

Introduction to the Codelink package

Diego Diez

May 1, 2014

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 backward compatibility, and information about its use can be found in the vignette `Codelink_Legacy.pdf`.

<i>Probe type</i>	<i>Description</i>	<i>Default weight</i>
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.

<i>Flag</i>	<i>Description</i>	<i>Default value set</i>	<i>Default weight</i>
G	Good signal ($\text{SNR} \geq 1$)	NA	1
L	Limit signal ($\text{SNR} < 1$)		1
S	Saturated signal		1
I	Irregular shape		0
M	MSR spot (-9999)		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"            "AssayData"
[5] "vectorORfactor"
> 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 controlled 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  1645.44  1175.08  1191.27  1127.00

```

```

1002    47.24    39.70    41.34    35.96
1004    42.80    39.59    44.35    36.60
1005   1008.04   482.79   638.27   482.16
1006    118.91    73.39    82.84    77.16
1007   1612.36  1122.67  1122.65  1074.53
>
> # 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   37.791  27.6105  28.0621  26.1874
1002    1.100   0.9287   0.9711   0.8586
1004    1.008   0.9299   1.0147   0.8476
1005   22.496  11.4868  14.9881  10.9126
1006    2.576   1.7566   2.0463   1.8041
1007   36.999  25.4020  25.9305  24.3570
>
> # 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 normalization 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 normalization 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. These images are useful to identify spatial artifact that may be affecting the analysis.

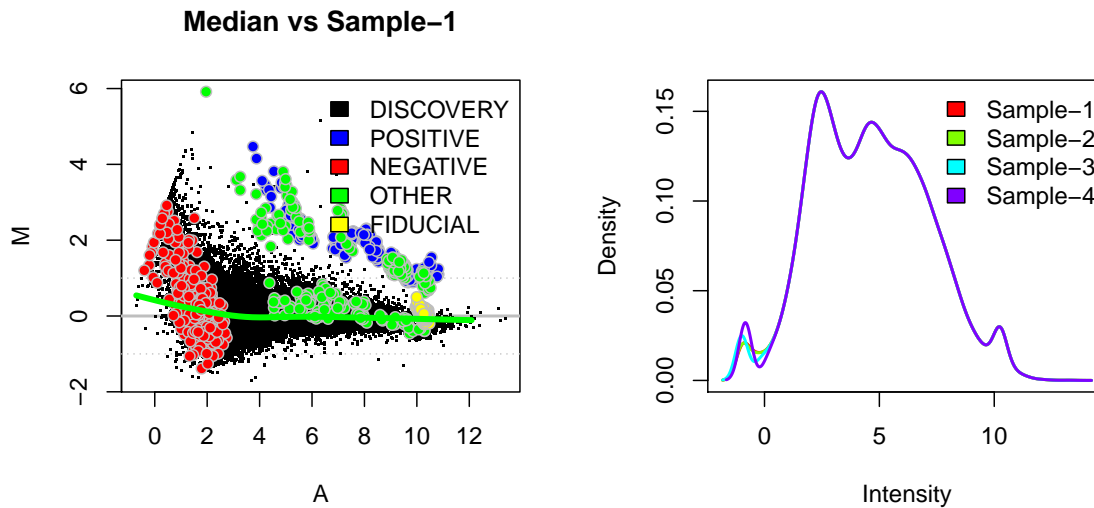


Figure 1: MA plot (left) and density plot (right).

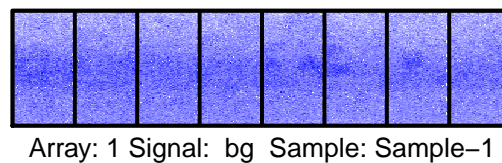


Figure 2: Pseudo image plot of an array

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

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	AveExpr
255020	GE1262775	DISCOVERY	255	20	224.67	7.985	13.33
311011	GE21204	DISCOVERY	311	11	197.73	7.968	13.22
31103	GE1152142	DISCOVERY	31	103	222.18	7.863	13.14
106027	GE1204034	DISCOVERY	106	27	130.05	7.468	12.45
24019	GE1126416	DISCOVERY	24	19	122.54	7.313	12.23

150024	GE20195	DISCOVERY	150	24	105.41	7.266	12.15
321107	GE1181836	DISCOVERY	321	107	98.69	7.260	12.07
242033	GE1221296	DISCOVERY	242	33	102.22	7.224	12.07
114068	GE22145	DISCOVERY	114	68	102.59	7.199	12.07
312012	GE19692	DISCOVERY	312	12	93.39	7.192	11.98
	t	P.Value	adj.P.Val	B			
255020	12.70	6.257e-37	1.506e-32	72.17			
311011	12.67	8.899e-37	1.506e-32	71.83			
31103	12.51	7.294e-36	8.228e-32	69.79			
106027	11.88	1.609e-32	1.362e-28	62.33			
24019	11.63	2.994e-31	2.026e-27	59.49			
150024	11.56	7.181e-31	3.856e-27	58.65			
321107	11.55	7.975e-31	3.856e-27	58.54			
242033	11.49	1.537e-30	6.502e-27	57.91			
114068	11.45	2.436e-30	9.161e-27	57.46			
312012	11.44	2.777e-30	9.396e-27	57.34			

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},
  url = {http://dx.doi.org/10.1093/bioinformatics/btm072},
}
```

8 Session info

```
> sessionInfo()
R version 3.1.0 (2014-04-10)
Platform: x86_64-apple-darwin13.1.0 (64-bit)
```

```

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

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

other attached packages:
[1] knitr_1.5 codetools_1.32.0 limma_3.20.1
[4] Biobase_2.24.0 BiocGenerics_0.10.0

loaded via a namespace (and not attached):
[1] AnnotationDbi_1.26.0 DBI_0.2-7 GenomeInfoDb_1.0.2
[4] IRanges_1.22.5 RSQLite_0.11.4 XML_3.98-1.1
[7] annotate_1.42.0 evaluate_0.5.5 formatR_0.10
[10] highr_0.3 stats4_3.1.0 stringr_0.6.2
[13] tools_3.1.0 xtable_1.7-3

```