Introduction to the Codelink package

Diego Diez

October 29, 2019

1 Introduction

This package implements methods to facilitate the preprocessing and analysis of Codelink microarrays. Codelink is a microarray platform for the analysis of gene expression that uses 30 base long oligonucleotides. Codelink is currently owned by Applied Microarrays, Inc. (previously was GE Healthcare and before that Amersham). There is a proprietary software for reading scanned images, perform spot intensity quantification and diagnostics. A Codelink microarray consists of a number of species-specific probes to measure gene expression, as well as some other control probes (see Table 1). The Codelink software assigns quality flags to each spot (see Table 2) 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.

The codelink package enables loading *Codelink* data into R, and stores it as a CodelinkSet object. The `CodelinkSet-class` inherits from `ExpressionSet` all methods, and enables straightforward interfacing with other Bioconductor structures, and packages.

NOTE: the old `Codelink-class` infrastructure is maintained for backbard compatibility, and information about its use can be found in the vignette Codelink_Legacy.pdf.

Probe type	*Description*	*Default weight*
DISCOVERY	Measure gene expression	1
POSITIVE	Positive control	0
NEGATIVE	Negative control	0
FIDUCIAL	Grid alignment	0
OTHER	Other controls and housekeeping genes	0

Table 1: Probe types for Codelink arrays.

Flag	*Description*	*Default value set*	*Default weight*
G	Good signal (SNR $\geq$ 1)		1
L	Limit signal (SNR $<$ 1)		1
S	Saturated signal		1
I	Irregular shape		0
M	MSR spot (-9999)	NA	0
C	Background contaminated		0
X	User excluded spots		0

Table 2: Quality Flag description. SNR: Signal to Noise Ratio.

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

2 Reading data

Only Codelink data exported as plain text from the Codelink software is supported. Unfortunately the Codelink exported text format can have arbitrary columns and header fields so depending of what has been exported, reading it into a `CodelinkSet` object may be more or less complicated. As a rule of thumb it is recommended to include in the exported files at least Spot_mean and Bkgd_median values so that background correction and normalization can be performed within R. In addition, Bkgd_stdev will be needed to compute the SNR. If Raw_intensity or Normalized_intensity columns are present then it is possible to avoid background correction and/or normalization, and use the ones performed by the Codelink software. The Feature_id column will be use to assign unique identifiers to each spot, so that CodelinkSet object can be read appropriately (or else will try to guess those). To read codelink data:

```
> # NOT RUN #
> library(codelink)
> # to read data as CodelinkSet object:
> f = list.files(pattern = "TXT")
> codset = readCodelinkSet(filename = f)
> # NOT RUN #
```

This assumes that the files have the extension "TXT" (uppercase) and are in the working directory. You can prepare a targets file with each file's name and additional phenotypic information, then pass this information to readCodelinkSet() so that it is stored in the CodelinkSet object.

```
> # NOT RUN #
> pdata = read.AnnotatedDataFrame("targets.txt")
> codset = readCodelinkSet(filename = pdata$FileName, phenoData = pdata)
> # NOT RUN #
```

```
> # sample dataset.
> data(codset)
> codset
CodelinkSet (storageMode: lockedEnvironment)
assayData: 35129 features, 4 samples
  element names: background, exprs, flag, snr, weight
protocolData: none
phenoData
  sampleNames: Sample-1 Sample-2 Sample-3 Sample-4
  varLabels: sample
  varMetadata: labelDescription
featureData
  featureNames: 1001 1002 ... 328112 (35129 total)
  fvarLabels: probeName probeType ... meanSNR (5 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: rwgcod
```

To convert old Codelink objects into the new CodelinkSet the handy function Codelink2CodelinkSet() can be used:

```
> data(codelink.example)
> print(is(codelink.example))
[1] "Codelink"         "list"             "vector"
[4] "AssayData"        "list_OR_List"     "vector_OR_factor"
[7] "vector_OR_Vector"
> tmp = Codelink2CodelinkSet(codelink.example)
> tmp
CodelinkSet (storageMode: lockedEnvironment)
assayData: 20469 features, 2 samples
  element names: background, exprs, flag, snr, weight
protocolData: none
phenoData
  rowNames: 1 2
  varLabels: sample
  varMetadata: labelDescription
featureData
  featureNames: 1 2 ... 20469 (20469 total)
  fvarLabels: probeName probeType ... meanSNR (5 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: h10kcod.db
```

2.1 Flags and weights

Traditionally the codelink package has used flag information to assign NAs to values. This behavior has been changed since the version released with Bioconductor 2.13 (October, 2013). To reproduce the old behavior call `readCodelinkSet()` with argument `old=TRUE`.

In the current implementation, only probes flagged as MSR spots (flag 'M'- which have an intensity value assigned of -9999), will be automatically converted to NA. This value cannot be adjusted since the value of the probes itself does not represent any measure of signal.

In addition to this, probe weights will be computed by default, based on the conversion table shown in tables 1 and 2. The weight computation follows this process. First, weights are assigned based on type, with DISCOVERY probes being assigned `weight=1` and other probes `weight=0`. Then, weights are adjusted based on flags. The worst weight (type or flag weights when multiple) is assigned to each probe. The weights assigned can be controled by the `type.weights` and `flag.weights` argument to `readCodelinkSet()`. It is possible also to reassign weights after reading with the function `createWeights()`. Weights can be used during preprocessing (background correction and normalization) and linear modeling.

```
> w = createWeights(codset)
> ## NOTE: a proper replacement function will be provided later:
> assayDataElement(codset, "weight") = w
```

2.2 Accessing data

Data stored in a CodelinkSet object can be accessed using several accessor functions:

```
> # get signal intensities. alias: getInt()
> head(exprs(codset))
      Sample-1  Sample-2  Sample-3  Sample-4
```

```
1001 1645.4359 1175.0750 1191.2703 1127.0000
1002   47.2364   39.7000   41.3415   35.9556
1004   42.7961   39.5862   44.3542   36.5977
1005 1008.0443  482.7928  638.2655  482.1574
1006  118.9067   73.3896   82.8421   77.1571
1007 1612.3613 1122.6693 1122.6522 1074.5339
> # get background intensities.
> head(getBkg(codset))
     Sample-1 Sample-2 Sample-3 Sample-4
1001       31       31       31       31
1002       31       31       31       31
1004       31       31       32       31
1005       31       30       31       32
1006       32       31       30       32
1007       31       30       31       32
> # get SNR values.
> head(getSNR(codset))
      Sample-1   Sample-2   Sample-3   Sample-4
1001 37.790711 27.6104937 28.0621444 26.1873780
1002  1.100202  0.9286658  0.9711291  0.8586443
1004  1.007958  0.9298767  1.0146951  0.8475702
1005 22.495867 11.4867798 14.9880533 10.9125875
1006  2.576429  1.7566020  2.0463101  1.8040827
1007 36.999348 25.4019705 25.9305364 24.3570387
> # get flags.
> head(getFlag(codset))
     Sample-1 Sample-2 Sample-3 Sample-4
1001 "G"      "G"      "G"      "G"
1002 "G"      "L"      "L"      "L"
1004 "G"      "L"      "G"      "L"
1005 "G"      "G"      "G"      "G"
1006 "G"      "G"      "G"      "G"
1007 "G"      "G"      "G"      "G"
> # get weights.
> head(getWeight(codset))
     Sample-1 Sample-2 Sample-3 Sample-4
1001        0        0        0        0
1002        1        1        1        1
1004        1        1        1        1
1005        1        1        1        1
1006        1        1        1        1
1007        0        0        0        0
> # get phenoData:
> head(pData(codset))
          sample
Sample-1 T3-5(3)
Sample-2 TX-1(3)
Sample-3 T3-2(3)
Sample-4 T3-4(1)
```

3 Background correction

If Spot_mean and Bkgd_median values are available then background correction can be performed with `codCorrect()`. Background correction methods are borrowed from the limma package, including methods *none*, *subtract*, *half* and *normexp*. The default is set to *half*, because it is very fast. However, more sensitive (although slower) methods like *normexp* are recommended. It is possible to assign an offset to avoid low intensity probes to have high M variances.

```
> codset = codCorrect(codset, method = "half", offset = 0)
```

4 Normalization

Normalization of the background corrected intensities is done by the wrapper function `normalize` (or the alias `codNormalize()`). Here again, normalization is borrowed from the limma package. Methods *median*, *quantile* (the default) and *loess* are available.

```
> codset = codNormalize(codset, method = "quantile")
```

Method *loess* performs CyclicLoess normalzation and accepts weights. Weights are used in a per-probe fashion (that is, one weight for one probe, not different weights for each sample). When weights are used for normalzation the minimum of all the weights for each probe along all the samples will be used. This is to ensure that for each array there is an equal contribution of each probe in the normalization process.

```
> # NOT RUN
> codset = codNormalize(codset, method = "loess", weights = getWeight(codset),
+     loess.method = "fast")
> # NOT RUN
```

5 Diagnostic plots

There are some plot facilities to help diagnose the effect of background correction and normalization, as well as identify putative faulty arrays. The most commonly used functions are MA plots, density plots and array images. All these functions can be accessed through the function codPlot(). The parameter `what` specifies the type of plot: *ma* (default), *density*, *scatter* and *image* are valid choices. Figures 1 and 2 below show examples of these plotting functions.

```
> codPlot(codset)  # by default MA plot.
> codPlot(codset, what = "density")
```

When the columns *Logical_row* and *Logical_col* are present in the original data files, this information is used to assign the physical location of each probe in the array to plot a pseudo image. 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.

```
> codPlot(codset, what = "image")
```

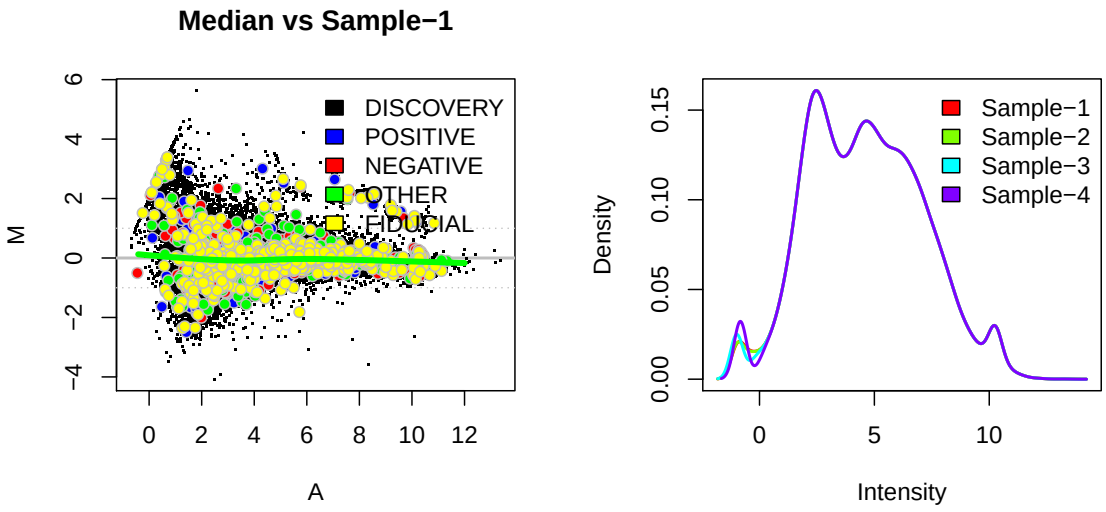

Figure 1: MA plot (left) and density plot (right).

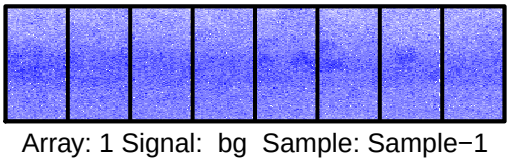

Figure 2: Pseudo image plot of an array

6 Fitting linear models

A typical analysis include the testing for differentially expressed probes between two populations. This can be performed using a variety of different R/Bioconductor packages, but the `limma` package is one of the most popular options. Limma can readily use `CodelinkSet` objects, and can take advantage of weights generated during data reading. In this case, weights will be use probe-wise (i.e. different weights for the same probe in different samples will be considered).

```
> fit = lmFit(codset, design = c(1, 1, 2, 2), weights = getWeight(codset))
> fit2 = eBayes(fit)
> topTable(fit2)
       probeName probeType logicalRow logicalCol   meanSNR    logFC
255020 GE1262775 DISCOVERY        255         20 224.66663 7.985204
311011   GE21204 DISCOVERY        311         11 197.73053 7.967831
31103  GE1152142 DISCOVERY         31        103 222.18072 7.863261
106027 GE1204034 DISCOVERY        106         27 130.05434 7.468283
24019  GE1126416 DISCOVERY         24         19 122.54245 7.312816
150024   GE20195 DISCOVERY        150         24 105.40705 7.265655
321107 GE1181836 DISCOVERY        321        107  98.68877 7.259977
242033 GE1221296 DISCOVERY        242         33 102.22355 7.224388
114068   GE22145 DISCOVERY        114         68 102.59456 7.199294
312012   GE19692 DISCOVERY        312         12  93.38577 7.192154
        AveExpr        t      P.Value    adj.P.Val        B
255020 13.33260 12.79005 1.994916e-37 4.824245e-33 73.30523
311011 13.21738 12.76222 2.851040e-37 4.824245e-33 72.95862
31103  13.14444 12.59473 2.406646e-36 2.714857e-32 70.88828
106027 12.44730 11.96209 5.914179e-33 5.003691e-29 63.31500
24019  12.23010 11.71307 1.146213e-31 7.758030e-28 60.44112
150024 12.14916 11.63753 2.783076e-31 1.496585e-27 59.58128
321107 12.06931 11.62844 3.095590e-31 1.496585e-27 59.47814
242033 12.07106 11.57144 6.020429e-31 2.546792e-27 58.83346
114068 12.06821 11.53124 9.604776e-31 3.611609e-27 58.38079
312012 11.98362 11.51981 1.096667e-30 3.711341e-27 58.25229
```

7 Citation

```
> citation(package = "codelink")

To cite codelink in publications use:

  Diego Diez, Rebeca Alvarez and Ana Dopazo. codelink: An R
  package for analysis of GE Healthcare Gene Expression Bioarrays.
  2007, Bioinformatics

A BibTeX entry for LaTeX users is

  @Article{,
    title = {codelink: an R package for analysis of GE Healthcare Gene
Expression Bioarrays},
    author = {{Diego Diez} and {Rebeca Alvarez} and {Ana Dopazo}},
```

```
    year = {2007},
    journal = {Bioinformatics},
    volume = {23},
    number = {9},
    pages = {1168-9},
    doi = {10.1093/bioinformatics/btm072},
  }
```

8 Session info

```
> sessionInfo()
R version 3.6.1 (2019-07-05)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.3 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.10-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.10-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] knitr_1.25          codelink_1.54.0     limma_3.42.0
[4] Biobase_2.46.0      BiocGenerics_0.32.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.2           formatR_1.7          highr_0.8
 [4] compiler_3.6.1       pillar_1.4.2         bitops_1.0-6
 [7] tools_3.6.1          zeallot_0.1.0        digest_0.6.22
[10] bit_1.1-14           annotate_1.64.0      RSQLite_2.1.2
[13] evaluate_0.14        memoise_1.1.0        tibble_2.1.3
[16] pkgconfig_2.0.3      rlang_0.4.1          DBI_1.0.0
[19] xfun_0.10            stringr_1.4.0        vctrs_0.2.0
[22] S4Vectors_0.24.0     IRanges_2.20.0       stats4_3.6.1
[25] bit64_0.9-7          AnnotationDbi_1.48.0 XML_3.98-1.20
[28] blob_1.2.0           magrittr_1.5         backports_1.1.5
[31] xtable_1.8-4         stringi_1.4.3        RCurl_1.95-4.12
[34] crayon_1.3.4
```