Processing Affymetrix Expression Data

Patrick Aboyoun

Fred Hutchinson Cancer Research Center

April 27, 2009
Lab Structure

- Interactive walkthrough of Chapter 3: Processing Affymetrix Expression Data from the book *Bioconductor Case Studies* by Hahne et al.
- Students encouraged to run R during lab and submit commands as we go because there will be exercises along the way.
Outline

1. Background

2. The input data: CEL files

3. Quality Assessment

4. Preprocessing

5. Ranking and filtering probe sets

6. Advanced preprocessing
Bioconductor Packages Covered

- **affy** - Methods for Affymetrix oligonucleotide arrays
- **CLL** - Chronic Lymphocytic Leukemia gene expression data
- **simpleaffy** - Simple high-level analysis of Affymetrix data
- **genefilter** - Methods for filtering genes from microarray experiments
- **affyPLM** - Methods for fitting probe-level models
- **limma** - Linear models for microarray data
- **annotate** - Annotation for microarrays
- **annaffy** - Annotation tools for Affymetrix biological metadata
- **hgu95av2.db** - Affymetrix Human Genome U95 Set annotation data (chip hgu95av2)
- **KEGG.db** - A set of annotation maps for KEGG
- **geneplotter** - Graphics related functions for Bioconductor
- **vsn** - Variance stabilization and calibration for microarray data
Affymetrix GeneChip® Microarray Overview

1. mRNA target sequence based off of public databases
   - 5' to 3'
   - ~16-20 probes designed to match public database sequence
   - 25 base probe designed to be a perfect match to the public database sequence

2. 25 base oligonucleotide Perfect Match probe
   - Target Sequence: 5' ATGGCTGCAATCGATCATCCGAC 3'
   - Perfect Match PM: ATGGCTGCAATCGATCATCCGAC
   - Mismatch MM: ATGGCTGCAATCGATCATCAGAC
   - Mismatch probe designed to mirror PM with the 13 bases switched

3. Chip scanned and fluorescence intensity recorded

4. Image generated
   - Image quantified by Affymetrix software

Data generated by Affymetrix software

Data output for analysis

Sample loaded onto chip

Labeled mRNA from cells

Cells

Biological Sample

PM and MM probes synthesized on GeneChip
Affymetrix GeneChip® Microarray Terminology

- Each gene or portion of a gene is represented by 1 to 20 oligonucleotides of 25 base-pairs.
- **Probe**: an oligonucleotide of 25 base-pairs, i.e., a 25-mer.
- **Perfect match (PM)**: A 25-mer complementary to a reference sequence of interest (e.g., part of a gene).
- **Mismatch (MM)**: same as PM but with a single base change for the middle (13th) base (transversion purine <-> pyrimidine, G <->C, A <->T). Used to measure non-specific binding and background noise.
- **Probe-pair**: a (PM,MM) pair.
- **Probe-pair set**: a collection of probe-pairs (1 to 20) related to a common gene or fraction of a gene.
- **Affy ID**: an identifier for a probe-pair set.
Affymetrix A to Z

Background

Affymetrix GeneChip® Files

- **DAT** file: Image file, $10^7$ pixels, 50 MB.
- **CEL** file: Cell intensity file, probe level PM and MM values.
- **CDF** file: Chip Description File. Describes which probes go in which probe sets and the location of probe-pair sets (genes, gene fragments, ESTs).
Expression Measures

- 10-20K genes represented by 11-20 pairs of probe intensities (PM & MM).
- Obtain expression measure for each gene on each array by summarizing these pairs.
- Background adjustment and normalization are important issues.
- There are many methods.
Importing Affymetrix GeneChip® Data ...

- Affymetrix GeneChip® CEL files are imported using `ReadAffy` from the `affy` package.

```r
> library("affy")
> myAB1 <- ReadAffy()
> myAB2 <- ReadAffy(filenames = c("a1.cel", "a2.cel", "a3.cel"))
```

- By default, all the CEL files from the current working directory (CWD) are imported. CWD can be found using `getwd` and changed using `setwd`.

- Alternatively, the `filenames` argument can be supplied.

- `list.celfiles` can be used to select the list CEL file in the directory.
... Into *AffyBatch* Objects

- Affymetrix GeneChip® probe-level data are stored in *AffyBatch* objects.
- For more information on this class see `help("AffyBatch-class")`.
- We will use pre-imported data from the *CLL* package.
Example Data Set

- Chronic Lymphocytic Leukemia Gene Expression Data
- 24 samples run on HG-U95Av2 Affymetrix GeneChip® arrays
- Large number of clinical measures collected, but we'll use only one.

```r
> library("CLL")
> data("CLLbatch")
> CLLbatch

AffyBatch object
size of arrays=640x640 features (91212 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=24
number of genes=12625
annotation=hgu95av2
notes=
```
The `sampleNames` function extracts the name of the samples.

The `sampleNames<-` replacement function overwrites the existing names.

```r
> head(sampleNames(CLLbatch))
[1] "CLL10.CEL" "CLL11.CEL" "CLL12.CEL"

> sampleNames(CLLbatch) <- sub("\\.CEL$", 
+ " 
+ "", sampleNames(CLLbatch))
> head(sampleNames(CLLbatch))
[1] "CLL10" "CLL11" "CLL12" "CLL13" "CLL14"
[6] "CLL15"
```
CLL Disease State

> data("disease")
> head(disease)

    SampleID Disease
  1    CLL10   <NA>
  2    CLL11  progres.
  3    CLL12   stable
  4    CLL13  progres.
  5    CLL14  progres.
  6    CLL15  progres.

> table(disease$Disease, useNA = "always")

              progres. stable <NA>
progres.           14         9      1
stable

> rownames(disease) <- disease$SampleID
Managing Phenotypic Data

- Phenotypic data are stored in `AnnotatedDataFrame` objects.
- These are essentially `data.frame` objects with metadata.
- They can be created from other R objects using `new` or imported from a file using `read.AnnotatedDataFrame`.
- The `phenoData` and `phenoData<-` functions get and set the phenotypic data in an `AffyBatch` object.

```r
> mt <- match(rownames(disease), sampleNames(CLlbatch))
> vmd <- data.frame(labelDescription = c("Sample ID", + "Disease status: progressive or stable disease"))
> phenoData(CLlbatch) <- new("AnnotatedDataFrame", + data = disease[mt, ], varMetadata = vmd)
```
Sample Removal

- Phenotypic data can be used to filter AffyBatch objects.
- In this case we will remove the sample with an unknown disease state.

```r
> CLLbatch <- CLLbatch[, !is.na(CLLbatch$Disease)]
```
QA/QC

- **Quality Assessment**: computation and interpretation of metrics that are intended to measure quality.

- **Quality Control**: possible subsequent actions, such as removing data from bad arrays or re-doing parts of an experiment.
Affymetrix Quality Assessment Metrics

- **Average Background**: the average of the background values.
- **Scale Factor**: The constant $i$ which is the ratio of the trimmed mean for array $i$ to the trimmed mean of the reference array.
- **Percent Present**: the percentage of spots that are present according to Affymetrix detection algorithm.
- **3'/5' ratios**: for different quality control probe sets, such as Actin and GAPDH, each represented by 3 probesets, one from the 5' end, one from the middle and one from the 3' end of the targeted transcript. The ratio of the 3' expression to the 5' expression for these genes serves as a measure of RNA quality.
Recommended QA/QC Packages

- The **simpleaffy** package computes a variety of statistics for QA/QC.
- The **affyPLM** package contains methods for fitting probe-level models.
- The **arrayQualityMetrics** and **affyQCRreport** packages contain recommended functionality for generating comprehensive QA reports.

```r
> library("simpleaffy")
> saqc <- qc(CLLbatch)
```
Quality Control Plot

> plot(saqc)

QC Stats

- actin3/actin5
- gapdh3/gapdh5

<table>
<thead>
<tr>
<th>Sample</th>
<th>actin3/actin5</th>
<th>gapdh3/gapdh5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLL9</td>
<td>40.5%</td>
<td>36.65%</td>
</tr>
<tr>
<td>CLL8</td>
<td>41.28%</td>
<td>37.39%</td>
</tr>
<tr>
<td>CLL7</td>
<td>40.7%</td>
<td>32.74%</td>
</tr>
<tr>
<td>CLL6</td>
<td>37.31%</td>
<td>36.65%</td>
</tr>
<tr>
<td>CLL5</td>
<td>37.8%</td>
<td>34.23%</td>
</tr>
<tr>
<td>CLL4</td>
<td>37.2%</td>
<td>34.38%</td>
</tr>
<tr>
<td>CLL3</td>
<td>39.6%</td>
<td>38.89%</td>
</tr>
<tr>
<td>CLL2</td>
<td>40%</td>
<td>34%</td>
</tr>
<tr>
<td>CLL24</td>
<td>43%</td>
<td>39.03%</td>
</tr>
<tr>
<td>CLL23</td>
<td>43%</td>
<td>39.03%</td>
</tr>
<tr>
<td>CLL22</td>
<td>43.35%</td>
<td>41.28%</td>
</tr>
<tr>
<td>CLL21</td>
<td>42.4%</td>
<td>41.5%</td>
</tr>
<tr>
<td>CLL20</td>
<td>41.6%</td>
<td>38.24%</td>
</tr>
<tr>
<td>CLL2</td>
<td>40.57%</td>
<td>39.6%</td>
</tr>
<tr>
<td>CLL1</td>
<td>39.6%</td>
<td>39%</td>
</tr>
<tr>
<td>CLL19</td>
<td>39.6%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL18</td>
<td>39.5%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL17</td>
<td>39.5%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL16</td>
<td>39.6%</td>
<td>39.6%</td>
</tr>
<tr>
<td>CLL15</td>
<td>39.5%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL14</td>
<td>39.5%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL13</td>
<td>39.5%</td>
<td>39.5%</td>
</tr>
<tr>
<td>CLL12</td>
<td>40.6%</td>
<td>40.6%</td>
</tr>
<tr>
<td>CLL11</td>
<td>41.3%</td>
<td>41.3%</td>
</tr>
</tbody>
</table>
Clustering Arrays by Expression Data

- Microarrays can be clustered based on expression profiles to determine outlining samples.
- `genefilter`'s `dist2` calculates pairwise distances that can be fed into cluster analysis software.
  - By default, `dist2` calculates the mean of the absolute differences between pairs of samples.

```r
> library("genefilter")
> dd <- dist2(log2(exprs(CLLbatch)))
```
Between-Array Distance Plot (1/2)

```r
> diag(dd) <- 0
> dd.row <- as.dendrogram(hclust(as.dist(dd)))
> row.ord <- order.dendrogram(dd.row)
> library("latticeExtra")
> legend <- list(top = list(fun = dendrogramGrob,
+   args = list(x = dd.row, side = "top")))
> lp <- levelplot(dd[row.ord, row.ord],
+   scales = list(x = list(rot = 90)),
+   xlab = "", ylab = "", legend = legend)
```
Between-Array Distance Plot (2/2)

> plot(lp)
The \texttt{affyPLM} package provides another set of QA/QC diagnostic measures:

- Relative Log Expression (RLE) - For each gene, the across array median log expression is subtracted from each individual measurement.
- Normalize Unscaled Standard Error (NUSE) - see documentation for mathematical definition.

For both these measures, boxplots are used to highlight aberrant samples.
Probe-Level Diagnostics (1/3)

```r
> library("affyPLM")
> dataPLM <- fitPLM(CLLbatch)

> boxplot(dataPLM, main = "NUSE", ylim = c(0.95, 1.22), outline = FALSE, col = "lightblue", las = 3, whisklty = 0, staplelty = 0)

> Mbox(dataPLM, main = "RLE", ylim = c(-0.4, 0.4), outline = FALSE, col = "mistyrose", las = 3, whisklty = 0, staplelty = 0)
```
Probe-Level Diagnostics (2/3)

NUSE

RLE
Both diagnostic boxplots indicate that array CLL1 is problematic.

We drop it from our further analysis.

```r
> badArray <- match("CLL1", sampleNames(CLLbatch))
> CLLB <- CLLbatch[, -badArray]
```
Exercise 3.1

- Repeat the calculation of the NUSE and RLE plots for the data with the array CLL1 removed.
Solution Exercise 3.1 (1/2)

```r
> dataPLMx <- fitPLM(CLLB)

> boxplot(dataPLMx, main = "NUSE", ylim = c(0.95, 1.3), outline = FALSE, col = "lightblue",
  +       las = 3, whisklty = 0, staplelty = 0)

> Mbox(dataPLMx, main = "RLE", ylim = c(-0.4, 0.4), outline = FALSE, col = "mistyrose",
  +       las = 3, whisklty = 0, staplelty = 0)
```
Solution Exercise 3.1 (2/2)

NUSE

RLE
Expression Microarray Preprocessing Tasks

- Background correction; increases sensitivity by removing non-specific signal
- Between-array normalization; adjusts for technical variability within an experiment. **Between experiment variability should be modeled, not normalized away.**
- Reporter summarization; summarize gene expression value for each gene from all array features that target its transcripts.
Robust multi-array analysis (RMA) techniques provides all three of the preprocessing tasks.

- Works with $n = 2$ or more chips.
- Method provided and documented in affy package as the rma function.
- The threestep function in the affyPLM package allows for more flexibility in performing these three tasks.

```r
> CLLrma <- rma(CLLB)
```
RMA Results

- The *rma* function returns an *ExpressionSet* object.
- These expression values are on the log$_2$ scale.
- The sample information is transferred to the output.

```r
> exprs(CLLrma)[1:3, 1:3]

    CLL11   CLL12   CLL13
100_g_at 7.997251 7.939522 8.068330
1000_at  8.351710 8.560025 8.206671
1001_at  4.565553 4.466520 4.645425
```

```r
> pData(CLLrma)[1:3,]

       SampleID Disease
CLL11    CLL11  progres.
CLL12    CLL12  stable
CLL13    CLL13  progres.
```
Exercise 3.2

- How many probe sets are there in this dataset?
Solution Exercise 3.2

■ There are lots of solutions. Here are some:

```r
> e <- exprs(CLLrma)
> dim(e)[1]
[1] 12625
> nrow(e)
[1] 12625
> dim(exprs(CLLrma))[1]
[1] 12625
> nrow(CLLrma)
Features
   12625
> length(featureNames(CLLrma))
[1] 12625
```
Non-Specific Filtering After RMA

- Genome-wide microarrays are sensitive up to 50% of the genes being differentially expressed.
- Non-informative genes add noise and filtering them out benefits downstream analyses.
- The `nsFilter` function from the `genefilter` package filters probe sets on various criteria.
- Let’s filter out genes with small variance across samples, no Entrez Gene identifiers, and Affymetrix control probes.

```r
> CLLf <- nsFilter(CLLrma, remove.dupEntrez = FALSE, +       var.cutoof = 0.5)$eset
```
Individual $t$-tests

- Average log-fold change within phenotypic groups are a good naive measure to compare.
- The `rowMeans` function can calculate these statistics from an expression matrix.
- The `rowttests` function from the `genefilter` package performs Student’s $t$-test on each row of a matrix using the average log-fold changes from two groups.
- Similarly, the `rowFtests` function performs $F$-tests for comparing multiple groups.

```r
> a <- rowMeans(exprs(CLLf))
> CLLtt <- rowttests(CLLf, "Disease")
> names(CLLtt)

[1] "statistic" "dm"       "p.value"
```
Exercise 3.3

- Does the variability of the log-ratio values depend on the average intensity?
- Plot the log-ratio against the average intensity
- Plot log-ratio versus the rank of the average intensity

```r
> par(mfrow = c(1, 2))
> myPlot <- function(...) {
+   plot(y = CLLtt$dm, pch = ".", ylim = c(-2, 2), ylab = "log-ratio", ...)
+   abline(h = 0, col = "blue")
+ }
> myPlot(x = a, xlab = "average intensity")
> myPlot(x = rank(a), xlab = "rank of average intensity")
```
Solution Exercise 3.3
Pooled \( t \)-tests

- Previous \( t \)-tests treated each probe set separately, resulting in little data used to estimate variance.
- \( t \)-statistics sensitive to variance estimate.
- Can improve testing using pooled probe set variances in classical linear model or in an empirical Bayesian approach.
- Both approaches tend to perform equally well with ten or more samples in each group.

- The \texttt{lmFit} and \texttt{eBayes} functions from the \texttt{limma} package perform the former and later respectively.

\begin{verbatim}
> library("limma")
> design <- model.matrix(~CLLf$Disease)
> CLLlim <- lmFit(CLLf, design)
> CLLeb <- eBayes(CLLlim)
\end{verbatim}
Exercise 3.4

- Compare the $t$-statistics obtained under the classical linear model and (moderated) empirical Bayes approaches.

```r
> plot(CLLtt$statistic, -CLLeb$t[, 2],
+     xlim = c(-6, 6), ylim = c(-6, 6),
+     xlab = "Classical Linear Model",
+     ylab = "Empirical Bayes", main = "Comparison of $t$ Statistics",
+     pch = ".")
> abline(a = 0, b = 1)
```
Solution Exercise 3.4

Comparison of t Statistics

Classical Linear Model

Empirical Bayes
Volcano Plot (1/2)

- A volcano plot displays statistic, like t-statistic $p$-value, used for ranking against log-fold change.
- Reference lines are often added to plot to determine statistical and/or biological significance.

```r
> plot(CELLtt$dm, -log10(CELLtt$p.value),
+       pch = ".", xlab = "log-ratio", ylab = expression(-log[10](p)),
+       p)
> abline(h = 2)
```
Volcano Plot (2/2)
Exercise 3.5

- Generate a volcano plot using a moderated empirical Bayesian $t$-statistic.
- Does it look similar to the plot using the classical $t$-statistic?
Solution Exercise 3.5 (1/2)

```r
> plot(CLtt$dm, -log10(CLLeb$p.value[, + 2]), pch = ".", xlab = "log-ratio", + ylab = expression(log[10] ~ p))
> abline(h = 2)
```
Solution Exercise 3.5 (2/2)
Exercise 3.6

- Use a volcano plot to highlight the top 25 genes with the smallest classical linear model \( p \)-value.

```r
> plot(CLLtt$dm, -log10(CLLtt$p.value),
+      pch = ".", xlab = "log-ratio", ylab = expression(log[10] ~ p))
> o1 <- order(abs(CLLtt$dm), decreasing = TRUE)[1:25]
> points(CLLtt$dm[o1], -log10(CLLtt$p.value)[o1],
+       pch = 18, col = "blue")
```
Solution Exercise 3.6
Multiple Testing Problem

- Due to high number of statistical tests, rule of thumb cutoffs like 0.01 for $p$-values yield many false positives.

- For example, using if none of the 6098 probe sets in CLLf were differentially expressed, the above cutoff would yield on average around 61 false positive significant test results.
Exercise 3.7

- How many probe sets with \( p \leq 0.01 \) are there in the classical linear model \( t \)-tests?
- How many with the moderated empirical Bayesian \( t \)-tests?
Solution Exercise 3.7

> sum(CLLt$t$p.value <= 0.01)
[1] 242

> sum(CLLe$b$p.value[, 2] <= 0.01)
[1] 260
Many methods exist for dealing with the multiple testing problem. 

The `multtest` package provides implementation of many of these methods.

Alternatively, the `topTable` function from the `limma` package contains multiple testing adjustment methods, including Benjamini and Hochberg’s false discovery rate FDR, simple Bonferroni correction, and several others.

```r
> tab <- topTable(CLLeb, coef = 2, adjust.method = "BH", +                   n = 10)
> genenames <- as.character(tab$ID)
> genenames
[1] "1303_at" "33791_at" "37636_at" "36131_at"
[5] "36939_at" "36129_at" "551_at" "41776_at"
[9] "39400_at" "36122_at"
```
Microarray Annotations

- The **annotate** package provides facilities to navigate microarray annotations.
- Many packages like **hgu95av2.db** contain platform-specific annotations.

```r
> library("annotate")
> annotation(CLLf)
[1] "hgu95av2"
> suppressMessages(library("hgu95av2.db"))
```
EntrezGene ID and Gene Symbol

The getEG and getSYMBOL functions from the *annotate* package return the EntrezGene ID and gene symbol for a specified annotation package respectively.

```r
> ll <- getEG(genenames, "hgu95av2")
1303_at 33791_at 37636_at 36131_at 36939_at
 "6452" "10301" "9767" "1192" "2823"
36129_at 551_at 41776_at 39400_at 36122_at
 "9905" "2033" "475" "23102" "5687"
> sym <- getSYMBOL(genenames, "hgu95av2")
1303_at 33791_at 37636_at 36131_at
 "SH3BP2" "DLEU1" "PHF16" "CLIC1"
36939_at 36129_at 551_at 41776_at
 "GPM6A" "SGSM2" "EP300" "ATOX1"
39400_at 36122_at
 "TBC1D2B" "PSMA6"
```
The `htmlpage` from the `annotate` package creates an HTML table containing both static information as well as links to various online annotation sources.

```r
> tab <- data.frame(sym, signif(tab[, -1], 3))
> htmlpage(list(ll), othernames = tab,
+    filename = "GeneList1.html", title = "HTML report",
+    table.center = TRUE, table.head = c("Entrez ID",
+        colnames(tab)))
> browseURL("GeneList1.html")
```
HTML Reporting of Top Genes (2/2)

- The aafTableAnn from the annaffy package creates an alternative HTML table containing both static information as well as links to various online annotation sources.
- The colnames argument can be used to select a subset of the metadata for the specified probe IDs.

```r
> library("annaffy")
> library("KEGG.db")
> atab <- aafTableAnn(genenames, "hgu95av2.db",
+    aaf.handler())
> saveHTML(atab, file = "GeneList2.html")
> atab <- aafTableAnn(genenames, "hgu95av2.db",
+    aaf.handler()[c(2, 5, 8, 12)])
> saveHTML(atab, file = "GeneList3.html")
> browseURL("GeneList2.html")
> browseURL("GeneList3.html")
```
Advanced Preprocessing

- Bioconductor provides functionality for performing custom preprocessing.
- This preprocessing task starts with the “raw” feature intensity data from the CEL files and the assignment of features to target genes information from CDF files.
Probe-Level Data

- Probe-level data from Affymetrix GeneChip® Microarrays originate from CEL files in the form of PM and MM probe intensities.
- The `pm` and `mm` functions from the `affy` package extract data from PM and MM probes respectively.

```r
> pms <- pm(CLLB)
> mms <- mm(CLLB)
```
Exercise 3.8 (1/3)

- For the first array in the CLL data, make a scatterplot of the PM values versus the MM values and interpret the results.
- How many MM probes have larger intensities than their corresponding PM probes?

```r
> smoothScatter(log2(mms[, 1]), log2(pms[, + 1]), xlab = expression(log[2] * "MM values"), + ylab = expression(log[2] * "PM values"), + asp = 1)
> abline(a = 0, b = 1, col = "red")
```
Exercise 3.8 (2/3)
Exercise 3.8 (3/3)

- In a large number of cases, the MM value is larger than the PM value.
- This complicates the simple story that hybridization to the perfect match probe exceeds that of the mismatch probe.

```r
> table(sign(pms - mms))
   -1   0   1
1414590 31828 2993182
```
Exercise 3.9 (1/2)

- For the second array, make a histogram of the MMs for which \( PM > 2000 \)?
- Compare it to the histogram where the PM values are less than 2000.

```r
> library("geneplotter")
> grouping <- cut(log2(pms)[, 2], breaks = c(-Inf, log2(2000), Inf), labels = c("Low", "High"))
> multidensity(log2(mms)[, 2] ~ grouping,
+ main = "", xlab = "", col = c("red", "blue"), lwd = 2)
> legend("topright", levels(grouping),
+ lty = 1, lwd = 2, col = c("red", "blue"))
```
Exercise 3.9 (2/2)
Optical noise and cross-hybridizations results in positive feature intensities from Affymetrix microarrays even when expected to be zero.

Background correction is essential to obtain good sensitivity.

We will explore robust multi-array analysis (RMA) and variance-stabilizing normalization (VSN) methods.
Background Correction using RMA

- In RMA, background modeled using a Normal-Exponential mixture.
- PM intensity values corrected by subtracting background estimate of each probe.
- Corrected PM values guaranteed to be positive.

```r
> bgrma <- bg.correct.rma(CLLB)
> exprs(bgrma) <- log2(exprs(bgrma))
```
Background Correction using VSN

- In VSN, one overall background estimate is computed for the whole array.
- This estimate can be larger than some of the smaller feature intensities on the array, resulting in some of the background-subtracted values \( \leq 0 \).
- Generalized logarithm transformation, which handles nonpositive values, applied to corrected values.
- The \texttt{justvsn} function from the \texttt{vsn} package fits the \texttt{vsn} model.

```r
> library("vsn")
> bgvsn <- justvsn(CLLB)
```
Exercise 3.10 (1/3)

- Compare the results of the two background-correction methods to the original values and between each other. Use a subset of 500 randomly selected PM probes to speed up calculations.

```r
> sel <- sample(unlist(indexProbes(CLLB, "pm")), 500)
> sel <- sel[order(exprs(CLLB)[sel, 1])]
> yo <- exprs(CLLB)[sel, 1]
> yr <- exprs(bgrma)[sel, 1]
> yv <- exprs(bgvsn)[sel, 1]
```
Exercise 3.10 (2/3)

```r
> par(mfrow = c(1, 3))
> plot(yo, yr, xlab = "Original", ylab = "RMA",
+     log = "x", type = "l", asp = 1)
> plot(yo, yv, xlab = "Original", ylab = "VSN",
+     log = "x", type = "l", asp = 1)
> plot(yr, yv, xlab = "RMA", ylab = "VSN",
+     type = "l", asp = 1)
```
Exercise 3.10 (3/3)
VSN Preprocessing

- The `vsnrma` function from the `vsn` package summarizes the probe sets after performing probe-wise background correction and between-array normalization using VSN.
- Its output is derived from the `affy`'s `rma` function.

```r
> vsnrma
> CLLvsn <- vsnrma(CLLB)
```
Non-Specific Filtering After VSN

- We can repeat the non-specific filtering and testing for differential expression that was performed after RMA.

```r
> CLLvsnf <- nsFilter(CLLvsn, remove.dupEntrez = FALSE, +                   var.cutoff = 0.5)$eset
> CLLvsntt <- rowttests(CLLvsnf, "Disease")
```
Exercise 3.11 (1/2)

- Compare the results in \texttt{CLLvsntt} with those obtained for the RMA derived \texttt{CLLtt}.
- Produce a scatterplot between the \textit{t}-statistics obtained for both cases.

```r
> inboth <- intersect(featureNames(CLLvsnf),
+                     featureNames(CLLf))

> plot(CLLtt[inboth, "statistic"], CLLvsntt[inboth,
+                                "statistic"], pch = ".", xlab = "RMA",
+      ylab = "VSN", asp = 1)
```
Exercise 3.11 (2/2)
Reporter Summarization

- Bioconductor provides functionality for performing custom probe set (reporter) summarization.
- This is performed using data are obtained from the CDF file.

```r
> pns <- probeNames(CLLB)
> indices <- split(seq(along = pns), pns)
> length(indices)
[1] 12625

> indices[["189_s_at"]]

[1] 15714 15715 15716 15717 15718 15719 15720
[8] 15721 15722 15723 15724 15725 15726 15727
[15] 15728 15729
```
Exercise 3.12 (1/2)

Can you plot the PM and MM intensities for the probes of one probe set across a set of arrays?

```r
> colors <- brewer.pal(8, "Dark2")
> Index <- indices["189_s_at"] [seq(along = colors)]
> matplot(t(pms[Index, 1:12]), pch = "P",
+       log = "y", type = "b", lty = 1, main = "189_s_at",
+       xlab = "samples", ylab = expression(log[2] ~ Intensity), ylim = c(50, 2000),
+       col = colors)
> matplot(t(mms[Index, 1:12]), pch = "M",
+       log = "y", type = "b", lty = 3, add = TRUE,
+       col = colors)
```
Exercise 3.12 (2/2)

189_s_at

samples

log₂ Intensity

samples

2 4 6 8 10 12
50 100 200 500 1000 2000
One naive (robust) summarization of the probe sets is to take the median difference between the PM and MM values for each sample.

```r
> newsummary <- t(sapply(indices, function(j) rowMedians(t(pms[j,] - mms[j,]))))
> dim(newsummary)
[1] 12625   22
```
Exercise 3.13

- What percent of probe sets, for each array, yield negative values for each array? Will this concern biologist who are unhappy with negative expression estimates?

\[
> \text{colMeans(newsummary} < 0) \times 100
\]

Looking Forward

- Chapter 4 addresses these same issues for two-color arrays.
- Chapter 5 contains more on variance stabilization and calibration for microarray data.
- Questions?