

# Analysis of Bead Summary Data using beadarray

March 26, 2007

## Introduction

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: Dunning, M.J et al, *Quality Control and Low-level Statistical Analysis of Illumina Beadarrays*, Revstat **4**, 1-30 and in a separate vignette of beadarray.

*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. This is the format of the data output by Illumina's BeadStudio application. The methods described within this document are for the analysis of bead summary data which can be obtained using either the BeadChip (6 or 8 arrays on a slide) or SAM (arrays organised in 96 well plates) technologies.

## 1 Citing beadarray

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

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

## 2 Getting help with beadarray

Wherever possible, please send all queries about beadarray to the Bioconductor mailing list at [bioconductor@stat.math.ethz.ch](mailto:bioconductor@stat.math.ethz.ch). This will help to maintain a searchable archive of questions and responses visible to all users of the package.

## 3 Importing Bead Summary Data

The beadarray package is able to read the output of BeadStudio versions 1 and 2 which comes in the form of an Excel file. We assume the file to have one row for each probe and a set of columns for each array, depending on which columns have been exported from BeadStudio. We prefer that the annotation columns are not exported from BeadStudio. These columns often contain unusual characters which cannot be easily read into R. Annotation information can be imported easily through Bioconductor.

An example data set is included with the beadarray package and can be found as a zipped folder data directory of the beadarray download. Inside this folder you will find three Excel data files and two text files. The Excel files are the raw non-normalised data, a sample sheet and a quality control file for an example experiment. These data were obtained as part of a pilot study into BeadArray technology and comprises of 3 Human-6 BeadChips with 6 different samples, I, MC, MD, MT, P and Norm hybridised.

MC, MD, MT and P are all tumours whereas Norm is a normal sample and I is a sample provided by Illumina.

### 3.1 Description of Files

Reading bead summary data into `beadarray` requires the three files as given for this example experiment and we now describe these in more detail.

- `raw_data.csv` - This contains the raw, non-normalised bead summary values as output by BeadStudio and is readable by Excel. Inside the file are several lines of header information followed by a data matrix with some 48,000 rows. Each row is a different gene in the experiment and the columns give different measurements for the gene. For each array, we record the summarised expression level (`AVG_Signal`), standard error of the bead replicates (`BEADSTDEV`), Number of beads used (`Avg_NBEADS`) and a Detection score which estimates the probability of a gene being detected above the background. Note that whilst this data has not been normalised, it has been subjected to local background correction at the bead level prior to summarising.
- `raw_data_sample_sheet` - Defines the array IDs and samples placed on each array. In order for this information to be read into `beadarray`, we require that the 4th column is a unique identifier for each array in the experiment. This is a file format that Illumina recommend for users of BeadStudio to specify the contents of each array.
- `raw_data_qc_info` - Gives the summarised expression values for each of the controls that Illumina place on arrays and hence extremely useful for diagnostic purposes. The format of the quality control files differs slightly between BeadStudio versions 1 and 2. Version 1 of the software gives one averaged value for each control type, whereas version 2 gives summarised values for each control of a particular type. The user does not have to know the version of BeadStudio used to generate the file.

The following code can be used to read the example data into R. We first define the names of the raw data file, sample sheet and quality control information. If the quality control file and sample sheet are not available, then the raw data can be read in on it's own.

The function `readBeadSummaryData` can be made to read the output of either versions 1 and 2 of BeadStudio. Users may need to change the argument `sep`, which specifies if the file is comma or tab delimited and `skip` which specifies the number of lines of header information at the top of the file. Equivalent arguments are used to read the quality control file (`qc.skip` and `qc.sep`).

The `columns` argument is used to decide which column headings to read from the file.

```
> dataFile = "raw_data.csv"
> sampleSheet = "raw_data_sample_sheet.csv"
> qcFile = "raw_data_qcinfo.csv"
> BSData <- readBeadSummaryData(dataFile, qcFile = qcFile, sampleSheet = sampleSheet,
+   skip = 7, columns = list(exprs = "AVG_Signal", BeadStDev = "BEAD_STDEV",
+     NoBeads = "Avg_NBEADS"), qc.columns = list(exprs = "AVG.Signal",
+     BeadStDev = "SeqVAR"), qc.sep = ",", sep = ",", qc.skip = 7)
```

## 4 The BSData object

`BSData` is an object of type `ExpressionSetIllumina` which is an extension of the `ExpressionSet` class developed by the Biocore team used as a container for high-throughput assays. The data from the the `raw_data` file has been written to the `assayData` slot of the object, whereas the `phenoData` slot contains information from `sample_sheet` and the `QC` slot contains the quality control information. For consistency with the definition of other *ExpressionSet* objects, we now refer to the expression values as

the *exprs* matrix which can be accessed using `exprs` and subset in the usual manner. The `BeadStDev` matrix can be accessed using `se.exprs`. The rows of `exprs` are named according to the row names of the original `raw_data` file.

```
> BSData
```

```
ExpressionSetIllumina (storageMode: list)
```

```
assayData: 47293 features, 18 samples
```

```
  element names: exprs, BeadStDev, NoBeads, Detection, Narrays, arrayStDev, DiffScore
```

```
phenoData
```

```
  rowNames: 1, 2, ..., 18 (18 total)
```

```
  varLabels and varMetadata:
```

```
    Sample_Name: Sample_Name
```

```
    Sample_Well: Sample_Well
```

```
    ...: ...
```

```
    Sentrix_Position: Sentrix_Position
```

```
    (7 total)
```

```
featureData
```

```
  rowNames:
```

```
  varLabels and varMetadata: none
```

```
experimentData: use 'experimentData(object)'
```

```
Annotation character(0)
```

```
QC Information
```

```
  Available Slots:  exprs BeadStDev
```

```
  featureNames:  AVG.Signal.biotin, AVG.Signal.cy3_hyb_high, ..., AVG.Signal.low_stringency_hyb_pm, AVG
```

```
  sampleNames:  1475542110_F, 1475542113_E, ..., 1475542113_D, 1475542113_F
```

```
> exprs(BSData)[1:10, 1:2]
```

	IH-1	IC-1
GI_10047089-S	87.8	131.8
GI_10047091-S	161.8	130.8
GI_10047093-S	481.2	401.4
GI_10047099-S	633.7	483.8
GI_10047103-S	1535.6	1186.5
GI_10047105-S	247.5	210.2
GI_10047121-S	113.0	101.3
GI_10047123-S	453.9	306.8
GI_10047133-A	103.6	114.5
GI_10047133-I	118.0	123.1

```
> se.exprs(BSData)[1:10, 1:2]
```

	IH-1	IC-1
GI_10047089-S	5.1	9.5
GI_10047091-S	12.0	7.9
GI_10047093-S	21.7	24.5
GI_10047099-S	21.6	20.9
GI_10047103-S	42.7	34.5
GI_10047105-S	12.7	11.8
GI_10047121-S	6.4	8.1
GI_10047123-S	14.0	13.1
GI_10047133-A	6.8	6.0
GI_10047133-I	5.6	7.2

```
> pData(BSData)[, 1:6]
```

	Sample_Name	Sample_Well	Sample_Plate	Sample_Group	Pool_ID	Sentrix_ID
1	NA	NA	NA	IH-1	NA	1475542114
2	NA	NA	NA	IC-1	NA	1475542114
3	NA	NA	NA	IH-2	NA	1475542114
4	NA	NA	NA	MC-1	NA	1475542114
5	NA	NA	NA	MD-1	NA	1475542114
6	NA	NA	NA	MT-1	NA	1475542114
7	NA	NA	NA	IC-2	NA	1475542110
8	NA	NA	NA	IH-3	NA	1475542110
9	NA	NA	NA	IC-3	NA	1475542110
10	NA	NA	NA	P-3	NA	1475542110
11	NA	NA	NA	P-3	NA	1475542110
12	NA	NA	NA	Norm-1	NA	1475542110
13	NA	NA	NA	MC-2	NA	1475542113
14	NA	NA	NA	MD-2	NA	1475542113
15	NA	NA	NA	MT-2	NA	1475542113
16	NA	NA	NA	P-1	NA	1475542113
17	NA	NA	NA	Norm-2	NA	1475542113
18	NA	NA	NA	P-2	NA	1475542113

```
> QCInfo(BSData)$exprs[1:5, 1:4]
```

	1475542110_F	1475542113_E	1475542114_A	1475542114_C
AVG.Signal.biotin	7551.0	6137.2	10255.0	9358.7
AVG.Signal.cy3_hyb_high	32436.0	28081.0	41451.7	41116.9
AVG.Signal.cy3_hyb_low	816.6	739.4	1040.9	1100.0
AVG.Signal.cy3_hyb_med	11178.2	9158.1	13176.7	13109.2
AVG.Signal.gene	205.8	176.6	320.3	395.2

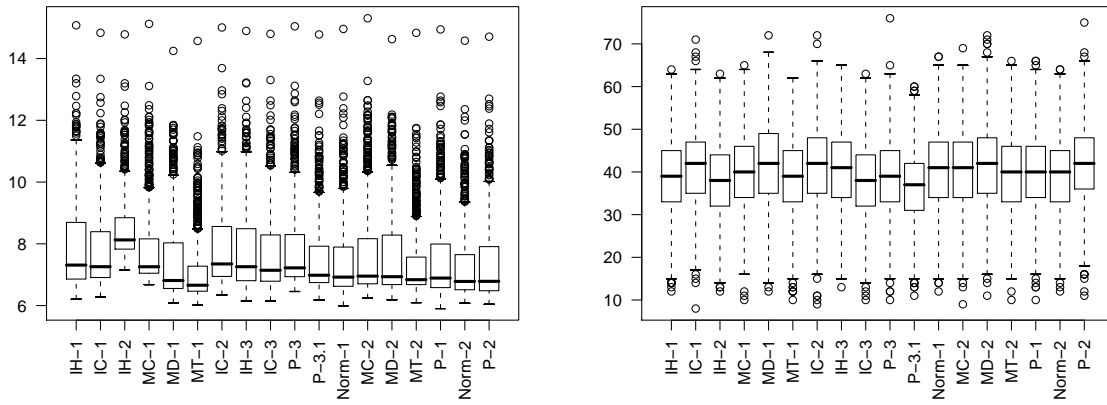
Boxplots of expression may be useful for quality control. Below we show the code to produce boxplots of the  $\log_2$  intensities of each array in the experiment. Recall that there are 6 arrays per BeadChip and that differences between chips hybridisations on different days may be expected. In this example the differences in intensity between arrays on the same chip and different chips do not seem too large. However, we can see that the first BeadChip seems to be more variable than the others and in particular the third array on the first BeadChip could be an outlier.

Boxplots of the other slots in BSData can be easily plotted. <sup>1</sup>

```
> par(mfrow = c(1, 2))
> boxplot(log2(exprs(BSData)[1:1000, ]), las = 2)
> boxplot(NoBeads(BSData)[1:1000, ], las = 2)
```

---

<sup>1</sup>We have restricted the number of points plotted in order to keep the size of this vignette small.



## 5 Normalisation and Quality Control

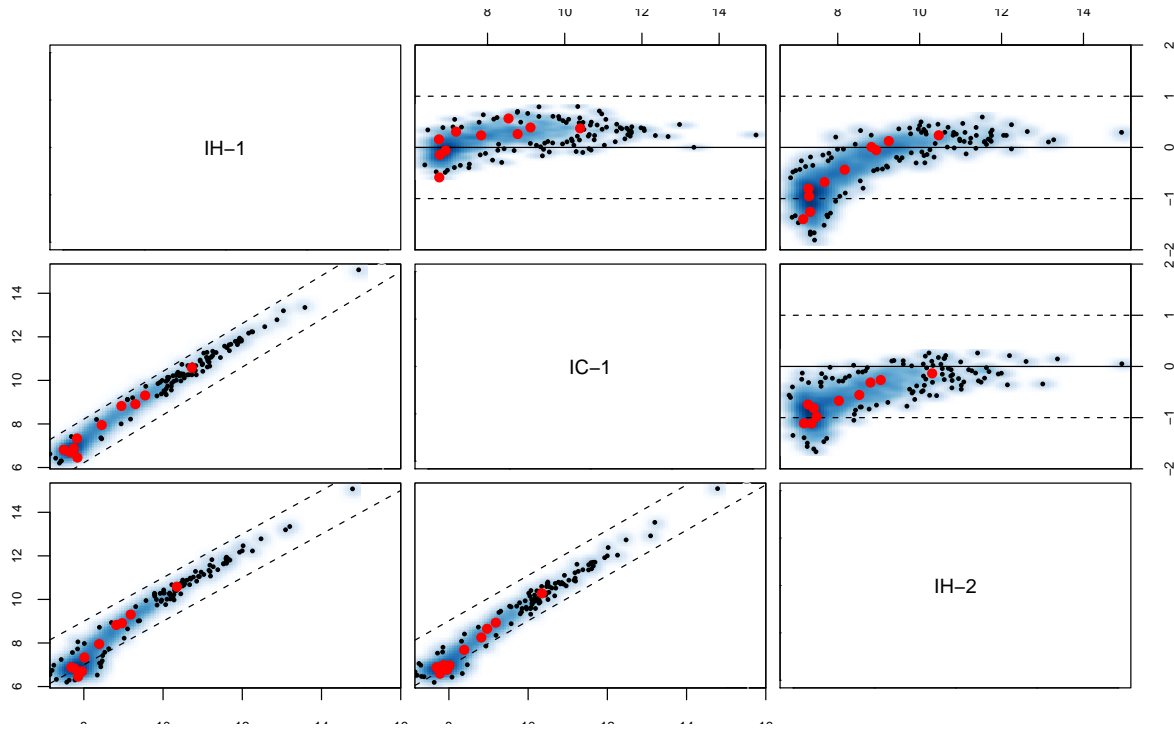
In the expression boxplots we notice that there are differences in expression level across a chip and between chips. Therefore we might want to normalise the arrays in the experiment comparable. We also see the the 3rd array has significantly different intensity. The sample on this array is replicated three times on the first chip, so comparing the MA and XY plots for the replicates of this sample can be informative.

Particular genes of interest may be highlighted on the MA and XY plots by using the `genesToLabel` argument which should match up with the row names in `BSData`. The `labelCol` argument can be used to specify a colour for each gene. For simplicity sake we simply highlight the first ten genes in the expression matrix, a possible application might be to highlight control genes on the plot or particular genes of interest.

```
> g = rownames(exprs(BSData))[1:10]
> g

[1] "GI_10047089-S" "GI_10047091-S" "GI_10047093-S" "GI_10047099-S"
[5] "GI_10047103-S" "GI_10047105-S" "GI_10047121-S" "GI_10047123-S"
[9] "GI_10047133-A" "GI_10047133-I"

> cols = rainbow(start = 0, end = 5/6, n = 10)
> plotMAXY(exprs(BSData)[1:1000, ], arrays = 1:3, genesToLabel = g,
+         labelCols = cols, labels = as.character(pData(BSData)[1:3,
+         4]), pch = 16)
```



In the top right corner we see the MA plots for all pairwise comparisons involving the 3 arrays. On an MA plot, for each gene we plot the average of the expression levels on the two arrays on the x axis and the difference in the measurements on the y axis. For replicate arrays we would expect all genes to be unchanged between the two samples and hence most points on the plot to lie along the line  $y=0$ . In the lower left corner of the MAXY plot we see the XY plot and for replicate arrays we would expect to see most points along the diagonal  $y = x$ . From this MAXY plot it is obvious that the third array is significantly different to the other replicates and requires normalisation.

Both XY and MA plots for a particular comparison of arrays are available separately using `plotXY` and `plotMA`

## 6 Using Quality Control Information

Quality control information from Illumina experiments can be exported from the BeadStudio application. This information can be read into beadarray using the `readBeadSummaryData` function or at a later time using `readQC`.

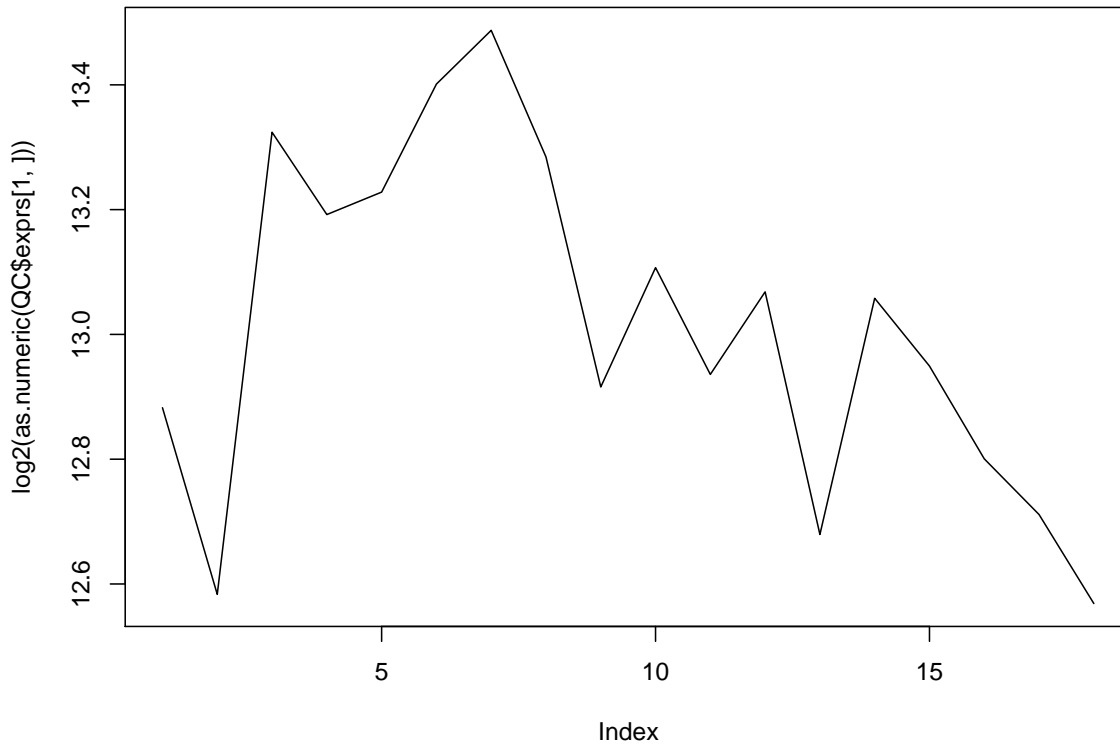
The quality control information which is read in by `readBeadSummaryData` can be plotted to provide useful diagnostic information. To retrieve this quality control data we can use the `QCInfo` function. Alternatively, quality control information can be read using `readQC`. `QC` is a list object and each item can be accessed using the `$` operator to give a matrix. The row names of the matrix gives an indication of the control types. In the following example we plot the values of the Biotin control across all arrays.

```
> QC = QCInfo(BSData)
> QC$exprs[1:3, ]
```

	1475542110_F	1475542113_E	1475542114_A	1475542114_C
AVG.Signal.biotin	7551.0	6137.2	10255.0	9358.7
AVG.Signal.cy3_hyb_high	32436.0	28081.0	41451.7	41116.9
AVG.Signal.cy3_hyb_low	816.6	739.4	1040.9	1100.0
	1475542110_B	1475542114_B	1475542110_A	1475542110_C

AVG.Signal.biotin	9595.1	10818.4	11483.9	9977.0
AVG.Signal.cy3_hyb_high	41957.6	44276.0	47204.4	45043.8
AVG.Signal.cy3_hyb_low	1017.8	1039.5	1037.6	953.8
	1475542114_D	1475542113_A	1475542114_E	1475542113_B
AVG.Signal.biotin	7727.4	8822.6	7835.9	8588.6
AVG.Signal.cy3_hyb_high	34646.3	36889.3	33330.6	40451.8
AVG.Signal.cy3_hyb_low	868.5	944.4	820.0	967.3
	1475542114_F	1475542113_C	1475542110_D	1475542110_E
AVG.Signal.biotin	6559.4	8527.5	7908.9	7134.3
AVG.Signal.cy3_hyb_high	27580.2	39325.6	33515.9	31132.7
AVG.Signal.cy3_hyb_low	697.4	949.6	901.8	797.9
	1475542113_D	1475542113_F		
AVG.Signal.biotin	6706.2	6075.0		
AVG.Signal.cy3_hyb_high	30452.9	25584.2		
AVG.Signal.cy3_hyb_low	828.7	730.8		

```
> plot(log2(as.numeric(QC$exprs[1, ])), type = "l")
```



The quality control information exported by Version 2 is somewhat different in that a summarised value is given for each control, rather than one value per control type. In this example, we import a file with just negative control information.

```
> QC2 = readQC("BeadStudioV2ControlInfo.txt", sep = "\t", skip = 0,
+ columns = list(exprs = "AVG_Signal", BeadStDev = "BEAD_STDERR",
```

```
+          controlType = "TargetID"))
> dim(QC2$exprs)
```

```
[1] 1616    6
```

```
> unique(QC2$controlType[, 1])
```

```
[1] negative
Levels: negative
```

It is possible to use the normalisation methods available in the affy such as quantile, qspline or others.

```
> library(affy)
> BSData.quantile = assayDataElementReplace(BSData, "exprs", normalize.quantiles(as.matrix(exprs(BSData))))
> BSData.qspline = assayDataElementReplace(BSData, "exprs", normalize.qspline(as.matrix(exprs(BSData))))
```

## 7 Differential Expression

Research into the best method for detecting differential expression for BeadArray data is still work in progress. In the meantime, users are able to use the `lmFit` and `eBayes` functions from `limma` on the matrix `exprs(BSData)` with a  $\log_2$  transformation applied.

The following code shows how to set up a design matrix for the example experiment combining the I, MC, MD, MT, P and Normal samples together. We then define contrasts comparing the I samples to the P samples and I to Normal and perform an empirical bayes shrinkage. In this particular experiment, the I and P samples are completely different so we would expect to see plenty of differentially expressed genes.

For more information about `lmFit` and `eBayes` please see the comprehensive `limma` documentation.

```
> design = matrix(nrow = 18, ncol = 6, 0)
> colnames(design) = c("I", "MC", "MD", "MT", "P", "Norm")
> design[which(strtrim(colnames(exprs(BSData))), 1) == "I", 1] = 1
> design[which(strtrim(colnames(exprs(BSData))), 2) == "MC", 2] = 1
> design[which(strtrim(colnames(exprs(BSData))), 2) == "MD", 3] = 1
> design[which(strtrim(colnames(exprs(BSData))), 2) == "MT", 4] = 1
> design[which(strtrim(colnames(exprs(BSData))), 1) == "P", 5] = 1
> design[which(strtrim(colnames(exprs(BSData))), 1) == "N", 6] = 1
> design
```

```
      I MC MD MT P Norm
[1,] 1  0  0  0  0   0
[2,] 1  0  0  0  0   0
[3,] 1  0  0  0  0   0
[4,] 0  1  0  0  0   0
[5,] 0  0  1  0  0   0
[6,] 0  0  0  1  0   0
[7,] 1  0  0  0  0   0
[8,] 1  0  0  0  0   0
[9,] 1  0  0  0  0   0
[10,] 0  0  0  0  1   0
[11,] 0  0  0  0  1   0
[12,] 0  0  0  0  0   1
[13,] 0  1  0  0  0   0
```



```
[14,] 0 0 1 0 0 0
[15,] 0 0 0 1 0 0
[16,] 0 0 0 0 1 0
[17,] 0 0 0 0 0 1
[18,] 0 0 0 0 1 0
```

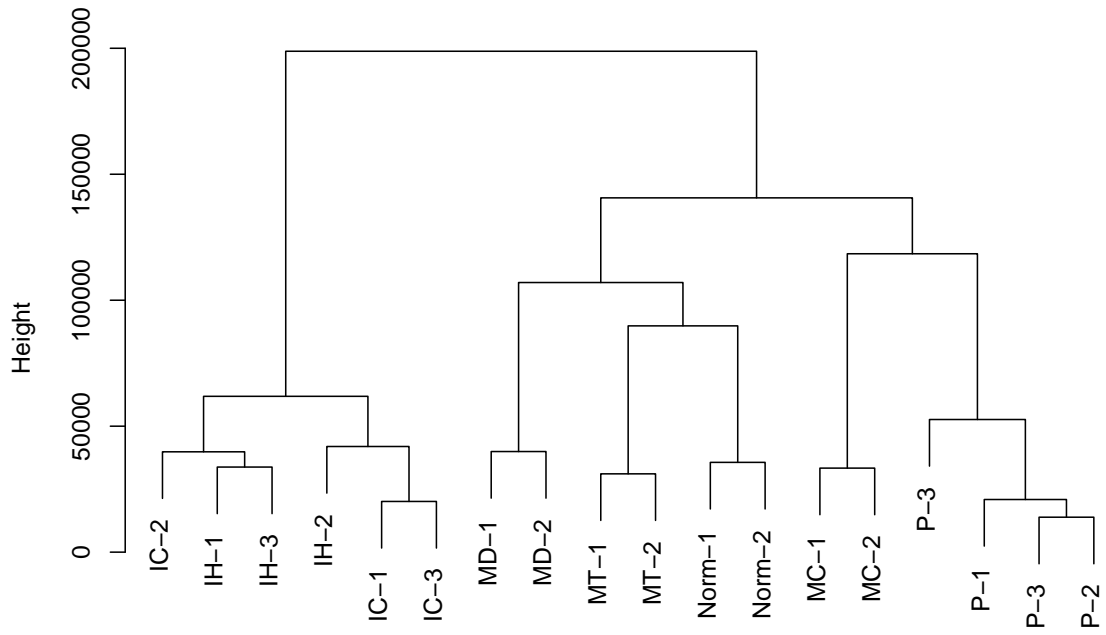
```
> fit = lmFit(log2(exprs(BSData)), design)
> cont.matrix = makeContrasts(IvsP = I - P, IvsNorm = I - Norm,
+   PvsNorm = P - Norm, levels = design)
> fit = contrasts.fit(fit, cont.matrix)
> ebFit = eBayes(fit)
> topTable(ebFit)
```

	ID	IvsP	IvsNorm	PvsNorm	F	P.Value
9259	GI_28302130-S	7.499572	7.361939	-0.137632534	937.0935	0.000000e+00
9260	GI_28302132-S	7.694362	7.544722	-0.149640427	985.6984	0.000000e+00
9258	GI_28302129-S	6.402578	6.512401	0.109823045	699.3457	1.896739e-304
24442	GI_6633805-S	6.288645	6.355043	0.066398690	671.6779	1.967835e-292
25430	GI_8392890-S	6.512861	5.830264	-0.682596858	669.7313	1.378516e-291
21840	GI_4501988-S	6.067112	5.813525	-0.253586595	604.0850	4.458776e-263
31186	Hs.449602-S	6.605590	1.619840	-4.985750409	598.4842	1.206731e-260
19978	GI_42542384-S	5.726441	5.629671	-0.096770366	546.8808	3.109321e-238
2128	GI_15149480-S	-5.669899	-5.671313	-0.001414323	542.2465	3.201302e-236
22143	GI_4503886-S	5.617491	5.751247	0.133756180	540.7956	1.366055e-235
	adj.P.Val					
9259	0.000000e+00					
9260	0.000000e+00					
9258	2.990083e-300					
24442	2.326621e-288					
25430	1.303883e-287					
21840	3.514482e-259					
31186	8.152844e-257					
19978	1.838114e-234					
2128	1.682213e-232					
22143	6.460483e-232					

## 8 Further Analysis

The clustering functionality available in BeadStudio can be easily performed through R using the `h1cust` once a distance matrix has been defined. In this example we see that the clusters correspond well to the different sample types. The `heatmap` function could also be used in a similar manner and principal components analysis is possible using `princomp`.

```
> d = dist(t(exprs(BSData)))
> p1clust(h1cust(d), labels = pData(BSData)[, 4])
```



d  
hclust (\*, "complete")