Hands-On: A Framework for Oligonucleotide Microarrays Preprocessing

Benilton S. Carvalho Rafael A. Irizarry

This document provides examples on the usage of the oligo package. The datasets used here can be downloaded by the user, who is expected to have the CEL and XYS files in the following directory structure:

Directory Name	Object Name	Description
expressionData[a]	affyExpressionFS	Latin Square - Affymetrix U95A
exonData[b]	affyExonFS	Exon Sample Dataset - Human
snpData[c]	affySnpFS	HapMap samples on XBA Array
tilingData[d]	nimbleTilingFS	Sample ChIP-chip dataset

Table 1: Directory structure used in this document to store CEL and XYS files.

[a] `http://www.affymetrix.com/support/technical/sample_data/datasets.affx`
[b] `http://www.affymetrix.com/support/technical/sample_data/exon_array_data.affx`
[c] `http://hapmap.ncbi.nlm.nih.gov/downloads/raw_data/affy100k/`
[d] Data available upon request and provided by NimbleGen

1 Preprocessing Expression Arrays

The dataset used in this example corresponds to the Latin Square Data for Expression Algorithm Assessment on the Human Genome U95 platform, made available by Affymetrix on their website[1]. To be used with oligo, requires the availability of the pd.hg.u95a annotation package, built with the pdInfoBuilder package.

[1] `http://www.affymetrix.com/support/technical/sample_data/datasets.affx`

After the annotation package is installed, the next step is to load oligo and identify the files to be used in the analysis. The `list.celfiles` function can be used to appropriately list Affymetrix CEL files. Similarly, the `list.xysfiles` can be used with NimbleGen XYS files. Both functions are built on top of `list.files`, therefore taking the same arguments as the latter, allowing more advanced use when necessary. Below, the `celFiles` contains the all the CEL file names, with full path, in the `expressionData` directory.

```
R> library(oligo)
R> celFiles <- list.celfiles("expressionData", full.names = TRUE)
```

Importing the CEL files is achieved with the `read.celfiles` function. An analogous function, `read.xysfiles`, is available for NimbleGen data, which is delivered via XYS files. Both functions will, in general, correctly identify the annotation package to be used with the experimental data being imported, but the user can specify the *pkgname* argument to force the use of a particular one, if for some reason this is required.

```
R> affyExpressionFS <- read.celfiles(celFiles, pkgname = "pd.hg.u95a")
```

The object `affyExpressionFS` belongs to the *ExpressionFeatureSet* class, as it corresponds to expression data. The object, like all *FeatureSet*-like objects, represents features in the rows and samples in the columns and can be easily subsetted, using the standard [operator. All the manipulation structure is inherited through te tight integration between oligo and Biobase, whose documentation we recommend to the interested reader.

```
R> class(affyExpressionFS)
[1] "ExpressionFeatureSet"
attr(,"package")
[1] "oligoClasses"
```

Figure 1 demonstrates how the *image* method can be used to generate pseudo-images of the samples. In this particular plot, we use the first sample as an example.

```
R> image(affyExpressionFS[, 1], col = gray((64:0)/64))
```

The user can evaluate the distribution of the observed data by using the *hist* method, which will produce smoothed histograms for each sample available in the dataset. Before plotting, the method transforms the data using the function passed to the *transfo* argument, whose default is `log2`, explaining why the plot is shown on the $\log_2$ scale.

Figure 1: Pseudo-image, used for visual assessment of the array, for sample `1521a99hpp_av06.CEL`.

```
R> hist(affyExpressionFS, col = darkColors(59), lty = 1,
     xlim = c(6, 12))
```

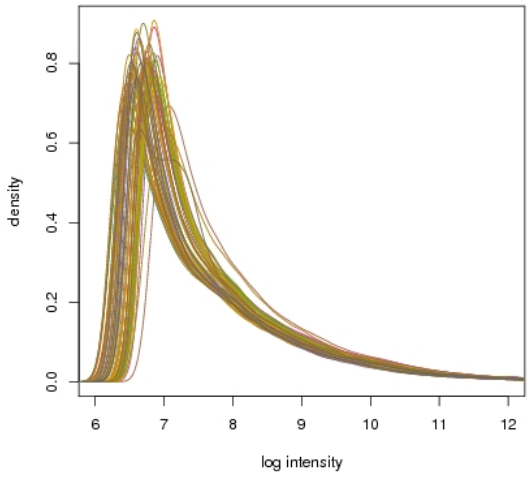


Figure 2: Smoothed histograms for samples in the dataset.

Another approach to assess the data distribution is to use the *boxplot* method. On the example below, we use only the first 10 samples in the dataset to simplify the visualization. On *FeatureSet* objects, the method will automatically transform the data to the $\log_2$ scale,

but this is easily modified through the *transfo* argument, which takes a function as a valid value.

```
R> boxplot(affyExpressionFS[, 1:10], col = darkColors(10),
     names = 1:10)
```

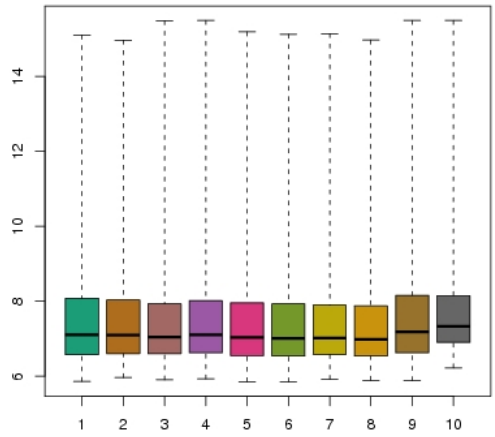


Figure 3: Boxplot showing the distribution of the observed $\log_2$-intensities on the sample dataset. The `boxplot` method implemented in oligo follows the standards of the original method used by R.

Plotting log-ratio *versus* average intensity can often reveal intensity effects on log-ratios, as shown by the MA plot on Figure 4. The argument *arrays* can be specified to determine which samples will be plotted and the *lowessPlot* is a logical flag to indicate that the user wants a lowess curve to be overlapped to the data points.

```
R> MAplot(affyExpressionFS, arrays = 1, lowessPlot = TRUE,
     ylim = c(-1, 1))
```

The annotation packages used by oligo store feature sequences. This is done through instances of *DNAStringSet* objects implemented in the Biostrings package. The sequences for PM probes can be easily accessed via the `pmSequence` function, as shown below.

```
R> pmSeq <- pmSequence(affyExpressionFS)
R> pmSeq[1:5]
  A DNAStringSet instance of length 5
    width seq
[1]    25 GCTGCCCACAGTGACCGACCAGGAG
[2]    25 GCAGCCACCAGTGGACCTAGCCTGG
```

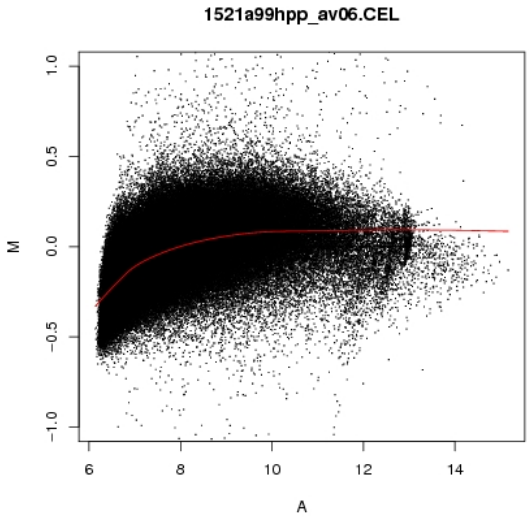


Figure 4: The MA plot can be used to assess the dependence of log-ratios on average log-intensities.

```
[3]     25 CAGCCACCAGTGGACCTAGCCTGGA
[4]     25 CGCATCCACGTGAACTTGAGCACTG
[5]     25 GGCTTCACAGTCACTCGGCTCAGTG
```

When importing the data, oligo does not impose any transformation, so one needs to manually apply, for example, the $\log_2$ transform to the intensities of PM probes, which can be accessed with the `pm` function, as needed. Below, we present how to centralize the $\log_2$-PM intensities for each sample in `affyExpressionFS`.

```
R> pmsLog2 <- log2(pm(affyExpressionFS))
```

The dependence of intensity on probe sequence is a well established fact on the microarray literature. The use of the oligo package simplifies significantly the observation of this event, as it provides simple access to both observed intensities and annotation. Below, we estimate the affinity splines coefficients (Wu et al., 2004).

```
R> coefs <- getAffinitySplineCoefficients(pmsLog2, pmSeq)
```

On Figure 5, we show how the results above can be used to estimate the base-position effects on the $\log_2$-intensities observed for the first sample in the dataset. The `getBasePro-file` function provides a simple way of using the affinity coefficients to estimate the effects of interest. It accepts a *plot* argument, which takes logical values, to make the plot and re-

turns, invisibly, the estimated effects. All the arguments that can be passed to the `matplot` function can also be passed to `getBaseProfile`.

```
R> colors <- darkColors(4)
R> xL <- "Base Position"
R> yL <- "Effect"
R> pchs <- c("A", "C", "G", "T")
R> getBaseProfile(coefs[, 1], plot = TRUE, pch = pchs, type = "b",
     xlab = xL, ylab = yL, lwd = 3, col = colors, ylim = c(-0.4,
         0.4))
```

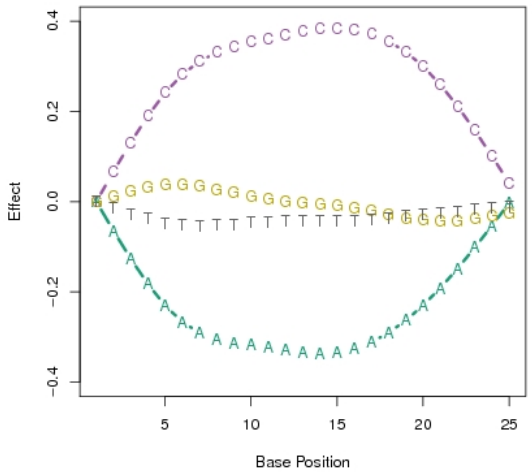


Figure 5: Sequence effect for the first sample in the dataset. These results have been reported in detail elsewhere, but can be easily reproduced with the use of the oligo package.

Tools implemented in other packages can be used in conjunction with oligo to investigate different hypothesis. The example below shows how the `alphabetFrequency` function, defined by the Biostrings can be used to determine the GC content of the probe sequences accessed by oligo.

```
R> counts <- Biostrings::alphabetFrequency(pmSeq, baseOnly = TRUE)
R> GCcontent <- ordered(counts[, "G"] + counts[, "C"])
```

In addition to Figure 5, we can also plot the $\log_2$-intensities as a function of the GC content computed above. Figure 6 presents the strong dependency of $\log_2$-intensities on GC contents for sample 1, which is also present in all other samples.

```
R> colors <- seqColors(nlevels(GCcontent))
R> xL <- "GC Frequency in 25-mers"
```

```
R> yL <- expression(log[2] ~ intensity)
R> boxplot(pmsLog2[, 1] ~ GCcontent, xlab = xL, ylab = yL,
     range = 0, col = colors)
```

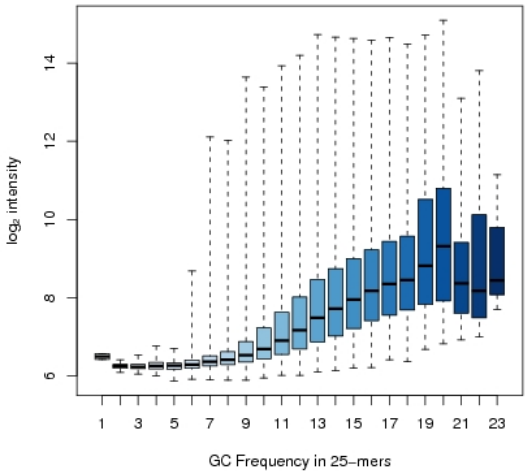


Figure 6: On this boxplot stratified by GC content, we can observe the strong dependency of $\log_2$-intensities on the number of G or C bases observed in the probe sequency.

To preprocess expression data, oligo implements the RMA algorithm (Irizarry et al., 2003a,b). The *rma* method, as shown below, proceeds with background subtraction, normalization and summarization using median polish.

```
R> ppData <- rma(affyExpressionFS)
```

The results are returned in an *ExpressionSet* instance and used in downstream analyses, as currently done by several strategies for microarray data analysis and described elsewhere.

```
R> class(ppData)
[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"
```

At this point, the user can proceed with, for example, differential expression analyses. The methodologies involved in this step make use of several other packages, like limma and genefilter. When preprocessing the data, oligo stores the summaries in a matrix called `exprs`, present in the `assayData` data slot of the *ExpressionSet* object. Therefore, the only restriction the additional strategies used with the preprocessed data have is to be aware that

the processed data can be easily accessed with the *exprs* method.

2 Obtaining Genotype Calls from SNP Arrays

The oligo package can genotype, using the CRLMM algorithm, several Affymetrix SNP arrays. To do so, the user will need, in addition to the oligo package, an annotation data package specific to the designed used in the experiment. Although these annotation packages are created using the pdInfoBuilder package, the CRLMM algorithm requires additional hand-curated data, which are included in the packages made available through the BioConductor website. Table 2 describes the supported designs and the respective annotation packages.

Design	Annotation Package
Mapping 50K XBA	pd.mapping50k.xba240
Mapping 50K HIND	pd.mapping50k.hind240
Mapping 250K NSP	pd.mapping250k.nsp
Mapping 250K STY	pd.mapping250k.sty
Genomewide SNP 5.0	pd.genomewidesnp.5
Genomewide SNP 6.0	pd.genomewidesnp.6

Table 2: SNP array designs currently supported by the oligo package and their respective annotation packages. These annotation packages are made available through the BioConductor website and contain hand-curated data, required by the CRLMM algorithm.

As an example, we will use the 269 CEL files, on the XBA array, available on the HapMap website[2], which were downloaded and saved, uncompressed, to a subdirectory called `snpData`. Therefore, we need to instruct the software to look for the files at the correct location. An output directory should also be defined and that is the place where the summary files, including genotype calls and confidences are stored. This output directory, which we chose to call `crlmmResults`, must not exist prior to the CRLMM call, the software will take care of this task.

```
R> library("oligo")
R> fullFilenames <- list.celfiles("snpData", full.names = TRUE)
R> outputDir <- file.path(getwd(), "crlmmResults")
```

[2] `http://www.hapmap.org`

Given the always increasing density of the SNP arrays, we developed efficient methods to process these chips, reducing the required amount of memory even for large studies. Using this approach, we process batches of SNPs at a time, saving partial results to disk. We refer the interested reader to Carvalho et al. (2007) for detailed information on the CRLMM algorithm. The genotyping strategy, in summary, has three steps: A) quantile normalizes against a known reference distribution; B) summarizes the data to the SNP-allele level using median polish; C) uses estimated parameters to classify the samples in genotype groups using Mahalanobis distance.

The summaries are average intensities and log-ratios, defined as across allele and within strand, ie:

$$A_s = \frac{\theta_{A,s} + \theta_{B,s}}{2} \quad (1)$$

$$M_s = \theta_{A,s} - \theta_{B,s}, \quad (2)$$

where s defines the strand (antisense or sense). On the genomewide designs, SNP 5.0 and 6.0, the strand information is dropped. These summaries can be obtained via *getA* and *getM* methods, which return arrays with dimensions corresponding to SNPs, samples and strands (if applicable), respectively. These measures are later used for genotyping.

CRLMM involves running an EM algorithm to adjust for average intensity and fragment length in the log-ratio scale. These adjustments may take long time to run, depending on the combination of number of samples and computer resources available. Below, we show the simplest way to call CRLMM, which requires only the file names and output directory.

```
R> if (!file.exists(outputDir)) crlmm(fullFilenames, outputDir)
```

The `crlmm` method does not return an object to the R session. Instead, it saves the objects to disk, as not all systems are guaranteed to meet the memory requirements that *SnpSuperSet* objects might need. For the user's convenience, the `getCrlmmSummaries` will read the information from disk and make a *SnpCallSetPlus* or *SnpCnvCallSetPlus* object available to the user.

```
R> crlmmOut <- getCrlmmSummaries(outputDir)
R> calls(crlmmOut[1:5, 1:2])
```

```
             NA06985.CEL NA06991.CEL
SNP_A-1507972           3           3
SNP_A-1510136           3           3
SNP_A-1511055           3           3
SNP_A-1518245           2           3
SNP_A-1641749           3           3
```

```
R> confs(crlmmOut[1:5, 1:2])
```

```
               NA06985.CEL  NA06991.CEL
SNP_A-1507972 0.0009994257 0.0009994060
SNP_A-1510136 0.0009993050 0.0009993744
SNP_A-1511055 0.0009994257 0.0009994257
SNP_A-1518245 0.0009990034 0.0009994257
SNP_A-1641749 0.0009984230 0.0009970634
```

The genotype calls are represented by 1 (AA), 2 (AB) and 3 (BB). The confidence is the predicted probability that the algorithm made the right call.

Summaries generated by the algorithm can also be accessed from the R session. The options for summaries are *"alleleA"*, *"alleleB"*, *"alleleA-sense"*, *"alleleA-antisense"*, *"alleleB-sense"*, *"alleleB-antisense"*. The options *"alleleA"* and *"alleleB"* are only available for SNP 5.0 and SNP 6.0 platforms. The other options are to be used with 50K and 250K arrays.

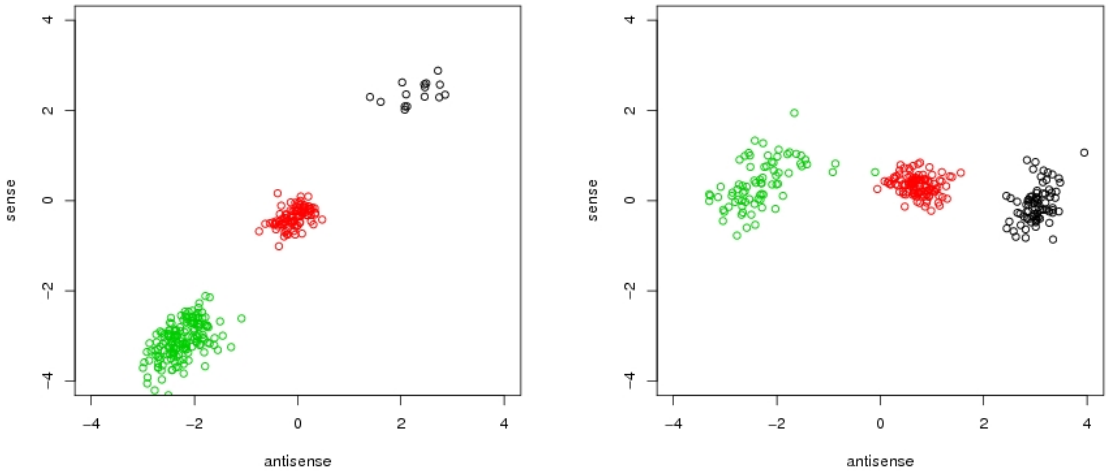
Below, we choose two SNPs to show the different configurations of the genotype groups.

```
R> snps <- paste("SNP_A-", c(1703121, 1725330), sep = "")
R> LIM <- c(-4, 4)
```

Figure 7(a) represents a SNP for which genotyping is simplified by the good discrimination of both strands. Figure 7(b) shows a SNP for which features on the antisense strand have very good discrimination power, while no information (for classification) can be extracted from the sense strand.

```
R> gtypes <- as.integer(calls(crlmmOut[snps[1], ]))
R> plotM(crlmmOut, snps[1], ylim = LIM, xlim = LIM, col = gtypes)
R> gtypes <- as.integer(calls(crlmmOut[snps[2], ]))
R> plotM(crlmmOut, snps[2], ylim = LIM, xlim = LIM, col = gtypes)
```

CRLMM was shown to outperform competing genotyping tools. We refer the reader to Lin et al. (2008) for further details on this subject. The genotypes provided by CRLMM,



(a) SNP_A-1703121 has very good discrimination on both strands and, as competing algorithms, CRLMM has excelent performance on scenarios like this. On this plot, genotype calls provided by oligo are represented in different colors (black: AA; red: AB; green: BB)

(b) SNP_A-1725330 presents poor discrimination on the sense strand. Because CRLMM does not average across strands, it can perfectly predict the genotype cluster each sample belongs to. On similar scenarios, competing algorithms are known to fail. Color scheme follows Figure 7(a).

and in this example stored in `crlmmOut`, can be easily used with other BioConductor tools, like the snpMatrix package, for downstream analyses.

3 Preprocessing Exon Arrays

On this section, we use colon cancer sample data for exon arrays, available on the Affymetrix website[3], to demonstrate the use of the oligo package to import and preprocess these data. The CEL files were downloaded to the `exonData` directory and, after loading the package, we use the `celFiles` variable to store the full CEL file names (including path), as shown below.

```
R> library(oligo)
R> celFiles <- list.celfiles("exonData", full.names = TRUE)
```

[3] `http://www.affymetrix.com/support/technical/sample_data/exon_array_data.affx`

The `read.celfiles` function is used to import CEL files. Its simplest use is shown below. In this example, the parser will read all CEL files present in the `exonData` directory and store the results in the `exonRawData` variable.

```
R> affyExonFS <- read.celfiles(celFiles)
```

As already noted, oligo implements different classes depending on the nature of the data. Therefore, `affyExonFS` is an *ExonFeatureSet* object. This is a especially interesting feature, as it allows methods to behave differently depending on the object class.

Generally, RMA will background correct, quantile normalize and summarize to the probeset level, as defined in the annotation packages. When working with an *ExonFeatureSet* object, processing to the probeset level provides expression summaries at the exon level and can be obtained by setting the argument *target* to `"probeset"`, as presented below.

```
R> probesetSummaries <- rma(affyExonFS, target = "probeset")
```

For Exon arrays, Affymetrix provides additional annotation files that define meta-probesets (MPSs), used to summarize the data to the gene level. These MPSs are classified in three groups – core, extended and full – depending on the level of confidence of the sources used to generate such annotations. Additional values allowed for the *target* argument are `"core"`, `"extended"` and `"full"`. The example below shows how gene level summaries can be obtained through oligo.

```
R> geneSummaries <- rma(affyExonFS, target = "core")
```

The results obtained from analyses performed with oligo can be easily combined with features offered by other packages. As an example, we use the biomaRt package to obtain IDs of probesets on the Human Exon array that map to Entrez Gene ID 10948 (ENSG00000131748).

```
R> library(biomaRt)
R> ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")
R> theIDs <- getBM(attributes = "affy_huex_1_0_st_v2", filters = "entrezgene",
     values = 10948, mart = ensembl)[[1]]
R> theIDs <- as.character(theIDs)
```

Combining this information with the annotation package associated to the data in `affyExonFS`, we can get detailed facts on the probesets found to map to Entrez Gene ID 10948. Below, we obtain, respectively, the MPS IDs, probeset IDs, probe IDs and start/stop posi-

tions for the probesets identified above.

```
R> library(AnnotationDbi)
R> conn <- db(affyExonFS)
R> sql <- paste("SELECT meta_fsetid, pmfeature.fsetid, fid, start, stop",
     "FROM featureSet, pmfeature, core_mps", "WHERE pmfeature.fsetid = featureSet.fset
     "AND featureSet.fsetid = core_mps.fsetid", "AND pmfeature.fsetid IN (",
     toSQLStringSet(theIDs), ")")
R> probesetInfo <- dbGetQuery(conn, sql)
```

The availability of start and stop positions of the probesets improves the visualization of the summaries at the exon level. If genomic coordinates were available for probes themselves, visualization could be improved even more. To achieve this, we first obtain the sequences for the probes identified above. We saw that the *pmSequence* method provides the sequences for all PM probes identified on the chip but, instead, we directly load the Biostrings object used to store the sequence information for these probes. This gives us access not only to the sequences, but also to the probe IDs linked to them.

```
R> library(Biostrings)
R> data(pmSequence, package = annotation(affyExonFS))
```

Because probe IDs are available in the `pmSequence` object, we can easily restrict our search to the probes listed in the `probesetInfo` object.

```
R> idx <- match(probesetInfo[["fid"]], pmSequence[["fid"]])
R> pmSequence <- pmSequence[idx, ]
```

The `pmSequence` object behaves like a *data.frame*, but it is comprised of complex data structures defined in Biostrings. Below, we modify its representation to make it a regular *data.frame* object.

```
R> pmSequence <- data.frame(fid = pmSequence[["fid"]], sequence = as.character(pmSeque
     stringsAsFactors = FALSE)
```

By joining the `probesetInfo` and `pmSequence` objects, we centralize the available probe annotation.

```
R> probeInfo <- merge(probesetInfo, pmSequence)
```

The genomic coordinates in `probeInfo` refer to the probesets. To better visualize the observed probe intensities, we would be better off if the coordinates were relative to the probes. Below, we use the BSgenome.Hsapiens.UCSC.hg18 to obtain up-to-date genomic coordinates. The coordinates are found by aligning the probe sequences to the reference

genome made available through the package. Because Entrez Gene ID 10948 is located on chromosome 17, the search is limited to this region.

```
R> library("BSgenome.Hsapiens.UCSC.hg18")
R> chr17 <- Hsapiens[["chr17"]]
R> seqs <- complement(DNAStringSet(probeInfo[["sequence"]]))
R> seqs <- PDict(seqs)
R> matches <- matchPDict(seqs, chr17)
```

After matching the sequences, we update the genomic coordinates.

```
R> probeInfo[["start"]] <- unlist(startIndex(matches))
R> probeInfo[["stop"]] <- unlist(endIndex(matches))
```

With the updated coordinates, we reorder the probe information object, `probeInfo`, and extract the probe intensities in the same order. The probe ID field, `fid` in `probeInfo`, provides direct access to the probes of interest. The *exprs* method is used to access the intensity matrix of the `affyExonFS` object and immediately subsetted to the probes of interest. After subsetting the observed intensities, we $\log_2$-transform the data.

```
R> probeInfo <- probeInfo[order(probeInfo[["start"]]), ]
R> probeData <- exprs(affyExonFS)[probeInfo[["fid"]], ]
R> probeData <- log2(probeData)
```

We use the updated genomic to estimate the probeset coverage. This information will be used when plotting the data and will provide approximate delimiters of the probesets.

```
R> attach(probeInfo)
R> probesetStart <- aggregate(as.data.frame(start), list(fsetid = fsetid),
     min)
R> names(probesetStart) <- c("fsetid", "start")
R> probesetStop <- aggregate(as.data.frame(stop), list(fsetid = fsetid),
     max)
R> names(probesetStop) <- c("fsetid", "stop")
R> detach(probeInfo)
```

The `psInfo` object will store the probeset information (probeset ID, start and stop positions), as shown below. After ordering appropriately the data, the `psInfo` probeset is attached, to simplify its usage during the R session.

```
R> psInfo <- merge(probesetStart, probesetStop)
R> psInfo <- psInfo[order(psInfo[["start"]]), ]
R> psInfo[["fsetid"]] <- as.character(psInfo[["fsetid"]])
R> attach(psInfo)
R> probesetData <- exprs(probesetSummaries[fsetid, ])
R> detach(psInfo)
```

To visualize the data processed by oligo, we will use the GenomeGraphs package. To match the genome build used to update the probe coordinates, an archived version of the database will be queried.

```
R> library(GenomeGraphs)
R> probeids <- as.character(probeInfo[["fsetid"]])
R> ensembl = useMart("ensembl_mart_51", dataset = "hsapiens_gene_ensembl",
     archive = T)
R> geneid <- "ENSG00000131748"
R> title <- makeTitle(text = geneid, color = "darkred")
```

The raw data, in the $\log_2$ scale, will be represented by the `raw` object below, created with the *makeExonArray* constructor.

```
R> attach(probeInfo)
R> raw <- makeExonArray(intensity = probeData, probeStart = start,
     probeEnd = stop, probeId = probeids, nProbes = rep(1,
         nrow(probeInfo)), dp = DisplayPars(color = "blue",
         mapColor = "dodgerblue2"), displayProbesets = FALSE)
R> detach(probeInfo)
```

The summarized data is also represented through an object created by *makeExonArray*. The structure is identical to the one used above.

```
R> attach(psInfo)
R> exon <- makeExonArray(intensity = probesetData, probeStart = start,
     probeEnd = stop, probeId = fsetid, nProbes = rep(1,
         nrow(psInfo)), dp = DisplayPars(color = "seagreen",
         mapColor = "seagreen"), displayProbesets = FALSE)
```

To represent the probesets designed by Affymetrix, we use an *AnnotationTrack* object.

```
R> affyModel <- makeAnnotationTrack(start = start, end = stop,
     feature = "gene_model", group = geneid, dp = DisplayPars(gene_model = "darkgreen"
R> detach(psInfo)
```

The gene and transcripts representations are build as follows. Affymetrix probes will be represented in green, while the gene will be in orange; transcripts are represented in blue.

```
R> gene <- makeGene(id = geneid, biomart = ensembl)
R> transcript <- makeTranscript(id = geneid, biomart = ensembl)
R> legend <- makeLegend(c("Affymetrix", "Gene"), fill = c("darkgreen",
     "orange"))
```

Figure 7, generated with the `gdPlot` function, shows the representation of the $\log_2$-intensities and summaries at the exon level. It also shows probesets, gene and transcripts on the region of interest.

```
R> gdPlot(list(title, raw, exon, affyModel, gene, transcript,
     legend), minBase = 35067500, maxBase = 35068900)
```

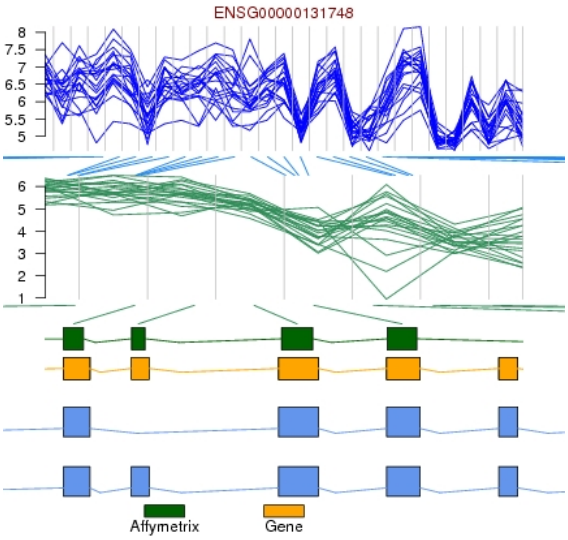


Figure 7: Visual representation of observed $\log_2$-intensities and summarized data at the exon level for gene ENSG00000131748. The probes, gene and transcript are also represented, respectively, in green, orange and blue.

Below, we identify the meta-probeset ID associated to the probes used above. Once that is known, we can extract the proper gene-level summaries stored in `geneSummaries`.

```
R> mps <- unique(probeInfo[["meta_fsetid"]])
R> mps <- as.character(mps)
R> mps
[1] "3720343"
```

Therefore, the standard accessors can be used to obtain the gene summaries for the unit above. Figure 8 shows the expressions for gene ENSG00000131748 across the 33 samples available on this dataset.

```
R> gSummaries <- exprs(geneSummaries[mps, ])
R> x <- 1:length(gSummaries)
R> plot(x, gSummaries, xlab = "Sample", ylab = "Expression",
     main = geneid)
```

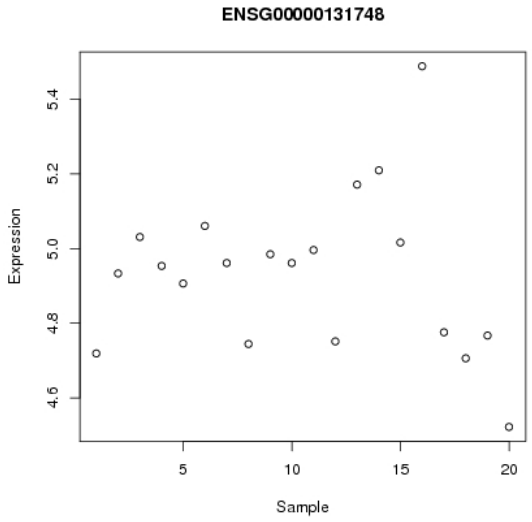


Figure 8: Expression levels estimated through RMA at the gene level.

4 Interfacing with ACME to Find Enriched Regions Using Tiling Arrays

On this Section, we demonstrate how oligo can be easily combined with tools that rely on the structure implemented in the Biobase package. Using a sample ChIP-chip dataset[4] provided by NimbleGen, we use the `getNgsColorsInfo` function to obtain the information regarding channels and sample names for the XYS files saved in the `tilingData` directory. The `getNgsColorsInfo` parses the file names and, using the `_532` and `_635` strings in the names, suggests channels and sample names for each XYS file available.

```
R> library(oligo)
R> info <- getNgsColorsInfo("tilingData", full = TRUE)
R> head(info)
```

```
                   color1                   color2 sampleNames
1 tilingData/92204_532.xys tilingData/92204_635.xys       92204
2 tilingData/92207_532.xys tilingData/92207_635.xys       92207
3 tilingData/92369_532.xys tilingData/92369_635.xys       92369
4 tilingData/94187_532.xys tilingData/94187_635.xys       94187
```

Combining the results in `info` with `read.xyfiles2`, we read the XYS files using a data structure that simplifies the data management across different channels.

[4]Available by request

```
R> nimbleTilingFS <- read.xysfiles2(info[, 2], info[, 1],
     sampleNames = info[, 3])
```

The user can access the channel specific data by calling the *channel* method. The resulting object is an *ExpressionSet* object that the user can use as required.

```
R> c1 <- channel(nimbleTilingFS, "channel1")
R> c2 <- channel(nimbleTilingFS, "channel2")
```

Detailed information on the PM probes available on the array can be obtained by directly querying the annotation package. The call below will extract the `fid`, `fsetid`, `chromosome` and `start` position of each probe from the annotation package and order the results by chromosome and start position.

```
R> sql <- paste("SELECT fid, fsetid, chrom as chromosome, position as start",
     "FROM pmfeature INNER JOIN featureSet USING(fsetid)",
     "ORDER BY chrom, position")
R> annotPM <- dbGetQuery(db(nimbleTilingFS), sql)
```

Using the probe sequence, the end position of the probe can be easily obtained. We load the sequences directly, so the `fid` field can be used to order the sequences appropriately.

```
R> data(pmSequence, package = annotation(nimbleTilingFS))
R> idx <- match(annotPM[["fid"]], pmSequence[["fid"]])
R> pmSequence <- pmSequence[idx, ]
```

To obtain the end position, we use *width*, defined in the Biostrings package.

```
R> attach(annotPM)
R> library(Biostrings)
R> annotPM[["end"]] <- start + width(pmSequence[["sequence"]]) -
     1
R> head(annotPM)
     fid fsetid chromosome start   end
1 392369   5622       chr1 56753 56808
2 286872   5622       chr1 56853 56909
3 229027   5622       chr1 56953 57007
4 386658   5622       chr1 57053 57114
5  85534   5622       chr1 57153 57202
6 170025   5622       chr1 57253 57307
```

The `fid` field corresponds to the row number in the `nimbleTilingFS` object. When applied to the raw data object, the *getM* function returns a matrix with the $\log_2$-ratio of the intensities. Below, we get the $\log_2$-ratios corresponding to the PM probes described in the `annotPM` object.

```
R> ratioPM <- getM(nimbleTilingFS)[fid, ]
R> dimnames(ratioPM) <- NULL
R> detach(annotPM)
R> class(ratioPM)
[1] "matrix"
```

By converting `annotPM` to an *AnnotatedDataFrame*, it can be used in the *featureData* slot of *eSet*-like objects.

```
R> annotPM <- as(annotPM, "AnnotatedDataFrame")
```

We will use the ACME package to calculate enrichment, using algorithms that are insensitive to normalization strategies and array noise. To work with this package, we need to create, first, an *ACMESet* object, which contains `chromosome`, `start` and `end` positions in the `featureData` slot.

```
R> library(ACME)
R> acme <- new("ACMESet", exprs = ratioPM, featureData = annotPM)
```

The `do.aGFF.calc` function processes the *ACMESet* object, using a window size and threshold to identify the positive probes in the object.

```
R> calc <- do.aGFF.calc(acme, window = 1000, thresh = 0.95)
```

The `calc` object is then used to find enriched regions with the `findRegions` function, as shown below.

```
R> regs <- findRegions(calc)
R> head(regs)
         Length    TF StartInd EndInd Sample Chromosome   Start
1.chr1.1   2179 FALSE        1   2179      1       chr1   56753
1.chr1.2      8  TRUE     2180   2187      1       chr1 7943079
1.chr1.3     18 FALSE     2188   2205      1       chr1 7943979
1.chr1.4      8  TRUE     2206   2213      1       chr1 8009343
1.chr1.5    251 FALSE     2214   2464      1       chr1 8010143
1.chr1.6      6  TRUE     2465   2470      1       chr1 9893303
             End       Median         Mean
1.chr1.1 7925574 5.164068e-01 5.290025e-01
1.chr1.2 7943879 1.451904e-05 3.231746e-05
1.chr1.3 8009243 4.002685e-01 3.273235e-01
1.chr1.4 8010043 5.670709e-08 3.615056e-05
1.chr1.5 9893203 5.438609e-01 5.414843e-01
1.chr1.6 9893803 2.471619e-05 4.113231e-05
```

References

Benilton S Carvalho, Henrik Bengtsson, Terence P Speed, and Rafael Irizarry. Exploration, normalization, and genotype calls of high-density oligonucleotide snp array data. *Biostatistics*, 8(2):485–99, Apr 2007. doi: 10.1093/biostatistics/kxl042. URL http://biostatistics.oxfordjournals.org/cgi/content/full/8/2/485.

Rafael Irizarry, Bridget Hobbs, Francois Collin, Yasmin D Beazer-Barclay, Kristen J Antonellis, Uwe Scherf, and Terence P Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, Apr 2003a. doi: 10.1093/biostatistics/4.2.249. URL http://dx.doi.org/10.1093/biostatistics/4.2.249.

Rafael Irizarry, Siew Loon Ooi, Zhijin Wu, and Jef D Boeke. Use of mixture models in a microarray-based screening procedure for detecting differentially represented yeast mutants. *Stat Appl Genet Mol Biol*, 2:Article1, 2003b. doi: 10.2202/1544-6115.1002. URL http://dx.doi.org/10.2202/1544-6115.1002.

S Lin, Benilton S Carvalho, D Cutler, D Arking, A Chakravarti, and Rafael Irizarry. Validation and extension of an empirical bayes method for snp calling on affymetrix microarrays. *Genome Biol*, 9(4):R63, Apr 2008. doi: 10.1186/gb-2008-9-4-r63. URL http://genomebiology.com/2008/9/4/R63.

Zhijin Wu, Rafael Irizarry, Robert C Gentleman, and F Martinez-Murillo .... A model-based background adjustment for oligonucleotide expression arrays. *Journal of the American Statistical Association*, 99(468):909–917, Dec 2004. doi: 10.1198/016214504000000683. URL http://pubs.amstat.org/doi/pdf/10.1198/016214504000000683.