Description of affy

Laurent Gautier, Rafael Irizarry, Leslie Cope, and Ben Bolstad

October 24, 2023

Contents

1 Introduction 2

2 Changes for affy in BioC 1.8 release 3

3 Getting Started: From probe level data to expression values 3

3.1 Quick start ........................................ 4

3.2 Reading CEL file information .................................. 5

3.3 Expression measures ...........................................

3.3.1 expresso ........................................ 7

3.3.2 MAS 5.0 ........................................ 9

3.3.3 Li and Wong’s MBEI (dchip) ................................ 9

3.3.4 C implementation of RMA . ................................ 10

4 Quality Control through Data Exploration 11

4.1 Accessing PM and MM Data .................................. 12

4.2 Histograms, Images, and Boxplots .......................... 14

4.3 RNA degradation plots ...................................... 18

5 Normalization 20

6 Classes 20

6.1 AffyBatch ........................................ 20

6.2 ProbeSet ........................................ 21

7 Location to ProbeSet Mapping 22

8 Configuring the package options 27

9 Where can I get more information? 27
1 Introduction

The *affy* package is part of the BioConductor project. It is meant to be an extensible, interactive environment for data analysis and exploration of Affymetrix oligonucleotide array probe level data.

The software utilities provided with the Affymetrix software suite summarizes the probe set intensities to form one *expression measure* for each gene. The expression measure is the data available for analysis. However, as pointed out by Li and Wong (2001), much can be learned from studying the individual probe intensities, or as we call them, the *probe level data*. This is why we developed this package. The package includes plotting functions for the probe level data useful for quality control, RNA degradation assessments, different probe level normalization and background correction procedures, and flexible functions that permit the user to convert probe level data to expression measures. The package includes utilities for computing expression measures similar to MAS 4.0’s AvDiff (Affymetrix 1999), MAS 5.0’s signal (Affymetrix 2001), DChip’s MBEI (Li and Wong 2001), and RMA (Irizarry et al. 2003b).

We assume that the reader is already familiar with oligonucleotide arrays and with the design of the Affymetrix GeneChip arrays. If you are not, we recommend the Appendix of the Affymetrix MAS manual Affymetrix (1999, 2001).

The following terms are used throughout this document:

- **probe** oligonucleotides of 25 base pair length used to probe RNA targets.
- **perfect match** probes intended to match perfectly the target sequence.
- **PM** intensity value read from the perfect matches.
- **mismatch** the probes having one base mismatch with the target sequence intended to account for non-specific binding.
- **MM** intensity value read from the mis-matches.
- **probe pair** a unit composed of a perfect match and its mismatch.
- **affyID** an identification for a probe set (which can be a gene or a fraction of a gene) represented on the array.
- **probe pair set** PMs and MM related to a common affyID.

CEL files contain measured intensities and locations for an array that has been hybridized.

CDF file contain the information relating probe pair sets to locations on the array.

Section 2 describes the main differences between version 1.5 and this version (1.6). Section 3 describes a quick way of getting started and getting expression measures. Section 4 describes some quality control tools. Section 5 describes normalization routines. Section 6 describes the different classes in the package. Section 7 describes our strategy to map probe locations to probe set membership. Section 8 describes how to change the package’s default options. Section ?? describes earlier changes.

Note: If you use this package please cite Gautier et al. (2003) and/or Irizarry et al. (2003a).

2 Changes for affy in BioC 1.8 release

There were relatively few changes.

- MAplot now accepts the argument plot.method which can be used to call smoothScatter.
- normalize.quantiles.robust has had minor changes.
- ReadAffy can optionally return the SD values stored in the cel file.
- The C parsing code has been moved to the affyio package, which is now a dependency of the affy package. This change should be transparent to users as affyio will be automatically loaded when affy is loaded.
- Added a cdfname argument to justRMA and ReadAffy to allow for the use of alternative cdf packages.

3 Getting Started: From probe level data to expression values

The first thing you need to do is load the package.

R> library(affy) ##load the affy package

This release of the affy package will automatically download the appropriate cdf environment when you require it. However, if you wish you may download and install the cdf environment you need from http://bioconductor.org/help/bioc-views/release/data/annotation/ manually. If there is no cdf environment currently built for your particular chip and you have access to the CDF file then you may use the makecdfenv package to create one yourself. To make the cdf packaes, Microsoft Windows users will need to use the tools described in http://www.murdoch-sutherlan.com/Rtools/.
3.1 Quick start

If all you want is to go from probe level data (CEL files) to expression measures here are some quick ways.

If you want is RMA, the quickest way of reading in data and getting expression measures is the following:

1. Create a directory, move all the relevant CEL files to that directory

2. If using linux/unix, start R in that directory.

3. If using the Rgui for Microsoft Windows make sure your working directory contains the CEL files (use “File -> Change Dir” menu item).

4. Load the library.

   R> library(affy) ##load the affy package

5. Read in the data and create an expression, using RMA for example.

   R> Data <- ReadAffy() ##read data in working directory
   R> eset <- rma(Data)

   Depending on the size of your dataset and on the memory available to your system, you might experience errors like ‘Cannot allocate vector ...’. An obvious option is to increase the memory available to your R process (by adding memory and/or closing external applications). An another option is to use the function justRMA.

   R> eset <- justRMA()

This reads the data and performs the ‘RMA’ way to preprocess them at the C level. One does not need to call ReadAffy, probe level data is never stored in an AffyBatch. rma continues to be the recommended function for computing RMA.

The rma function was written in C for speed and efficiency. It uses the expression measure described in [Irizarry et al. (2003b)].

For other popular methods use expresso instead of rma (see Section 3.3.1). For example for our version of MAS 5.0 signal uses expresso (see code). To get mas 5.0 you can use

   R> eset <- mas5(Data)

which will also normalize the expression values. The normalization can be turned off through the normalize argument.

---

2 UNIX-like systems users might also want to check ulimit and/or compile R and the package for 64 bits when possible.
In all the above examples, the variable eset is an object of class ExpressionSet described in the Biobase vignette. Many of the packages in BioConductor work on objects of this class. See the genefilter and geneplotter packages for some examples.

If you want to use some other analysis package, you can write out the expression values to file using the following command:

\[ \text{R} \text{> write.exprs(eset, file="mydata.txt")} \]

### 3.2 Reading CEL file information

The function ReadAffy is quite flexible. It lets you specify the filenames, phenotype, and MIAME information. You can enter them by reading files (see the help file) or widgets (you need to have the tkWidgets package installed and working).

\[ \text{R} \text{> Data <- ReadAffy(widget=TRUE) \#read data in working directory} \]

This function call will pop-up a file browser widget, see Figure 1 that provides an easy way of choosing cel files.
Figure 1: Graphical display for selecting CEL files. This widget is part of the tkWidgets package. (function written by Jianhua (John) Zhang).

Next, a widget (not shown) permits the user to enter the phenoData. Finally the a widget is presented for the user to enter MIAME information.

Notice that it is not necessary to use widgets to enter this information. Please read the help file for more information on how to read it from flat files or to enter it programmatically.

The function ReadAffy is a wrapper for the functions read.affybatch, tkSampleNames, read.AnnotatedDataFrame, and read.MIAME. The function read.affybatch has some nice feature that make it quite flexible. For example, the compression argument permit the user to read compressed CEL files. The argument compress set to TRUE will inform the readers that your files are compressed and let you read them while they remain compressed. The compression formats zip and gzip are known to be recognized.

A comprehensive description of all these options is found in the help file:

R> ?read.affybatch
R> ?read.AnnotatedDataFrame
R> ?read.MIAME
3.3 Expression measures

The most common operation is certainly to convert probe level data to expression values. Typically this is achieved through the following sequence:

1. reading in probe level data.
2. background correction.
3. normalization.
4. probe specific background correction, e.g. subtracting MM.
5. summarizing the probe set values into one expression measure and, in some cases, a standard error for this summary.

We detail what we believe is a good way to proceed below. As mentioned the function `expresso` provides many options. For example,

```
R> eset <- expresso(Dilution, normalize.method="qspline",
                   bgcorrect.method="rma", pmcorrect.method="pmonly",
                   summary.method="liwong")
```

This will store expression values, in the object `eset`, as an object of class `ExpressionSet` (see the `Biobase` package). You can either use R and the BioConductor packages to analyze your expression data or if you rather use another package you can write it out to a tab delimited file like this

```
R> write.exprs(eset, file="mydata.txt")
```

In the `mydata.txt` file, row will represent genes and columns will represent samples/arrays. The first row will be a header describing the columns. The first column will have the `affyIDs`. The `write.exprs` function is quite flexible on what it writes (see the help file).

3.3.1 expresso

The function `expresso` performs the steps background correction, normalization, probe specific correction, and summary value computation. We now show this using an `AffyBatch` included in the package for examples. The command `data(Dilution)` is used to load these data.

Important parameters for the `expresso` function are:

`bgcorrect.method` . The background correction method to use. The available methods are

```
> bgcorrect.methods()
```
The normalization method to use. The available methods can be queried by using \texttt{normalize.methods}.

\begin{verbatim}
> library(affydata)
>
> Package    LibPath           Item
> [1,] "affydata"  "/home/biocbuild/bbs-3.18-bioc/R/site-library" "Dilution"
> Title
> [1,] "AffyBatch instance Dilution"

> data(Dilution) ## data included in the package for examples
> normalize.methods(Dilution)

[1] "constant"   "contrasts"  "invariantset"  "loess"
[5] "methods"    "qspline"    "quantiles"   "quantiles.robust"
\end{verbatim}

The method for probe specific correction. The available methods are

\begin{verbatim}
> pmcorrect.methods()

[1] "mas"     "methods"    "pmonly"    "subtractmm"
\end{verbatim}

The summary method to use. The available methods are

\begin{verbatim}
> express.summary.stat.methods()

[1] "avgdiff"  "liwong"    "mas"     "medianpolish"  "playerout"
\end{verbatim}

Here we use \texttt{mas} to refer to the methods described in the Affymetrix manual version 5.0.

Making the \texttt{widget} argument \texttt{TRUE}, will let you select missing parameters (like the normalization method, the background correction method or the summary method). Figure 2 shows the widget for the selection of preprocessing methods for each of the steps.

\begin{verbatim}
R> expresso(Dilution, widget=TRUE)
\end{verbatim}

There is a separate vignette \texttt{affy: Built-in Processing Methods} which explains in more detail what each of the preprocessing options does.
3.3.2 MAS 5.0

To obtain expression values that correspond to those from MAS 5.0, use mas5, which wraps expresso and affy.scalevalue.exprSet.

```r
> eset <- mas5(Dilution)
```

background correction: mas
PM/MM correction: mas
expression values: mas
background correcting...done.
12625 ids to be processed
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
This gives the current \( PM \)-only default. The reduced model (previous default) can be obtained using \texttt{pmcorrect.method="subtractmm"}.

### 3.3.4 C implementation of RMA

One of the quickest ways to compute expression using the \texttt{affy} package is to use the \texttt{rma} function. We have found that this method allows a user to compute the RMA expression measure in a matter of minutes for datasets that may have taken hours in previous versions of \texttt{affy}. The function serves as an interface to a hard coded C implementation of the RMA method \cite{irizarry03b}. Generally, the following would be sufficient to compute RMA expression measures:

```r
> eset <- rma(Dilution)
```

**Background correcting**

**Normalizing**

**Calculating Expression**

Currently the \texttt{rma} function implements RMA in the following manner

1. Probe specific correction of the PM probes using a model based on observed intensity being the sum of signal and noise

2. Normalization of corrected PM probes using quantile normalization \cite{bolstad03}

3. Calculation of Expression measure using median polish.

The \texttt{rma} function is likely to be improved and extended in the future as the RMA method is fine-tuned.
4 Quality Control through Data Exploration

For the users convenience we have included the Dilution sample data set:

```r
> Dilution

AffyBatch object
size of arrays=640x640 features (38422 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=4
number of genes=12625
annotation=hgu95av2
notes=
```

This will create the Dilution object of class AffyBatch. print (or show) will display summary information. These objects represent data from one experiment. The AffyBatch class combines the information of various CEL files with a common CDF file. This class is designed to keep information of one experiment. The probe level data is contained in this object.

The data in Dilution is a small sample of probe sets from 2 sets of duplicate arrays hybridized with different concentrations of the same RNA. This information is part of the AffyBatch and can be accessed with the phenoData and pData methods:

```r
> phenoData(Dilution)

An object of class 'AnnotatedDataFrame'
sampleNames: 20A 20B 10A 10B
varLabels: liver sn19 scanner
varMetadata: labelDescription
```

```r
> pData(Dilution)

liver sn19 scanner
20A 20 0 1
20B 20 0 2
10A 10 0 1
10B 10 0 2
```

Several of the functions for plotting summarized probe level data are useful for diagnosing problems with the data. The plotting functions boxplot and hist have methods for AffyBatch objects. Each of these functions presents side-by-side graphical summaries of intensity information from each array. Important differences in the distribution of intensities are often evident in these plots. The function MAplot (applied, for example, to pm(Dilution)), offers pairwise graphical comparison of intensity data. The option
pairs permits you to choose between all pairwise comparisons (when TRUE) or compared to a reference array (the default). These plots can be particularly useful in diagnosing problems in replicate sets of arrays. The function argument plot.method can be used to create a MA plot using a smoothScatter, rather than the default method which is to draw every point.

```r
> data(Dilution)
> MAplot(Dilution, pairs=TRUE, plot.method="smoothScatter")
```

![MVA plot](image)

**Figure 3: Pairwise MA plots**

### 4.1 Accessing PM and MM Data

The PM and MM intensities and corresponding affyID can be accessed with the pm, mm, and probeNames methods. These will be matrices with rows representing probe pairs
and columns representing arrays. The gene name associated with the probe pair in row \( i \) can be found in the \( i \)th entry of the vector returned by `probeNames`.

```r
> Index <- c(1,2,3,100,1000,2000)  ##6 arbitrary probe positions
> pm(Dilution)[Index,]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20A</td>
<td>20B</td>
<td>10A</td>
<td>10B</td>
<td></td>
</tr>
<tr>
<td>358160</td>
<td>468.8</td>
<td>282.3</td>
<td>433.0</td>
<td>198.0</td>
</tr>
<tr>
<td>118945</td>
<td>430.0</td>
<td>265.0</td>
<td>308.5</td>
<td>192.8</td>
</tr>
<tr>
<td>323731</td>
<td>182.3</td>
<td>115.0</td>
<td>138.0</td>
<td>86.3</td>
</tr>
<tr>
<td>281340</td>
<td>264.0</td>
<td>151.0</td>
<td>167.0</td>
<td>103.3</td>
</tr>
<tr>
<td>361988</td>
<td>152.3</td>
<td>113.0</td>
<td>135.0</td>
<td>88.8</td>
</tr>
<tr>
<td>310732</td>
<td>275.0</td>
<td>155.5</td>
<td>194.3</td>
<td>124.5</td>
</tr>
</tbody>
</table>
```

```r
> mm(Dilution)[Index,]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20A</td>
<td>20B</td>
<td>10A</td>
<td>10B</td>
<td></td>
</tr>
<tr>
<td>358800</td>
<td>1123.5</td>
<td>673.0</td>
<td>693.5</td>
<td>434.5</td>
</tr>
<tr>
<td>119585</td>
<td>259.0</td>
<td>175.3</td>
<td>194.0</td>
<td>110.3</td>
</tr>
<tr>
<td>324371</td>
<td>160.0</td>
<td>95.0</td>
<td>119.3</td>
<td>72.5</td>
</tr>
<tr>
<td>281980</td>
<td>180.3</td>
<td>102.5</td>
<td>109.0</td>
<td>74.0</td>
</tr>
<tr>
<td>362628</td>
<td>178.8</td>
<td>126.8</td>
<td>156.3</td>
<td>83.5</td>
</tr>
<tr>
<td>311372</td>
<td>478.0</td>
<td>284.0</td>
<td>305.0</td>
<td>212.3</td>
</tr>
</tbody>
</table>
```

```r
> probeNames(Dilution)[Index]

[1] "1000_at" "1000_at" "1000_at" "1006_at" "1057_at" "1114_at"
```

`Index` contains six arbitrary probe positions.

Notice that the column names of `PM` and `MM` matrices are the sample names and the row names are the `affyID`, e.g. `1001_at` and `1000_at` together with the probe number (related to position in the target sequence).

```r
> sampleNames(Dilution)

[1] "20A" "20B" "10A" "10B"
```

**Quick example:** To see what percentage of the `MM` are larger than the `PM` simply type

```r
> mean(mm(Dilution)>pm(Dilution))

[1] 0.2746048
```

The `pm` and `mm` functions can be used to extract specific probe set intensities.
> gn <- geneNames(Dilution)
> pm(Dilution, gn[100])

<table>
<thead>
<tr>
<th>Gene</th>
<th>20A</th>
<th>20B</th>
<th>10A</th>
<th>10B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1090_f_at1</td>
<td>115.0</td>
<td>74.0</td>
<td>94.0</td>
<td>61.0</td>
</tr>
<tr>
<td>1090_f_at2</td>
<td>129.3</td>
<td>80.3</td>
<td>108.0</td>
<td>70.3</td>
</tr>
<tr>
<td>1090_f_at3</td>
<td>152.3</td>
<td>81.0</td>
<td>97.5</td>
<td>67.5</td>
</tr>
<tr>
<td>1090_f_at4</td>
<td>105.3</td>
<td>76.3</td>
<td>100.3</td>
<td>62.3</td>
</tr>
<tr>
<td>1090_f_at5</td>
<td>153.0</td>
<td>111.5</td>
<td>118.0</td>
<td>76.3</td>
</tr>
<tr>
<td>1090_f_at6</td>
<td>1984.3</td>
<td>1207.0</td>
<td>1331.3</td>
<td>728.0</td>
</tr>
<tr>
<td>1090_f_at7</td>
<td>290.5</td>
<td>181.0</td>
<td>220.0</td>
<td>134.3</td>
</tr>
<tr>
<td>1090_f_at8</td>
<td>882.3</td>
<td>532.0</td>
<td>525.0</td>
<td>347.0</td>
</tr>
<tr>
<td>1090_f_at9</td>
<td>157.3</td>
<td>105.3</td>
<td>125.0</td>
<td>69.3</td>
</tr>
<tr>
<td>1090_f_at10</td>
<td>103.5</td>
<td>83.0</td>
<td>90.0</td>
<td>67.0</td>
</tr>
<tr>
<td>1090_f_at11</td>
<td>100.0</td>
<td>75.5</td>
<td>89.0</td>
<td>71.0</td>
</tr>
<tr>
<td>1090_f_at12</td>
<td>143.5</td>
<td>92.8</td>
<td>104.3</td>
<td>78.0</td>
</tr>
<tr>
<td>1090_f_at13</td>
<td>111.0</td>
<td>70.8</td>
<td>97.0</td>
<td>60.5</td>
</tr>
<tr>
<td>1090_f_at14</td>
<td>381.5</td>
<td>255.0</td>
<td>289.0</td>
<td>198.0</td>
</tr>
<tr>
<td>1090_f_at15</td>
<td>650.0</td>
<td>389.3</td>
<td>415.0</td>
<td>275.0</td>
</tr>
<tr>
<td>1090_f_at16</td>
<td>262.0</td>
<td>157.0</td>
<td>194.8</td>
<td>131.5</td>
</tr>
</tbody>
</table>

The method `geneNames` extracts the unique `affyIDs`. Also notice that the 100th probe set is different from the 100th probe! The 100th probe is not part of the the 100th probe set.

The methods `boxplot`, `hist`, and `image` are useful for quality control. Figure 4 shows kernel density estimates (rather than histograms) of PM intensities for the 1st and 2nd array of the `Dilution` also included in the package.

### 4.2 Histograms, Images, and Boxplots

As seen in the previous example, the sub-setting method `[` can be used to extract specific arrays. **NOTE:** Sub-setting is different in this version. One can no longer subset by gene. We can only define subsets by one dimension: the columns, i.e. the arrays. Because the `Cel` class is no longer available `[[` is no longer available.

The method `image()` can be used to detect spatial artifacts. By default we look at log transformed intensities. This can be changed through the `transfo` argument.

> `par(mfrow=c(2,2))`
> `image(Dilution)`

These images are quite useful for quality control. We recommend examining these images as a first step in data exploration.

The method `boxplot` can be used to show `PM`, `MM` or both intensities. As discussed in the next section this plot shows that we need to normalize these arrays.
Figure 4: Histogram of $PM$ intensities for 1st and 2nd array
Figure 5: Image of the log intensities.
> par(mfrow=c(1,1))
> boxplot(Dilution, col=c(2,3,4))

Small part of dilution study

![Boxplot of arrays in Dilution data.](image)

Figure 6: Boxplot of arrays in Dilution data.
4.3 RNA degradation plots

The functions AffyRNAdeg, summaryAffyRNAdeg, and plotAffyRNAdeg aid in assessment of RNA quality. Individual probes in a probeset are ordered by location relative to the 5' end of the targeted RNA molecule [1999]. Since RNA degradation typically starts from the 5' end of the molecule, we would expect probe intensities to be systematically lowered at that end of a probeset when compared to the 3' end. On each chip, probe intensities are averaged by location in probeset, with the average taken over probesets. The function plotAffyRNAdeg produces a side-by-side plots of these means, making it easy to notice any 5' to 3' trend. The function summaryAffyRNAdeg produces a single summary statistic for each array in the batch, offering a convenient measure of the severity of degradation and significance level. For an example

```r
> deg <- AffyRNAdeg(Dilution)
> names(deg)

[1] "N"               "sample.names" "means.by.number" "ses"
[5] "slope"           "pvalue"
```

does the degradation analysis and returns a list with various components. A summary can be obtained using

```r
> summaryAffyRNAdeg(deg)

            20A  20B  10A  10B
slope  -0.0239 0.0363 0.0273 0.0849
pvalue  0.8920 0.8400 0.8750 0.6160
```

Finally a plot can be created using plotAffyRNAdeg, see Figure 7.
> plotAffyRNAdeg(deg)

RNA degradation plot

Figure 7: Side-by-side plot produced by plotAffyRNAdeg.
5 Normalization

Various researchers have pointed out the need for normalization of Affymetrix arrays. See for example Bolstad et al. (2003). The method `normalize` lets one normalize at the probe level

```r
> Dilution.normalized <- normalize(Dilution)
```

For an extended example on normalization please refer to the vignette in the affydata package.

6 Classes

`AffyBatch` is the main class in this package. There are three other auxiliary classes that we also describe in this Section.

6.1 AffyBatch

The AffyBatch class has slots to keep all the probe level information for a batch of .CEL files, which usually represent an experiment. It also stores phenotypic and MIAME information as does the `ExpressionSet` class in the Biobase package (the base package for BioConductor). In fact, `AffyBatch` extends `ExpressionSet`.

The expression matrix in `AffyBatch` has columns representing the intensities read from the different arrays. The rows represent the cel intensities for all position on the array. The cel intensity with physical coordinates\(^3\)(x, y) will be in row

\[
i = x + \text{nrow} \times (y - 1)
\]

The `ncol` and `nrow` slots contain the physical rows of the array. Notice that this is different from the dimensions of the expression matrix. The number of row of the expression matrix is equal to `ncol x nrow`. We advice the use of the functions `xy2indices` and `indices2xy` to shuttle from X/Y coordinates to indices.

For compatibility with previous versions the accessor method `intensity` exists for obtaining the expression matrix.

The `cdfName` slot contains the necessary information for the package to find the locations of the probes for each probe set. See Section 7 for more on this.

\(^3\)Note that in the .CEL files the indexing starts at zero while it starts at 1 in the package (as indexing starts at 1 in R).
6.2 ProbeSet

The ProbeSet class holds the information of all the probes related to an affyID. The components are pm and mm.

The method probeset extracts probe sets from AffyBatch objects. It takes as arguments an AffyBatch object and a vector of affyIDs and returns a list of objects of class ProbeSet.

```r
> gn <- featureNames(Dilution)
> ps <- probeset(Dilution, gn[1:2])
> #this is what i should be using: ps
> show(ps[[1]])

ProbeSet object:
  id=1000_at
  pm= 16 probes x 4 chips
```

The pm and mm methods can be used to extract these matrices (see below).

This function is general in the way it defines a probe set. The default is to use the definition of a probe set given by Affymetrix in the CDF file. However, the user can define arbitrary probe sets. The argument locations lets the user decide the row numbers in the intensity that define a probe set. For example, if we are interested in redefining the AB000114_at and AB000115_at probe sets, we could do the following:

First, define the locations of the PM and MM on the array of the 1000_at and 1001_at probe sets.

```r
> mylocation <- list("1000_at"=cbind(pm=c(1,2,3),mm=c(4,5,6)),
+  "1001_at"=cbind(pm=c(4,5,6),mm=c(1,2,3)))
```

The first column of the matrix defines the location of the PMs and the second column the MMs.

Now we are ready to extract the ProbeSets using the probeset function.

```r
> ps <- probeset(Dilution, genenames=c("1000_at","1001_at"),
+ locations=mylocation)
```

Now, ps is list of ProbeSets. We can see the PMs and MMs of each component using the pm and mm accessor methods.

```r
> pm(ps[[1]])
```

```
 20A 20B 10A 10B
1 149.0 112.0 129.0 60.0
2 1153.5 575.3 1262.3 564.8
3 142.0 98.0 128.0 56.0
```
This can be useful in situations where the user wants to determine if leaving out certain probes improves performance at the expression level. It can also be useful to combine probes from different human chips, for example by considering only probes common to both arrays.

Users can also define their own environment for probe set location mapping. More on this in Section 7.

An example of a ProbeSet is included in the package. A spike-in data set is included in the package in the form of a list of ProbeSets. The help file describes the data set. Figure 8 uses this data set to demonstrate that the MM also detect transcript signal.

7 Location to ProbeSet Mapping

On Affymetrix GeneChip arrays, several probes are used to represent genes in the form of probe sets. From a CEL file we get for each physical location, or cel, (defined by x and y coordinates) an intensity. The CEL file also contains the name of the CDF file needed for the location-probe-set mapping. The CDF files store the probe set related to each location on the array. The computation of a summary expression values from the probe intensities requires a fast way to map an affyId to corresponding probes. We store this mapping information in R environments. They only contain a part of the information that can be found in the CDF files. The cdfenvs are sufficient to perform

---

4Please refer to the R documentation to know more about environments.
> data(SpikeIn)  ##SpikeIn is a ProbeSets
> pms <- pm(SpikeIn)
> mms <- mm(SpikeIn)
> ##pms follow concentration
> par(mfrow=c(1,2))
> concentrations <- matrix(as.numeric(sampleNames(SpikeIn)),20,12,byrow=TRUE)
> matplot(concentrations,pms,log="xy",main="PM",ylim=c(30,20000))
> lines(concentrations[1,],apply(pms,2,mean),lwd=3)
> ##so do mms
> matplot(concentrations,mms,log="xy",main="MM",ylim=c(30,20000))
> lines(concentrations[1,],apply(mms,2,mean),lwd=3)

Figure 8: PM and MM intensities plotted against SpikeIn concentration
the numerical processing methods included in the package. For each CDF file there is a package, available from [http://bioconductor.org/help/bioc-views/release/data/annotation/](http://bioconductor.org/help/bioc-views/release/data/annotation/) that contains exactly one of these environments. The 

In instances of AffyBatch, the cdfName slot gives the name of the appropriate CDF file for arrays represented in the intensity slot. The functions read.celfile, read.affybatch, and ReadAffy extract the CDF filename from the CEL files being read. Each CDF file corresponds to exactly one environment. The function cleancdfname converts the Affymetrix given CDF name to a BioConductor environment and annotation name. Here are two examples:

These give environment names:

```r
> cat("HG_U95Av2 is",cleancdfname("HG_U95Av2"),"\n")
HG_U95Av2 is hgu95av2cdf

> cat("HG-133A is",cleancdfname("HG-133A"),"\n")
HG-133A is hg133acdf
```

This gives annotation name:

```r
> cat("HG_U95Av2 is",cleancdfname("HG_U95Av2",addcdf=FALSE),"\n")
HG_U95Av2 is hgu95av2
```

An environment representing the corner of an Hu6800 array is available with the package. In the following, we load the environment, look at the names for the first 5 objects defined in the environment, and finally look at the first object in the environment:

```r
> data(cdfenv.example)
> ls(cdfenv.example)[1:5]
[1] "A28102_at" "AB000114_at" "AB000115_at" "AB000220_at" "AB002314_at"
> get(ls(cdfenv.example)[1],cdfenv.example)
   mm
[1,] 102 203
[2,] 104 205
[3,] 106 207
[4,] 108 209
[5,] 110 211
[6,] 112 213
[7,] 114 215
```
The package needs to know what locations correspond to which probe sets. The `cdfName` slot contains the necessary information to find the environment with this location information. The method `getCdfInfo` takes as an argument an `AffyBatch` and returns the necessary environment. If `x` is an `AffyBatch`, this function will look for an environment with name `cleancdfname(x@cdfName)`.

```r
> print(Dilution@cdfName)
[1] "HG_U95Av2"
> myenv <- getCdfInfo(Dilution)
> ls(myenv)[1:5]
[1] "1000_at"  "1001_at"  "1002_f_at"  "1003_s_at"  "1004_at"
```

By default we search for the environment first in the global environment, then in a package named `cleancdfname(x@cdfName)`.

Various methods exist to obtain locations of probes as demonstrated in the following examples:

```r
> Index <- pmindex(Dilution)
> names(Index)[1:2]
[1] "1000_at"  "1001_at"
> Index[1:2]

$1000_at$
[1] 358160 118945 323731 223978 313420 349209 199525 213669 236739 298099

$1001_at$
[1] 340142 236569 327449 203508 300798 276193 354374 400320 250783 379851
```
pmindex returns a list with probe set names as names and locations in the components. We can also get specific probe sets:

```r
> pmindex(Dilution, genenames=c("1000_at","1001_at"))

$ `1000_at`
  [1]  358160 118945 323731 223978 313420 349209 199525 213669 236739 298099

$ `1001_at`
  [1]  340142 236569 327449 203508 300798 276193 354374 400320 250783 379851

The locations are ordered from 5' to 3' on the target transcript. The function mmindex performs in a similar way:

```r
> mmindex(Dilution, genenames=c("1000_at","1001_at"))

$ `1000_at`
  [1]  358800 119585 324371 224618 314060 349849 200165 214309 237379 298739

$ `1001_at`
  [1]  340782 237209 328089 204148 301438 276833 355014 400960 251423 380491

They both use the method indexProbes

```r
> indexProbes(Dilution, which="pm")[1]

$ `1000_at`
  [1]  358160 118945 323731 223978 313420 349209 199525 213669 236739 298099

> indexProbes(Dilution, which="mm")[1]

$ `1000_at`
  [1]  358800 119585 324371 224618 314060 349849 200165 214309 237379 298739

> indexProbes(Dilution, which="both")[1]

$ `1000_at`
  [1]  358160 118945 323731 223978 313420 349209 199525 213669 236739 298099
  [21] 314060 349849 200165 214309 237379 298739 283384 282083 349838 298593
  [31]  317694 404709

The which="both" options returns the location of the PMs followed by the MMs.
8 Configuring the package options

Package-wide options can be configured, as shown below through examples.

- Getting the names for the options:

  ```
  > opt <- getOption("BioC")
  > affy.opt <- opt$affy
  > print(names(affy.opt))
  [1] "compress.cdf" "compress.cel" "use.widgets" "probesloc"
  [9] "xy.offset"
  ```

- Default processing methods:

  ```
  > opt <- getOption("BioC")
  > affy.opt <- opt$affy
  > affy.opt$normalize.method <- "constant"
  > opt$affy <- affy.opt
  > options(BioC=opt)
  ```

- Compression of files: if you are always compressing your CEL files, you might find annoying to specify it each time you call a reading function. It can be specified once for all in the options.

  ```
  > opt <- getOption("BioC")
  > affy.opt <- opt$affy
  > affy.opt$compress.cel <- TRUE
  > opt$affy <- affy.opt
  > options(BioC=opt)
  ```

- Priority rule for the use of a cdf environment: The option `probesloc` is a list. Each element of the list is itself a list with two elements `what` and `where`. When looking for the information related to the locations of the probes on the array, the elements in the list will be looked at sequentially. The first one leading to the information is used (an error message is returned if none permits to find the information). The element `what` can be one of `package, environment`.

9 Where can I get more information?

There are several other vignettes addressing more specialised topics related to the `affy` package.
- **affy: Custom Processing Methods (HowTo):** A description of how to use custom preprocessing methods with the package. This document gives examples of how you might write your own preprocessing method and use it with the package.

- **affy: Built-in Processing Methods:** A document giving fuller descriptions of each of the preprocessing methods that are available within the affy package.

- **affy: Import Methods (HowTo):** A discussion of the data structures used and how you might import non standard data into the package.

- **affy: Loading Affymetrix Data (HowTo):** A quick guide to loading Affymetrix data into R.

- **affy: Automatic downloading of cdfenvs (HowTo):** How you can configure the automatic downloading of the appropriate cdfenv for your analysis.

## A Previous Release Notes

**A.1 Changes in versions 1.6.x**

There were very few changes.

- The function `MAplot` has been added. It works on instances of AffyBatch. You can decide if you want to make all pairwise MA plots or compare to a reference array using the `pairs` argument.

- Minor bugs fixed in the parsers.

- The path of celfiles is now removed by `ReadAffy`.

**A.2 Changes in versions 1.5.x**

There are some minor differences in what you can do but little functionality has disappeared. Memory efficiency and speed have improved.

- The widgets used by `ReadAffy` have changed.

- The path of celfiles is now removed by `ReadAffy`.

**A.3 Changes in versions 1.4.x**

There are some minor differences in what you can do but little functionality has disappeared. Memory efficiency and speed have improved.
• For instances of AffyBatch the subsetting has changed. For consistency with exprSets one can only subset by the second dimension. So to obtain the first array, abatch[1] and abatch[1,] will give warnings (errors in the next release). The correct code is abatch[,1].

• mas5calls is now faster and reproduces Affymetrix’s official version much better.

• If you use pm and mm to get the entire set of probes, e.g. by typing pm(abatch) then the method will be, on average, about 2-3 times faster than in version 1.3.

References


