

Introduction to the Codelink package

Diego Diez

November 22, 2024

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] "list_OR_List"      "vector_OR_factor"  "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 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 1645.4359 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 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.

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

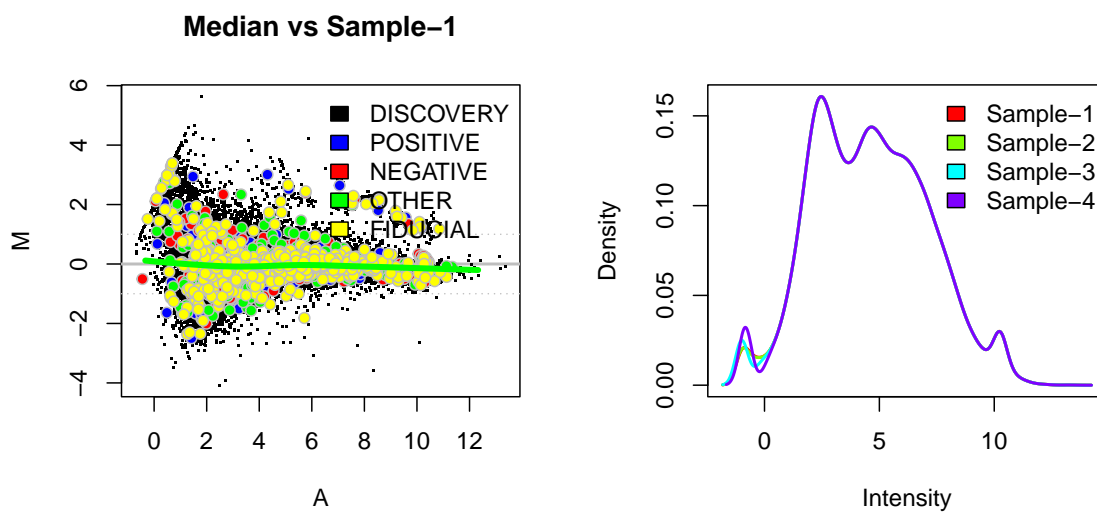


Figure 1: MA plot (left) and density plot (right).



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 | AveExpr |
|--------|-----------|--------------|--------------|------------|-----------|----------|----------|
| 255020 | GE1262775 | DISCOVERY | 255 | 20 | 224.66663 | 7.985204 | 13.33260 |
| 311011 | GE21204 | DISCOVERY | 311 | 11 | 197.73053 | 7.967831 | 13.21738 |
| 31103 | GE1152142 | DISCOVERY | 31 | 103 | 222.18072 | 7.863261 | 13.14444 |
| 106027 | GE1204034 | DISCOVERY | 106 | 27 | 130.05434 | 7.468283 | 12.44730 |
| 24019 | GE1126416 | DISCOVERY | 24 | 19 | 122.54245 | 7.312816 | 12.23010 |
| 150024 | GE20195 | DISCOVERY | 150 | 24 | 105.40705 | 7.265655 | 12.14916 |
| 321107 | GE1181836 | DISCOVERY | 321 | 107 | 98.68877 | 7.259977 | 12.06931 |
| 242033 | GE1221296 | DISCOVERY | 242 | 33 | 102.22355 | 7.224388 | 12.07106 |
| 114068 | GE22145 | DISCOVERY | 114 | 68 | 102.59456 | 7.199294 | 12.06821 |
| 312012 | GE19692 | DISCOVERY | 312 | 12 | 93.38577 | 7.192154 | 11.98362 |
| | t | P.Value | adj.P.Val | B | | | |
| 255020 | 12.73139 | 8.863450e-37 | 2.074213e-32 | 71.77767 | | | |
| 311011 | 12.70555 | 1.225821e-36 | 2.074213e-32 | 71.46363 | | | |
| 31103 | 12.53708 | 9.994042e-36 | 1.127395e-31 | 69.43157 | | | |
| 106027 | 11.90935 | 1.965818e-32 | 1.663180e-28 | 62.08972 | | | |
| 24019 | 11.66107 | 3.563186e-31 | 2.411707e-27 | 59.28627 | | | |
| 150024 | 11.58612 | 8.446908e-31 | 4.466440e-27 | 58.45125 | | | |
| 321107 | 11.57832 | 9.238545e-31 | 4.466440e-27 | 58.36459 | | | |
| 242033 | 11.52059 | 1.788894e-30 | 7.567469e-27 | 57.72537 | | | |
| 114068 | 11.47996 | 2.843083e-30 | 1.069062e-26 | 57.27724 | | | |
| 312012 | 11.46988 | 3.188422e-30 | 1.079026e-26 | 57.16636 | | | |

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 Under development (unstable) (2024-11-20 r87352)
Platform: x86_64-apple-darwin20
Running under: macOS Monterey 12.7.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRlapack.dylib; LAPACK v

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

time zone: America/New_York
tzcode source: internal

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

other attached packages:
[1] knitr_1.49          codelink_1.75.0     limma_3.63.2
[4] Biobase_2.67.0      BiocGenerics_0.53.3 generics_0.1.3

loaded via a namespace (and not attached):
[1] crayon_1.5.3          vctrs_0.6.5          httr_1.4.7
[4] cli_3.6.3             rlang_1.1.4          xfun_0.49
[7] highr_0.11            DBI_1.2.3            UCSC.utils_1.3.0
[10] png_0.1-8             jsonlite_1.8.9       xtable_1.8-4
[13] bit_4.5.0             S4Vectors_0.45.2     statmod_1.5.0
[16] Biostrings_2.75.1     XML_3.99-0.17        formatR_1.14
[19] stats4_4.5.0          KEGGREST_1.47.0      evaluate_1.0.1
[22] fastmap_1.2.0         GenomeInfoDb_1.43.1  IRanges_2.41.1
[25] memoise_2.0.1         compiler_4.5.0       RSQLite_2.3.8
[28] blob_1.2.4            XVector_0.47.0       R6_2.5.1
[31] GenomeInfoDbData_1.2.13 AnnotationDbi_1.69.0  annotate_1.85.0
[34] tools_4.5.0           bit64_4.5.2          zlibbioc_1.53.0
[37] cachem_1.1.0

```