Analysis of Bead-level Data using beadarray

Mark Dunning

January 15, 2016

Introduction

beadarray is a package for the pre-processing and analysis of Illumina BeadArray. The main advantage is being able to read raw data output by Illumina's scanning software. Data presented in this form are in the same format regardless of the assay (i.e expression, genotyping, methylation) being performed. Thus, beadarray is able to handle all these types of data. Many functions within beadarray have been written to cope with this flexibility.

The BeadArray technology involves randomly arranged arrays of beads, with beads having the same probe sequence attached colloquially known as a bead-type. BeadArrays are combined in parallel on either a rectangular chip (BeadChip) or a matrix of 8 by 12 hexagonal arrays (Sentrix Array Matrix or SAM). The BeadChip is further divided into strips on the surface known as sections, with each section giving rise to a different image when scanned by BeadScan. These images, and associated text files, comprise the raw data for a beadarray analysis. However, for BeadChips, the number of sections assigned to each biological sample may vary from 1 on HumanHT12 chips, 2 on HumanWG6 chips or sometimes ten or more for SNP chips with large numbers of SNPs being investigated.

This vignette demonstrates the processing of bead-level data using beadarray using data from the *beadarrayExampleData* package. A more comprehensive commentary on the analysis of Illumina BeadArray package is given in the vignette of *BeadArrayUseCases*, including other analysis tools that are not part of *beadarray*.

```
library(beadarrayExampleData)
library(beadarray)
data(exampleBLData)
```

Citing beadarray

If you use *beadarray* for the analysis or pre-processing of BeadArray data please cite:

Dunning MJ, Smith ML, Ritchie ME, Tavaré S, **beadarray: R classes and methods for Illumina bead-based data**, *Bioinformatics*, **23**(16):2183-2184

1 Asking for help on beadarray

Wherever possible, questions about *beadarray* should be sent to the Bioconductor mailing list[1]. This way, all problems and solutions will be kept in a searchable archive. When posting to this mailing list, please first consult the *posting guide*. In particular, state the version of *beadarray* and R that you are using[2], and try to provide a reproducible example of your problem. This will help us to diagnose the problem.

2 Reading bead-level data into beadarray

2.1 File formats

The raw images and text files required to perform a bead-level analysis are produced by Illumina's BeadScan or iScan software. Usually, it will be necessary for you to modify BeadScan's default settings to obtain bead-level data, see `http://www.compbio.group.cam.ac.uk/Resources/illumina`.

The command to read bead-level data from the current working directory is as follows. However, raw data are not included with *beadarrayExampleData* or *beadarray*. See the *BeadArrayUseCases* package for some example data to try out this function.

```
BLData = readIllumina(useImages=FALSE, illuminaAnnotation = "Humanv3")
```

The `useImages` argument specifies whether beadarray will read foreground and background intensities from the TIFF images present in the directory, allowing users to experiment with strategies for image processing. Such strategies are described in greater detail in the `imageProcessing.pdf` vignette. In this example we set `useImages=FALSE` (often a convenient choice), and locally background corrected intensities will simply be extracted from the `txt` files. The *optical* background-correction that is referred to here is done by subtracting the *background* pixel intensities surrounding each bead. It should not to be confused with another background correction further along the analysis pipeline, which may involve negative control beads to account for non-specific binding. *beadarray* is able to use some of Illumina's files during analysis. These include `.locs` files, which contain the locations of *all* beads on an array (not just those that were decoded), and `.sdf` files, which contain information about the physical layout of the chip. In combination, using these files can result in significant time improvements to the detection of spatial artifacts and add additional information to some QA plots. These files are not read automatically, but if present, the path to these files is stored by beadarray for future use. If the *metrics* file generated by BeadScan is present in the directory, it will be read unaltered and stored.

[1] `http://www.bioconductor.org`

[2] This can be done by pasting the output of running the function `sessionInfo()`.

2.2 A note for those with iScan data

Data from Illumina's newer iScan system come in a different format to the previous BeadScan data. The scanner itself is capable of producing higher-resolution images and there are two images of each array section (along with two .locs files), which are labeled Swath1 and Swath2. These two images are of the two halves of the array section, with an overlapping region in the middle. However, there is only one bead-level text file (with the extension perBeadFile.txt), with no indication as to which of the two images each entry comes from. Given this, simply reading the bead-level text file will result in any function that uses bead locations performing undesirably. However, the `readIllumina` function is able to detect that iScan data is present and will advise the user to run the `processSwathData` function, which will try and deconvolute the bead-level data and create two files, one per swath, which can then be read independently into beadarray.

2.3 Array annotation

The text files produced by the scanning software give a very limited annotation for each bead that was scanned. All beads are associate with position on the array and a numeric code (`ArrayAddress` that refers to the *decoding* oligonucleotide sequence attached to the bead[3]. A collection of beads with the same ArrayAddress are known as a bead-type and have the same 50-base sequence attached. However, the sequence and the region of the genome that it targets cannot be inferred from bead-level data alone. The mapping between ArrayAddress and hybridization sequence is provided on Illumina's FTP site in a series of flat-files and we have built Bioconductor packages that can be accessed from within *beadarray*. In order that the correct mappings are performed, users must specify an annotation name for their data, which requires knowing the organism being investigated and annotation revision number (e.g. Humanv4, Humanv3, Humanv2, Humanv1, Mousev2, Mousev1p1, Mousev1 or Ratv1). The `suggestAnnotation` function may be used if you are unsure of which string to use. This checks the overlap between the bead IDs found in the data with a collection of IDs extracted from Illumina's annotation files. For the example data stored with the `beadarrayExampleData`, the suggested annotation is `Humanv3`. Provided that the *illuminaHumanv3.db* package is present, *beadarray* will be able to annotate the `beadarrayExampleData` object.

```
suggestAnnotation(exampleBLData,verbose=TRUE)
```

```
## Percentage of overlap with IDs on this array and known expression platforms
## HUMANREF8_V3_0_R1_11282963_A_WGDASL        HumanHT12_V3_0_R3_11283641_A
##                             48.78621                             97.16521
##        HumanHT12_V4_0_R1_15002873_B        HumanHT12_V4_0_R2_15002873_B
##                             83.70782                             84.05715
## HumanHT12_V4_0_R2_15002873_B_WGDASL                         HumanRef8_V1
##                             54.68237                             37.33872
##        HumanRef8_V2_0_R2_11223162_A        HumanRef8_V2_0_R4_11223162_A
##                             38.88553                             38.88470
```

[3]These decoding sequences are required due to the random nature of each array. However, the sequences themselves have never been disclosed

```
##        HumanRef8_V3_0_R0_11282963_A        HumanRef8_V3_0_R3_11282963_A
##                            48.78621                            48.78621
##                          HumanWG6_V1         HumanWG6_V2_0_R2_11223189_A
##                            37.33872                            81.32701
##          HumanWG6_V2_0_R4_11223189_A              HumanWG6_V2_11223189_B
##                            81.32619                            81.32619
##          HumanWG6_V3_0_R3_11282955_A                        MouseRef8_V1
##                            97.16521                            36.67984
##        MouseRef8_V1_1_R4_11234312_A        MouseRef8_V2_0_R3_11278551_A
##                            38.10629                            43.77615
##                          MouseWG6_V1         MouseWG6_V1_1_R4_11234304_A
##                            36.67984                            38.10629
##                       MouseWG6_V1_B         MouseWG6_V2_0_R3_11278593_A
##                            36.58768                            78.45060
##          RatRef12_V1_0_R5_11222119_A
##                            36.06644
## [1] "Humanv3"
```

```
annotation(exampleBLData) <-"Humanv3"
```

The verbose output of `suggestAnnotation` shows high overlap between the ArrayAddress IDs in the `exampleBLData` object and the ArrayAddress IDs in the official annotation files `HumanHT12_V3_0_R3_11283641_A` and `HumanWG6_V3_0_R3_11282955_A`. However, both HT12 and WG6 arrays have the same probe sequences on them and the difference is the number of sections on a chip. Hence, we can assign the `Humanv3` label to data from either platform.

3 The beadLevelData class

Once imported, the bead-level data are stored in an object of class *beadLevelData*. This class can handle raw data from both single channel and two-colour BeadArrays. Due to the random nature of the technology, each array generally has a variable number of rows of intensity data, and we use an R environment variable to store this information in a memory efficient way.

The beadLevelData class contains a number of slots useful for describing Illumina data. The data that have been extracted from the text files are found in the beadData slot. This can be thought of as a list, which can be indexed by name or a numeric value representing a particular array-section. A data frame holds the data for that array-section, with the number of rows being the number of beads on the section. For convenience, the function `getBeadData` is used to access data held in the beadData slot. The function `insertBeadData` can be used to assign new data to this slot.

Data types with one value per array-section can be stored in the sectionData slot. For instance, any metrics information present in the directory used by `readIllumina` will be stored here. This is also a convenient place to store any QC information derived during the pre-processing of the data, as we will see.

The numeric identifiers for the bead-types in the *beadLevelData* are known as ArrayAddress IDs in Illumina's annotation files. For downstream analysis it is convenient to convert these into the form ILMN_... used in most annotation packages. Mapping objects to convert these IDs are supplied with beadarray in the `extdata` directory, but this conversion may be performed automatically if the annotation of the *beadLevelData* object is known. For two-channel data, the intensities from the Red channel and associated coordinates are also stored in the object.

```
class(exampleBLData)
```

```
## [1] "beadLevelData"
## attr(,"package")
## [1] "beadarray"
```

```
slotNames(exampleBLData)
```

```
## [1] "beadData"       "sectionData"    "experimentData" "history"
```

```
##Get the beadData for array-section 1
exampleBLData[[1]][1:10,]
```

```
##       ProbeID      GrnX      GrnY Grn wts
##  [1,]   10008  900.6661 10781.320 355   1
##  [2,]   10008 1992.5400 11352.000 377   1
##  [3,]   10008 1257.4790  7559.513 452   1
##  [4,]   10008 1700.1600  6351.157 267   1
##  [5,]   10008 1814.5210  3299.495 431   1
##  [6,]   10008 2060.3440  8471.688 357   1
##  [7,]   10008  609.0356  6028.458 408   1
##  [8,]   10008 1487.7190 15933.790 431   1
##  [9,]   10008 1517.5080 14928.520 351   1
## [10,]   10008 1619.3640 17650.690 235   1
```

```
##Alternative using accessor function
getBeadData(exampleBLData, array=1, what="Grn")[1:10]
```

```
##  [1] 355 377 452 267 431 357 408 431 351 235
```

```
##Get unique ProbeIDs. These are the ArrayAddressIDs
uIDs = unique(getBeadData(exampleBLData, array=1, what="ProbeID"))
uIDs[1:10]
```

```
##  [1] 10008 10010 10017 10019 10020 10021 10025 10035 10037 10039
```

4 Scan Metrics

The first view of array quality can be assessed using the metrics calculated by the scanner. These include the 95th (P95) and 5th (P05) quantiles of all pixel intensities on the image. A signal-to-noise ratio (SNR) can be calculated as the ratio of these two quantities. These metrics can be viewed in real-time

as the arrays themselves are being scanned. By tracking these metrics over time, one can potentially halt problematic experiments before they even reach the analysis stage. The metrics information for the `exampleBLData` object can retrieved in the following way. Illumina recommend that the SNR ratio should be above 10, so these arrays are acceptable. However, the P95 and P05 values will fluctuate over time and are dependant upon the scanner setup. Including SNR values for arrays other than those currently being analysed will give a better indication of whether any outlier arrays exist.

```
metrics(exampleBLData)
```

```
##                    Date     Matrix Section RegGrn FocusGrn SatGrn P95Grn P05Grn
## 1  3/13/2009 6:45:04 PM 4613710017       B   0.13     0.70      0    704     36
## 12       04/01/09 04:50 4616494005       A   0.13     0.59      0    678     38
##    RegRed FocusRed SatRed P95Red P05Red
## 1       0        0      0      0      0
## 12      0        0      0      0      0
```

```
p95(exampleBLData, "Grn")
```

```
## [1] 704 678
```

```
snr(exampleBLData, "Grn")
```

```
## [1] 19.55556 17.84211
```

5 Transformation Functions

A more flexible way to obtain per-bead data from a beadLevelData object is to define a transformation function that takes as arguments the beadLevelData object and an array index. The function then manipulates the data in the desired manner and returns a vector the same length as the number of beads on the array. The `logGreenChannelTransform` is the default transformation in many plotting / QA functions within beadarray. Users with two-channel data may also wish to experiment with the similarly defined `logRedChannelTransform` or `logRatioTransform` when plotting.

```
log2(exampleBLData[[1]][1:10,2])
```

```
##  [1]  9.814849 10.960393 10.296319 10.731455 10.825373 11.008670  9.250383
##  [8] 10.538886 10.567488 10.661212
```

```
logGreenChannelTransform
```

```
## function (BLData, array)
## {
##     x = getBeadData(BLData, array = array, what = "Grn")
##     return(log2.na(x))
## }
## <environment: namespace:beadarray>
```

```
logGreenChannelTransform(exampleBLData, array=1)[1:10]
```

```
##  [1] 8.471675 8.558421 8.820179 8.060696 8.751544 8.479780 8.672425 8.751544
##  [9] 8.455327 7.876517
```

```
logRedChannelTransform
```

```
## function (BLData, array)
## {
##     x = getBeadData(BLData, array = array, what = "Red")
##     return(log2.na(x))
## }
## <environment: namespace:beadarray>
```

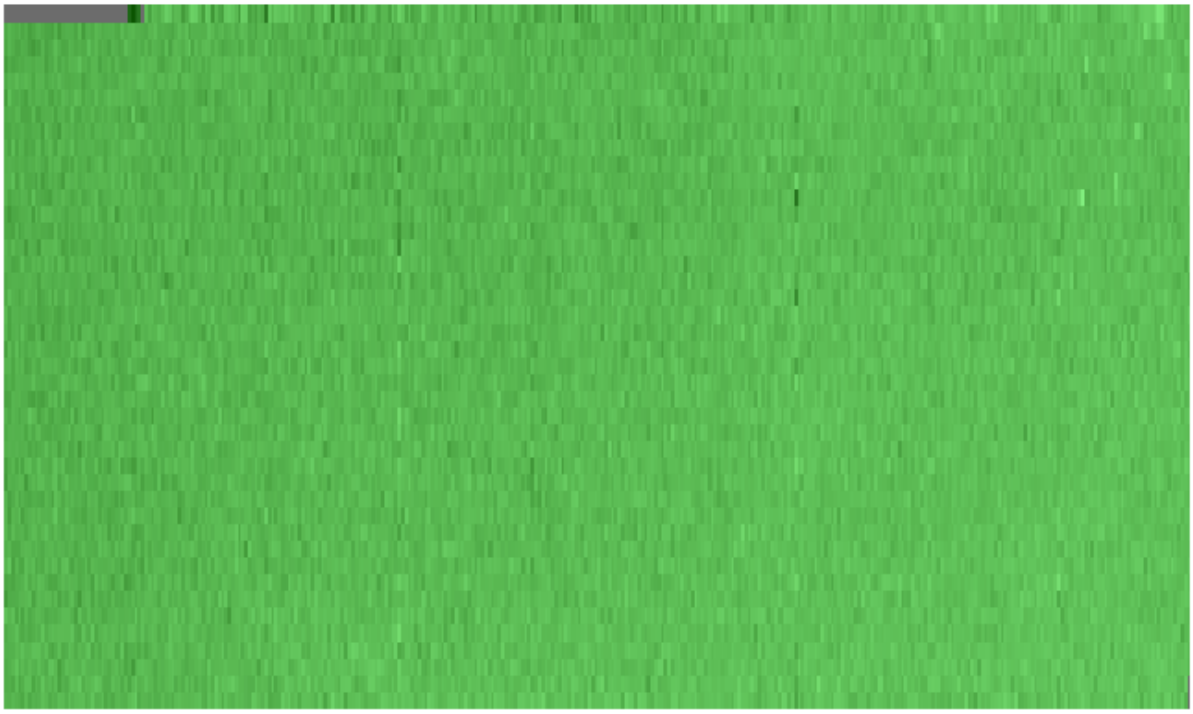
6 Boxplots and imageplots

Two standard quality assessment plots supported by *beadarray* are the imageplot and boxplot. Boxplots can be used to compare foreground and background intensities between arrays. Image plots can be used to identify spatial artifacts on the array surface that can occur from mis-handling or scanning problems. With the raw bead-level data, we can plot false images of each array. This kind of visualization is not possible when using the summarized BeadStudio output, as the summary values are averaged over spatial positions. Image plots in R are also more convenient than scrutinizing the original tiffs, as multiple arrays can be visualized on the one page. By default, the array surface is plotted with the longest edge going horizontally. Both the `boxplot` and `imageplot` functions take a transformation function as an argument, with the default to do a $\log_2$ transformation on the green channel.

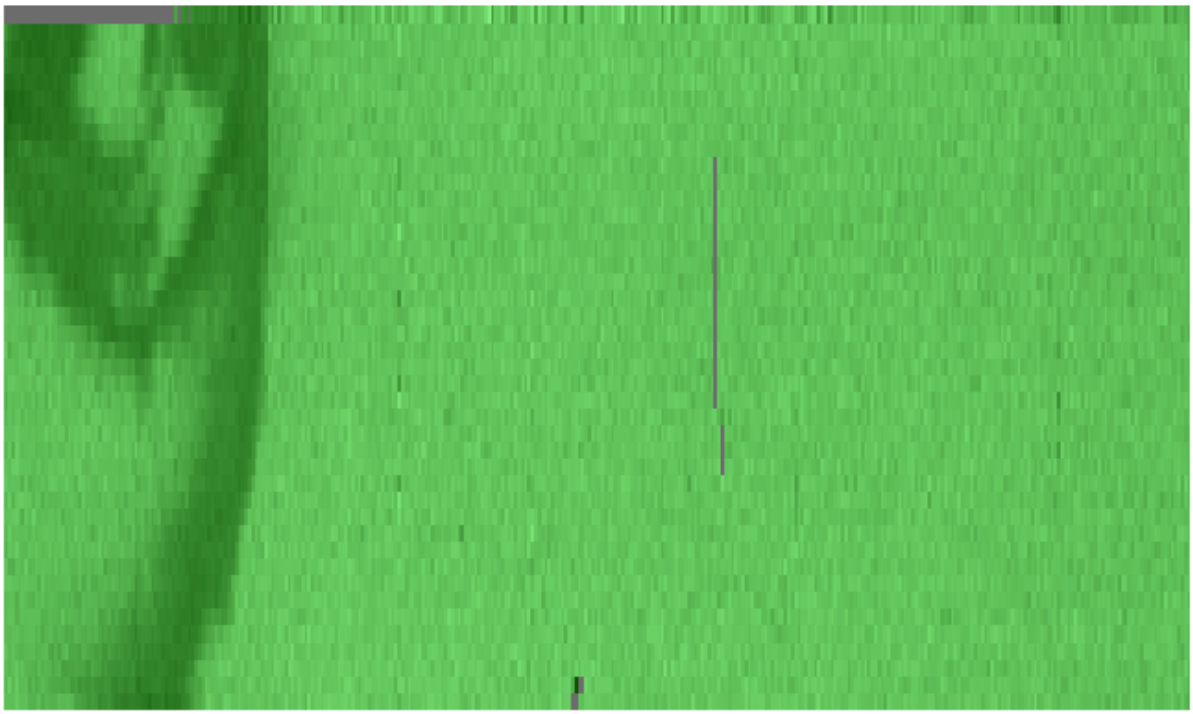
The imageplot can be configured in many ways (see manual page for more details). Sections from a BeadChip often have one edge that is much longer than the other, and it is important to recognise this when producing the plots. By default, *beadarray* makes imageplots with the longest edge on the x-axis (suitable for widescreen monitors). However, with `horizontal = FALSE`, the imageplot will be displayed in the same orientation as the original TIFF image from the directory. With the `squareSize` we can control how many pixels from the original image make up the pixels in the resulting imageplot. The following code produces imageplots for all array-sections in the example dataset. Note that we also change the colour scheme to represent low and high intensities by light and dark green respectively.

If `.locs` information is available to *beadarray*, it will be able to determine the optimal `squareSize` parameter. If not (as with our example dataset), the user may have to experiment with different values for `squareSize`.

```
imageplot(exampleBLData, array=1, low="lightgreen", high="darkgreen")
```



```
imageplot(exampleBLData, array=2, low="lightgreen", high="darkgreen")
```



7 BASH

BASH is a method for managing the spatial artefacts that may be found on an array as described in Cairns et al (2008). BASH uses the methodology developed for the Harshlight package, but altered to exploit the availability of replicated observations on the same array. The algorithm first identifies Extended defects, where an array has gradual but significant shifts across the surface. BASH also seeks to find more localized artifacts on arrays by classifying features that have unusual intensities as outliers and then finding outliers close to each other on the array. Two separate algorithms then search for areas with a larger numbers of outliers than would be expected by chance (Diffuse Defects) and large connected clusters of outliers (Compact defects). The random nature (both in position and numbers of each feature type) of Illumina arrays mean that the Harshlight algorithm must proceed in a different way to the original Harshlight implementation. Whereas Affymetrix probes have replicates on other arrays, Illumina beads are replicated on the same array. We can therefore generate an error image based on how much each bead differs from the median of its replicates' intensities, instead of replicates on other arrays. Having performed manipulations to the error image, we can then find outliers on this image by bead type, determining which beads are more than 3 Median Absolute Devations, or MADs, from the median.

Finally, since Illumina arrays are randomly arranged and use a hexagonal grid rather than rectangular, BASH has it's own method for creating networks of beads on the array. However, if `.locs` files are available to *beadarray* the time taken for this step will be improved considerably.

The following command can be used to run BASH with the default settings

```
bsh = BASH(exampleBLData, array=1:2)
```

The weights and QC can be stored using `setWeights` and `insertSectionData`.

```
for(i in 1:2){

        BLData <- setWeights(exampleBLData, wts=bsh$wts[[i]], array=i)

}

BLData <- insertSectionData(exampleBLData, what="BASHQC", data = bsh$QC)
```

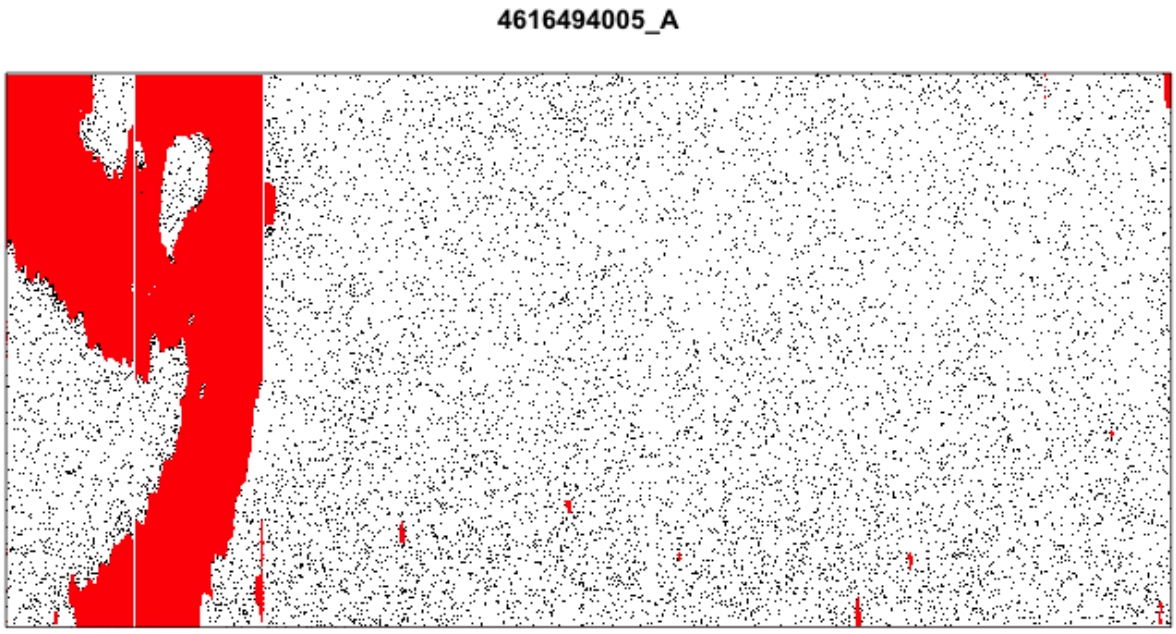
We have already saved the weights into the `exampleBLData` object and they can be retrieved in the following way. A weight of zero meaning that the bead will be excluded from an outlier calculations or summarisation procedures.

```
table(getBeadData(exampleBLData, array=1, what="wts"))

##
##       0       1
##   13380 1074989

table(getBeadData(exampleBLData, array=2, what="wts"))

##
```

```
##      0      1
## 143923 956850
```

Before combining the observations for each bead-type on an array, Illumina remove any observations with outlying intensity (more than 3 median absolute deviations from the median). This step can be repeated in *beadarray* and can be adjusted so that other outlier removal schemes can be run. It is useful to see where these outliers are located on the array surface. Often, they will coincide with beads masked by BASH or with any spatial artefacts that may be seen. The locations of beads that have been masked by BASH can be visualised using the `showArrayMask` function.

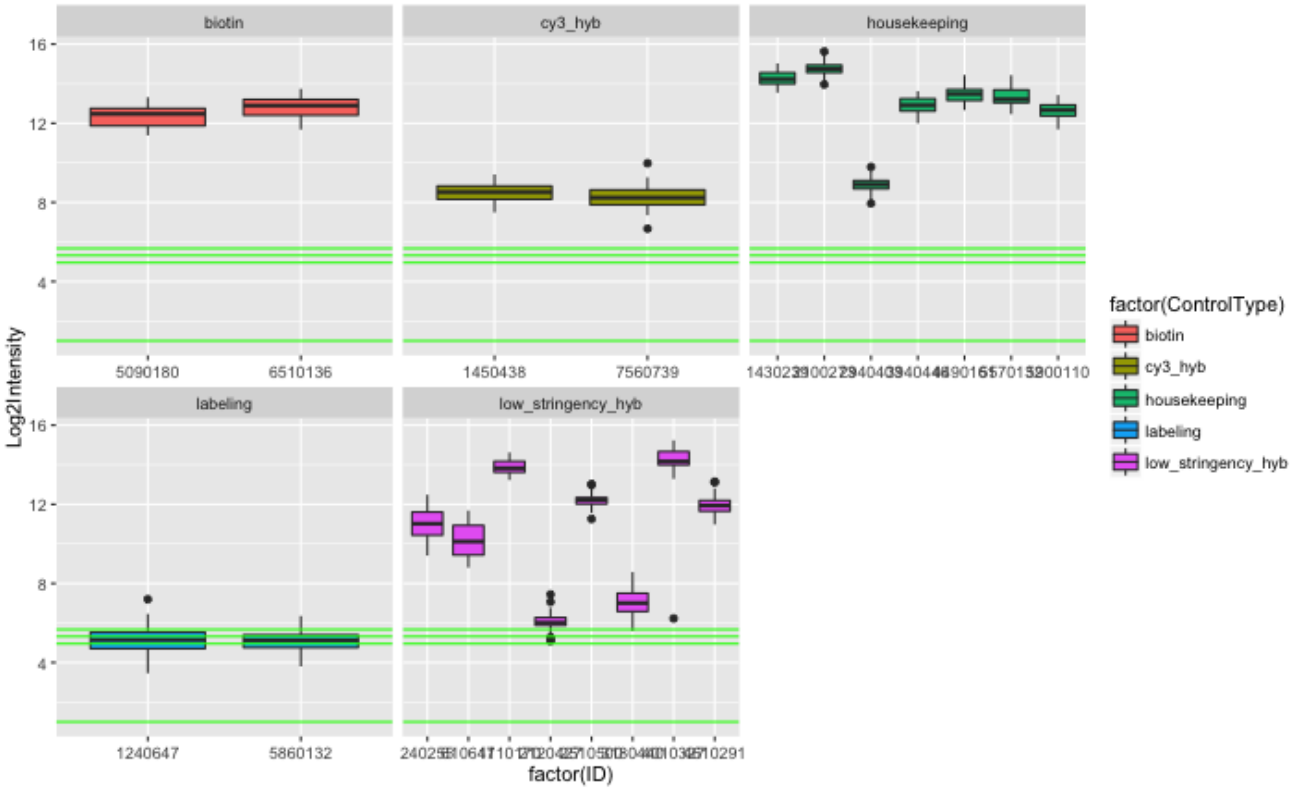
```
showArrayMask(exampleBLData, array=2)
```



8 Using control information

Illumina have designed a number of control probes for each expression platform. Two particular controls on expression arrays are housekeeping and biotin controls. With the `poscontPlot` function, we can plot the intensities of any ArrayAddressIDs that are annotated as belonging to the Housekeeping or Biotin group in the `ExpressionControlData` object. The mapping of controls to ArrayAddressID is possible by having the *illuminaHumanv3.db* package installed and setting the annotation of the object accordingly.

```
p <- combinedControlPlot(exampleBLData)
```



9 Summarization

The summarization procedure takes the BLData object, where each bead-type is represented by differing numbers of observations on each array, and produces a summarized object to make comparisons between arrays. For each array section represented in the BLData object, all observations are extracted, transformed, and then grouped together according to their ArrayAddressID. Outliers are removed and the mean and standard deviation of the remaining beads are calculated.

The *illuminaChannel* class is used to define how summarization proceeds with specification of a transformation function, a function to remove outliers and function to calculate the means and standard deviation. The default options to summarize apply a log2 transformation, remove outliers using the Illumina 3 M.A.D cut-off, and report the mean and standard deviation for each bead type.

```
BSData <- summarize(exampleBLData)

## No sample factor specified. Summarizing each section separately
## Finding list of unique probes in beadLevelData
## 49895  unique probeIDs found
## Number of unmapped probes removed:  319
## Summarizing  G  channel
```

```
## Processing Array 1
## Removing outliers
## Using exprFun
## Using varFun
## Summarizing  G  channel
## Processing Array 2
## Removing outliers
## Using exprFun
## Using varFun
## Making  summary object
```

The code below creates a different summarized object; one which reports median and standard errors and does not log transform the data.

```
myMedian <- function(x) median(x, na.rm=TRUE)
myMad <- function(x) mad(x, na.rm=TRUE)

greenChannel2 <- new("illuminaChannel", greenChannelTransform, illuminaOutlierMethod,
myMedian, myMad,"G")

BSData2 <- summarize(exampleBLData, list(greenChannel2))
```

The `BSData` object is very similar to the *ExpressionSet* class in Biobase. However, to accommodate the unique features of Illumina data we have added an `nObservations` slot, which gives the number of beads that we used to create the summary values for each bead-type on each array after outlier removal.

```
BSData
```

```
## ExpressionSetIllumina (storageMode: list)
## assayData: 49576 features, 2 samples
##   element names: exprs, se.exprs, nObservations
## protocolData: none
## phenoData
##   rowNames: 4613710017_B 4616494005_A
##   varLabels: sampleID SampleFac
##   varMetadata: labelDescription
## featureData
##   featureNames: ILMN_1802380 ILMN_1893287 ... ILMN_1846115 (49576
##     total)
##   fvarLabels: ArrayAddressID IlluminaID Status
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation: Humanv3
## QC Information
##  Available Slots:
```

```
##   QC Items: Date, Matrix, ..., SampleGroup, numBeads
##   sampleNames: 4613710017_B, 4616494005_A
```

It is possible to have multiple channels, each of which is summarized in a different manner, in the same `ExpressionSetIllumina` object. This is achieved by passing a list of illuminaChannel objects to `summarize`. This would be especially useful for two-channel data, where the Red and Green channels, and some combination of the two would be of interest in the analysis.

The detection score is a standard measure for Illumina expression experiments, and can be viewed as an empirical estimate of the p-value for the null hypothesis that there is no expression. These can be calculated for summarized data provided that the identity of the negative controls on the array is known. For further analysis of the summarized object, see the separate beadsummary vignette `beadsummary.pdf`.

```
det = calculateDetection(BSData)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
head(det)
```

```
##              4613710017_B 4616494005_A
## ILMN_1802380   0.00000000   0.00000000
## ILMN_1893287   0.27309237   0.43658211
## ILMN_1736104   0.55555556   0.73564753
## ILMN_1792389   0.00000000   0.00000000
## ILMN_1854015   0.05756359   0.01869159
## ILMN_1904757   0.21686747   0.40987984
```

```
Detection(BSData) <- det
```

```
sessionInfo()
```

```
## R version 3.2.3 (2015-12-10)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
```

```
##  [1] illuminaHumanv3.db_1.26.0  org.Hs.eg.db_3.2.3
##  [3] RSQLite_1.0.0              DBI_0.3.1
##  [5] AnnotationDbi_1.32.3       IRanges_2.4.6
##  [7] S4Vectors_0.8.9            beadarrayExampleData_1.8.0
##  [9] beadarray_2.20.1           ggplot2_2.0.0
## [11] Biobase_2.30.0             BiocGenerics_0.16.1
## [13] knitr_1.12
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.3          XVector_0.10.0       magrittr_1.5
##  [4] zlibbioc_1.16.0      GenomicRanges_1.22.3 munsell_0.4.2
##  [7] colorspace_1.2-6     highr_0.5.1          stringr_1.0.0
## [10] plyr_1.8.3           GenomeInfoDb_1.6.1   tools_3.2.3
## [13] base64_1.1           grid_3.2.3           gtable_0.1.2
## [16] digest_0.6.9         reshape2_1.4.1       formatR_1.2.1
## [19] evaluate_0.8         labeling_0.3         limma_3.26.5
## [22] stringi_1.0-1        BeadDataPackR_1.22.0 scales_0.3.0
## [25] BiocStyle_1.8.0      illuminaio_0.12.0
```