiagram shows that the genes are mostly responding in the same direction in the mutant and wt, but to different degrees:

```
> heatDiagram(results, fit2$coef, primary = "Int")
```



11.6 Bob Mutant Data: 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. The theory behind the approach is explained in [19]. This approach assumes that all genes are replicated the same number of times on the array and that the spacing between the replicates is entirely regular.

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.

The data used for this case study can be downloaded from `http://bioinf.wehi.edu.au/limmaGUI/DataSets.html`.

```
> 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
```

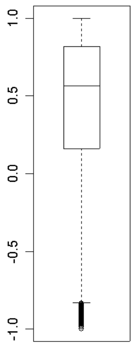
Although there are only four arrays, we have a total of eight spots for each gene, and more for the controls. Naturally the two M-values obtained from duplicate spots on the same array are highly correlated. The problem is how to make use of the duplicate spots in the best way. The approach taken here 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)
```

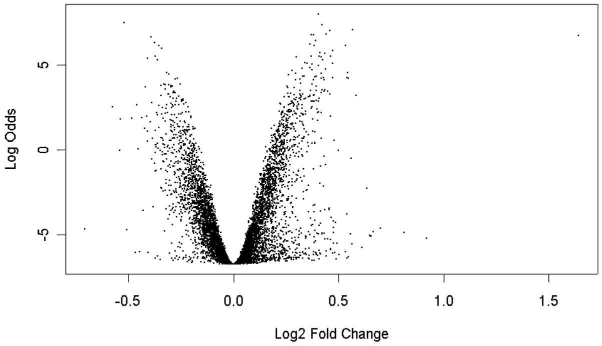
Now estimate the spatial correlation. We estimate a correlation term by REML for each gene, and then take a trimmed mean on the atanh scale to estimate the overall correlation. This command takes a lot of time, perhaps as much as an hour for a series of arrays.

```
> cor <- duplicateCorrelation(MA,design,ndups=2) # A slow computation!
> cor$consensus.correlation
[1] 0.571377
> boxplot(cor$all.correlations)
```



```
> fit <- lmFit(MA,design,ndups=2,correlation=0.571377)
> fit <- eBayes(fit)
> topTable(fit,n=30,adjust="BH")

     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

> volcanoplot(fit)
```



Notes

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. Laurent Gautier made the ecoliLeucine data set available on Bioconductor. Lynn Corcoran provided the Bob Mutant data.

The limma package has benefited from many people who have made suggestions or reported bugs including Naomi Altman, Henrik Bengtsson, Lourdes Peña Castillo, Marcus Davy, Ramon Diaz-Uriarte, Robert Gentleman, Wolfgang Huber, William Kenworthy, Kevin Koh, Erik Kristiansson, Mette Langaas, Gregory Lefebvre, Andrew Lynch, James MacDonald, Martin Maechler, Ron Ophir, Francois Pepin, Hubert Rehrauer, Matthew Ritchie, Ken Simpson, Laurentiu Adi Tarca, Björn Usadel, James Wettenhall, Chris Wilkinson, Yee Hwa (Jean) Yang, John Zhang.

Conventions

Where possible, limma tries to use the convention that class names are in upper *CamelCase*, i.e., the first letter of each word is capitalized, while function names are in lower *camelCase*, i.e., first word is lowercase. When periods appear in function names, the first word should be an action while the second word is the name of a type of object on which the function acts.

Software Projects Using limma

The limma package is used as a building block or as the underlying computational engine by a number of software projects designed to provide user-interfaces for microarray data analysis including [6, 23, 24, 2], the KTH Package [17], and SKCC WebArray [27]. The LCB-BASE project provides a limma plug-in for the BASE database [11]. The Stanford Microarray Database `http://genome-www5.stanford.edu` calls out to limma for background correction options.

Citations

Biological studies using the limma package include [7, 16, 15, 1, 13, 22]. Methodological studies using the limma package include [9, 10].

Bibliography

[1] P. C. Boutros, I. D. Moffat, M. A. Franc, N. Tijet, J. Tuomisto, R. Pohjanvirta, and A. B. Okey. Identification of the DRE-II gene battery by phylogenetic footprinting. *Biochem Biophys Res Commun*, 321(3):707–715, 2004.

[2] Andreas Buness, Wolfgang Huber, Klaus Steiner, Holger Sltmann, and Annemarie Poustka. arrayMagic: two-colour cDNA microarray quality control and preprocessing. *Bioinformatics*, 21(4):554–556, 2005.

[3] 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*, 10:2022–2029, 2000.

[4] P. Dalgaard. *Introductory Statistics with R.* Springer, New York, 2002.

[5] E. Diaz, Y. Ge, Y. H. Yang, K. C. Loh, T. A. Serafini, Y. Okazaki, Y. Hayashizaki, T. Speed, J. P., Ngai, and P. Scheiffele. Molecular analysis of gene expression in the developing pontocerebellar projection system. *Neuron*, 36:417–434, 2002.

[6] Steffen Durinck, Joke Allemeersch, Vincent J. Carey, Yves Moreau, and Bart De Moor. Importing MAGE-ML format microarray data into BioConductor. *Bioinformatics*, 20(18):3641–3642, 2004.

[7] R. Golden, T. and S. Melov. Microarray analysis of gene expression with age in individual nematodes. *Aging Cell*, 3:111–124, 2004.

[8] S. Hung, P. Baldi, and G. W. Hatfield. Global gene expression profiling in Escherichia coli K12: The effects of leucine-responsive regulatory protein. *Journal of Biological Chemistry*, 277(43):40309–40323, 2002.

[9] C. Kendziorski, R. A. Irizarry, K.-S. Chen, J. D. Haag, and M. N. Gould. On the utility of pooling biological samples in microarray experiments. *PNAS*, 102(12):4252–4257, 2005.

[10] Charles Kooperberg, Aaron Aragaki, Andrew D. Strand, and James M. Olson. Significance testing for small microarray experiments. *Statistics in Medicine*, 24:2281–2298, 2005.

[11] Linnaeus Centre for Bioinformatics, Uppsala University, Sweden. BASE plug-ins. Software package, `http://www.lcb.uu.se/baseplugins.php`, 2005.

[12] G. A. Milliken and D. E. Johnson. *Analysis of Messy Data, Volume 1: Designed Experiments.* Chapman & Hall, New York, 1992.

[13] M. J. Peart, G. K. Smyth, R. K. van Laar, V. M. Richon, A. J. Holloway, and R. W. Johnstone. Identification and functional significance of genes regulated by structurally diverse histone deacetylase inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 102(10):3697–3702, 2005.

[14] A. Reiner, D. Yekutieli, and Y. Benjamini. Identifying differentially expressed genes using false discovery rate controlling procedures. *Bioinformatics*, 19:368–375, 2003.

[15] S. C. P. Renn, N. Aubin-Horth, and H. A. Hofmann. Biologically meaningful expression profiling across species using heterologous hybridization to a cdna microarray. *BMC Genomics*, 5(42), 2004.

[16] M. W. Rodriguez, A. C. Paquet, Y. H. Yang, and D. J. Erle. Differential gene expression by integrin $\beta 7+$ and $\beta 7$- memory T helper cells. *BMC Immunology*, 5(13), 2004.

[17] Royal Institute of Technology, Sweden. KTH-package for microarray data analysis. Software package, `http://www.biotech.kth.se/molbio/microarray/pages/kthpackagetransfer.html`, 2005.

[18] G. K. Smyth. Paper 116: Individual channel analysis of two-colour microarrays. In *55th Session of the International Statistics Institute, 5-12 April 2005, Sydney Convention & Exhibition Centre, Sydney, Australia (CD).* International Statistical Institute, Bruxelles, 2005.

[19] G. K. Smyth, J. Michaud, and H. Scott. The use of within-array replicate spots for assessing differential expression in microarray experiments. *Bioinformatics*, 21(9):2067–2075, 2005.

[20] G. K. Smyth and T. P. Speed. Normalization of cDNA microarray data. *Methods*, 31(4):265–273, 2003.

[21] G.K. Smyth. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3:Article 3, 2004.

[22] Srinivasa Rao Uppalapati, Patricia Ayoubi, Hua Weng, David A. Palmer, Robin E. Mitchell, William Jones, and Carol L. Bender. The phytotoxin coronatine and methyl jasmonate impact multiple phytohormone pathways in tomato. *The Plant Journal*, 42(2):201–217, April 2005.

[23] Juan M. Vaquerizas, Joaqun Dopazo, and Ramn Daz-Uriarte. DNMAD: web-based diagnosis and normalization for microarray data. *Bioinformatics*, 20(18):3656–3658, 2004.

[24] Hua Weng and Patricia Ayoubi. GPAP (GenePix Pro Auto-Processor) for online preprocessing, normalization and statistical analysis of primary microarray data. Software package, Microarray Core Facility, Oklahoma State University, `http://darwin.biochem.okstate.edu/gpap3`, 2004.

[25] J. M. Wettenhall and G. K. Smyth. limmaGUI: a graphical user interface for linear modeling of microarray data. *Bioinformatics*, 20:3705–3706, 2004.

[26] R. D. Wolfinger, G. Gibson, E. D. Wolfinger, L. Bennett, H. Hamadeh, P. Bushel, C. Afshari, and R. S. Paules. Assessing gene significance from cDNA microarray expression data via mixed models. *Journal of Computational Biology*, 8:625–637, 2001.

[27] Xiaoqin Xia, Michael McClelland, and Yipeng Wang. Webarray: an online platform for microarray data analysis. *BMC Bioinformatics*, 6:306, 2005.

[28] 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):e15, 2002.

[29] Y. H. Yang, S. Dudoit, P. Luu, and T. P. Speed. Normalization for cDNA microarray data. In M. L. Bittner, Y. Chen, A. N. Dorsel, and E. R. Dougherty, editors, *Microarrays: Optical Technologies and Informatics*, pages 141–152. Proceedings of SPIE, Volume 4266, 2001.

[30] Y. H. Yang and T. P. Speed. Design and analysis of comparative microarray experiments. In T. P. Speed, editor, *Statistical Analysis of Gene Expression Microarray Data*, pages 35–91. Chapman & Hall/CRC Press, 2003.

[31] Y. H. Yang and N. P. Thorne. Normalization for two-color cDNA microarray data. In D. R. Goldstein, editor, *Science and Statistics: A Festschrift for Terry Speed*, pages 403–418. Institute of Mathematical Statistics Lecture Notes – Monograph Series, Volume 40, 2003.