HELP Microarray Analytical Tools

Reid F. Thompson

October 17, 2016

Contents

1 Introduction

The *HELP* package provides a number of tools for the analysis of microarray data, with particular application to DNA methylation microarrays using the Roche Nimblegen format and HELP assay protocol (**?**). The package includes plotting functions for the probe level data useful for quality control, as well as flexible functions that allow the user to convert probe level data to methylation measures.

In order to use these tools, you must first load the *HELP* package:

```
> library(HELP)
```

It is assumed that the reader is already familiar with oligonucleotide arrays and with the design of Roche Nimblegen arrays in particular. If this is not the case, consult the NimbleScan User's Guide for further information (**?**).

Throughout this vignette, we will be exploring actual data for 3 samples from a small corner of a larger microarray chip.

2 Changes for HELP in current BioC release

- This is the first public release of the *HELP* package.

3 Data import and Design information

3.1 Pair files and probe-level data

The package is designed to import matched Roche Nimblegen formatted .pair files, which contain the raw numerical output for each signal channel from each microarray scan. For applications to the HELP assay in particular, the Cy3 (532nm) channel is reserved for the reference sample (MspI) and the Cy5 (635nm) channel is reserved for the experimental half of the co-hybridization (HpaII).

In addition to gridding and other technical controls supplied by Roche NimbleGen, the microarrays also report random probes (50-mers of random nucleotides) which serve as a metric of non-specific annealing and background fluorescence. By design, all probes are randomly distributed across each microarray.

Signal intensity data for every spot on the array is read from each .pair file and stored in an object of class `ExpressionSet`, described in the Biobase vignette. The following code will import three sets of example .pair files included with the *HELP* package:

```
> msp1.files <- dir(data.path, pattern="532[._]pair", full.names=TRUE)
> hpa2.files <- dir(data.path, pattern="635[._]pair", full.names=TRUE)
> N <- length(msp1.files)
> for (i in 1:N) {
+         if (i == 1) {
+                 pairs <- readPairs(msp1.files[i],hpa2.files[i])
+         }
+         else {
+                 pairs <- readPairs(msp1.files[i],hpa2.files[i],pairs)
+         }
+ }
> class(pairs)

[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"

> dim(pairs)

Features  Samples
    1109        3

> sampleNames(pairs)

[1] "70989" "71083" "71420"
```

3.2 Sample Key

The `readSampleKey()` function, which can be used at any point in time, provides the ability to apply a user-defined map of chip names to numeric identifiers, so as to provide human-readable aliases for each set of pair files that are imported. The format of a standard sample key file is tab-delimited text, and contains two columns, CHIP_ID and SAMPLE, with CHIP_ID representing the numeric chip identifier (supplied by NimbleGen) and SAMPLE representing the user-defined alias or human-readable chip name.

```
> chips <- sub("[_.]*532[_.]*pair.*","",basename(msp1.files))
> chips ## CHIP_IDs

[1] "70989" "71083" "71420"

> samplekey <- file.path(data.path, "sample.key.txt")
> chips <- readSampleKey(file=samplekey,chips=chips)
> chips ## SAMPLEs (from supplied key)

[1] "Brain1" "Brain2" "Brain3"

> sampleNames(pairs) <- chips
```

3.3 Design files and information

Roche NimbleGen formatted design files (.ndf and .ngd) are then used to link probe identifiers to their corresponding HpaII fragments, and provide genomic position and probe sequence information, stored as `featureData`. Design file import should be used following .pair file import. File names should end with either .ndf or .ngd, but can contain additional extensions so long as the file formats are appropriate to the relevant design file.

```
> ndf.file <- file.path(data.path, "HELP.ndf.txt")
> ngd.file <- file.path(data.path, "HELP.ngd.txt")
> pairs <- readDesign(ndf.file, ngd.file, pairs)
> pData(featureData(pairs))[1:10,c("CHR","START","STOP")]
```

```
                      CHR     START      STOP
CHRXP81294872_1_1    chrX  81294565  81295582
CHR7P94610020_25_1   chr7  94609060  94610216
CHR7P77493528_27_1   chr7  77493338  77494071
CHR6P06276136_29_1   chr6   6275566   6276664
CHRXP99065670_31_1   chrX  99064104  99065795
CHRXP133973448_33_1 chrX 133973208 133974942
CHR6P07762362_35_1   chr6   7761911   7762945
CHRXP33314570_37_1   chrX  33314570  33314771
CHR7P61447495_39_1   chr7  61447137  61448980
CHR7P61243254_41_1   chr7  61243100  61243312
```

```
> getFeatures(pairs,"SEQUENCE")[1]
```

```
[1] "ATCTAGGAAGATTTAGAGAGGCAATGTGTCATTTAGCATCTAATTTTACC"
```

3.4 Melting temperature (Tm) and GC content

Oligonucleotide melting temperatures can be calculated with the `calcTm()` function. Currently, the only supported method for Tm calculation is the nearest-neighbor base-stacking algorithm (**?**) and the unified thermodynamic parameters (**?**). This functionality can be used for individual or groups of sequences; however, it can also be applied to an object of class `ExpressionSet` containing sequence information.

```
> calcTm("ATCTAGGAAGATTTAGAGAGGCAATGTGTCATTTAGCATCTAATTTTACC")

[1] 76.9773

> calcTm(getFeatures(pairs,"SEQUENCE")[1:4])

[1] 76.97730 89.14768 83.89483 80.47045
```

GC content can be calculated with the `calcGC()` function, which returns values expressed as a percent. This functionality can be used for individual or groups of sequences, and may also be applied to `ExpressionSet` objects containing sequence information.

```
> calcGC("ATCTAGGAAGATTTAGAGAGGCAATGTGTCATTTAGCATCTAATTTTACC")

[1] 0.34

> calcGC(getFeatures(pairs,"SEQUENCE")[1:4])

[1] 0.34 0.60 0.46 0.40
```

4 Quality Control and Data Exploration

4.1 Calculating prototypes

Consideration of probe signal in the context of its performance across multiple arrays improves the ability to discriminate finer deviations in performance. Prototypical signal intensities and ratios can be defined using the `calcPrototype()` function provided with this package. The approach is analagous to one described by Reimers and Weinstein (**?**). Data from each array are (optionally) mean-centered and each probe is then assigned a summary measure equivalent to the (20%-) trimmed mean of its values across all arrays. This defines a prototype with which each individual array can be compared.

```
> getSamples(pairs)[1:4,]
```

```
                     Brain1  Brain2  Brain3
CHRXP81294872_1_1  2182.33 2093.89 2002.56
CHR7P94610020_25_1 4595.11 4822.11 3435.00
CHR7P77493528_27_1 2430.44 1663.56 2246.33
CHR6P06276136_29_1 5897.11 8591.00 4895.33
```

```
> calcPrototype(pairs,center=FALSE)[1:4]
```

```
 CHRXP81294872_1_1 CHR7P94610020_25_1 CHR7P77493528_27_1 CHR6P06276136_29_1
          2092.927           4284.073           2113.443           6461.147
```

4.2 Chip image plots

The `plotChip()` function can be used to display spatial variation of microarray data contained in `ExpressionSet` objects or in a matrix format. As noted previously, the data being explored throughout this vignette represents only a small corner from a larger microarray chip.
The figure shown below is produced using default parameters and therefore shows the data from signal channel 1 (MspI) for the specified sample (Brain2). Note the white blocks on the chip plot, which correspond to coordinates on the array that do not contain probe-level measurements (this is due to the use of .pair file reports, which typically exclude gridding controls).

```
> plotChip(pairs, sample="Brain2")
```

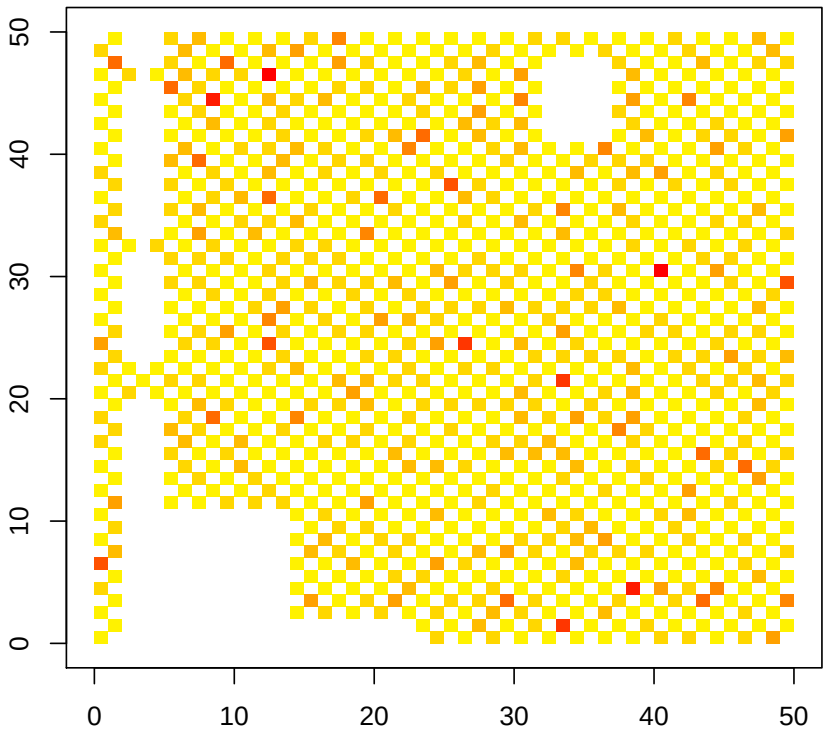

Figure 1: Plot of actual microarray data

The magnitude of data needed to demonstrate quality control analysis for an entire microarray set is too large to include within the scope of this vignette and package distribution. However, please refer to our published work for a further discussion of chip plots and quality control of Roche NimbleGen microarrays (**?**). The following code, included as an example within the R documentation for the `plotChip()` function, demonstrates what one may see in some cases of poor hybridization with high spatial heterogeneity. However, it is important to note that the following figure is generated from synthetic data, as follows:

```
> x <- rep(1:100,100)
> y <- rep(1:100,each=100)
> z <- x*(1001:11000/1000)
> z <- z-mean(z)
> z <- z*(sample(1:10000/10000)+1)
> plotChip(x,y,z,main="Curved gradient",xlab="x",ylab="y")
```

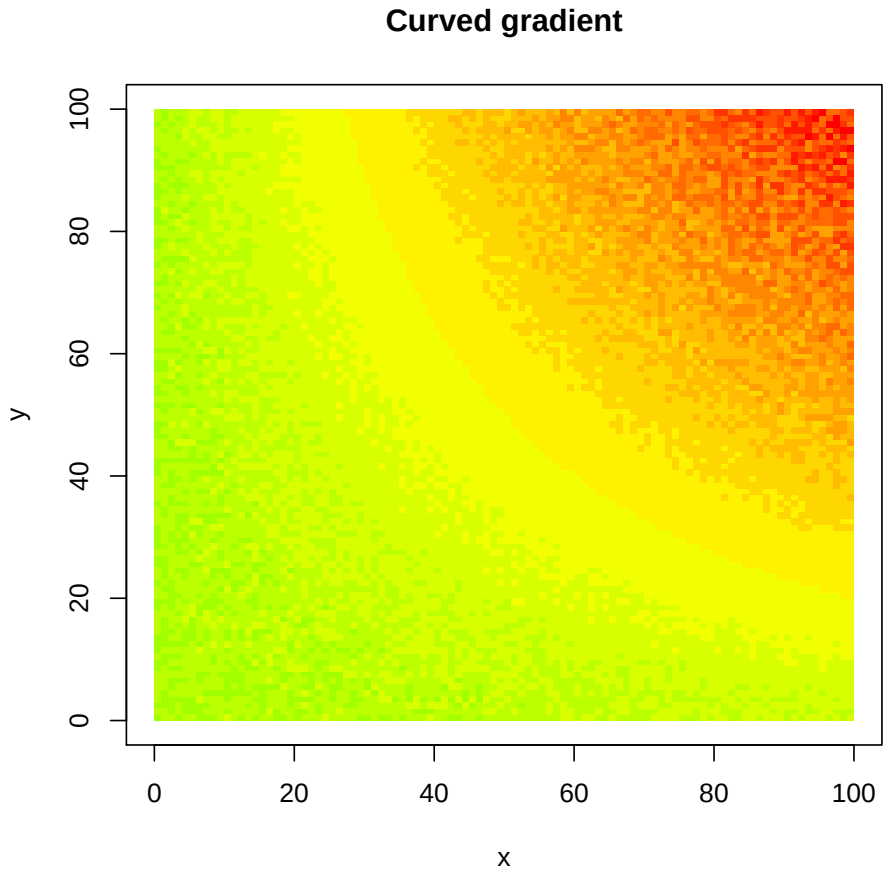

Figure 2: Spatial heterogeneity

4.3 Fragment size v. signal intensity

Visualization of signal intensities as a function of fragment size reveals important behavioral characteristics of the HELP assay. MspI-derived representations show amplification of all HpaII fragments (HTFs) and therefore high signal intensities across the fragment size distribution. The HpaII-derived representation shows a second variable population of probes with low signal intensities across all fragment sizes represented, corresponding to DNA sequences that are methylated (figure below). For a further discussion, refer to **?** and/or **?**. Background signal intensity is measured by random probes. "Failed" probes are defined as those where the level of MspI and HpaII signals are indistinguishable from random probe intensities, defined by a cutoff of 2.5 median absolute deviations above the median of random probe signals.

```
> plotFeature(pairs[,"Brain2"],cex=0.5)
```

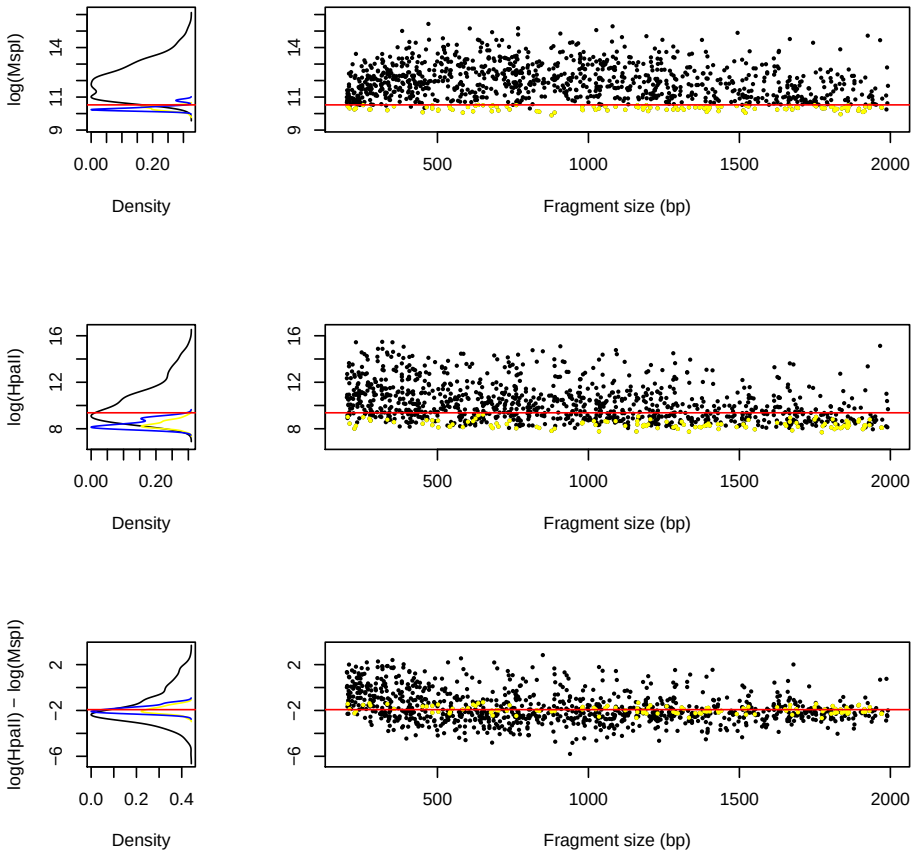

Figure 3: Fragment size v. intensity

5 Single-sample quantile normalization

5.1 Concept

Fragment size v. signal intensity plots demonstrate a size bias that can be traced back to the LM-PCR used in the HELP assay. The *HELP* package makes use of a novel quantile normalization approach, similar to the RMA method described by **?**. The `quantileNormalize()` function, which performs intra-array quantile normalization, is used to align signal intensities across density-dependent sliding windows of size-sorted data. The algorithm can be used for any data whose distribution within each binning window should be identical (i.e. the data should not depend upon the binning variable). For a further discussion of the actual algorithm, please refer to **?**. The figure shown (below) demonstrates a single sample (black distribution) whose components divide into 20 color-coded bins, each of which has a different distribution.

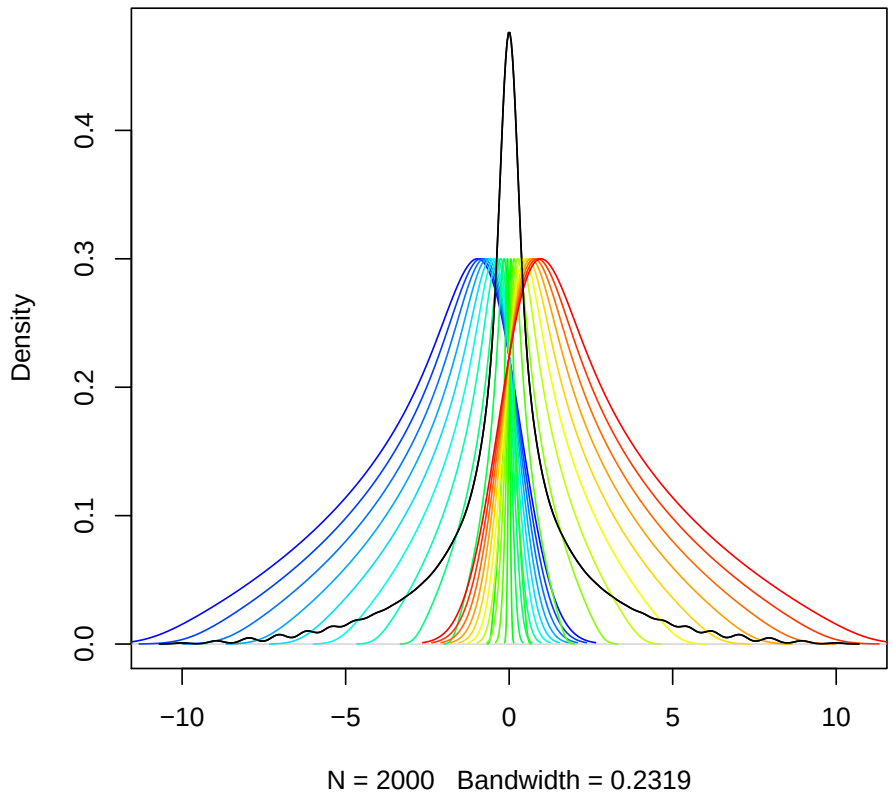

Figure 4: Twenty bins with different distributions, before normalization

With normalization, the different bins are each adjusted to an identical distribution, calculated as the average distribution for all of the component bins. This gives a new sample, normalized across its component bins, shown in the figure (below). Note that the `plotBins()` function (included) can be used to explore bin distributions in a manner similar to the figure depicted (below). Also note that the individual bin densities are stacked (for easier visualization), the alternative being complete overlap and a loss of ability to visually resolve independent bin distributions.

```
> quantileNormalize(x, y, num.bins=20, num.steps=1, ...)
```

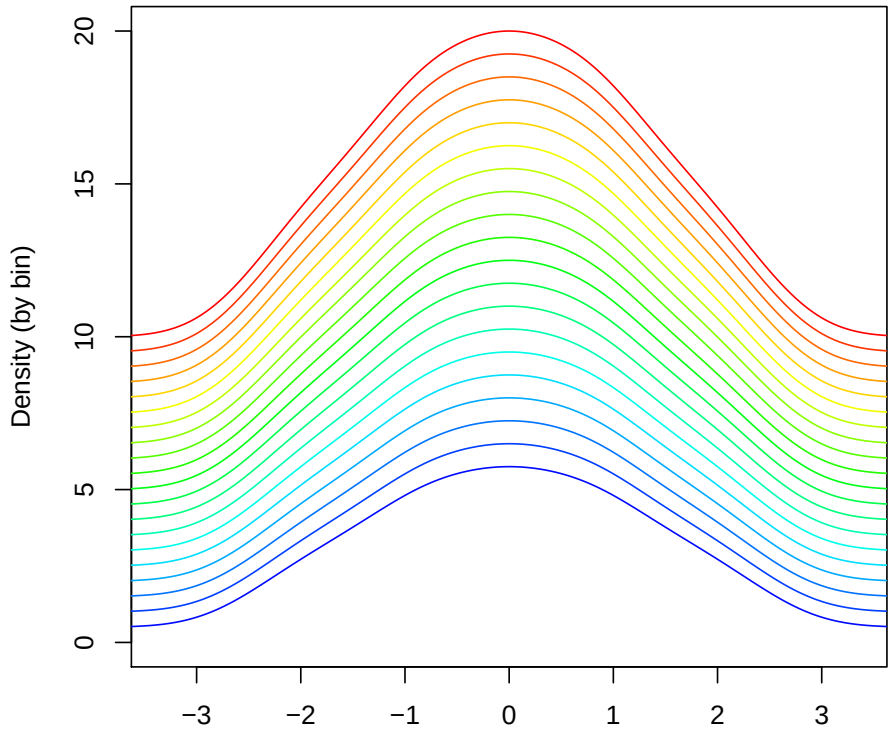

Figure 5: Twenty bins with identical distributions, after normalization

5.2 Application

To apply the normalization to actual HELP data, the data must be considered in terms of their component signals. Specifically, HpaII and MspI must be treated individually, and the signals that exist above background noise must be treated separately from those that fall within the distribution of noise (defined by random probes). Note that data should already be loaded appropriately (as above).

- Identify background noise (note that MspI data is stored in element "exprs" while HpaII is stored in element "exprs2"):

```
> rand <- which(getFeatures(pairs, "TYPE")=="RAND")
> msp.rand <- getSamples(pairs, element="exprs")[rand,]
> hpa.rand <- getSamples(pairs, element="exprs2")[rand,]
```

- Define background cutoffs:

```
> msp.rand.med <- apply(msp.rand, 2, median)
> msp.rand.mad <- apply(msp.rand, 2, mad)
> hpa.rand.med <- apply(hpa.rand, 2, median)
> hpa.rand.mad <- apply(hpa.rand, 2, mad)
> msp.fail <- msp.rand.med + 2.5*msp.rand.mad
> hpa.fail <- hpa.rand.med + 2.5*hpa.rand.mad
> hpa.meth <- apply(hpa.rand, 2, quantile, 0.99)
```

- MspI normalization: handle one sample at a time (in this case, "Brain2") and remove "failed" probes from consideration:

```
> norand <- which(getFeatures(pairs, "TYPE")=="DATA")
> size <- as.numeric(getFeatures(pairs, "SIZE"))[norand]
> msp <- getSamples(pairs, "Brain2", element="exprs")[norand]
> hpa <- getSamples(pairs, "Brain2", element="exprs2")[norand]
> nofail <- which(msp>msp.fail["Brain2"] | hpa>hpa.fail["Brain2"])
> msp.norm <- msp
> msp.norm[nofail] <- quantileNormalize(msp[nofail],size[nofail])
```

- HpaII normalization: handle probe-level data that fall within background distribution separately from high signals:

```
> meth <- which(msp>msp.fail["Brain2"] & hpa<=hpa.meth["Brain2"])
> hpa.norm <- hpa
> hpa.norm[meth] <- quantileNormalize(hpa[meth],size[meth])
> nometh <- which(hpa>hpa.meth["Brain2"])
> hpa.norm[nometh] <- quantileNormalize(hpa[nometh],size[nometh])
```

- Create normalized `ExpressionSet` object:

```
> pairs.norm <- pairs
> exprs(pairs.norm)[norand, "Brain2"] <- msp.norm
> exprs2(pairs.norm)[norand, "Brain2"] <- hpa.norm
> getSamples(pairs, element="exprs")[1:5,]

                     Brain1   Brain2   Brain3
CHRXP81294872_1_1  11.09165 11.03197 10.96763
CHR7P94610020_25_1 12.16588 12.23545 11.74609
CHR7P77493528_27_1 11.24700 10.70006 11.13335
CHR6P06276136_29_1 12.52579 13.06861 12.25719
CHRXP99065670_31_1 10.76377 10.70755 10.89263

> getSamples(pairs.norm, element="exprs")[1:5,]

                     Brain1   Brain2   Brain3
CHRXP81294872_1_1  11.09165 10.95601 10.96763
CHR7P94610020_25_1 12.16588 12.22624 11.74609
CHR7P77493528_27_1 11.24700 10.67728 11.13335
CHR6P06276136_29_1 12.52579 12.89556 12.25719
CHRXP99065670_31_1 10.76377 11.03563 10.89263
```

6 Data summarization

The methylation status of each HpaII fragment is typically measured by a set of probes (number is variable, depending on the array design). Thus, probe-level data must be grouped and summarized. The `combineData()` function employs a simple mean (by default) to each group of probe-level datapoints. This functionality should be applicable to any dataset defined by containers consisting of multiple (and potentially variable numbers of) instances of probe-level data which require summarization. For application to HELP, MspI signal intensities are supplied as a weighting matrix and the 20%-trimmed mean is calculated for each group of probes. Here we show the first five results:

```
> data <- getSamples(pairs,element="exprs2")
> seqids <- getFeatures(pairs,'SEQ_ID')
> weight <- getSamples(pairs,element="exprs")
> combineData(data, seqids, weight, trim=0.2)[1:5,]

                    Brain1    Brain2    Brain3
MM6MSPIS00690755  9.295631  8.688460  8.273516
MM6MSPIS00292185 11.059568 10.962412 10.179499
MM6MSPIS00288594  9.884674  9.382257  9.089239
MM6MSPIS00229944 10.859193 11.190028 10.277287
MM6MSPIS00694154  8.523562  8.429072  7.546894
```

The summarization can also be applied directly to `ExpressionSet` objects. The following code generates an unweighted summarization of MspI signal intensity data, again we show the first five results:

```
> combineData(pairs, feature.group='SEQ_ID', trim=0.2)[1:5,]

                   Brain1   Brain2   Brain3
MM6MSPIS00690755 11.09165 11.03197 10.96763
MM6MSPIS00292185 12.16588 12.23545 11.74609
MM6MSPIS00288594 11.24700 10.70006 11.13335
MM6MSPIS00229944 12.52579 13.06861 12.25719
MM6MSPIS00694154 10.76377 10.70755 10.89263
```

7 Data Visualization

Sample-to-sample relationships can be explored at the global level using both pairwise (Pearson) correlation and unsupervised clustering approaches, among other techniques. The `cor()` and `hclust()` functions perform these tasks in particular. However, the `plotPairs()` function included with this package is particularly suited for pairwise visualization of sample relationships. For instance, HpaII signals are compared in the following figure:

```
> plotPairs(pairs,element="exprs2")
```

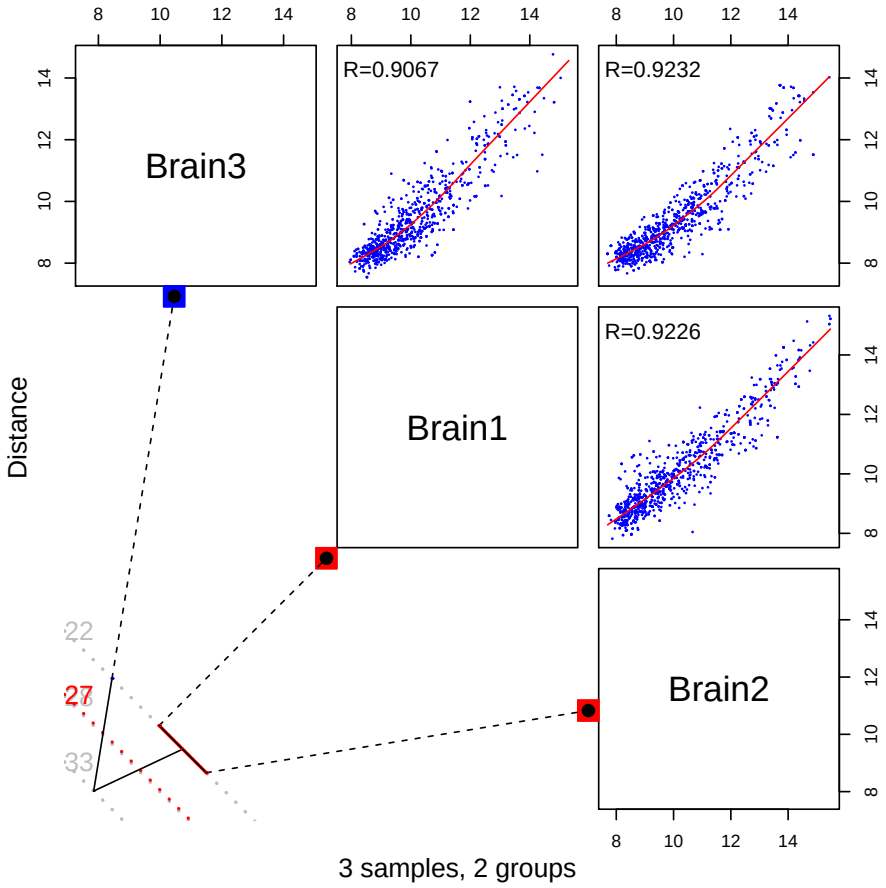

Figure 6: Pairwise comparison of samples

A Previous Release Notes

- No previous releases to date.