1 Introduction

Codelink™ is a platform for the analysis of gene expression on biological samples using short (30 base long) oligonucleotide probes. There is a proprietary software for reading scanned images, spot intensity quantization and some diagnostics. Quality flags (Table 1) are assigned to the spot based on signal to noise ratio (SNR) computation (Eq: 1) and other morphological characteristics as irregular shape of the spots, saturation of the signal or manufacturer spots removed. The results can be exported in many formats as XML, Excel, plain text, etc. This library allows to read Codelink plain text exported data into R [3] for analysis of gene expression with any of the available tools in R+Bioconductor[1]. A new class is defined for convenient storing Codelink data as exprSet class is not convenient for this purpose.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Good signal (SNR $\geq 1$)</td>
</tr>
<tr>
<td>L</td>
<td>Limit signal (SNR $&lt; 1$)</td>
</tr>
<tr>
<td>I</td>
<td>Irregular shape</td>
</tr>
<tr>
<td>S</td>
<td>Saturated signal</td>
</tr>
<tr>
<td>M</td>
<td>MSR spot</td>
</tr>
<tr>
<td>C</td>
<td>Background contaminated</td>
</tr>
<tr>
<td>X</td>
<td>User excluded spots</td>
</tr>
</tbody>
</table>

Table 1: Quality Flag description. SNR: Signal to Noise Ratio.

\[
SNR = \frac{S_{\text{mean}}}{(B_{\text{median}} + 1.5 \times B_{\text{stddev}})} \tag{1}
\]

2 Reading data

Currently only data exported as plain text from Codelink software can be used. The Codelink text format can have arbitrary columns and header fields so depending of what you have exported you can read it or not. The suggestion
<table>
<thead>
<tr>
<th>Probe type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCOVERY</td>
<td>Gene expression testing probes</td>
</tr>
<tr>
<td>POSITIVE</td>
<td>Positive control probes</td>
</tr>
<tr>
<td>NEGATIVE</td>
<td>Negative control probes</td>
</tr>
<tr>
<td>FIDUCIAL</td>
<td>Grid alignment probes</td>
</tr>
<tr>
<td>OTHER</td>
<td>Other controls and housekeeping gene probes</td>
</tr>
</tbody>
</table>

Table 2: Probe types for Codelink arrays.

is that you put on the files Spot_mean and Bkgd_median values so you can do background correction and normalization in R. If you put Raw_intensity or Normalized_intensity columns then you can also read it directly and avoid background correction and/or normalization but this is not recommended. To read some Codelink files you do:

```r
> library(codelink)
> data <- readCodelink()
```

This suppose that your files have the extension “TXT” (uppercase) and they are in your working directory. If this is not the case you can specify files to be read with the ‘file’ argument. The function `readCodelink` returns and object of Codelink:

```r
> library(codelink)
> data(codelink.example)
> codelink.example
```

An object of class "Codelink"

$product

[1] "UniSet Human 20K I"

$sample

[1] "Sample 1" "Sample 2"

$file

[1] "T001-2006-12-25_Sample 1.TXT" "T002-2006-12-25_Sample 2.TXT"

$name

[1] "NM_012429.1_PROBE1" "NM_003980.2_PROBE1" "AY044449_PROBE1"

[4] "NM_005015.1_PROBE1" "AB037823_PROBE1"

20464 more elements ...

$type
$flag
   1  2
  1 "L" "L"
  2 "L" "L"
  3 "G" "L"
  4 "G" "M"
  5 "G" "G"
20464 more rows ...

$method
$background
[1] "NONE"

$normalization
[1] "NONE"

$merge
[1] "NONE"

$log
[1] FALSE

$snr
   1     2
 1 0.7920656 0.7856675
 2 0.8160216 0.7862189
 3 1.0744066 0.9331916
 4 3.8865153   NA
 5 4.6647164 1.7993536
20464 more rows ...

$logical
   row col
  1  1   9
  2  1  10
  3  1  11
  4  1  12
  5  1  13
20464 more rows ...

$Smean
   1   2
 1   3

3
1  49.0588  46.7304
2  48.6395  45.8333
3  65.0781  52.4917
4  243.3116  NA
5  267.6458 102.0803
20464 more rows ...

$Bmedian
  1  2
  1  43  42
  2  42  42
  3  42  40
  4  44  NA
  5  42  42
20464 more rows ...

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>Chip name description</td>
</tr>
<tr>
<td>sample</td>
<td>Sample names vector</td>
</tr>
<tr>
<td>file</td>
<td>File names vector</td>
</tr>
<tr>
<td>name</td>
<td>Probe names vector</td>
</tr>
<tr>
<td>type</td>
<td>Probe types vector</td>
</tr>
<tr>
<td>method</td>
<td>Methods applied to data</td>
</tr>
<tr>
<td>method$background</td>
<td>Background correction method used</td>
</tr>
<tr>
<td>method$normalization</td>
<td>Normalization method used</td>
</tr>
<tr>
<td>method$merge</td>
<td>Merge method used</td>
</tr>
<tr>
<td>method$log</td>
<td>Logical: If data is in log scale</td>
</tr>
<tr>
<td>flag</td>
<td>Quality flag matrix</td>
</tr>
<tr>
<td>Smean</td>
<td>Mean signal intensity matrix</td>
</tr>
<tr>
<td>Bmedian</td>
<td>Median background intensity matrix</td>
</tr>
<tr>
<td>Ri</td>
<td>Raw intensity matrix</td>
</tr>
<tr>
<td>Ni</td>
<td>Normalized intensity matrix</td>
</tr>
<tr>
<td>snr</td>
<td>Signal to Noise Ratio matrix</td>
</tr>
<tr>
<td>cv</td>
<td>Coefficient of Variation matrix</td>
</tr>
</tbody>
</table>

Table 3: Description of Codelink object slots.

The chip type (product slot) is read from the PRODUCT field in the header of Codelink files. If it is not found then a warning message is shown and product slot is set to "Unknown". If one product type disagree with the others an error message is shown and reading of files is terminated.

By default, all spots flagged with M, I, and S flags are set to NA. This can be changed with the flag argument in `readCodelink`. The flag argument is a list that can contain a valid flag identifier and a value for that flag. For example,
if you want to set all M flagged spots to 0.01 and let other spot untouched you do:

```r
> data <- readCodelink(flag=list(M=0.01))
```

It is possible to find probes wit more that one flag assigned, i.e. CL for a probe labeled as C and L, CLI for a probe labeled as C, L and I, and so on. As a regular expression is used to find flag types it is possible to manage all this situations. When two user modified flags fall in the same probe the smallest (or NA if applicable) is assigned.

3 Background correction

Smean intensity values can be processed into Ri through background correction:

```r
> data <- bkgdCorrect(data, method="half")
```

The default method used is based on the half method from limma [4] package. Median background intensity (Bmedian) is subtracted from mean spot intensity (Smean) and if the result is less than 0.5 then it is set to 0.5 to ensure no negative numbers are obtained.

4 Normalization

Normalization of Ri values are done with the wrapper function normalize(). The default method is quantile normalization that in fact call normalizeQuantiles() from limma package (that allows for NAs). There is also the possibility to use a modified version of cyclic loess from affy [2] package that also allows NA values.

```r
> data <- normalize(data, method="quantile")
```

By default, normalize return log2 intensity values. This could be changed setting the parameter log.it to FALSE.

5 Plotting

There are some plotting functions available that can use directly the Codelink objects. These are functions for producing MA plots (plotMA), scatterplots
(plotCorrelation) and density plots (plotDensities). All functions use the available intensity value (i.e. Smean, R1 or N1) to make the plot.

The function plotMA can highlight spots based on type values (by default) or SNR values setting the argument label. It requires array1 and array2 arguments and compute M and A values based on equations 2 and 3.

> plotMA(codelink.example, legend.x = "topright")

\[
M = \text{Array2} - \text{Array1} \tag{2}
\]

\[
A = \frac{\text{Array2} + \text{Array1}}{2} \tag{3}
\]

The function plotDensities plot the density of intensity values of all arrays. If the subset argument is supplied it can use only a subset of the arrays in the Codelink object.

> plotDensities(codelink.example)
6 Miscellaneous

There are also some miscellaneous functions used in some analysis that could be useful for someone.

6.1 Using weights

The `createWeights` function creates a matrix of weights based on probe type labels to be used, for example, in fitting a linear model with `limma` [4].

```r
> w <- createWeights(codelink.example, type = list(FIDUCIAL = 0.01, NEGATIVE = 0.1))
> w[1:10, ]

[,1] [,2]
[1,] 1 1
[2,] 1 1
[3,] 1 1
[4,] 1 1
[5,] 1 1
[6,] 1 1
[7,] 1 1
```
6.2 Merging arrays

In case you want to merge array intensities the `mergeArray` function help on this task. It computes the mean of $N_i$ values on arrays of the same class. The grouping is done by means of the `class` argument (numerical vector of classes). New sample names should be assigned to sample slot with the `names` argument. The function also returns the coefficient of variation in the `cv` slot. The distribution of coefficients of variations can be checked with the function `plotCV`.

```r
> data <- mergeArray(data, group=c(1,1,2,2),
+                   names=c("A", "B"))

> plotCV(data)
```

References


