# Chapter 6

# Microarray Analysis

The analysis of whole-genome gene expression in bioinformatics has revolutionized the fields of genetics and molecular biology. The regulated process of expressing genes through RNA synthysis is the basis for all known life. The ability to accurately measure levels of transcript production for different cell types and environmental conditions or cell stresses has greatly increased our understanding of the regulatory networks and causal mechanisms that govern phenotypic responses to cell signals, genetic mutations, and epigenetic changes. This revolution began through the advances in microarray technology and is still progressing through advances in next-generation sequencing.

In this chapter, we will first learn how to preprocess and then study large microarray datasets in order to find genomic markers for phenotypic changes. Then we will learn how to perform *gene ontology (GO)* enrichment analysis to find possible differences under different conditions in either a molecular function, biological process, or within a cellular component. We will also learn how to: (1) filter for classes of genes using GO identifiers, (2) program for various visualizations, (3) find and load publicly available gene expression data, and lastly, (4) summarize results in html output.[1]

## 6.1 Probe data

In general, the microarray technique takes advantage of the hybridization properties of nucleic acids. To assay gene expression, a DNA microarray

[1]It may be helpful to explore the possibilities of the `limmaGUI`. Our approach, however, will be to concentrate on the programming aspects using the commandline.

(also called a *DNA chip* or *biochip*) is used to simultaneously measure the abundances of RNA oligos (called targets) that have been transcribed from cells under experimental conditions. On a DNA microarray, there are tens of thousands of spots on a glass slide or silicon chip, where each spot contains picomoles of a specific DNA molecule attached to the chip surface at one end through a covalent bond. The DNA oligos are typically $\geq 25$ base pairs and are specifically designed to maximize unique mapping to *untranslated regions (UTRs)* or *coding sequence (CDS)* of a gene. Confusingly, the individual DNA oligos within a spot and the spots themselves are often both referred to as *probes*. Typically, each gene within a genome will have a few sequence-matching probes in the array, and this combined *probe set* is used to measure the level of expression of the corresponding gene.

In the DNA chip assay, cDNA or cRNA (also called anti-sense RNA) samples (also called targets) hybridize with the DNA oligos that are attached to the chip. Free, unbound targets are gently washed off and the levels of hybridization at each spot is measured typically by detection of the fluorophore-labeled targets. The DNA chips are designed to minimize cross-hybridization between similar probes from different spots on the array. In an attempt to normalize for both cross-hybridization and non-specific binding, on many chip designs the probes often come in pairs. In a given *probe pair*, the *perfect match (PM)* probe perfectly matches the corresponding genomic sequence, while the *mismatch (MM)* probe contains one nucleotide substitution in the middle of the probe. The *mismatch (MM)* probe is intended to capture background noise possibly due to cross-hybridization or non-specific binding. However, multiple studies have shown that the mismatch probe alone is not sufficient to properly capture these sources of noise in microarray data. As a consequence, some background correction methods (e.g. *RMA*) ignore the mismatch probes altogether.

The raw fluoresence intensity data captured by an Affymetrix scanner is stored in a so-called *DAT file*, which is then processed into a *CEL file*. In this chapter we will work with the CEL files. We will take advantage of the `affy` package that provides many useful methods to read and analyze the gene expression data stored in these CEL files.

**Example 1: Application to the Ross et al. 2003 data.** We will start with a built-in dataset called `MLL.B` from the `ALLMLL` package. This dataset is from the Ross et al. 2003 study that used gene expression profiles from human acute lymphocytic leukemia (ALL) samples in order to accurately

diagnose subtypes and outcomes. The `MLL.A` and `MLL.B` datasets provide raw, unprocessed, probe-level gene expression data for 22,645 genes (probe sets) from 20 HGU133A and 20 HGU133B human microarray assays. Two microarray designs (A and B) were required to cover all $\approx 22K$ genes in the human genome. Both `MLL.A` and `MLL.B` are `AffyBatch` objects. After loading the data, execute `?MLL.B` and `?AffyBatch` for more information about the `MLL.B` dataset and `AffyBatch` objects.

To retrieve this data, we must load the ALLMLL package and then load the MLL.B data that's contained in the package into our R environment:

```
> library(affy)
> library(ALLMLL)
> data(MLL.B)
```

It's useful to print the class, help, contents, and structure of the loaded `MLL.B` object using `class(MLL.B)`, `?MLL.B`, `MLL.B`, `str(MLL.B)`, respectively:

```
> class(MLL.B)
[1] "AffyBatch"
attr(,"package")
[1] "affy"
> ?MLL.B
MLL                    package:ALLMLL                    R Documentation

AffyBatch instances MLL.A and MLL.B

Description:

     These 'AffyBatch' objects contain a subset of arrays from a large
     acute lymphoblastic leukemia (ALL) study.

Usage:

     data(MLL.A)
     data(MLL.B)

Format:

     Each are 'AffyBatch' containing 20 arrays.

Source:

     This package provides probe-level data for 20 HGU133A and 20
     HGU133B arrays which are a subset of arrays from a large ALL
     study. The data is for the MLL arrays. This data was published in:

     Mary E. Ross, Xiaodong Zhou, Guangchun Song, Sheila A. Shurtleff,
     Kevin Girtman, W. Kent Williams, Hsi-Che Liu, Rami Mahfouz, Susana
     C. Raimondi, Noel Lenny, Anami Patel, and James R. Downing (2003)
     _Classification of pediatric acute lymphoblastic leukemia by gene
     expression profiling_ Blood 102: 2951-2959
> MLL.B
AffyBatch object
```

```
size of arrays=712x712 features (21 kb)
cdf=HG-U133B (22645 affyids)
number of samples=20
number of genes=22645
annotation=hgu133b
notes=
> str(MLL.B)
Formal class 'AffyBatch' [package "affy"] with 10 slots
  ..@ cdfName          : chr "HG-U133B"
  ..@ nrow             : num 712
  ..@ ncol             : num 712
  ..@ assayData        :<environment: 0x00000000134393a8>
  ..@ phenoData        :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  .. .. ..@ varMetadata      :'data.frame':     1 obs. of  1 variable:
  .. .. .. ..$ labelDescription: chr "arbitrary numbering"
  .. .. ..@ data             :'data.frame':     20 obs. of  1 variable:
  .. .. .. ..$ sample: int [1:20] 1 2 3 4 5 6 7 8 9 10 ...
  .. .. ..@ dimLabels        : chr [1:2] "sampleNames" "sampleColumns"
  .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots
  .. .. .. .. ..@ .Data:List of 1
  .. .. .. .. .. ..$ : int [1:3] 1 1 0
  ..@ featureData      :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  .. .. ..@ varMetadata      :'data.frame':     0 obs. of  1 variable:
  .. .. .. ..$ labelDescription: chr(0)
  .. .. ..@ data             :'data.frame':     506944 obs. of  0 variables
  .. .. ..@ dimLabels        : chr [1:2] "featureNames" "featureColumns"
  .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots
  .. .. .. .. ..@ .Data:List of 1
  .. .. .. .. .. ..$ : int [1:3] 1 1 0
  ..@ experimentData   :Formal class 'MIAME' [package "Biobase"] with 13 slots
  .. .. ..@ name             : chr ""
  .. .. ..@ lab              : chr ""
  .. .. ..@ contact          : chr ""
  .. .. ..@ title            : chr ""
  .. .. ..@ abstract         : chr ""
  .. .. ..@ url              : chr ""
  .. .. ..@ pubMedIds        : chr ""
  .. .. ..@ samples          : list()
  .. .. ..@ hybridizations   : list()
  .. .. ..@ normControls     : list()
  .. .. ..@ preprocessing    :List of 2
  .. .. .. ..$ filenames  : chr [1:20] "/tmp/MLL/JD-ALD009-v5-U133B.CEL" "/tmp/MLL/JD-
    ↪ ALD051-v5-U133B.CEL" "/tmp/MLL/JD-ALD052-v5-U133B.CEL" "/tmp/MLL/JD-ALD057-v5-
    ↪ U133B.CEL" ...
  .. .. .. ..$ affyversion: chr "1.4.32"
  .. .. ..@ other            :List of 1
  .. .. .. ..$ : chr ""
  .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots
  .. .. .. .. ..@ .Data:List of 1
  .. .. .. .. .. ..$ : int [1:3] 1 0 0
  ..@ annotation       : chr "hgu133b"
  ..@ protocolData     :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
  .. .. ..@ varMetadata      :'data.frame':     0 obs. of  1 variable:
  .. .. .. ..$ labelDescription: chr(0)
  .. .. ..@ data             :'data.frame':     20 obs. of  0 variables
  .. .. ..@ dimLabels        : chr [1:2] "sampleNames" "sampleColumns"
  .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots
```

```
.. .. .. .. ..@ .Data:List of 1
.. .. .. .. .. ..$ : int [1:3] 1 1 0
..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots
.. .. ..@ .Data:List of 4
.. .. .. ..$ : int [1:3] 2 10 0
.. .. .. ..$ : int [1:3] 2 5 5
.. .. .. ..$ : int [1:3] 1 3 0
.. .. .. ..$ : int [1:3] 1 2 0
```

From these commands, we can see that the `MLL.B` object is an `AffyBatch` object that contains microarry data for 20 samples using the human, whole-genome HG-U133B chip design. The HG-U133B chip contains probe sets from 22,645 human genes. We can also see that an AffyBatch object can also contain a plethora of corresponding phenotypic, experimental, and other annotation data. We also see that the `AffyBatch` object is a Formal Class that contains 10 differently named slots (of data). The `slotNames()` function retrieves the names of all the slots contained in the object:

```
> slotNames(MLL.B)
 [1] "cdfName"           "nrow"              "ncol"              "assayData"         "
    ↪ phenoData"
 [6] "featureData"       "experimentData"    "annotation"        "protocolData"      "__
    ↪ classVersion__"
```

The raw probe intensities can be retrieved with `exprs(MLL.B)`, which extracts the probe intensities (expression set) from the `MLL.B` object. The number of rows and columns of the expression values of MLL.B can be obtained by the `dim()` function:

```
> dim(exprs(MLL.B))
[1] 506944     20
```

Above, the `dim(exprs())` functions tell us that there are 506,944 probes (spots) on each of the 20 MLL.B arrays. Therefore, there are on average $506,944/22,645 = 22.4$ probes $= 11.2$ probe pairs in each probe set on the microarray. To find the name of the chip design for the microarray data we use the `annotation()` function:

```
> annotation(MLL.B)
[1] "hgu133b"
```

To print the names of the probe sets that each probe belongs to, we use the `probeNames()` function:

```
> probeNames(MLL.B)[1:10]
 [1] "200000_s_at" "200000_s_at" "200000_s_at" "200000_s_at" "200000_s_at"
 [6] "200000_s_at" "200000_s_at" "200000_s_at" "200000_s_at" "200000_s_at"
```

Note that the probe set names are the same as those obtained by `geneNames()`, which shows that for Affy data the genes are named by their probe set names.

The PM and MM values for each probe within a probe set are retrieved using the functions `pm()` and `mm()`. For example, to print the PM values of the first four probes (rows) for the probe set with identifier `200000_s_at`, we could do the following:

```
> pm(MLL.B,"200000_s_at")[1:4,1:3]
          JD-ALD009-v5-U133B.CEL JD-ALD051-v5-U133B.CEL JD-ALD052-v5-U133B.CEL
200000_s_at1                661.5                  321.5                  312.5
200000_s_at2                838.8                  409.3                  395.3
200000_s_at3                865.3                  275.5                  341.3
200000_s_at4                425.8                  253.5                  196.8
```

We can use the matrix plot function `matplot()`, which plots columns from different matrices against each other, to view the variability of hybridization for the 11 probes within the `200000_s_at` probeset across the 20 samples. We see that for this probeset on the 20 DNA chips, there is considerable fluorescence-intensity variability between the probes within a probe set and between the 20 different probe sets:

```
> par(mfrow=c(2,2)) # divide the canvas into a 2 by 2 plots
> matplot(pm(MLL.B,"200000_s_at"),type="l", xlab="Probe No.",
 + ylab="PM Probe intensity")
```

The resulting plot in Figure 6.1 shows that the variability is substantial. Data normalization is needed to factor out the average intensity differences between the experiments. Also, for each individual experiment the substantial variability in the intensities could be due to artifacts (e.g. cross-hybridization) or alternative splicing.

Histograms of the log of the probe values for each experiment can be obtained by using the `hist()` function:

```
> hist(MLL.B)
```

From the histograms of the log of the intensity data in Figure 6.2, we can see that these intensity distributions are heavily skewed to the right. Therefore, the majority of the probes form a baseline of relatively low intensity measurements. In addition, since the intensity peaks are staggered, we know that these baseline levels of intensity differ across the experiments.

The *MA plot* visualises the differences between measurements of two samples $X_1$ and $X_2$. Plotted on the y-axis is their log ratio $M$ and on the x-axis their mean log intensities $A$:

Probe intensities within a probe set for all 20 samples (Matrix plot)

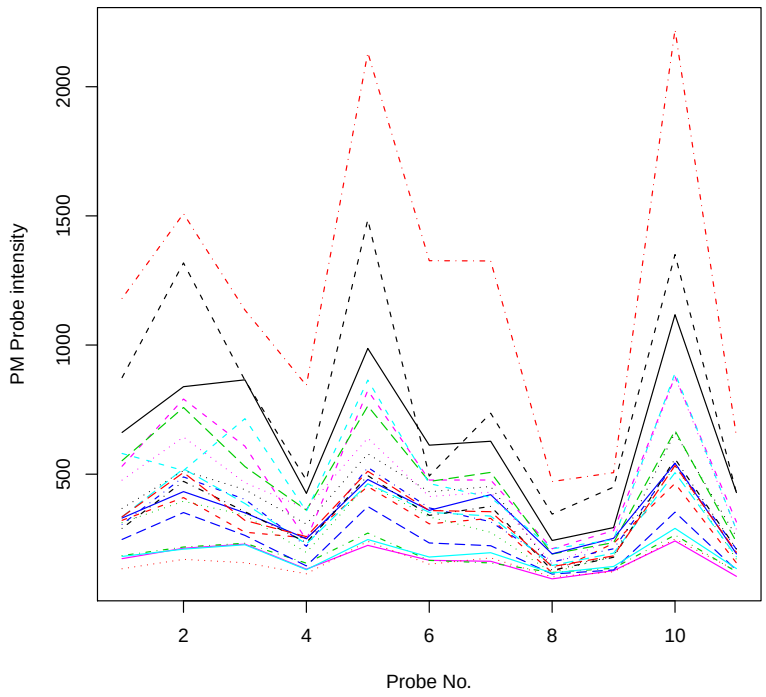


Figure 6.1: Plot of the 11 probe intensities within the probe set 200000_*s*_*at* across the 20 samples. Data normalization is needed to factor out the average intensity differences between the experiments (lines). For each individual experiment (line), the substantial variability in the intensities could be due to artifacts (e.g. cross-hybridization) or alternative splicing.

$$\begin{aligned} M &= log_2(X_1/X_2) &&= log_2(X_1) - log_2(X_2) \\ A &= \frac{1}{2} log_2(X_1 \cdot X_2) &&= \frac{1}{2}\left(log_2(X_1) + log_2(X_2)\right) \end{aligned}$$

MA plots are very useful to look for biases between experiments, and also for quality assurance to see that the spread of the log ratios decreases as the

Histogram of raw probe intensities for all 20 experiments

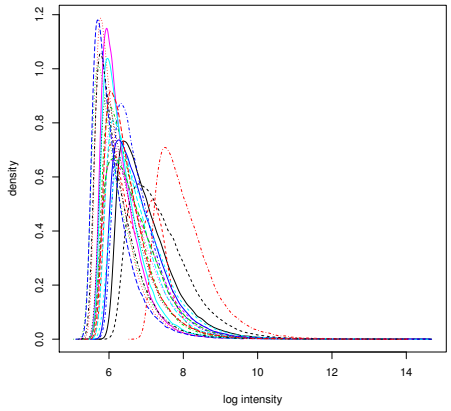


Figure 6.2: Histogram of raw probe intensities for all 20 microarray samples.

Histogram of probe intensities after RMA normalization

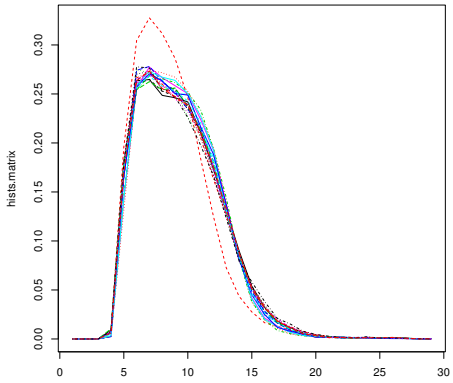


Figure 6.3: Histogram of probe intensities after performing the RMA normalization over all 20 microarray samples.

average intensities increase from left to right. The decrease in the spread indicates that the signal-to-noise ratio improves as the intensity measurements increase. Therefore, we trust high measurements much more than low measurements which are of the same magnitude as the noise (i.e. in the noise regime).

We will see later that the LOESS line through this plot can be used to normalize an experiment based on a reference. In Figure 6.4, we use the `MAplot()` function from the `affy` package to create a typical scatter MA plot of the unnormalized experiment 6 intensities relative to the pseudo-median reference. The LOESS line is plotted in red:

```
> MAplot(MLL.B, which=c(6), cex.lab=1.5)
```

Since the red LOESS line is well below the blue pseudo-median reference line, we know that the average intensities for experiment 6 are well below the median intensities for all 20 experiments. In this chapter, we will learn how to normalize the experiment 6 intensities so that their baseline intensities are in agreement with the other experiments.

In Figure 6.6, we use the `MAplot()` function again to produce smooth-scatter MA plots of the first 4 samples relative to the pseudo-median reference. The LOESS line is again plotted in red:

```
> par(mfrow=c(2,2))   # split the window into 4 parts to plot to sequentially
> MAplot(MLL.B, which=c(1,2,3,4), plot.method= "smoothScatter")
```

MA plot of the raw intensities from experiment 6

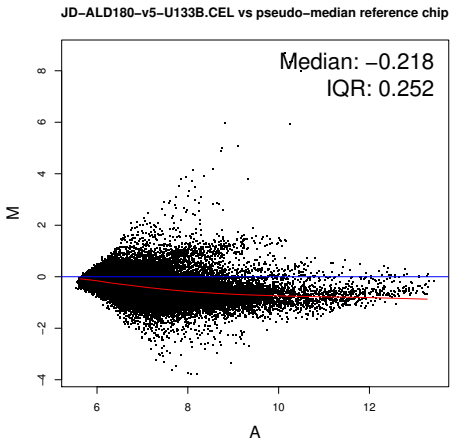


Figure 6.4: MA plot of the raw, unnormalized experiment 6 data relative to the pseudo-median reference. The LOESS line is plotted in red.

MA plot of the RMA-normalized intensities from experiment 6

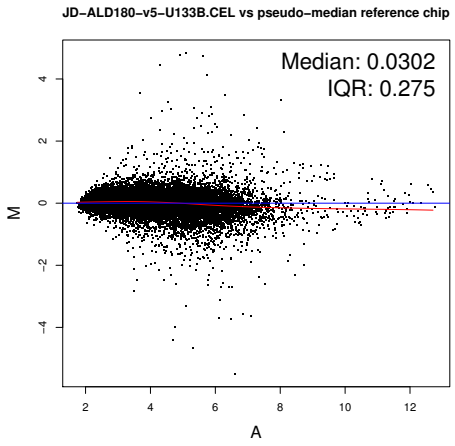


Figure 6.5: MA plot of the RMA-normalized experiment 6 data relative to the pseudo-median reference. The LOESS line is plotted in red

Lastly, it's also useful to visualize the raw fluorescence intensities on the DNA chip using the `image()` function:

```
> par(mfrow=c(1,1))  # revert the canvas back to just one plot per window
> image(MLL.B)       # hit the escape key to exit out
```

Viewing the raw fluorescence intensities can help to find experimental artifacts such as fingerprints or air bubbles on the arrays. For example, in Figure 6.7 we see that the second array does indeed have a fingerprint! As a consequence, we should be wary of that experiment and consider removing it from the analysis. Some preprocessing methods attempt to normalize out any local artifacts on the array like fingerprints or air bubbles - a process called *background correction* or *spatial detrending*. However, later we will discover that even after RMA background correction the data from this experiment is still not like the others.

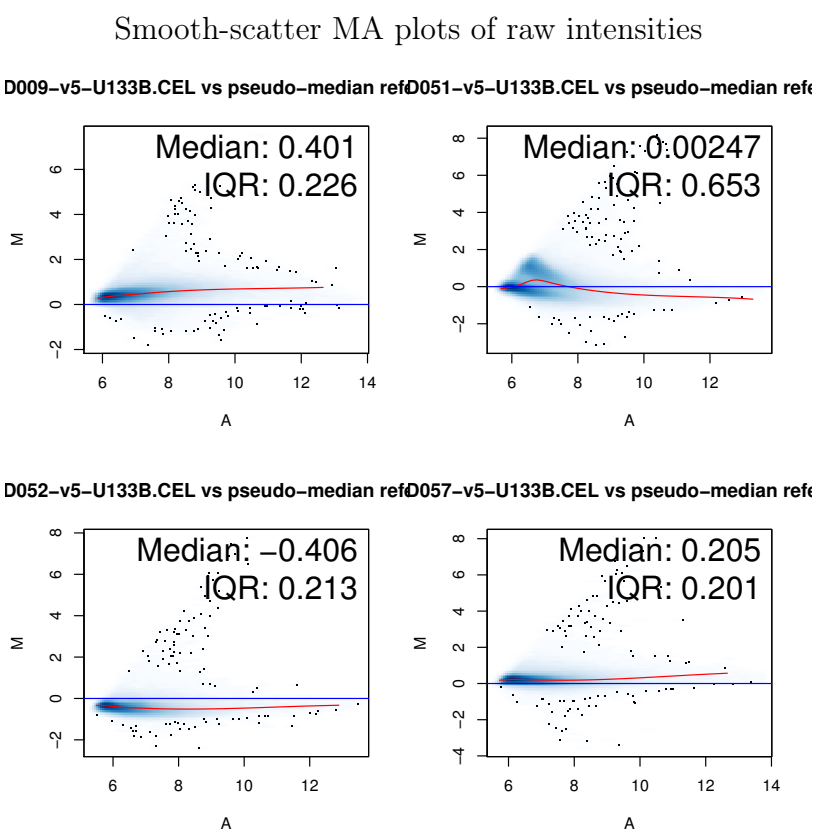


Figure 6.6: Smooth-scatter MA plots of the first 4 samples relative to the pseudo-median reference. The LOESS line through the data is plotted in red.

## 6.2 Preprocessing methods

Using the visualization methods above, it's clear that preprocessing of the probe intensities is necessary before making biologically relevant conclusions from our 20 samples. The Bioconductor packages provide facilities for various preprocessing methods. Here we will show how to use the most common methods for proper preprocessing. Preprocessing consists of three major steps: background correction, normalization, and summarization. To obtain the available background and pm correction methods, use the following:

```
> bgcorrect.methods
```

Image of raw fluorescence intensities with a fingerprint

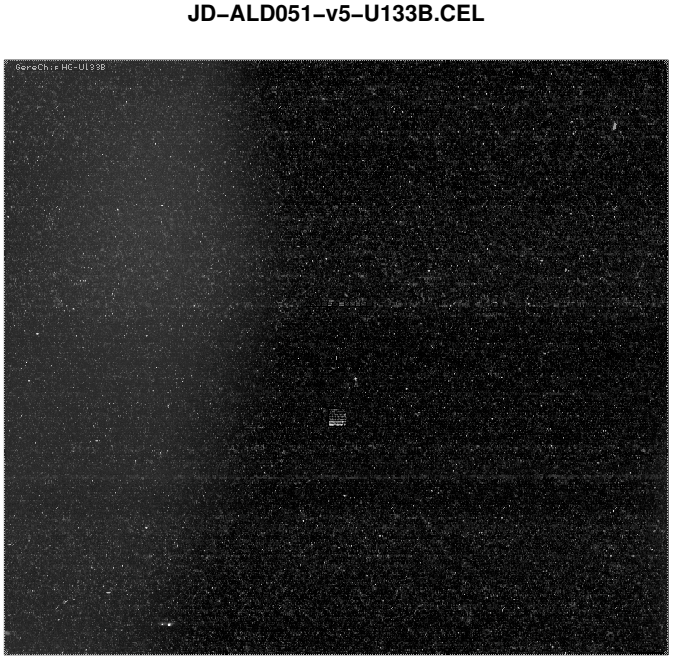


Figure 6.7: The image of the raw fluorescence intensities shows a fingerprint on the left-hand side. If this major artifact cannot be normalized out through background correction, then this experiment should be discarded.

```
[1] "mas"  "none" "rma"  "rma2"
```

The "mas" background correction is part of the MAS Affymetrix software and is based on the 2% lowest probe values. The "rma" correction uses only the PM values, rejects the MM values totally, is based on conditional expectation, and assumes normality of the probes values. There are also a number of correction methods available for just the PM values:

```
> pmcorrect.methods
[1] "mas"        "pmonly"     "subtractmm"
```

Next, the following normalization methods are available:

```
> normalize.methods(MLL.B)
[1] "constant"         "contrasts"        "invariantset"     "loess"
[5] "qspline"          "quantiles"        "quantiles.robust"
```

The "constant" normalization method is a scaling method equivalent to linear regression on a reference array without an intercept term. More general are the non-linear normalization methods such as "loess", "qspline", "quantiles", and "quantiles.robust". "Loess" is a nonlinear method based on local regression of MA plots. The method of "contrasts" is based on loess regression as well. Quantile normalization is an inverse transformation of the empirical distribution with respect to an averaged sample quantile in order to impose the same distribution to each array. The method "qspline" uses quantiles from each array and a target array to fit a system of cubic splines. The target should be the mean (geometric) or median of each probe, but could also be the name of a particular group.

The final step of preprocessing is to aggregate (or summarize) the multiple probe intensities within a probe set into a single gene expression value. The available summarization methods are:

```
> express.summary.stat.methods
[1] "avgdiff" "liwong"  "mas" "medianpolish" "playerout"
```

The first method, "avgdiff", is the simplest as it is based on averaging.

There is no single best method for all preprocessing problems. However, since microarray data can be very noisy due to cross-hybridization and other platform specific aritifacts, it's always wise to use methods robust against outliers together with non-linear normalization methods.

**Example 1: Using expresso().** The three pre-processing steps can be employed one after the other by the function `expresso()`. To combine the background correction from RMA, constant normalization, and average differences summarization for the estimation of mRNA expression values, we may execute the following:

```
> eset <- expresso(MLL.B,bgcorrect.method="rma",
  normalize.method="constant",pmcorrect.method="pmonly",
  summary.method="avgdiff")
```

**Example 2: Using rma().** A frequently applied complete preprocessing method is *RMA*. It combines convolution background correction, quantile normalization, and summarization based on a robust, multi-array model fit called the *median polish algorithm*.

```
> library(affy)
> data(MLL.B, package = "ALLMLL")
> eset2 <- rma(MLL.B)
Background correcting
Normalizing
Calculating Expression
> mybreaks = seq(from=0, to=max(exprs(eset2)), length.out=30)
> hists = apply(exprs(eset2), 2, function(x) {h = hist(x, breaks=mybreaks, plot=FALSE); h$
    ↪ density})
> hists.matrix = matrix(unlist(hists), ncol = 20, byrow = FALSE)
> matplot(hists.matrix, type="l")
> MAplot(eset2, which=c(6), cex.lab=1.5)
```

The three stages of preprocessing used by `rma()` are part of the output. Also notice that before a histogram plot can be constructed, the expression values need to be extracted from the `eset2` object by using the `exprs()` function. In Figure 6.3, the resultant histograms of the RMA-normalized probe intensities for all 20 samples shows only one outlier - which is from the experiment with the fingerprint. Also, in Figure 6.5 the MA plot shows that the RMA-normalized intensities have considerably less bias relative to the pseudo-median reference as compared to the unnormalized intensities in Figure 6.4.

After the three main preprocessing steps above, it is often desirable to further preprocess the data in order to remove patient specific means or medians. One benefit of zero-median rescaling is that testing for a gene to have mean expression value different from zero then becomes meaningful.

**Example 3: Application to the Chiaretti et al. (2004) data.** In the rest of this chapter, we shall frequently work with the Chiaretti et al. (2004) `ALL` data from the `ALL` package of Bioconductor. Here, the data set is briefly introduced (see also Section 1.1) and post-preprocessing steps are illustrated. The raw data have been jointly normalized by RMA and are available in the form of an exprSet object. In this data set, $12,625$ gene expression values are available from microarray experiments using samples from 128 patients suffering from acute lymphoblastic leukemia (ALL). A number of interesting phenotypical co-variates are also available. For instance, the `ALL$t(9;22)` variable has TRUE/FALSE values for each of the 128 patients depending on whether a reciprocal translocation occurred between the long arms of chromosomes 9 and 22 - which has been associated with both chronic and acute leukemia. We can also execute `table(ALL$BT)` to obtain an overview of the numbers of patients which are in certain phases of a disease. Use the

general `?ALL` help command for further information on the data or the article by Chiaretti et al. (2004).

```
> data(ALL, package = "ALL")
> slotNames(ALL)
[1] "assayData"         "phenoData"         "featureData"
[4] "experimentData"    "annotation"        ".__classVersion__"
> row.names(exprs(ALL))[1:10]
 [1] "1000_at"   "1001_at"   "1002_f_at" "1003_s_at" "1004_at"   "1005_at"
 [7] "1006_at"   "1007_s_at" "1008_f_at" "1009_at"
```

In the case when the gene expression values of the patients are non-normally distributed, one may want to subtract the median and divide by the MAD (as opposed to subtracting the mean, and dividing by the standard deviation). An efficient manner to do so is to first use an `apply()` function to compute the column MAD and median, and then the `sweep()` function to subtract the median from each column and then divide by the MAD:

```
> ALL1pp <- ALL1 <- ALL[,ALL$mol == "ALL1/AF4"]
> mads <- apply(exprs(ALL1), 2, mad)
> meds <- apply(exprs(ALL1), 2, median)
> dat <- sweep(exprs(ALL1), 2, meds)
> exprs(ALL1pp) <- sweep(dat, 2, mads, FUN="/")
> boxplot(data.frame(exprs(ALL1)), col="blue")
> boxplot(data.frame(exprs(ALL1pp)), col="magenta")
```

Above, we first select the patients with the "molecular biology" labeled as "ALL1/AF4". ALL1/AF4 is a fusion protein due to a chromosomal translocation between chromosomes 4 and 11: $t(4;11)$. Then `ALL1` variable is copied in order to overwrite the expression values in a later stage. The median and the MAD are computed per column by the specification 2 (column index) in the `apply()` function. Then the first `sweep()` function subtracts the medians from the expression values, while the second `sweep()` divides these by the corresponding MAD. By comparing the box plots in Figure 6.8 and 6.9 the effect of preprocessing can be observed. The medians of the preprocessed data are equal to zero and the variation is smaller due to the division by their MAD. Notice that with box plots we can get some quick first impressions of the distributions of the columns in a data.frame.

## 6.3 Gene filtering

Now we will show a few common methods for filtering genes. Also, keep in mind that there are statistical as well as biological criteria for filtering genes,

Raw probe intensities before robust rescaling

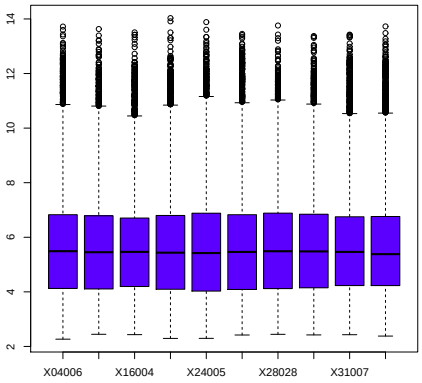


Figure 6.8: Box plot of 10 probes from the ALL1/AF4 patients before median subtraction and MAD division.

Raw probe intensities after robust rescaling

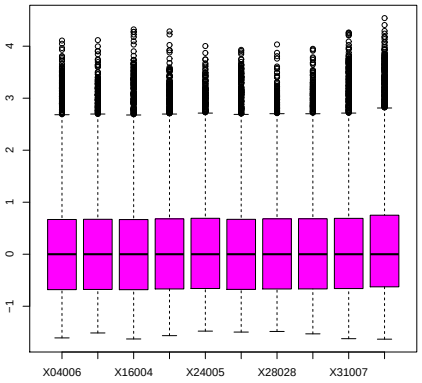


Figure 6.9: Boxplot of the same 10 probes from the ALL1/AF4 patients after median subtraction and MAD division.

and that a combination of these often gives the most satisfactory results. The following examples stress the importance of careful thinking.

**Example 1.: Filtering by the coefficient of variation.** One manner to filter genes is by the coefficient of variation, which is defined as the standard deviation divided by the absolute value of the mean: $cv = \sigma/|\mu|$. If $cv = 1$, then the standard deviation equals the mean, so that the experimental effect is small relative to the precision of measurement. If, however, $cv < 0.2$, then the mean is five times larger than the standard deviation, so that both the experimental effect and the measurement precision are large. Below, we compute the coefficient of variation per gene for the `ALL1pp` data of the previous section. Then, using the `sum()` function yields 4751 genes with a coefficient of variation smaller than 0.2:

```
> cvval <- apply(exprs(ALL1pp),1,function(x){sd(x)/abs(mean(x))})
> sum(cvval<0.2)
[1] 4751
```

**Example 2: Combining several filters.** It's often desirable to combine several filters - which can be performed iteratively on the data set without

any helper function. However, it's often convenient to use the helper function `filterfun()` to combine several filters into one combined filter and then perform the actual filtering just once. The code below is an example of when it's useful to first combine the several filtering functions and then apply them all at once on a data set:

```
> library("genefilter")
> f1 <- function(x)(IQR(x)>0.5)
> f2 <- pOverA(.25, log2(100))
> f3 <- function(x) (median(2^x) > 300)
> f4 <- function(x) (shapiro.test(x)$p.value > 0.05)
> f5 <- function(x) (sd(x)/abs(mean(x))<0.1)
> f6 <- function(x) (sqrt(10)* abs(mean(x))/sd(x) >  qt(0.975,9))
> ff <- filterfun(f1,f2,f3,f4,f5,f6)
> library("ALL"); data(ALL)
> selected <- genefilter(exprs(ALL[,ALL$BT=="B"]), ff)
> sum(selected)
[1] 317
```

After excuting the above gene filtering, we obtain 317 genes that pass the combined filter. The first function returns TRUE if the interquartile range is larger than 0.5, the second if 25% of the gene expression values is larger than 6.643856, the third if the median of the expression values taken as powers to the base two is larger than 300, the fourth if it passes the Shapiro-Wilk normality test, the fifth if the coefficient of variation is smaller than 0.1, and the sixth if the one-sample $t$-value is significant. The filter functions are combined by `filterfun()` and the function `genefilter()` returns a logical vector indicating whether the gene passed all the filters or failed at least one of them. In order to use filters properly, it's important to think them through. For example, several of the filters above focus on similar properties and are redundant. In particular, since the IQR divided by 1.349 is a robust estimator of the standard deviation, the first filter selects genes with a certain minimal standard deviation. With respect to the third filter, note that $2^x > 300$ is equivalent to $x > {}^2\log(300) \approx 8.228819$, which is highly similar to the second filter. Furthermore, $s/|\overline{x}| < 0.1$ is equivalent to $\sqrt{10}|\overline{x}|/s > 1/\sqrt{10}$, so the last two filters are highly similar.

**Example 3: Filtering by $t$-Test and normality.** One may also want to select genes with respect to $p$-values of a two-sample $t$-Test. Let's use the example of B-cell ALL versus T-cell ALL. This particular $t$-Test can be combined with a normality test in such a way that the $t$-Test is only applied to those genes that pass the Shapiro-Wilk normality test - which saves considerable execution time. The normality test will be applied separately for

both the B-cell ALL patients and for the T-cell ALL patients. To accomplish this, we write a filter function that will be used twice. First, however, we create a logical factor `patientB` indicating patients with B-cell ALL (TRUE) and with T-cell ALL (FALSE). The filter selects genes that have their $p$-value from the Welch two-sample $t$-Test smaller than the significance level 0.05. A logical variable named `preSelected` is defined which attains TRUE only if `select1` and `select2` have the value TRUE. Then the final gene filter `f2` is applied to only those genes which have been preSelected:

```
> library("genefilter");library("ALL"); data(ALL)
> patientB <- factor(ALL$BT %in% c("B","B1","B2","B3","B4"))
> f1 <- function(x) (shapiro.test(x)$p.value > 0.05)
> f2 <- function(x) (t.test(x ~ patientB)$p.value < 0.05)
> select1 <- genefilter(exprs(ALL[,patientB==TRUE]), filterfun(f1))
> select2 <- genefilter(exprs(ALL[,patientB==FALSE]), filterfun(f1))
> preSelected <- select1 & select2
> preSelectedALLs <- ALL[preSelected,]
> select3 <- genefilter(exprs(preSelectedALLs), filterfun(f2))
> selectedALLs <- preSelectedALLs[sel3,]
> dim(selectedALLs)
Features  Samples
    1817      128
```

We have 1817 genes that pass all three filters. For these genes, it holds that the expression values for B-cell ALL patients as well as for T-cell ALL patients are normally distributed (in the sense of non-rejection).

A common technique to visualize how the genes are divided between filters is to construct a Venn diagram. This can conveniently be done by using the `venncounts()` and `vennDiagram()` functions from the `limma` package (Smyth, 2005):

```
> library(limma)
> x <- matrix(c(as.integer(sel1),as.integer(sel2),as.integer(sel3)),ncol = 3,byrow=FALSE)
> colnames(x) <- c("Normally-distributed\nB-cell\n","Normally-distributed\nT-cell\n",
+ "Different\nB-cell vs T-cell\nmeans")
> vc <- vennCounts(x, include="both")
> vennDiagram(vc, circle.col=c("magenta", "cyan", "green"), lwd=15, cex=1.2)
```

From the resulting Venn diagram in Figure 6.10, we see that 1817 genes pass all three filters, 1780 genes pass none, 3406 genes pass the normality tests but not the $t$-Test filter, and so on...

Venn diagram of selected genes

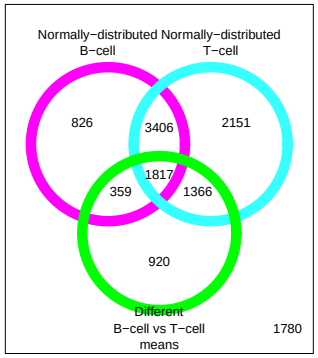


Figure 6.10: Venn diagram of selected ALL genes that pass the three filters represented as circles.

Raw probe intensities after robust rescaling

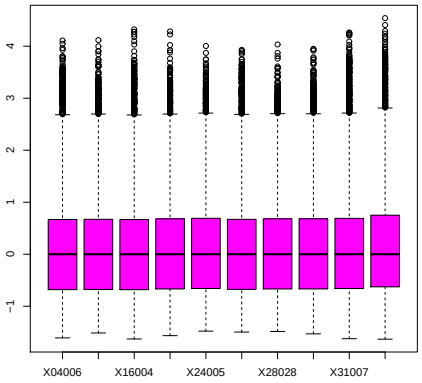


Figure 6.11: Boxplot of the ALL1/AF4 patients after median subtraction and MAD division.

## 6.4 Applications of linear models

The `limma` package is frequently used for analyzing microarray data using linear models, such as ANOVA.

**Example 1: Analysis of variance.** Suppose we are interested in those genes that exhibit the greatest differences between 3 different B-cell leukemia stages: B, B1, and B2. First, we select only those patients with B-cell leukemia in the beginning stage B and in more progressive stages B1 and B2. We specify only those either B, B1, or B2 stage patients by using a factor that defines our model (design) matrix. Then the linear model is fitted to this subset of the data by using `lmFit()`. Next, the `eBayes()` empirical Bayes procedure is used to adapt the gene-specific variances with a global variance estimator (Smyth, 2004)[2]. Next, the `topTable()` function is used to perform multiple hypothesis correction on the $p$-values using the *false discovery rate (FDR)*. Lastly, `topTable()` also sorts our results according to increasing $p$-values:

```
> library("ALL"); library("limma");
```

[2]To obtain the appropriate number of levels we make a factor of ALLB$BT.

```
> data(ALL, package = "ALL")
> allB <- ALL[,which(ALL$BT %in% c("B","B1","B2"))]
> design.ma <- model.matrix(~ 0 + factor(allB$BT)) # treatment contrast matrix without
    ↪ intercept
> colnames(design.ma) <- c("B","B1","B2")
> fit <- lmFit(allB, design.ma)
> fit <- eBayes(fit)
> toptable <- topTable(fit, coef=2,5,adjust.method="fdr")
> print(toptable[,1:5],digits=4)
                  ID logFC AveExpr     t   P.Value
12586 AFFX-hum_alu_at 13.42   13.50 326.0 3.165e-99
2488         32466_at 12.68   12.70 306.3 1.333e-97
2773         32748_at 12.08   12.11 296.3 9.771e-97
5328         35278_at 12.44   12.45 295.5 1.146e-96
4636       34593_g_at 12.64   12.58 278.0 4.431e-95
```

Let's call the mean of the $B$ patients $\mu$, that of B1 $\mu_1$, and that of B2 $\mu_2$. In the example above, we're using the default *treatment contrasts* design matrix which tests the hypotheses $H_0 : \mu - \mu_1$ and $H_0 : \mu - \mu_2$. However, we're not interested in the hypothesis $H_0 : \mu - \mu_2$, because this is the difference between Stage 0 and Stage 3. Rather, we are interested in the hypotheses $H_0 : \mu - \mu_1$ and $H_0 : \mu_1 - \mu_2$. To test the hypotheses we care about, we need to create our own contrast matrix, which can be specified as follows:

```
> cont.ma <- makeContrasts(B-B1,B1-B2, levels=factor(allB$BT))
> cont.ma
      Contrasts
Levels B - B1 B1 - B2
    B       1       0
    B1     -1       1
    B2      0      -1
```

Observe that the contrast matrix specifies the difference between the levels B and B1 as well as between B1 and B2. We can use this contrast matrix to test the right hypotheses as follows:

```
> fit1 <- contrasts.fit(fit, cont.ma)
> fit1 <- eBayes(fit1)
> toptabcon <- topTable(fit, coef=2,5,adjust.method="fdr")
> print(toptabcon[,1:5],digits=4)
> toptabcon <- topTable(fit1, coef=2,5,adjust.method="fdr")
> print(toptabcon[,1:5],digits=4)
          ID   logFC AveExpr      t   P.Value
3389 33358_at  1.4890   5.260  7.374 5.737e-10
419   1389_at -1.7852   9.262 -7.081 1.816e-09
1016  1914_at  2.0976   4.939  7.019 2.315e-09
6939 36873_at  1.8646   4.303  6.426 2.361e-08
7542 37471_at  0.8701   6.551  6.106 8.161e-08
```

Again, we have applied the *false discovery rate* (FDR) multiple hypothesis correction to reduce the number of false positives. Notice that even after FDR correction, the top five genes have very significant $p$-values.

A very convenient manner to summarize, collect, and communicate various types of results is in the form of an HTML table.

**Example 2: Summarizing output in HTML format.** It is often beneficial to combine the typical output from a function like `topTable()` with that of an HTML output page containing various types of information. To illustrate this, we load the `annaffy` annotation package and proceed with the object `toptabcon` from the previous example:

```
> library("annaffy");library("hgu95av2.db")
> anntable <- aafTableAnn(as.character(toptabcon$ID), "hgu95av2.db",
  aaf.handler())
> saveHTML(anntable, "ALLB123.html", title = "B-cell 012 ALL")
```

The function `aafTableAnn()` gathers various types of information available from the output of `topTable()`. The information collected contains the following: Probe, Symbol, Description, Function, Chromosome, Chromosome Location, GenBank, LocusLink, Cytoband, UniGene, PubMed, Gene Ontology, and Pathway. The resulting `anntable` is saved in HTML format in the working directory or the Desktop. It contains a wealth of information on chromosome location, KEGG mappings, summaries from Pubmed articles, etc.

**Example 3: Using ANOVA.** It is also possible to summarize results in an HTML table on the basis of $p$-values from a particular hypothesis test. That is, the selected genes can directly be used as input for `aafTableAnn`:

```
> library("multtest"); library("annaffy"); library("hgu95av2.db")
> library("ALL"); data(ALL, package = "ALL")
> ALLB <- ALL[,which(ALL$BT %in% c("B","B1","B2"))]
> panova <- apply(exprs(ALLB), 1, function(x) anova(lm(x ~ ALLB$BT))$Pr[1])
> genenames <- featureNames(ALLB)[panova<0.000001]
> atab <- aafTableAnn(genenames, "hgu95av2.db", aaf.handler()[c(1:3,8:9,11:13)])
> saveHTML(atab, file="ANOVAonB-cellGroups.html")
```

`hgu95av2.db` is a metadata annotation package providing the information requested by `aaf.handler()`. The meaning of the columns can be obtained from the help page of the function. The resulting `table` is saved as an HTML file in the working directory or desktop. (The `getwd()` function will print the current working directory.) Also, in a similar manner the $p$-values from the Kruskal-Wallis test can be used to select genes.

Bioconductor has a useful facility to download publicly available microarray data sets from NCBI.

**Example 4: Analyzing publicly available data.** The GDS1365 data contain primed macrophage response to IFN-gamma restimulation after different time periods. Note that data taken at different time points is often referred to as *time course* data and almost always calls for paired statistical tests since the identity of the samples has not changed. The purpose of the study is to gain insight into the influence of IFN-gamma priming on IFN-gamma induced transcriptional responses. Among the phenotypical covariates of the data, there is a factor "time" with levels 0, 3 and 24 hours and a factor "protocol" with the levels "IFN-gamma primed" and "unprimed", which can be extracted by the function `pData()`. Since researchers are often interested in the interaction between factors, we shall select genes with a significant interaction effect:

```
> library(GEOquery); library(limma); library(hgu95av2.db);library(annaffy)
> gds <- getGEO("GDS1365")
> eset <- GDS2eSet(gds,do.log2=T)
> prot <- pData(eset)$protocol
> time <- pData(eset)$time
> pval <- apply(exprs(eset)[1:12625,], 1,
   function(x) anova(lm(x ~ prot * time))$Pr[1:3])
> pvalt <- data.frame(t(pval))
> colnames(pvalt) <- c("meffprot","mefftime","interaction")
> genenames <- featureNames(eset)[pvalt$meffprot< 0.01 &
   pvalt$mefftime < 0.01 & pvalt$interaction < 0.01]
> atab <- aafTableAnn(genenames,"hgu95av2.db",aaf.handler()[c(1:3,8:9,11:13)])
> saveHTML(atab, file="Two-way ANOVA protocol by time.html")
```

With the `getGEO()` function, the data are downloaded to the disk and then loaded into the R environment. Next, with the `GDS2eSet()` function the GDS data are transformed to ("2") an expression set so that these can be analyzed statistically. The function `pData()` extracts the factors from the expression set `eset`. Lastly, the function `anova()` extracts the $p$-value of the interaction effect from the estimated linear model. We restrict the analysis to the first 12625 rows because the additional ones contain NA values. The resulting html file contains many interesting genes for further study.

## 6.5 Searching an annotation package

Detailed information about different microarray designs is stored in corresponding annotation packages. As an example, we will look at the annotation package for the human Affy *hgu95av2* gene expression DNA chip:

```
> library("ALL"); data(ALL)
> annotation(ALL)
[1] "hgu95av2"
```

Hence, the annotation package we need is `hgu95av2.db`. Let's load it and obtain an overview of its functionality:

```
> library(hgu95av2.db)
> ls("package:hgu95av2.db")
 [1] "hgu95av2"                "hgu95av2_dbconn"         "hgu95av2_dbfile"
 [4] "hgu95av2_dbInfo"         "hgu95av2_dbschema"       "hgu95av2ACCNUM"
 [7] "hgu95av2ALIAS2PROBE"     "hgu95av2CHR"             "hgu95av2CHRLENGTHS"
[10] "hgu95av2CHRLOC"          "hgu95av2CHRLOCEND"       "hgu95av2ENSEMBL"
[13] "hgu95av2ENSEMBL2PROBE"   "hgu95av2ENTREZID"        "hgu95av2ENZYME"
[16] "hgu95av2ENZYME2PROBE"    "hgu95av2GENENAME"        "hgu95av2GO"
[19] "hgu95av2GO2ALLPROBES"    "hgu95av2GO2PROBE"        "hgu95av2MAP"
[22] "hgu95av2MAPCOUNTS"       "hgu95av2OMIM"            "hgu95av2ORGANISM"
[25] "hgu95av2PATH"            "hgu95av2PATH2PROBE"      "hgu95av2PFAM"
[28] "hgu95av2PMID"            "hgu95av2PMID2PROBE"      "hgu95av2PROSITE"
[31] "hgu95av2REFSEQ"          "hgu95av2SYMBOL"          "hgu95av2UNIGENE"
[34] "hgu95av2UNIPROT"
```

The annotation package contains environments (hash tables) with different types of information. An easy manner to make the content of an environment available is by converting it into a list and printing part of it to the screen:

```
> ChrNrOfProbe <- as.list(hgu95av2CHR)
> ChrNrOfProbe[1]
$`1000_at`
[1] "16"
```

We recognize the manufacturer's identifiers of genes and the corresponding chromosome. Asking information with `?hgu95av2CHR` reveals that it is an environment (hash table) which provides mappings between identifiers and chromosomes. From these we obtain various types of information on the basis of the manufacturer's identifier, such as `"1389_at"`. Below, we obtain the GenBank accession number, Entrez Gene identifier, gene abbreviation, gene name, brief summaries of functions of the gene products, and UniGene identifier, respectively. We use the `get()` function to search an environment for the value to a particular ID (key). Data in environments are commonly referred to as key-value pairs:

```
> get("1389_at", env = hgu95av2ACCNUM)
[1] "J03779"
> get("1389_at", env = hgu95av2ENTREZID)
[1] 4311
> get("1389_at", env = hgu95av2SYMBOL)
[1] "MME"
> get("1389_at", env = hgu95av2GENENAME)
[1] "membrane metallo-endopeptidase (neutral endopeptidase,
```

```
   enkephalinase, CALLA, CD10)"
> get("1389_at", env = hgu95av2SUMFUNC)
[1] NA
> get("1389_at", env = hgu95av2UNIGENE)
[1] "Hs.307734"
```

Let's use the GenBank accession number to search the GenBank nucleotide database:

```
> library(annotate)
> genbank("J03779",disp="browser")
```

From this we obtain the corresponding GI:179833 number, which can be used to obtain a complete XML document:

```
> genbank(1430782,disp="data",type="uid")
```

Obviously, probes correspond to genes and frequently we are interested in their chromosome location, and, specifically, its starting position(s):

```
> get("1389_at", env = hgu95av2CHRLOC)
       3        3        3
156280152 156280327 156280748
```

The cytoband location can also be obtained:

```
> get("1389_at", env = hgu95av2MAP)
[1] "3q25.1-q25.2"
```

Hence, we see that the gene is on Chromosome 3 on q arm band 25, sub-band 1 and 2.

## 6.6 Using annotation to search literature

Given the manufacturerss probe identifier, it's possible to search the literature by collecting Pubmed ID's and then use them to find relevant articles:

```
> library(hgu95av2.db);library(annotate); library(ALL); data(ALL)
> pmid <- get("1389_at",env=hgu95av2PMID)
> pubmed(pmid,disp="browser")
```

Another possibility is to collect a list containing the PubMed ID, authors, abstract, title, journal, and publication date using the `getabst()` and `titles()` functions:

```
> absts <- pm.getabst("1389_at", "hgu95av2")
> pm.titles(absts)
```

Also, the `abstGrep()` function can be used to search the database with regular expressions:

```
> ne <- pm.abstGrep("neutral (endo|exo)peptidase",absts[[1]])
```

Another possibility is to use `pmAbst2HTML()` to construct an HTML table with the titles:

```
> pmAbst2HTML(absts[[1]],filename="pmon1389_at.html")
```

## 6.7 Searching GO numbers and evidence

An *ontology* is any structured language that annotates a conceptual domain. The gene ontology (GO) consortium defines three gene ontologies for categorizing the Molecular Function (MF), Biological Process (BP), and Cellular Component (CC) for the protein coded by a gene. A Molecular Function (MF) describes a phenomenon at the biochemical level such as *enzyme*, *transporter*, or *ligand*. A Biological Process (BP) may coordinate various related molecular functions such as *DNA replication* or *signal transduction*. A Cellular Component (CC) is a unit within a part of the cell such as a *chromosome*, *nucleus*, or *ribosome*.

Each term is identified by a unique GO number. To find GO numbers and their dependencies we use `get()` to extract a list from a GO annotation file. In our example, we want to use the `hgu95av2GO` annotation file. We then use an `apply()` type of function to extract another list containing GO identification numbers:

```
> go1389 <- get("1389_at", env = hgu95av2GO)
> idl <- lapply(go1389,function(x) x$GOID)
> idl[[1]]
[1] "GO:0006508"
```

The list `idl` contains 8 members of which only the first is printed to the screen. We can also use the `getOntology()` function with the `GOID` to extract the `Ontology` which contains more specific information pertaining to the ontology. Also, From the `annotate` package we may now select the GO numbers which are related to a biological process:

```
> library(annotate)
> getOntology(go1389,"BP")
[1] "GO:0006508" "GO:0007267"
```

There are various types of evidence for categorizing the MF, BP, or CC of a gene, such as "inferred from genetic interaction (IGI)", "inferred from electronic annotation (IEA)", "traceable author statement (TAS)", and others. We can use the `getEvidence()` with the GO identifier to obtain the types of evidence.

```
> getEvidence(go1389)
GO:0004245 GO:0005886 GO:0005887 GO:0006508 GO:0007267 GO:0008237 GO:0008270
     "IEA"      "TAS"      "TAS"      "TAS"      "TAS"      "TAS"      "IEA"
GO:0046872
     "IEA"
```

When we now want to select the GO numbers with a particular type of evidence, such as "traceable author statement", we can use the `subset()` function to create a list:

```
> go1389TAS <- subset(go1389,getEvidence(go1389)=="TAS")
```

To extract information from this list we can use the `apply()` type of function `sapply()`:

```
> sapply(go1389TAS,function(x) x$GOID)
> sapply(go1389TAS,function(x) x$Evidence)
> sapply(go1389TAS,function(x) x$Ontology)
```

## 6.8 GO parents and children

The term "transmembrane receptor protein-tyrosine kinase" is more specific and therefore a 'child' of the more general parent term "transmembrane receptor" (Gentleman, et. al, 2005).

**Example 1: Collecting GO information.** There are functions to obtain parents and children from a GO identifier:

```
> GOMFPARENTS$"GO:0003700"
        isa          isa
"GO:0003677" "GO:0030528"
> GOMFCHILDREN$"GO:0003700"
        isa
"GO:0003705"
```

In the case when we have a list of GO identifiers for which we want to collect the ontology, parents, and children identifiers in a vector, we can do the following:

```
> go1389 <- get("1389_at", env = hgu95av2GO)
> gonr <- getOntology(go1389, "BP")
> gP <- getGOParents(gonr)
> gC <- getGOChildren(gonr)
> gPC <- c(gonr,gP,gC)
> pa <- sapply(gP,function(x) x$Parents)
> ch <- sapply(gC,function(x) x$Children)
> gonrc <- c(gonr,unlist(pa),unlist(ch))
```

**Example 2: Probe selection by GO.** A research strategy may be to look at the gene expression of genes with a similar biological process as a certain particularly over-expressed gene. We can start with a probe set number for that gene, find the GO identifiers of the biological process, obtain its parents, and then transform these to probe sets. We can accomplish this with the following:

```
> library(GO); library(annotate); library("ALL"); data(ALL)
> go1389 <- get("1389_at", env = hgu95av2GO)
> gonr <- getOntology(go1389, "BP")
> gP <- getGOParents(gonr)
> pa <- sapply(gP,function(x) x$Parents)
> probes <- mget(pa,hgu95av2GO2ALLPROBES)
> probeNames <- unlist(probes)
> ALLpr <- ALL[probeNames,]
> > dim(exprs(ALLpr))
[1] 7745  128
```

With this approach you may end up with many genes with similar biological processes for further analysis.

## 6.9 Gene filtering by a biological term

An application of working with GO numbers is to filter for genes which are related to a biological term.

**Example 1: Filtering genes by a term.** From a biological point of view it is most interesting to select genes which are related to a certain biological process, such as "transcriptional repression". We can combine this with the previous filter. First, we use `annotation(ALL)` to obtain the GO annotation for the ALL data set. Then we define a function (Gentleman, et al., 2005, p. 123) to collect appropriate GO numbers from that annotation environment `GOTERM`:

```
> library("GO"); library("annotate"); library("hgu95av2.db")
> GOTerm2Tag <- function(term) {
 GTL <- eapply(GOTERM, function(x) {grep(term, x@Term, value=TRUE)})
 Gl <- sapply(GTL, length)
 names(GTL[Gl>0])
}
>  GOTerm2Tag("transcriptional repressor")
[1] "GO:0016564" "GO:0016565" "GO:0016566" "GO:0017053"
```

Above, we're using the functions `eapply()` and `sapply()` to apply the `grep()` function to each term in order to search for possible matches to each

term in the environment GOTERM. A precaution is taken to select only those names which are not empty. The result are the GO terms which can now be translated into probe sets in the ALLs data.

```
> tran1 <- hgu95av2GO2ALLPROBES$"GO:0016564"
> tran2 <- hgu95av2GO2ALLPROBES$"GO:0016566"
> tran3 <- hgu95av2GO2ALLPROBES$"GO:0017053"
> tran <- c(tran1,tran2,tran3)
> inboth <- tran %in% row.names(exprs(ALLs))
> ALLtran <- ALLs[tran[inboth],]
> dim(exprs(ALLtran))
[1] 26
```

The GO translated probe names are intersected with the row names of the data giving the logical variable inboth. The variable tran[inboth] gives the IDs by which genes can be selected. Next, gene IDs for which inboth equals TRUE are selected and the corresponding data are collected in the data.frame ALLtran. More information can be obtained by GOTERM$"GO:0016564". Through dim(exprs(ALLtran)) we observe that 26 genes which passed the normality filter are related to "transcriptional repression".

## 6.10 Significance per chromosome

After a statistical analysis to filter and order genes, it is often quite useful to post the results of the analysis. In particular, after collecting $p$-values from a $t$-Test, one may wonder whether genes with significant $p$-values occur more often within a certain chromosome. To test for such over or under-representation, the Fisher exact test is very useful (see Section 4.1.8).

**Example 1: Two sample $t$-Test.** From the expression values of the ALL data, we want to perform a two sample $t$-Test using (1) the patient group for which remission was achieved and (2) the patient group for which it was not achieved. Per chromosome it can be tested whether the odds ratio differs from 1 or, equivalently, whether there is independence. The data for the test consist of (1) the number of significant probes on Chromosome 19, (2) the number of non-significant probes on Chromosome 19, (3) the number of remaining significant probes, and (4) the number of remaining non-significant probes.

```
> library("ALL"); data(ALL); library("hgu95av2.db")
> rawp <- apply(exprs(ALL), 1, function(x) t.test(x ~ ALL$remission)$p.value)
> xx <- as.list(hgu95av2CHR)
```

```
> AffimIDChr <- names(xx[xx=="19"])
> names(rawp) <-  featureNames(ALL)
> f <- matrix(NA,2,2)
> f[1,1] <- sum(rawp[AffimIDChr]<0.05); f[1,2] <- sum(rawp[AffimIDChr]>0.05)
> f[2,1] <- sum(rawp<0.05) - f[1,1]   ; f[2,2] <- sum(rawp>0.05) - f[1,2]
> print(f)
     [,1]  [,2]
[1,]  106   638
[2,]  832 11049
> fisher.test(f)

        Fisher's Exact Test for Count Data

data:  f
p-value = 4.332e-11
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 1.757949 2.748559
sample estimates:
odds ratio
  2.206211

> chisq.test(f)

        Pearson's Chi-squared test with Yates' continuity correction

data:  f
X-squared = 52.3803, df = 1, p-value = 4.573e-13
```

The number of significant probes is larger for Chromosome 19 resulting in an odds ratio of 2.2. The hypothesis of independence is rejected by both tests.

## 6.11 Overview and concluding remarks

In this chapter, we looked at many examples that used analysis of variance (ANOVA) or $t$-Tests for selecting genes with large experimental effects between different patient groups. Although not complete, we covered many statistical methods used to answer important biological questions with commonly gathered biological data.

## 6.12 Exercises

1. **Gene filtering on normality per group of B-cell ALL patients.**

   (a) Use `genefilter()` to program the Shapiro-Wilk normality test separately for each gene over the groups "B1","B2","B3","B4".

(b) How many pass the filter for all the groups?

(c) Contruct a Venn diagram for groups "B2", "B3", and "B4", plot it, and give a correct interpretation for each number.

2. **Analysis of gene expressions of B-cell ALL patients using Limma.**

   (a) Construct a data.frame containing the expression values for the B-cell ALL patients in stage B, B1, B2, B3, B4 from the `ALL` data.

   (b) Construct the design matrix and an appropriate contrast matrix.

   (c) Compute the twenty best differentially expressed genes with `eBayes()` and `topTable()`.

   (d) Collect information on the twenty best genes in an HTML page.

3. **Finding a row number.** Use grep to find the row number of gene `1389_at`. Hint: Use `row.names()` or `featureNames()`.

4. **Remission from acute lymphocytic leukemia (ALL).** In the `ALL` data from the `ALL` library there is a phenotypic variable called `remission` indicating either complete remission `CR` or refractory `REF` - meaning alleviated from the disease after treatment (at least temporarily) or no response to treatment, respectively.

   (a) How many persons are classified as CR and REF, respectively? Hint: Use `pData()` to extract a data.frame with phenotypical data.

   (b) Program the two-sample $t$-Test, not assuming equal variances, to select genes with $p$-values smaller than 0.001. How many are there? Hint: You may have to select the patients in remission and then exclude not available data (NAs).

   (c) Collect and give the manufacturer's probe names of the genes with $p$-values smaller than 0.001.

   (d) Use the probe names to find the corresponding gene names.

   (e) Is the famous tumor-suppressor protein p53 among them?

   (f) How many unique gene names are there?

5. **Remission achieved.** In the ALL data, the patients are checked for achieving remission after treatment. The variable `ALL$CR` has values CR (became healthy) and REF (did not respond to therapy and remained ill).

   (a) Construct a separate data.frame consisting of only those gene expression values from patients that have values CR or REF.

   (b) How many genes have a $p$-value smaller than 0.0001 from the two-sample $t$-Test, not assuming equal variances? Hint: Use the `apply()` function to program the test.

   (c) Give the Affymetrix names (symbols) of the genes that pass the selection criterion of having a $p$-value smaller than 0.0001.

   (d) Use the Affymtrix names to find the biological names.

   (e) How many oncogenes are there in this set? (Hint: Use `grep()` to search the biological names for "oncogene".)

   (f) Perform the Fisher exact test on the number of oncogenes out of the total versus the number of significant oncogenes out of those selected.

6. **Gene filtering of ALL data.** The patients with T-cell leukemia which are in stages T2 and T3 can be selected by using the variable `ALL$BT`. You can use the function "table" to find the frequencies of the patient types and leukemia stages. Use the functions from the library "genefilter" to answer the questions below:

   (a) Program a gene-filter step separately for T2 and T3 patients such that only normally distributed genes can pass.

   (b) Program a second gene-filter step which passes only those genes with a significant $p$-value from the two sample $t$-Test between two samples.

   (c) How many genes pass all the filter steps?

   (d) How many genes just pass normality?

7. **Stages of B-cell ALL.** Use the limma package to answer the questions below:

(a) Select the patients with T-cell leukemia which are in stage B1, B2, B3, and B4.

(b) What contrast matrix would you like to suggest in this situation? Construct your contrast matrix in R.

(c) Perform analysis of variance (ANOVA) to test the hypothesis of equal population means for all 4 stages. Use the Benjamini & Hochberg (1995) ("BH") adjustment method for the false discovery rate with `topTable()` to report the five best genes.

(d) For how many genes do you reject the null-hypothesis?

8. **Analysis of microarray data on rheumatoid arthritis.**

(a) Download the GDS486 data and transform it into eset form. Here we meet a missing data problem. A manner to solve it is as follows. First, use the function `function(x) sum(is.na(x))` in `apply()` on the rows to count the number of missing values per row. Then select the rows without missing values to perform a two-sample $t$-Test with the groups in `cell.line`. Overwrite the vector with the number of missing values with the $p$-values in a suitable manner.

(b) Download GDS711 and repeat the above procedure using ANOVA $p$-values with the covariate `disease.state` to indicate the groups.

(c) Download GDS2126 and repeat the above procedure using ANOVA $p$-values with the covariate `disease.state` to indicate the groups.

(d) Compute the symbols of the twenty best genes in the sense of having smallest sum of $p$-values.

(e) Summarize information of the twenty best genes in an HTML table. Does p53 play a role in the pathway of the best gene?

9. **Analysis of genes from a GO search.**

(a) Select the patients on the covariate `mol.biol` with values `ALL1/AF4`, `BCR/ABL`, and `NEG`.

(b) Collect the ANOVA $p$-values with contrast between `NEG` and `ALL1/AF4`, and between `NEG` and `BCR/ABL`. Report the total number of significant genes and Affy number for the significant Affy ID's. Hint: Reorder the columns into "NEG", "ALL1/AF4", and "BCR/ABL".

(c) Find the GO ID's referring to the term "protein-tyrosine kinase" (It mediates many steps due to BCR/ABL translocation.)

(d) Select the Affy ID's corresponding to the GO ID's and report the number of Affy ID's in each GO category and how many of them are significant genes according to your test.

(e) Perform a Fisher exact test to test for enrichment of significant genes in the GO categories.