Codelink

Diego Diez

April 13, 2011

1 Introduction

Codelink is a platform for the analysis of gene expression on biological samples property of Applied Microarrays, Inc. (previously was GE Healthcare and Amersham). The hybridization reagents are still supplied by GE Healthcare.

The system uses 30 base long oligonucleotide probes for expression testing. There is a proprietary software for reading scanned images, doing spot intensity quantization and some diagnostics. The software assigns quality flags (see Table 1) to each spot on the basis of a 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. By default, the software performs background correction (subtract) followed by median normalization. The results can be exported in several formats as XML, Excel, plain text, etc. This library allows to read Codelink plain text exported data into R [3] for the analysis of gene expression with any of the available tools in R+Bioconductor[1].

For storing the available data, a new class `Codelink` was designed. The now defunt `exprsSet` in the *Biobase* package was not a convenient store class for Codelink data, because it didn't allow probes with duplicated names, as those names are stored as rownames in the expression matrix.

Currently there is experimental support for a `ExpressionSet` derived class that can accomodate duplicated probe names in an `AnnotatedDataFrame` whereas the feature ids would be used for row names when available.

Flag	*Description*
G	Good signal (SNR >= 1)
L	Limit signal (SNR < 1)
I	Irregular shape
S	Saturated signal
M	MSR spot
C	Background contaminated
X	User excluded spots

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

$$SNR = \frac{Smean}{(Bmedian + 1.5 * Bstdev)} \quad (1)$$

Probe type	*Description*
DISCOVERY	Gene expression testing probes
POSITIVE	Positive control probes
NEGATIVE	Negative control probes
FIDUCIAL	Grid alignment probes
OTHER	Other controls and housekeeping gene probes

Table 2: Probe types for Codelink arrays.

2 Reading data

Currently only data exported as plain text from Codelink software is supported. Unfortunately the Codelink exported text format can have arbitrary columns and header fields so depending of what you have exported you can read it or not. The suggestion is that you put on the files everything you can, including Spot_mean and Bkgd_median values so you can do background correction and normalization in R. In addition, Bkgd_stdev is needed to compute the SNR. 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:

```
# NOT RUN #

> library(codelink)
> foo <- readCodelink()
> summaryFlag(foo) # will show a summary of flag values.

# NOT RUN #
```

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

```
> 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
[1] "DISCOVERY" "DISCOVERY" "DISCOVERY" "DISCOVERY" "DISCOVERY"
20464 more elements ...

$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  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 ...
```

The `Codelink-class` is basically a list that stores information in several slots (see Table 3). The chip type (product slot) is read from the PRODUCT field if found 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 the product is not the same in all the files the reading is canceled with an error message.

By default, all spots flagged with M, I, and S flags are set to NA. This can be controlled with the flag argument of `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:

```
# NOT RUN #

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

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

Table 3: Description of Codelink object slots.

```
# NOT RUN #
```

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. A a regular expression is used to find flag types in an attempt to to manage all the possible situations. When two user modified flags fall in the same probe the more restricting (NA being the most) is assigned.

3 Background correction

If you have Spot_mean values Bkgd_median values the you can apply one of the several background correction methos interfaced. This is done by the function `bkgdCorrect`. To see the different options look at ?bkgdCorrect. For instance, if you want to apply *half* method you do:

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

The default method used is *half* and is based in the same method applied in the *limma* [4] package to two channel microarrays. In this method, the median background intensity (Bmedian) is subtracted from mean spot intensity (Smean) and any value smaller that 0.5 is shift to 0.5 to ensure no negative numbers are

obtained that would prevent to transform the data into log scale. Other available methods are *none* that let the spot intensities untouched, *subtract* that is analog to the default method used in the Codelink software and *normexp* and interface to the method available in the *limma* package.

4 Normalization

Normalization of the background corrected intensities is done by the wrapper function `normalize`. The default method is *quantile* normalization that in fact call `normalizeQuantiles()` from *limma* package (allowing for NAs). There is also the possibility to use a modified version of CyclicLoess from *affy* [2] package that allow using weights and missing values. Finally, the *median* normalization allows to normalize using a method analog to the default method in the Codelink software. To normalize you usually do:

```
> foo <- normalize(foo, method = "quantiles")
```

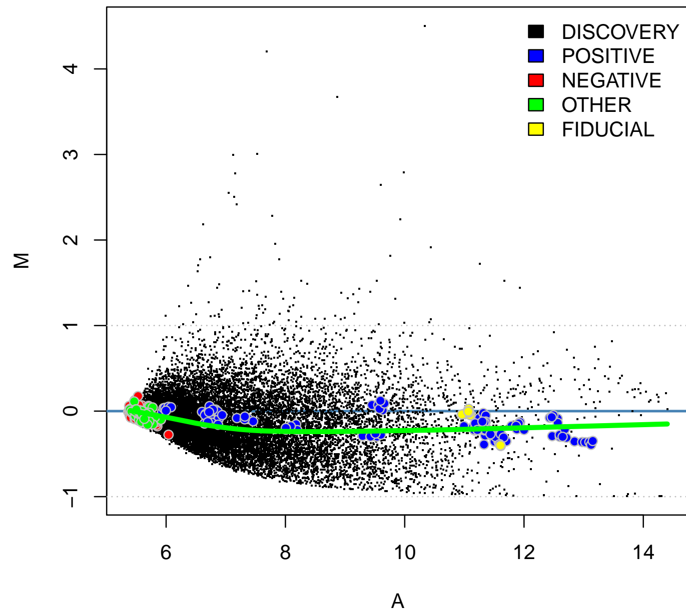
By default, `normalize` return log2 intensity values. This could be controlled setting the parameter log.it to FALSE.

5 Plotting

There are some diagnostic plots available for the `Codelink` object. These are functions for producing MA plots (`plotMA`), scatterplots (`plotCorrelation`) and density plots (`plotDensities`). All functions use the available intensity value (i.e. Smean, Ri or Ni) to make the plot.

The functions `plotMA` and `plotCorrelation` can highlight points based on the Spot Type, which is the default behavior or using the SNR values. The mode is controlled with argument *label*. `plotCorrelation` requires arguments *array1* and *array2* to be set in order to select which arrays are going to be plotted. For `plotMA` if only *array1* is specified, the values are plotted againts a pseudoarray constructed with the mean of the probe intensities along all available arrays. M and A values are computed following equations 2 and 3.

```
> plotMA(codelink.example)
```

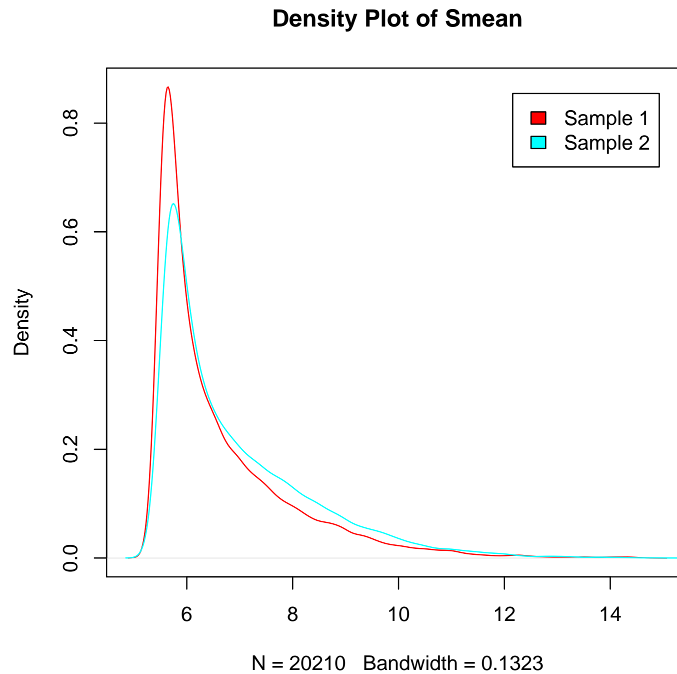


$$M = Array2 - Array1 \tag{2}$$

$$A = \frac{Array2 + Array1}{2} \tag{3}$$

The function `plotDensities` plot the density of intensity values of all arrays. It can plot only a subset of arrays if the *subset* argument is supplied.

```
> plotDensities(codelink.example)
```



When Logical_row and Logical_col columns are exported into they are stored into the *logical* slot. This information stores the physical location of each probe in the array, and can be used to plot pseudo images of the array intensities. To plot a pseudo image you should use:

```
> imageCodelink(foo)
> imageCodelink(foo, what = "snr")
```

It is possible to plot the background intensities (default), the spot mean, raw and normalized intensities and the SNR values. This images are useful to identify spatial artifact that may be affecting the analysis.

6 Miscellaneous

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

6.1 Export to file

The function `writeCodelink` exports a `Codelink` object to a file. The file will contain probe intensities and, if specified flag = TRUE, probe quality flags.

6.2 Using weights

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

```
> 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
 [8,]    1    1
 [9,]    1    1
[10,]    1    1
```

6.3 Merging arrays

In case you want to merge array intensities the `mergeArray` function help on this task. It computes the mean of Ni 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 the sample slot using 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`.

```
> foo <- mergeArray(foo, class = c(1, 1, 2, 2),
+ names = c("A", "B"))

> plotCV(foo)
```

7 Problems reading data

Some updated version of the Codelink software changed the order in which probes are printed in the exported text files. That makes files from these different software version impossible to be analyzed together. The function `readCodelink` have a new argument *check* TRUE by default, that check the Probe_name columns to see if they have the same order in all the arrays at the cost is a little extra loading time. This behaviour can be turn off by setting

check to FALSE. If an ordering problem is found, a warning message is print but the reading of data is not stopped, allowing the visual examination of the data. In this case, the best option is to export the old files again using the updated version of the software. If this is impossible for whatever reason, the codelink package can try to fix the order of the files.

To the aim, a new argument *fix = TRUE* can be passed to `readCodelink`. The method will try to order the probes using the Feature_id column, which is a combination of Logical_row and Logical_col and for this, a unique identifier of each probe. This is the optimal fix, and the new data should be perfectly ordered. If that column is missing, the it will try to order the data using the Probe_name column. This is a sub-optimal solution because as the fiducial, control and some discovery probes have duplicated Probe_name, they may be end messed up. In this case, the best solution again, is to try to export again the data with the updated version of the software.

```
> foo <- readCodelink(fix = TRUE)
```

8 Future improvements

As the new classes in the Biobase package (`eSet` and `ExpressionSet`) allow more flexible data structures, a reimplementation of the `Codelink-class` based on `ExpressionSet-class` will be the next major feature added to this package. This will allow a better integration with other tools available through the Bioconductor project.

Right now, there is some experimental implementation of the new codebase. You need to use the wrapper function `readCodelink2`.

```
> foo <- readCodelink2()
```

References

[1] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[2] Rafael A. Irizarry, Laurent Gautier, Benjamin Milo Bolstad, , Crispin Miller with contributions from Magnus Astrand <Magnus.Astrand@astrazeneca.com>, Leslie M. Cope, Robert Gentleman,

Jeff Gentry, Conrad Halling, Wolfgang Huber, James MacDonald, Benjamin I. P. Rubinstein, Christopher Workman, and John Zhang. *affy: Methods for Affymetrix Oligonucleotide Arrays*, 2005. R package version 1.8.1.

[3] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[4] Gordon K Smyth. *Limma: linear models for microarray data*, pages 397–420. Springer, New York, 2005.