1 Analysis of Bead Summary Data using beadarray

There are two methods for describing the results of a BeadArray experiment. Firstly, we can use *bead-level data* whereby the position and intensity of each individual bead on an array is known. The methods available for processing bead level data are discussed in a seperate vignette.

Bead summary data can also be used whereby a summary intensity for each bead type on an array is given. The summarised values for a particular bead type can then be compared between different arrays within an experiment.

Whilst the *beadarray* package includes methods for processing data of both kinds, bead summary data is far more widely available at the present time. As such, the methods described within this document focus exclusivly on dealing with bead summary data. The bead summary data can be data obtained using either the BeadChip (6 or 8 arrays on a slide) or SAM (arrays organised in 96 well plates) technologies. This document uses a SAM experiment as an example although BeadChip can be read in the same manner.

2 Citing beadarray

For description of bead-level analysis and pre-processing use:

Dunning,M.J et al, *Quality Control and Low-level Statistical Analysis of Illumina Beadarrays*, Revstat **4**, 1-30.

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

Dunning M, Smith M, Thorne N, Tavaré, *beadarray: An R package to Analyse Illumina BeadArrays*, R News, submitted

3 Reading and analysing bead summary data

Bead summary data produced by BeadStudio (Illumina's application for reading raw BeadArray data) can be read directly into our library. The function used is `readBeadSummaryData` and requires a set of text files as an input. Each text file can describe the bead summary data for a particular array in the experiment, or it may describe all arrays. Example files are provided at the following URL with each file describing a single array from a SAM experiment

www.damtp.cam.ac.uk/user/jcm68/beadarray.html

Once the example files have been downloaded they can be read into R. By default, the files are read from the current working directory in R, but this can be changed by setting the *path* parameter. The `targets` object is used to define a vector of file names to be read and is created by reading the beadSummaryTargets.txt file (see example file). For experiments with a large number of arrays, it may be inconvenient to create the text file. Therefore, the `readBeadSummaryData` is able to read all files in the working directory if the *targets* parameter is omitted. By default, the function looks for each of the column headings as they are listed below. It is possible to use alternative names for headings (eg nobeads instead of NoBeads). For more details on this please see the appropriate help file.

- TargetID - an identifier for each bead type (probe type) on the array
- AVG_Signal - Summary intensity produced by averaging bead intensities of all beads of a particular type
- BEAD_STDEV - Standard deviation of all beads of a particular type, outliers excluded
- Avg_NBEADS - Number of beads used to produce average
- Detection - Average detection score for each bead type

These column names are the default names given to files produced by BeadStudio. Any other column names of interest can be read by Usage of `readBeadSummaryTargets` is as follows:

```
> targets = readBeadSummaryTargets()
> BSData = readBeadSummaryData(targets, sep = "\t")

Reading file  plate1_1
Reading file  plate1_1
Reading file  plate1_2
Reading file  plate1_3
Reading file  plate1_4
```

The resulting object, a *BeadSummaryList*, is a list based object with the following sublists:

- R - averaged foreground intensities
- ProbeID - unique identifer for the probe
- BeadStDev - standard deviation of all beads of a particular type
- Nobeads - number of beads used to produce average
- Detection - average detection score for each bead type

The first entries in a *BeadSummaryList* are all matrices with the each row corresponding to a particular bead type and the columns to individual arrays.

A common cause of error when reading files is for the column names found in the files to not match the headings that R expects to find. The columns parameter is used to change which column headings to look for in the input file.

In the example bead summary file, each file gives data for a seperate array in a experiment. It is also possible to read files containing data for more than one array using `readBeadSummaryData`. Files of this type are assumed to have a number of rows equal to the number of bead types in the experiment (eg 1500 for SAM or 24,000 for BeadChip) and the same columns for each array. The column headings are assumed to be of the form AVG_Signal-1, AVG_Signal-2,...AVG_Signal-n for n arrays. This is the standard output produced by BeadStudio. See below for a screenshot of an exampe file containing two arrays.

	A	B	C	D	E	F	G	H	I
1	TargetID	AVG_Signal-1	BEAD_STDEV-1	Detection-1	Avg_NBEADS-1	AVG_Signal-2	BEAD_STDEV-2	Detection-2	Avg_NBEADS-2
2	GI_10047089-S	106.4	4.5	0.33644494	35	80.9	3.3	0.95437491	39
3	GI_10047091-S	190.5	10.4	0.99998383	38	130.8	7.5	1	47
4	GI_10047093-S	675	33.4	1	37	546.8	21.6	1	42
5	GI_10047097-S	432.6	16.9	1	43	323	14.1	1	44
6	GI_10047099-S	664	19.3	1	30	710.2	25.8	1	35
7	GI_10047103-S	3108.8	71.8	1	38	2214.1	76.7	1	41
8	GI_10047105-S	146.9	11.7	0.96259936	30	85.5	7	0.98882627	48
9	GI_10047115-S	1945.5	77.8	1	49	1436.9	47.6	1	59
10	GI_10047117-S	140.8	8	0.92632108	52	109.5	6.3	0.99999996	45
11	GI_10047121-S	105.5	5.2	0.3185542	32	63.9	5	0.31269067	38
12	GI_10047123-S	418.9	15.8	1	50	278.5	9.4	1	43
13	GI_10047133-A	158.8	5.7	0.99239624	40	97.2	5.9	0.99991986	40
14	GI_10047133-I	129.5	10.5	0.79732306	38	78.3	4.2	0.91234824	36
15	GI_10048402-S	88.8	5.4	0.08345505	51	70.9	5.3	0.660507	46
16	GI_10092578-S	113.1	7	0.47603397	38	73.9	5.6	0.78504785	58
17	GI_10092585-S	161.1	9.2	0.99473016	39	93.5	9.7	0.99952363	39
18	GI_10092596-S	180.8	7.3	0.99985835	34	126.9	6.6	1	38
19	GI_10092602-S	155.9	9.8	0.98839642	44	85.6	5.4	0.98899854	26
20	GI_10092603-S	144.1	7.7	0.94810805	44	74.6	5.4	0.81219389	39
21	GI_10092611-A	270.3	7.2	1	45	152.6	7.7	1	46
22	GI_10092616-S	235.9	11.2	1	59	149.1	7.9	1	63
23	GI_10092618-S	685.5	25.6	1	51	590.3	19.2	1	41

All functions described from now on use a *BeadSummaryList* object as a parameter. This object can either be created using the steps described above, or can be created from bead-level data by using the `createBeadSummaryData` function (see Dunning et al). An example *BeadSummaryList* object is provided with this library and can be loaded at any time using the command.

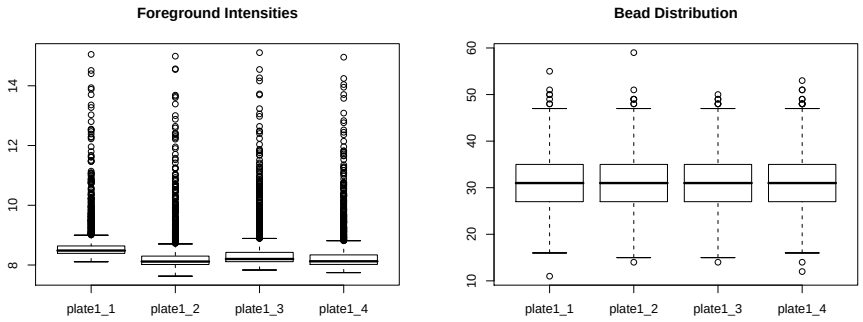
```
> data(BSData)

> names(BSData)

[1] "R"         "Nobeads"   "Detection" "BeadStDev" "ProbeID"   "targets"
```

We can use the detection score as a preliminary indicator of the quality of each array. Using the following commands we can construct a boxplot of the intensity for each probe on an array. This may reveal systematic differences between arrays, although in this four array example it is not very informative. Boxplots of the average number of beads for each bead type can also be constructed.

```
> par(mfrow = c(1, 2))
> boxplot(log2(BSData$R), main = "Foreground Intensities")
> boxplot(BSData$Nobeads, main = "Bead Distribution")
```



4 Plotting Values Across Whole SAMs and Use of Control Information

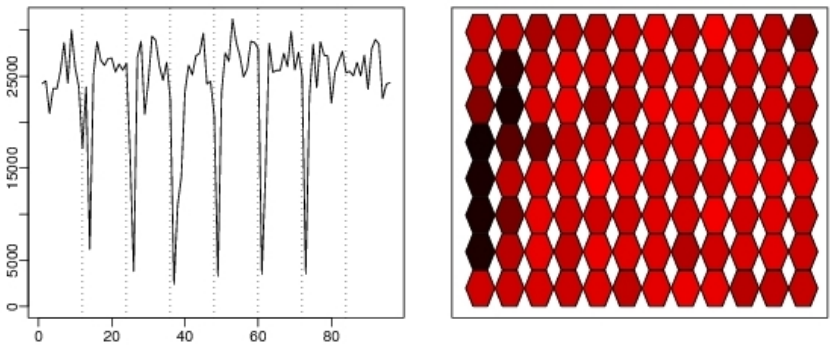
The `plotProbeVariation` function can be used to plot the variation in a particular probe / bead type on all arrays in an experiment. The input to the function is simply an *BeadSummaryList* object, and a ProbeID. The result is not very interesting for this data set as we only have 4 arrays.

```
> plotProbeVariation(BSData, ProbeID = 2)
```

We have provided a SAM summary plot for when we want to compare the intensity of a particular bead type across all arrays on one SAM. In its most simple form we use `plotOnSAM(v)` where `v` is simply a vector of numeric values with length 96. To create the vector `v` we could make use of the function `getMeanIntensities(BSData, probe)` which will return the mean intensity of beads with probeID on each of the arrays in the experiment. For instance probeID 4279 is a housekeeping control in the example experiment, so we can do.

```
> getMeanIntensities(BSData, 4279)
```

As an example we show the variation in a hybridisation control across 96 arrays



On the left-hand plot we have array index on the x axis and on the y axis we have the corresponding value of v for each of the 96 arrays. Instantly we can see the numbers of arrays for which the value of `v` is lower.

On the right-hand plot we relate the vector `v` to the position of each array on the SAM. The arrays are numbered from 1 in top-left corner to 96 in the bottom-right corner. The colour of each hexagon is related to its value of `v`, the higher this value the brighter the shade of red (a greyscale version of the plot can also be made).

In the figure above we can see that the line in the left hand plot is very erratic and the colour of the hexagons range from black through to bright red. Both of these indicate that the values in vector `v` change greatly across the SAM. Using the right hand plot we can quickly identify which probes have the lowest intensities allowing us to easily go and investigate the possible reasons. The BeadStudio application provided by Illumina is able to produce the plot seen in the left panel of the `plotOnSAM` function output, but we feel that our method is more flexible. With our function we can plot values of any probe (not just

controls) and can plot intensities on both raw and logged scale. We can also see whereabouts any potential problem arrays are located on the SAM. In the examples above we found the values of a particular bead type across all arrays and used as input to `plotOnSAM`. This plotting function is flexible because it allows any vector of numeric values with length 96 as input. For instance we could also use the number of outliers on each of the 96 arrays or the number of unregistered beads as input.

5 Comparing Samples

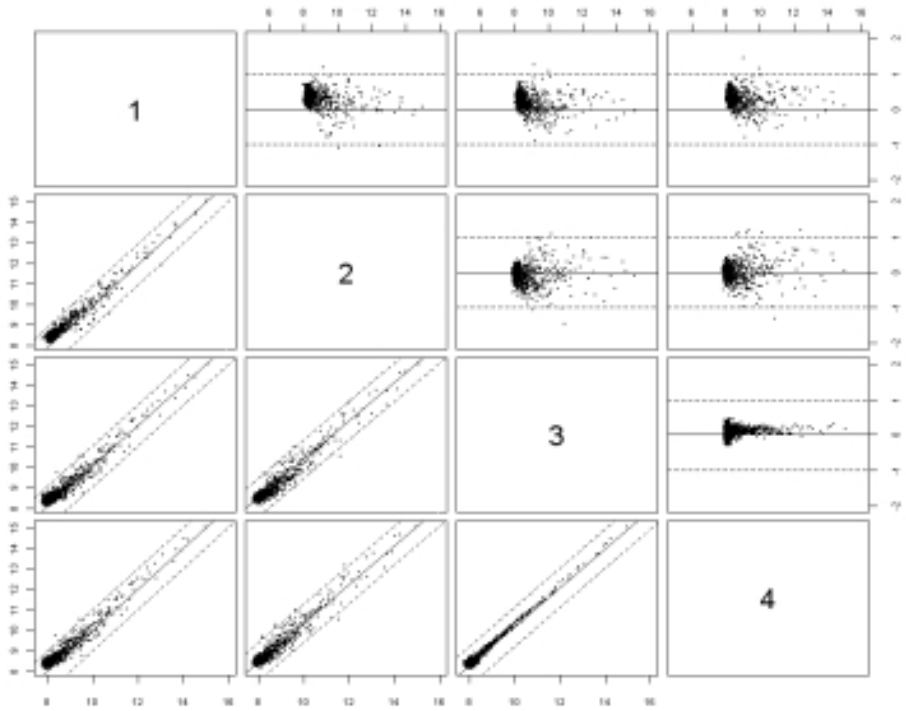
Now that all the arrays in the experiment have been averaged, we can see how particular bead types vary between different arrays or samples. We have implemented both XY plots and MA plots to achieve this and these can be viewed simultaneously for a series of arrays (the MAXY plot). In an XY plot, for a particular gene, we simply plot the value obtained from two different samples against each other with one sample on the x axis and one sample on the y axis. For an MA plot we plot the average intensity of each gene from the two different samples against the difference. For conventional microarrays, the MA plot can often reveal important differences between the two dyes used for hybridisation and give us an idea of the amount of noise generated by experiments.

The functions created for the library are capable of making comparisons between the red and green channels for the same array as well as between two arrays from a one-colour experiment. However, we did not have any two colour data when creating this document, so the following examples will be for comparing two different arrays from a one-colour experiment.

We might want to know about the variation between replicates of the same sample in an experiment. The function `pairwiseMA` is capable of producing XY plots and MA plots for a defined set of arrays with XY and MA plots shown for all pairwise comparisons between the arrays

In our experiment arrays 1,2,3 & 4 were replicates of the same sample.

```
> vec = c(1, 2, 3, 4)
> plotMAXY(BSData, vec)
```



The resulting graphic is in a 4 x 4 grid. In the first row we have MA-plots of the first array compared to arrays 2,3 and 4 and in the first column there are XY plots of the first array compared to arrays 2,3 and 4.

As we are comparing replicates of the sample we would expect to see very little variation in the plots so XY plots should be centred around the diagonal and MA plots about the horizontal

The `plotMAXY` can be made to highlight particular control types or genes of interest. This is done by defining a `genes` matrix which contains information about each gene in an experiment. To define the `genes` we require an Excel file that defines the ProbeID and name of each gene. Part of an example file is shown below and was created by adapting a bead set manifest file supplied by Illumina.

A	B	C	D	E	F	G
Search Key	Target	ProbeID	Gid	Transcript	Accession	Symbol
NM_003060	GI_24497491-S	1104	GI_244974	GI_244974	NM_00306	SLC22A5
YWHAB-001	GI_31742479-I	4153	GI_317424	GI_317424	NM_00340	YWHAB
YWHAB-001	GI_31742479-I	6016	GI_317424	GI_317424	NM_00340	YWHAB
YWHAB-001	GI_31742480-A	1588	GI_317424	GI_317424	NM_13932	YWHAB
SPINT3-001	SPINT3-001-S	4123		SPINT3-001		SPINT3-00
SPINT3-001	SPINT3-001-S	1002		SPINT3-001		SPINT3-00
MMP9-001	GI_4826835-S	3790	GI_482683	GI_482683	NM_00499	MMP9
MMP9-001	GI_4826835-S	4091	GI_482683	GI_482683	NM_00499	MMP9
NM_004414	GI_33620724-S	1250	GI_336207	GI_336207	NM_00441	DSCR1

Such a file (in this example called *my gene list.csv*) can be read into the library by using the `readProbeInfo` function which creates an extra matrix - $genes in the *BeadSummaryList* object. We read the Target, ProbeID and Symbol columns from the file.

```
> names(BSData)

[1] "R"         "Nobeads"   "Detection" "BeadStDev" "ProbeID"   "targets"

> BSData = readProbeInfo(file = "my gene list.csv", BSData, columns = c("Target",
+     "ProbeID", "Symbol"))
> names(BSData)

[1] "R"         "Nobeads"   "Detection" "BeadStDev" "ProbeID"   "targets"
[7] "genes"

> BSData$genes[1:10, ]

          Target ProbeID     Symbol
1  GI_24497491-S    1104    SLC22A5
2  GI_31742479-I    4153      YWHAB
3  GI_31742479-I    6016      YWHAB
4  GI_31742480-A    1588      YWHAB
5   SPINT3-001-S    4123 SPINT3-001
6   SPINT3-001-S    1002 SPINT3-001
7   GI_4826835-S    3790       MMP9
8   GI_4826835-S    4091       MMP9
9  GI_33620724-S    1250      DSCR1
10 GI_33620724-S    1595      DSCR1
```

We can use a separate file to define which genes we are interested in highlighting on the plots. In the screenshot below we have defined a set of housekeeping genes by their ProbeIDs and a gene with a particular symbol that we might be interested in.

	A	B	C	D	E
1	SpotType	ProbeID	Symbol	Name	Colour
2	housekeeping	751	*	*	red
3	housekeeping	1375	*	*	red
4	housekeeping	2458	*	*	red
5	housekeeping	3001	*	*	red
6	housekeeping	3087	*	*	red
7	housekeeping	3804	*	*	red
8	housekeeping	4268	*	*	red
9	housekeeping	4276	*	*	red
10	housekeeping	4278	*	*	red
11	housekeeping	4279	*	*	red
12	housekeeping	4285	*	*	red
13	housekeeping	4290	*	*	red
14	housekeeping	4295	*	*	red
15	housekeeping	4297	*	*	red
16	MY GENE	*	MPG	*	green

To set which genes we are interested in we use functions found in the *limma* package.

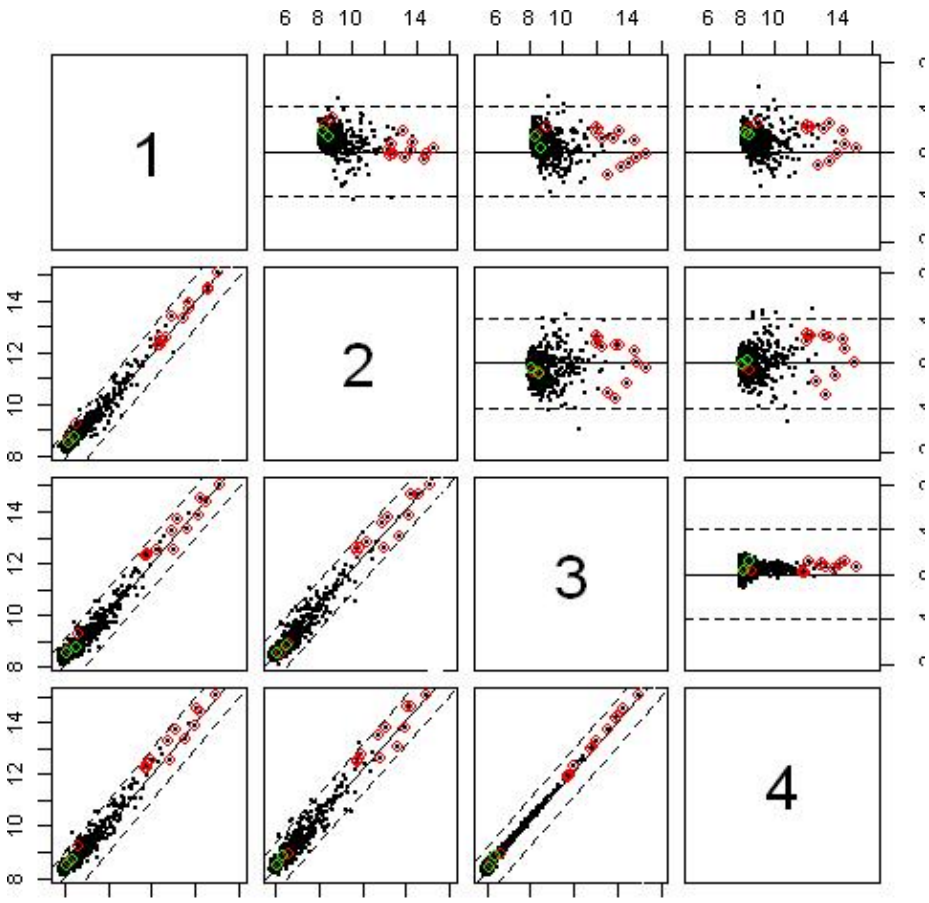
```
> types = readSpotTypes("control types.csv", sep = ",")
> BSData$genes$Status = controlStatus(types, BSData$genes)

Matching patterns for: ProbeID Symbol
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
```

```
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 1 housekeeping
Found 2 MY GENE
Setting attributes: values Name Colour
```

The output of `controlStatus` says that we have found each of the genes. Now to see the genes plotted...

```
> plotMAXY(BSData, vec = 1:4, label = TRUE)
```



The housekeeping genes are labelled in red and the other gene we specified in the file is labelled in green.

6 Normalisation Methods

The normalisation methods currently implemented in beadarray are

- `medianNormalise` - scaling the medians of arrays to make them comparable
- `quantileNormalise` - fitting each array to have the same distribution
- `qsplineNormalise` - fit a smoothing curve to each array using quantiles [?]

The `plotMAXY` function can be used to assess the performance of normalisation methods on the data by seeing if the normalisation removes any systematic trends that exist prior to normalisation. `plotMAXY` combines two functions for creating XY and MA plots. These functions can be called separately to compare two arrays in more detail.

```
> par(mfrow = c(1, 2))
> plotMA.beads(BSData, array1 = 1, array2 = 2)
> plotXY.beads(BSData, array1 = 1, array2 = 2, log = TRUE)
```

7 Further Analysis

Additional methods that can be applied to bead summary data include cluster analysis, principal components analysis and linear modelling and use existing functions from other libraries. In this section we will give a brief overview demonstrating how the object types present in the *beadarray* can be analysed.

Clustering can be achieved by using existing functions on the expression matrix of the `BSData` object. The following code can be used to cluster samples

```
> d = dist(t(BSData$R))
> plclust(hclust(d))
```

See help files for the various options available for `hclust`. The `heatmap` function could also be used

```
> heatmap(BSData$R)
```

```
> pca = princomp(d)
> biplot(pca)
```

References

[1] MJ Dunning, NP Thorne, I Camilier, M Smith, S Tavaré. Quality control and low-level statistical analysis of Illumina BeadArrays. *Revstat*, 4:1–30, 2006. http://www.ine.pt/revstat/pdf/rs060101.pdf

[2] MJ Dunning, M Smith, N Thorne, S Tavaré. beadarray: An R Package to Analyse Illumina BeadArrays. *R News*, submitted

[3] KL Gunderson, S Kruglyak, MS Graige, F Garcia, BG Kermani, C Zhao, D Che, T Dickinson, E Wickham, J Bierle, D Doucet, M Milewski, R Yang, C Siegmund, J Haas, L Zhou, A Oliphant, JB Fan, S Barnard, and MS Chee. Decoding randomly ordered DNA arrays. *Genome Res*, 14:870–877, 2004.

[4] K Kuhn, SC Baker, E Chudin, MH Lieu, S Oeser, H Bennett, P Rigault, D Barker, TK McDaniel, and MS Chee. A novel, high-performance random array platform for quantitative gene expression profiling. *Genome Res*, 14:2347–2356, 2004.

[5] M Barnes, J Freudenberg, S Thompson, B Aronow, and P Pavlidis. Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms. *Nucleic Acids Res*, 33:5914–5923, 2005.

[6] L Gautier, L Cope, BM Bolstad, and RA Irizarry. affy–analysis of Affymetrix GeneChip data at the probe level. *Bioinformatics*, 20(3):307–15, 2004.

[7] GK Smyth. Limma: linear models for microarray data. In R Gentleman, V Carey, W Huber, R Irizarry, and S Dudoit, editors, *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York, 2005.