23

Limma: Linear Models for Microarray Data

Gordon K. Smyth

Abstract

A survey is given of differential expression analyses using the linear modeling features of the limma package. The chapter starts with the simplest replicated designs and progresses through experiments with two or more groups, direct designs, factorial designs and time course experiments. Experiments with technical as well as biological replication are considered. Empirical Bayes test statistics are explained. The use of quality weights, adaptive background correction and control spots in conjunction with linear modelling is illustrated on the $\beta 7$ data.

23.1 Introduction

Limma[1] is a package for differential expression analysis of data arising from microarray experiments. The package is designed to analyze complex experiments involving comparisons between many RNA targets simultaneously while remaining reasonably easy to use for simple experiments. The central idea is to fit a linear model to the expression data for each gene. The expression data can be log-ratios, or sometimes log-intensities, from two color microarrays or log-intensity values from one channel technologies such as Affymetrix™. Empirical Bayes and other shrinkage methods are used to borrow information across genes making the analyses stable even for experiments with small number of arrays [1, 2].

[1] 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.

Limma is designed to be used in conjunction with the affy or affyPLM packages for Affymetrix™ data as described in chapters 2 and 25. With two color microarray data, the marray package may be used for pre-processing as described in Chapter 4. Limma itself also provides input and normalization functions which support features especially useful for the linear modeling approach.

This chapter gives a survey of differential expression analyses starting with the simplest replicated designs and progressing through experiments with two or more groups, factorial designs and time course experiments. For the most part, this chapter does not analyze specific data sets but gives instead generic code which can be applied to any data set arising from the designs described. Analyses of specific data sets are given in Chapters 4, 14, 16 and 25. One purpose of this chapter is to place these analyses in context and to indicate how the methods would be extended to more complex designs. This chapter was prepared using limma version 1.8.18.

23.2 Data Representations

The starting point for this chapter and many other chapters in this book is that an experiment has been performed using a set of microarrays hybridized with two or more different RNA sources. The arrays have been scanned and image-analyzed to produce output files containing raw intensities, usually one file for each array. The arrays may be *one-channel* with one RNA sample hybridized to each array or they may be *two-channel* or *two-color* with two RNA samples hybridized competitively to each array.

Expression data from experiments using one-channel arrays can be represented as a data matrix with rows corresponding to probes and columns to arrays. The `rma()` function in the affy package produces such a matrix for Affymetrix™ arrays. The output from `rma()` is an *exprSet* object with the matrix of log-intensities in the `exprs` slot. See Chapters 2 and 25 for details.

Experiments using two-color arrays produce two data matrices, one each for the green and red channels. The green and red channel intensities are usually kept separate until normalization, after which they are summarized by a matrix of log-ratios (M-values) and a matrix of log-averages (A-values). See Chapter 4 for details.

Two-color experiments can be divided into those for which one channel of every array is a common reference sample and those which make direct comparisons between the RNA samples of interest without the intermediary of a common reference. Common reference experiments can be treated similarly to one-channel experiments with the matrix of log-ratios taking the place of the matrix of log-intensities. Direct two-color designs require some special techniques. Many features of limma are motivated by the desire to obtain full information from direct designs and to treat all types of experiment in a unified way.

Sections 23.3 to 23.11 will assume that a normalized data object called `MA` or `eset` is available. The object `eset` is assumed to be of class *exprSet* containing normalized probe-set log-intensities from an Affymetrix™ experiment while `MA` is assumed to contain normalized M and A-values from an experiment using two-color arrays. The data object `MA` might be an *marrayNorm* object produced by `maNorm()` in the marray package or an *MAList* object produced by `normalizeWithinArrays()` or `normalizeBetweenArrays()` in the limma package, although *marrayNorm* objects usually need some further processing after normalization before being used for linear modeling as explained in Section 23.4. The examples of Sections 23.4 to 23.11 remain valid if `eset` or `MA` is just a matrix containing the normalized log-intensities or log-ratios.

Apart from the expression data itself, microarray data sets need to include information about the *probes* printed on the arrays and information about the *targets* hybridized to the arrays. The targets are of particular interest when setting up a linear model. In this chapter the target labels and any associated covariates are assumed to be available in a *targets frame* called `targets`, which is just a `data.frame` with rows corresponding to arrays in the experiment. In an *exprSet* object this data frame is often stored as part of the `phenoData` slot, in which case it can be extracted by `targets <- pData(eset)`. Despite the name, there is no implication that the covariates are phenotypic in nature, in fact they often indicate genotypes such as wild-type or knockout. In an *marrayNorm* object the targets frame is often stored as part of the `maTargets` slot, in which case it can be extracted by `targets <- maInfo(maTargets(MA))`. Limma provides the function `readTargets()` for reading the targets frame directly from a text file, and doing so is often the first step in a microarray data analysis.

23.3 Linear Models

Limma uses *linear models* to analyze designed microarray experiments [3, 1]. This approach allows very general experiments to be analyzed nearly as easily as a simple replicated experiment. The approach requires two matrices to be specified. The first is the *design matrix* which provides a representation of the different RNA targets which have been hybridized to the arrays. The second is the *contrast matrix* which allows the coefficients defined by the design matrix to be combined into contrasts of interest. Each contrast corresponds to a comparison of interest between the RNA targets. For very simple experiments the contrast matrix may not need to be specified explicitly.

The first step is to fit a linear model using `lmFit()` which fully models the systematic part of the data. Each row of the design matrix corresponds to an array in the experiment and each column corresponds to a coefficient. With one-channel data or common reference data, the number of coefficients will be equal to the number of distinct RNA sources. With direct-design two-color data there will be one fewer coefficient than distinct RNA targets, or the same number if a dye-effect is included. One purpose of this step is to estimate the variability in the data.

In practice one might be interested in more or fewer comparisons between the RNA targets than there are coefficients. The contrast step, which uses the function `contrasts.fit()`, allows the fitted coefficients to be compared in as many ways as there are questions to be answered, regardless of how many or how few these might be.

Mathematically we assume a linear model $\mathrm{E}[\mathbf{y}_j] = \mathbf{X}\boldsymbol{\alpha}_j$ where $\mathbf{y}_j$ contains the expression data for the gene j, $\mathbf{X}$ is the design matrix and $\boldsymbol{\alpha}_j$ is a vector of coefficients. Here $\mathbf{y}_j^T$ is the jth row of the expression matrix and contains either log-ratios or log-intensities. The contrasts of interest are given by $\boldsymbol{\beta}_j = \mathbf{C}^T\boldsymbol{\alpha}_j$ where $\mathbf{C}$ is the contrasts matrix. The `coefficients` component of the fitted model produced by `lmFit()` contains estimated values for the $\boldsymbol{\alpha}_j$. After applying `contrasts.fit()`, the `coefficients` component now contains estimated values for the $\boldsymbol{\beta}_j$.

With one-channel or common reference microarray data, linear modeling is much the same as ordinary ANOVA or multiple regression except that a model is fitted for every gene. With data of this type, design matrices can be created in the same way that one would do when modeling univariate data. With data from two-color direct designs, linear modeling is very flexible and powerful but the formation of the design matrix may be less familiar. The function `modelMatrix()` is provided to simplify the construction of appropriate design matrices for two-color data.

23.4 Simple Comparisons

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 frame might contain the following entries:

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

A list of the top genes which show evidence of differential expression between the mutant and wild-type might be found for this experiment by

```
> fit <- lmFit(MA)
> fit <- eBayes(fit)
> topTable(fit, adjust = "fdr")
```

where MA holds the normalized data. The default design matrix used here is just a single column of ones. This experiment estimates the fold change of mutant over wild-type. Genes which have positive M-values are more highly expressed in the mutant 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 empirical Bayes methods have been used to borrow information between genes.

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

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

and the analysis would be

```
> design <- c(1, -1, 1)
> fit <- lmFit(MA, design)
> fit <- eBayes(fit)
> topTable(fit, adjust = "fdr")
```

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

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

where targets is the targets frame.

If there are at least two arrays with each dye orientation, for example

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

then it may be useful to estimate probe-specific *dye-effects*. The dye-effect is estimated by an intercept term in the linear model. Including the dye-effect uses up one degree of freedom, which might otherwise be used to estimate the residual variability, but may be valuable if many genes show non-negligible dye-effects.

Integrin $\alpha 4\beta 7$ experiment. Chapter 4 introduces a data example with six replicate arays including three dye-swaps in which integrin β7+ memory T help cells play the role of "mutant" and β7- cells play the role of "wild-type". Here we continue from the "quick start" section of that chapter where an *marrayNorm* object `normdata` is created and a top table gene listing is presented. For this data it proves important to include a dye effect.

```
> design <- cbind(Dye = 1, Beta7 = c(1,-1,-1,1,1,-1))
> fit <- lmFit(normdata, design, weights = NULL)
> fit <- eBayes(fit)
```

Now

```
> topTable(fit, coef = "Dye", adjust = "fdr")
```

reveals significant dye effects for many genes.

Note the use of `weights=NULL` in the above fit. This is needed because the function `read.GenePix` used to input the data populates the `maW` slot of the data object with GenePix® spot quality flags rather than with weights. The flags are just indicators which take on various negative values to indicate suspect spots with zero representing a normal spot, whereas functions in R which accept "weights" expect them to be numeric non-negative values with zero indicating complete unreliability. For this reason it was necessary to use `weights=NULL` to tell `lmFit()` to ignore the weights slot in `normdata`. Much better however is to convert the flags into quantitative weights. The code below gives weight zero to all spots with negative flags and weight one to all unflagged spots. This improves the power of the analysis and increases the number of apparently differentially expressed genes.

```
> w <- 0 + (maW(normdata) >= 0)
> fit <- lmFit(normdata, design, weights = w)
> fit <- eBayes(fit)
> tab <- topTable(fit, coef = "Beta7", adjust = "fdr")
> tab$Name <- substring(tab$Name, 1, 20)
> tab
```

```
              ID                 Name     M    A     t P.Value   B
6647  H200017286 GPR2 - G protein-cou -2.45  7.8 -14.7   0.011 5.6
11025 H200018884 Homo sapiens cDNA FL -1.60  6.6 -12.1   0.024 4.7
6211  H200019655 KIAA0833 - KIAA0833  -1.61  8.0 -10.9   0.034 4.1
11431 H200015303 CCR9 - Chemokine (C-  1.50 10.1   9.9   0.049 3.6
4910  H200003784 SEMA5A - Sema domain -1.35  6.8  -8.7   0.101 2.8
3152  H200012024 ITGA1 - Integrin, al  1.32  7.0   9.7   0.101 2.5
22582 H200017325 IFI27 - Interferon,   1.32  7.2   7.7   0.105 2.2
7832  H200004937 Homo sapiens cDNA FL -1.23  6.3  -9.0   0.105 2.2
20941 H200008015 PTPRJ - Protein tyro -0.88 11.0  -7.6   0.105 2.0
9314  H200006462     LMNA - Lamin A/C -1.21  7.9  -8.6   0.105 2.0
```

In this table, `M` is the $\log_2$ fold change, with positive values indicating higher expression in the $\beta 7+$ cells. For the meaning of the other columns see Section 23.12.

23.5 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 = "fdr")
```

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 [4, 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 [2].

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 = "fdr")
```

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 = "fdr")
```

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 = "fdr")
```

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 = "fdr")
```

23.6 Within-Array Replicate Spots

Robotic printing of spotted arrays can be set up to print more than one spot from each well of the DNA plates. Such printing means that each probe is printed two or more times a fixed distance apart. The most common printing is in duplicate, with the duplicates being either side-by-side or in the top and bottom halves of the array. The second option means that the arrays are printed with two sub-arrays each containing a complete set of probes.

Duplicate spots can be effectively handled by estimating a common value for the intra-duplicate correlation [2]. Suppose that each probe is printed twice in adjacent positions, side-by-side by columns. Then the correlation may be estimated by

```
> corfit <- duplicateCorrelation(MA, design, ndups = 2, spacing = "columns")
```

Here `spacing="rows"` would indicate replicates side-by-side by rows and `spacing="topbottom"` would indicate replicates in the top and bottom halves of the arrays. The spacing may alternatively be given as a numerical value counting the number of spots separating the replicate spots.

The estimated common correlation is `corfit$consensus`. This value should be a large positive value, say greater than 0.4. The correlation is then specified at the linear modeling step:

```
> fit <- lmFit(MA, design, ndups = 2, spacing = 1, cor = corfit$consensus)
```

The object `fit` contains half as many rows as does `MA`, i.e., results for the multiple printings of each probe have been consolidated. See Chapter 14 for an example of this analysis applied to the Kidney cancer study. The analysis given there demonstrates the greater power of the duplicate correlation approach compared to simply averaging the log-ratios from the replicate spots.

23.7 Two Groups

Suppose now that we wish to compare two wild-type (Wt) mice with three mutant (Mu) mice using two-color 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 is to compare 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 might be

```
> design

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

This is sometimes called the *treatment-contrasts* parametrization. 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. For the remaining arrays the fitted values will be `WTvsREF + MUvsWT`, which is equivalent to mutant vs reference. Differentially expressed genes can be found by

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

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
```

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

```
> fit <- lmFit(MA, design)
> fit2 <- contrasts.fit(fit, c(-1, 1))
> fit2 <- eBayes(fit2)
> topTable(fit2, adjust = "fdr")
```

The genelist will be the same as for the first approach.

The design matrices can be constructed manually or 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

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

Suppose now that the experiment had been conducted with one-channel arrays such as Affymetrix™ rather than with a common reference, so the targets frame might be

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

The one-channel data can be analyzed exactly as for the common reference experiment. For the treatment-contrasts parametrization, the design matrix is as before

```
> design

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

except that the first coefficient estimates now the mean log-intensity for wild-type mice rather than the wild-type versus reference log-ratio. For the group-means parametrization, the design matrix is as before but the coefficients now represent mean log-intensties for wild-type and mutant rather than log-ratios versus the wild-type. Design and contrasts matrices are computed exactly as for the common reference experiment.

See Chapter 25 for a complete data analysis of a two group experiment with six Affymetrix™ arrays, three in each group, from the Affymetrix™ spike-in experiment.

23.8 Several Groups

The above approaches for two groups extend easily to any number of groups. Suppose that three RNA targets are to be compared using Affymetrix™ 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 = "fdr")
```

Acceptance or rejection of each hypothesis test can be decided by

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

A Venn diagram showing numbers of genes significant in each comparison is 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 across genes. Small p-values identify genes which vary in any way between the three RNA targets. The following code displays information on 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™ 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™ experiment.

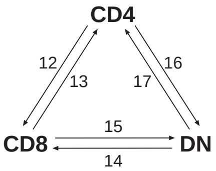

Figure 23.1. A direct design to compare three DC populations using six two-color microarrays. Each arrow represents an array, the head pointing towards the target labelled Cy5. Figure by Suzanne Thomas and James Wettenhall.

23.9 Direct Two-Color Designs

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 WEHI to compare gene expression in three different populations of dendritic cells (DC). This experiment involved six cDNA microarrays in three dye-swap pairs, with each pair used to compare two DC types (Figure 23.1). The targets frame was:

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

There are many valid choices for a design matrix for such an experiment. We chose the design matrix as

```
> design <- modelMatrix(targets, ref = "CD4")
> 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). A linear model was fit using

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

All pairwise comparisons between the three DC populations were made by

```
> cont.matrix <- cbind(
+      "CD8-CD4" = c(1, 0),
+      "DN-CD4" = c(0, 1),
+      "CD8-DN" = c(1, -1))
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit2 <- eBayes(fit2)
```

23.10 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 one-channel arrays such as Affymetrix™. This section explains the form of the analysis for a hypothetical experiment. A detailed analysis of an actual factorial experiment, the Estrogen data, is given Chapter 14. 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. 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 a traditional statistical approach using factorial model formulas. The two approaches 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
Intercept: baseline level of unstimulated wt	WT.U
StrainMu: unstimulated strain effect	Mu.U-WT.U
TreatmentS: stimulation effect for wt	WT.S-WT.U
StrainMu:TreatmentS: interaction	(Mu.S-Mu.U)-(WT.S-WT.U)

This is called the *treatment-contrast* parametrization. Note 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 the same results as the previous analysis. It differs from the previous approach only in the parametrization chosen for the linear model, i.e., in the coefficients chosen to represent the four distinct RNA targets.

23.11 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 23.8. 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%.

```
> sel.wt <- p.adjust(fit2$F.p.value, method = "fdr") < 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-mu.6hr",
+                          levels = design)
> fit2 <- contrasts.fit(fit, cont.mu)
> fit2 <- eBayes(fit2)
> sel.mu <- p.adjust(fit2$F.p.value, method = "fdr") < 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)
> sel.dif <- p.adjust(fit2$F.p.value, method = "fdr") < 0.05
```

23.12 Statistics for Differential Expression

Limma provides functions `topTable()` and `decideTests()` which summarize the results of the linear model, perform hypothesis tests and adjust the p-values for multiple testing. Results include (log) fold changes, standard errors, t-statistics and p-values. The basic statistic used for significance analysis is the *moderated t-statistic*, which is computed for each probe and for each contrast. 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, using a simple Bayesian model. This has the effect of borrowing information from the ensemble of genes to aid with inference about each individual gene [1]. Moderated t-statistics lead to p-values in the same way that ordinary t-statistics do except that the degrees of freedom are increased, reflecting the greater reliability associated with the smoothed standard errors. Chapter 25 demonstrates the effectiveness of the moderated t approach on a test data set for which the differential expression status of each probe is known.

A number of summary statistics are presented by `topTable()` for the top genes and the selected contrast. The M-value (`M`) is the value of the contrast. Usually this represents a $\log_2$-fold change between two or more experimental conditions although sometimes it represents a $\log_2$-expression level. The A-value (`A`) is the average $\log_2$-expression level for that gene across all the arrays and channels in the experiment. Column `t` is the moderated t-statistic. Column `p-value` is the associated p-value after adjustment for multiple testing. The most popular form of adjustment is `"fdr"` which is Benjamini and Hochberg's method to control the false discovery rate [5]. The meaning of `"fdr"` adjusted p-values is as follows. If all genes with p-value below a threshold, say 0.05, are selected as differentially expressed, then the expected proportion of false discoveries in the selected group is controled to be less than the theshold value, in this case 5%.

The B-statistic (`lods` or `B`) is the log-odds that the gene is differentially expressed [1, Section 5]. 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 in the call to `eBayes()`, are expected to be differentially expressed. The p-values and B-statistics will normally rank genes in the same order. In fact, if the data contains no missing values or quality weights, then the order will be precisely the same.

As with all model-based methods, the p-values depend on normality and other mathematical assumptions which are never exactly true for microarray data. It has been argued that the p-values are useful for ranking genes even in the presence of large deviations from the assumptions [6, 2]. Benjamini and Hochberg's control of the false discovery rate assumes independence between genes, although Reiner et al [7] have argued that it works for many forms of dependence as well. The B-statistic probabilities depend on the same assumptions but require in addition a prior guess for the proportion of differentially expressed genes. The p-values may be preferred to the B-statistics because they do not require this prior knowledge.

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 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 denominator degrees of freedom is the same as that of 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.

23.13 Fitted Model Objects

The output from `lmFit()` is an object of class *MArrayLM*. This section gives some mathematical details describing what is contained in such objects, following on from the Section 23.3. This section can be skipped by readers not interested in such details.

The linear model for gene j has residual variance σ_j^2 with sample value s_j^2 and degrees of freedom f_j. The output from `lmFit()`, `fit` say, holds the s_j in component `fit$sigma` and the f_j in `fit$df.residual`. The covariance matrix of the estimated $\hat{\boldsymbol{\beta}}_j$ is $\sigma_j^2\mathbf{C}^T(\mathbf{X}^T\mathbf{V}_j\mathbf{X})^{-1}\mathbf{C}$ where $\mathbf{V}_j$ is a weight matrix determined by prior weights, any covariance terms introduced by correlation structure and any iterative weights introduced by robust estimation. The square-roots of the diagonal elements of $\mathbf{C}^T(\mathbf{X}^T\mathbf{V}_j\mathbf{X})^{-1}\mathbf{C}$ are called unscaled standard deviations and are stored in `fit$stdev.unscaled`. The ordinary t-statistic for the kth contrast for gene j is $t_{jk} = \hat{\beta}_{jk}/(u_{jk}s_j)$ where u_{jk} is the unscaled standard deviation. The ordinary t-statistics can be recovered by

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

after fitting a linear model if desired.

The empirical Bayes method assumes an inverse Chisquare prior for the σ_j^2 with mean s_0^2 and degrees of freedom f_0. The posterior values for the residual variances are given by

$$\tilde{s}_j^2 = \frac{f_0 s_0^2 + f_j s_j^2}{f_0 + f_j}$$

where f_j is the residual degrees of freedom for the jth gene. The output from `eBayes()` contains s_0^2 and f_0 as `fit$s2.prior` and `fit$df.prior` and the $\tilde{s}_j^2$ as `fit$s2.post`. The moderated t-statistic is

$$\tilde{t}_{jk} = \frac{\hat{\beta}_{jk}}{u_{jk}\tilde{s}_j}$$

This can be shown to follow a t-distribution on $f_0 + f_j$ degrees of freedom if $\beta_{jk} = 0$ [1]. The extra degrees of freedom f_0 represent the extra information which is borrowed from the ensemble of genes for inference about each individual gene. The output from `eBayes()` contains the $\tilde{t}_{jk}$ as `fit$t` with corresponding p-values in `fit$p-value`.

23.14 Pre-processing Considerations

This section discusses some aspects of pre-processing which are often neglected but which are important for linear modeling and assessing differential expression for two-color data. The construction of spot quality weights, has already been briefly addressed in Section 23.4. Other important issues are the type of background correction used and the treatment of control spots on the arrays.

Background correction is more important than often appreciated because it impacts markedly on the variability of the log-ratios for low intensity spots. Chapter 4 shows an MA-plot for the $\beta7$ data illustrating the fanning out of log-ratios at low intensities when ordinary background subtraction is used. Many more spots are not shown on the plot because the background corrected intensities are negative leading to `NA` log-ratios. Fanning out of the log-ratios is undesirable for two reasons. Firstly it is undesirable than any log-ratios should be very variable, because this might lead those genes being falsely judged to be differentially expressed. Secondly, the empirical Bayes analysis implemented in `eBayes()` delivers most benefit when the variability of the log-ratios is as homogeneous as possible across genes. Chapter 4 shows that simply ignoring the background is a viable option. Another option is `"vsn"` normalization, a model-based method of stabilizing the variances which includes background correction [8, 9]. Here we illustrate a third option using the $\beta7$ data, the model-based background correction method `"normexp"` implemented in the `backgroundCorrect()` function. This method uses the available background estimates but avoids negative corrected intensities and reduces variability in the log-ratios. Background correction is still an active research area and the optimal method has not yet been determined, but the adaptive methods `"normexp"` and `"vsn"` have been found to perform well in many cases.

For convenience, we read in the $\beta7$ data again using the limma function `read.maimages`. A filter `f` is defined so that any spot which is flagged as "bad" or "absent" is given zero weight.

```
> beta7.dir <- system.file("beta7", package = "beta7")
> targets <- readTargets("TargetBeta7.txt", path = beta7.dir)
> f <- function(x) as.numeric(x$Flags > -75)
> RG <- read.maimages(targets$FileName, source = "genepix",
+                      path = beta7.dir, wt.fun = f)
> RG$printer <- getLayout(RG$genes)
```

Here `RG` is an *RGList* data object. The data read by `read.maimages()` differs slightly from `read.GenePix()` because `read.maimages()` reads mean foreground intensities for each spot while `read.GenePix()` reads median foreground intensities, although this difference should not be important here. The following code applies `"normexp"` background correction and then applies an offset of 25 to the intensities to further stabilize the log-ratios.

```
> RGne <- backgroundCorrect(RG, method = "normexp", offset = 25)
```

Now normalize and prepare for a linear model fit as in Section 23.4.

```
> MA <- normalizeWithinArrays(RGne)
> design <- cbind(Dye = 1, Beta7 = c(1, -1, -1, 1, 1, -1))
```

It is usually wise to remove uninteresting control spots from the data before fitting the linear model. Control spots can be identified on arrays by setting the `controlCode` matrix in the marray package before using `read.GenePix()` or by using `controlStatus()` in the limma package. For the β7 data, control codes have already been set in the `mraw` object, so we can restrict the fit to interesting probes by

```
> isGene <- maControls(mraw) == "probes"
> fit <- lmFit(MA[isGene, ], design)
> fit <- eBayes(fit)
> tab <- topTable(fit, coef = "Beta7", adjust = "fdr")
> tab$Name <- substring(tab$Name, 1, 20)
> tab[, -(1:3)]
```

```
              ID                 Name     M    A     t P.Value   B
5626  H200019655 KIAA0833 - KIAA0833  -1.48  8.6 -11.4   0.011 5.3
6029  H200017286 GPR2 - G protein-cou -1.98  8.2 -10.9   0.011 5.0
10115 H200018884 Homo sapiens cDNA FL -1.04  7.2 -10.8   0.011 5.0
10488 H200015303 CCR9 - Chemokine (C-  1.31 10.6  10.5   0.011 4.8
19217 H200001929 EPLIN - Epithelial p -0.86  8.9  -9.2   0.026 4.0
19346 H200008015 PTPRJ - Protein tyro -0.85 11.0  -8.8   0.030 3.7
20200 H200005842 GFI1 - Growth factor  0.76 11.7   8.6   0.033 3.5
10500 H200015731 SCYA5 - Small induci  1.54 11.1   8.3   0.033 3.4
3561  H200007572 Homo sapiens, clone   0.86  7.4   8.3   0.033 3.4
18292 H200000831 LRRN3 - Leucine-rich  0.80  9.5   8.0   0.039 3.1
```

Comparing this table to that in Section 23.4 shows more significant results overall suggesting that the adaptive background correction has reduced variability and improved power. The `"vsn"` method could have been applied here by substituting

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

for the background correction and normalization steps above. This also gives good results for the β7 data, with fewer significant results but with less attenuated fold change estimates compared to `"normexp"`.

23.15 Conclusion

This chapter has demonstrated the ability of the linear modeling aproach to handle a wide range of experimental designs. The method is applicable to both one and two-channel microarray platforms. The method is flexible and extensible in principle to arbitrarily complex designs. Some ability has also been demonstrated to accommodate both technical and biological replication in the assessment of differential expression, although here only simple experimental structures can so far be accommodated. The survey of different designs given in this chapter complements the treatments of individual data sets given in other chapters of the book.

References

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

[2] Smyth, G. K., Michaud, J., and Scott, H.: The use of within-array replicate spots for assessing differential expression in microarray experiments.Bioinformatics.21: to appear, 2005.

[3] Yang, Y. H. and Speed, T. P.Design and analysis of comparative microarray experiments.In Speed, T. P. (Ed.): Statistical Analysis of Gene Expression Microarray Data, pp 35–91. Chapman & Hall/CRC Press, 2003.

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

[5] Benjamini, Y. and Hochberg, Y.: Controlling the false discovery rate: a practical and powerful approach to multiple testing.J. R. Statist. Soc. B.57: 289–300, 1995.

[6] Smyth, G. K., Yang, Y. H., and Speed, T.: Statistical issues in cDNA microarray data analysis.Methods Mol Biol.224: 111–36, 2003.

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

[8] Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M.: Variance stabilization applied to microarray data calibration and to the quantification of differential expression.Bioinformatics.18 Suppl. 1: S96–S104, 2002.ISMB 2002.

[9] Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M.: Parameter estimation for the calibration and variance stabilization of microarray data.Statistical Applications in Genetics and Molecular Biology.2(1), 2003.