

Analysis of Bead-summary Data using beadarray

Mark Dunning

January 15, 2016

Contents

1 Introduction

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 analysis of bead summary data using beadarray. The recommended approach to obtain these data is to start with bead-level data and follow the steps illustrated in the vignette `beadlevel.pdf` distributed with [beadarray](#). If bead-level data are not available, the output of Illumina's BeadStudio or GenomeStudio can be read by [beadarray](#). Example code to do this is provided at the end of this vignette. However, the same object types are produced from either of these routes and the same functionality is available.

To make the most use of the code in this vignette, you will need to install the [beadarrayExampleData](#) and [illuminaHumanv3.db](#) packages from Bioconductor.

```
source("http://www.bioconductor.org/biocLite.R")
biocLite(c("beadarrayExampleData", "illuminaHumanv3.db"))
```

The code used to produce these example data is given in the vignette of [beadarrayExampleData](#), which follow similar steps to those described in the `beadlevel.pdf` vignette of [beadarray](#). The following commands give a basic description of the data.

```
library("beadarray")

require(beadarrayExampleData)
```

```
data(exampleSummaryData)

exampleSummaryData

## ExpressionSetIllumina (storageMode: list)
## assayData: 49576 features, 12 samples
##   element names: exprs, se.exprs, nObservations
## protocolData: none
## phenoData
##   rowNames: 4613710017_B 4613710052_B ... 4616494005_A (12 total)
##   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, 4613710052_B, ..., 4616443136_A, 4616494005_A
```

Summarized data are stored in an object of type *ExpressionSetIllumina* which is an extension of the *ExpressionSet* class developed by the Bioconductor team as a container for data from high-throughput assays. Objects of this type use a series of slots to store the data. For consistency with the definition of other *ExpressionSet* objects, we refer to the expression values as the *exprs* matrix (this stores the probe-specific average intensities) which can be accessed using *exprs* and *subset* in the usual manner. The *se.exprs* matrix, which stores the probe-specific variability can be accessed using *se.exprs*. You may notice that the expression values have already been transformed to the \log_2 scale, which is an option in the *summarize* function in *beadarray*. Data exported from BeadStudio or GenomeStudio will usually be un-transformed and on the scale 0 to 2^{16} .

```
exprs(exampleSummaryData)[1:5,1:5]

##           G:4613710017_B G:4613710052_B G:4613710054_B G:4616443079_B G:4616443093_B
## ILMN_1802380      8.454468      8.616796      8.523001      8.420796      8.527748
## ILMN_1893287      5.388161      5.419345      5.162849      5.133287      5.221987
## ILMN_1736104      5.268626      5.457679      5.012766      4.988511      5.284026
## ILMN_1792389      6.767519      7.183788      6.947624      7.168571      7.386435
## ILMN_1854015      5.556947      5.721614      5.595413      5.520391      5.558717

se.exprs(exampleSummaryData)[1:5,1:5]

##           G:4613710017_B G:4613710052_B G:4613710054_B G:4616443079_B G:4616443093_B
## ILMN_1802380      0.2833023      0.3367157      0.2750020      0.4141796      0.3581862
```

## ILMN_1893287	0.3963681	0.3882834	0.5516421	0.6761106	0.4448673
## ILMN_1736104	0.4704854	0.4951260	0.4031143	0.5276266	0.4864355
## ILMN_1792389	0.4038533	0.4728013	0.5032908	0.3447242	0.3951935
## ILMN_1854015	0.5663066	0.3783570	0.5511991	0.5358812	0.6748219

2 feature and pheno data

The `fData` and `pData` functions are useful shortcuts to find more information about the features (rows) and samples (columns) in the summary object. These annotations are created automatically whenever a bead-level data is summarized (see `beadlevel.pdf`) or read from a BeadStudio file. The `fData` will be added to later, but initially contains information on whether each probe is a control or not. In this example the `phenoData` denotes the sample group for each array; either Brain or UHRR (Universal Human Reference RNA).

```
head(fData(exampleSummaryData))
```

```
##           ArrayAddressID  IlluminaID  Status
## ILMN_1802380          10008 ILMN_1802380 regular
## ILMN_1893287          10010 ILMN_1893287 regular
## ILMN_1736104          10017 ILMN_1736104 regular
## ILMN_1792389          10019 ILMN_1792389 regular
## ILMN_1854015          10020 ILMN_1854015 regular
## ILMN_1904757          10021 ILMN_1904757 regular
```

```
table(fData(exampleSummaryData)[,"Status"])
```

```
##
##           biotin           cy3_hyb cy3_hyb,low_stringency_hyb
##                2                2                        4
##      housekeeping      labeling      low_stringency_hyb
##                7                2                        4
##      negative           regular
##                759           48796
```

```
pData(exampleSummaryData)
```

```
##           sampleID SampleFac
## 4613710017_B 4613710017_B    UHRR
## 4613710052_B 4613710052_B    UHRR
## 4613710054_B 4613710054_B    UHRR
## 4616443079_B 4616443079_B    UHRR
## 4616443093_B 4616443093_B    UHRR
## 4616443115_B 4616443115_B    UHRR
## 4616443081_B 4616443081_B    Brain
## 4616443081_H 4616443081_H    Brain
```

```
## 4616443092_B 4616443092_B      Brain
## 4616443107_A 4616443107_A      Brain
## 4616443136_A 4616443136_A      Brain
## 4616494005_A 4616494005_A      Brain
```

3 Subsetting the data

There are various way to subset an *ExpressionSetIllumina* object, each of which returns an *ExpressionSetIllumina* with the same slots, but different dimensions. When bead-level data are summarized by *beadarray* there is an option to apply different transformation options, and save the results as different channels in the resultant object. For instance, if summarizing two-colour data one might be interested in summarizing the red and green channels, or some combination of the two, separately. Both \log_2 and un-logged data are stored in the *exampleSummaryData* object and can be accessed by using the *channel* function. Both the rows and columns in the resultant *ExpressionSetIllumina* object are kept in the same order.

```
channelNames(exampleSummaryData)

## [1] "G"      "G.ul"

exampleSummaryData.log2 <- channel(exampleSummaryData, "G")
exampleSummaryData.unlogged <- channel(exampleSummaryData, "G.ul")

sampleNames(exampleSummaryData.log2)

## [1] "4613710017_B" "4613710052_B" "4613710054_B" "4616443079_B" "4616443093_B"
## [6] "4616443115_B" "4616443081_B" "4616443081_H" "4616443092_B" "4616443107_A"
## [11] "4616443136_A" "4616494005_A"

sampleNames(exampleSummaryData.unlogged)

## [1] "4613710017_B" "4613710052_B" "4613710054_B" "4616443079_B" "4616443093_B"
## [6] "4616443115_B" "4616443081_B" "4616443081_H" "4616443092_B" "4616443107_A"
## [11] "4616443136_A" "4616494005_A"

exprs(exampleSummaryData.log2)[1:10,1:3]

##           4613710017_B 4613710052_B 4613710054_B
## ILMN_1802380      8.454468      8.616796      8.523001
## ILMN_1893287      5.388161      5.419345      5.162849
## ILMN_1736104      5.268626      5.457679      5.012766
## ILMN_1792389      6.767519      7.183788      6.947624
## ILMN_1854015      5.556947      5.721614      5.595413
## ILMN_1904757      5.421553      5.320500      5.522316
## ILMN_1740305      5.417821      5.623998      5.720007
```

```
## ILMN_1665168      5.321087      5.155455      4.967601
## ILMN_2375156      5.894207      6.076418      5.638877
## ILMN_1705423      5.426463      4.806624      5.357688
```

```
exprs(exampleSummaryData.unlogged)[1:10,1:3]
```

```
##           4613710017_B 4613710052_B 4613710054_B
## ILMN_1802380      356.88235      396.46875      367.81481
## ILMN_1893287       40.85000       44.29167       38.42105
## ILMN_1736104       40.53333       46.50000       33.46154
## ILMN_1792389      112.90909      153.17647      122.65000
## ILMN_1854015       50.47059       53.26087       51.57143
## ILMN_1904757       41.45833       42.10000       49.92593
## ILMN_1740305       38.45455       51.50000       46.21429
## ILMN_1665168       42.38889       37.95000       30.46154
## ILMN_2375156       61.47368       72.73913       52.46154
## ILMN_1705423       42.38889       28.14286       38.62500
```

As we have seen, the expression matrix of the `ExpressionSetIllumina` object can be subset by column or row. In fact, the same subset operations can be performed on the `ExpressionSetIllumina` object itself. In the following code, notice how the number of samples and features changes in the output.

```
exampleSummaryData.log2[,1:4]
```

```
## ExpressionSetIllumina (storageMode: list)
## assayData: 49576 features, 4 samples
##   element names: exprs, se.exprs, nObservations
## protocolData: none
## phenoData
##   rowNames: 4613710017_B 4613710052_B 4613710054_B 4616443079_B
##   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, 4613710052_B, 4613710054_B, 4616443079_B
```

```
exampleSummaryData.log2[1:10,]
```

```
## ExpressionSetIllumina (storageMode: list)
## assayData: 10 features, 12 samples
##   element names: exprs, se.exprs, nObservations
```

```
## protocolData: none
## phenoData
##   rowNames: 4613710017_B 4613710052_B ... 4616494005_A (12 total)
##   varLabels: sampleID SampleFac
##   varMetadata: labelDescription
## featureData
##   featureNames: ILMN_1802380 ILMN_1893287 ... ILMN_1705423 (10 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, 4613710052_B, ..., 4616443136_A, 4616494005_A
```

The object can also be subset by a vector of characters which must correspond to the names of features (i.e. row names). Currently, no analogous functions is available to subset by sample.

```
randIDs <- sample(featureNames(exampleSummaryData), 1000)

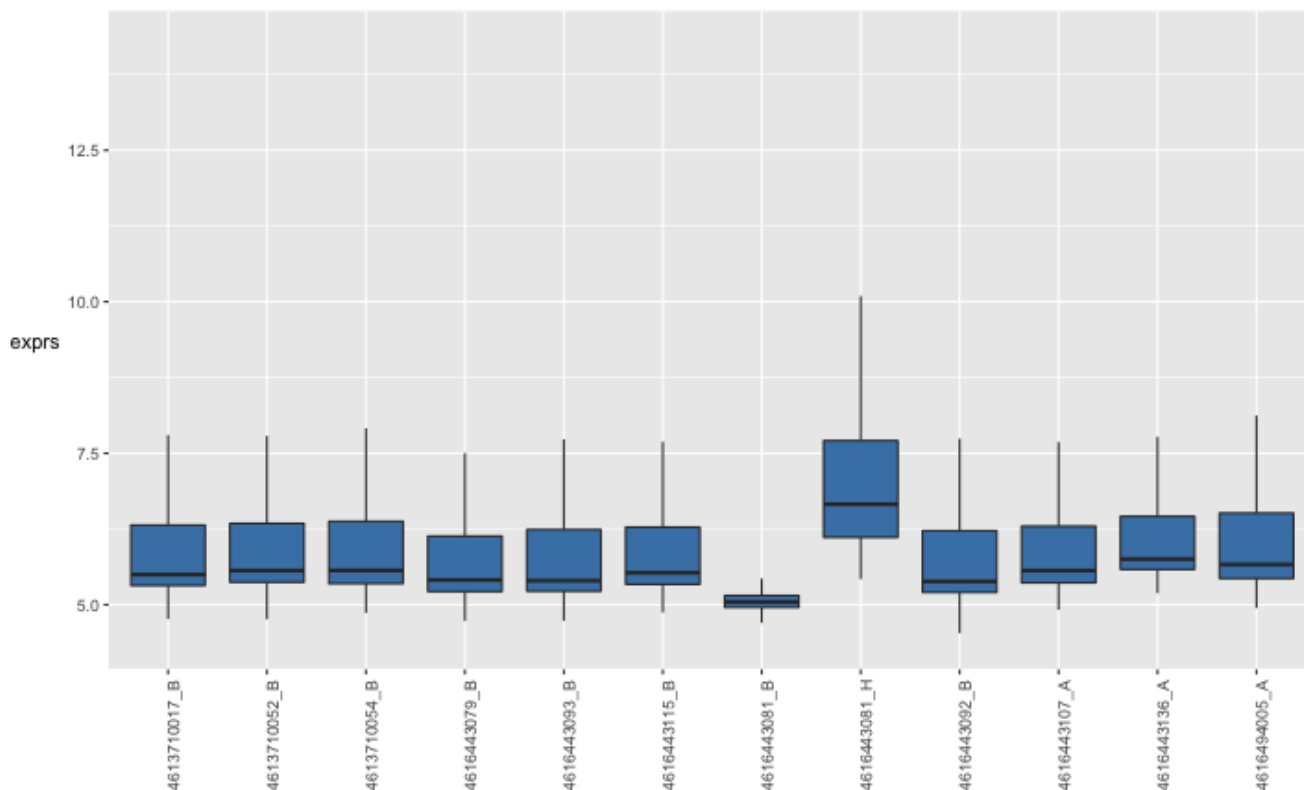
exampleSummaryData[randIDs,]

## ExpressionSetIllumina (storageMode: list)
## assayData: 1000 features, 12 samples
##   element names: exprs, se.exprs, nObservations
## protocolData: none
## phenoData
##   rowNames: 4613710017_B 4613710052_B ... 4616494005_A (12 total)
##   varLabels: sampleID SampleFac
##   varMetadata: labelDescription
## featureData
##   featureNames: ILMN_1679157 ILMN_1700240 ... ILMN_2346856 (1000 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, 4613710052_B, ..., 4616443136_A, 4616494005_A
```

4 Exploratory analysis using boxplots

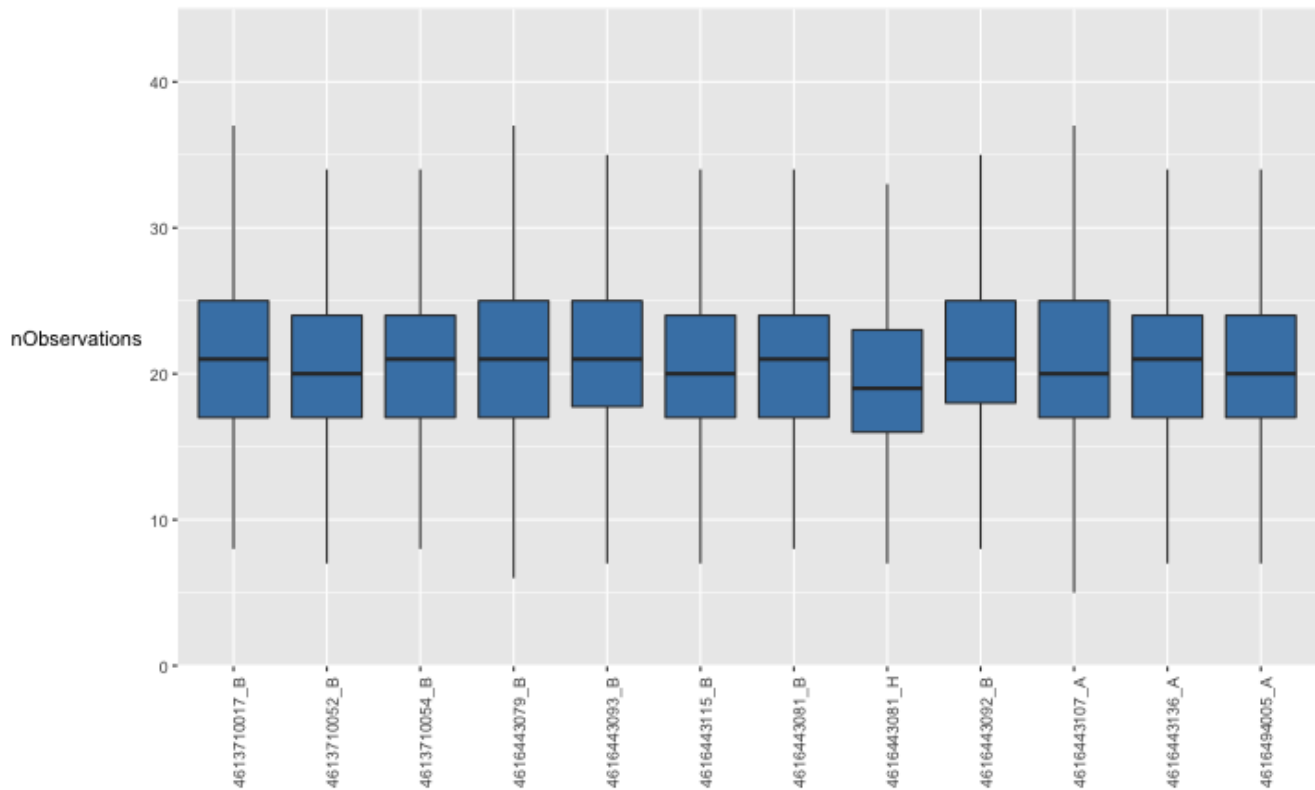
Boxplots of intensity levels and the number of beads are useful for quality assessment purposes. *beadarray* includes a modified version of the `boxplot` function that can take any valid *ExpressionSetIllumina* object and plot the expression matrix by default. For these examples we plot just a subset of the original `exampleSummaryData` object using random row IDs.

```
boxplot(exampleSummaryData.log2[randIDs,])
```



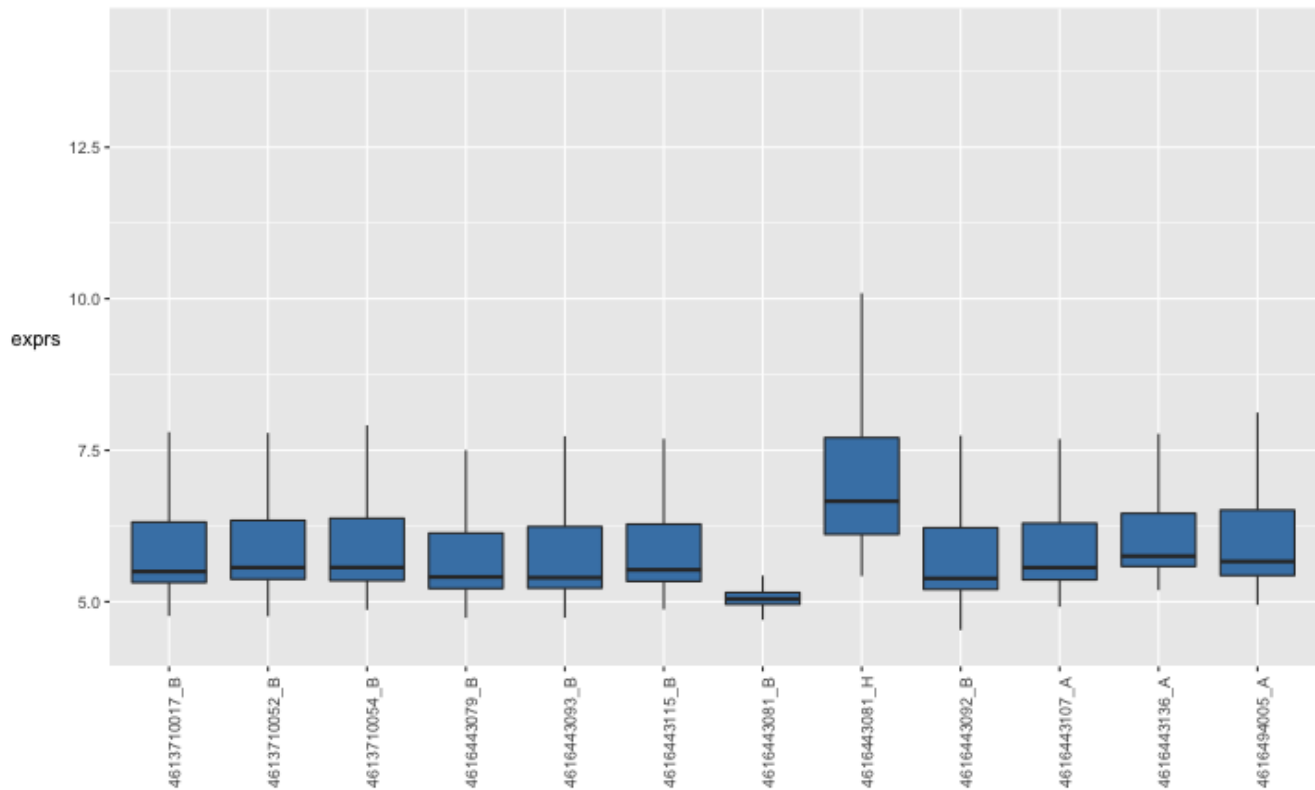
The function can also plot other `assayData` items, such as the number of observations.

```
boxplot(exampleSummaryData.log2[randIDs,], what="nObservations")
```



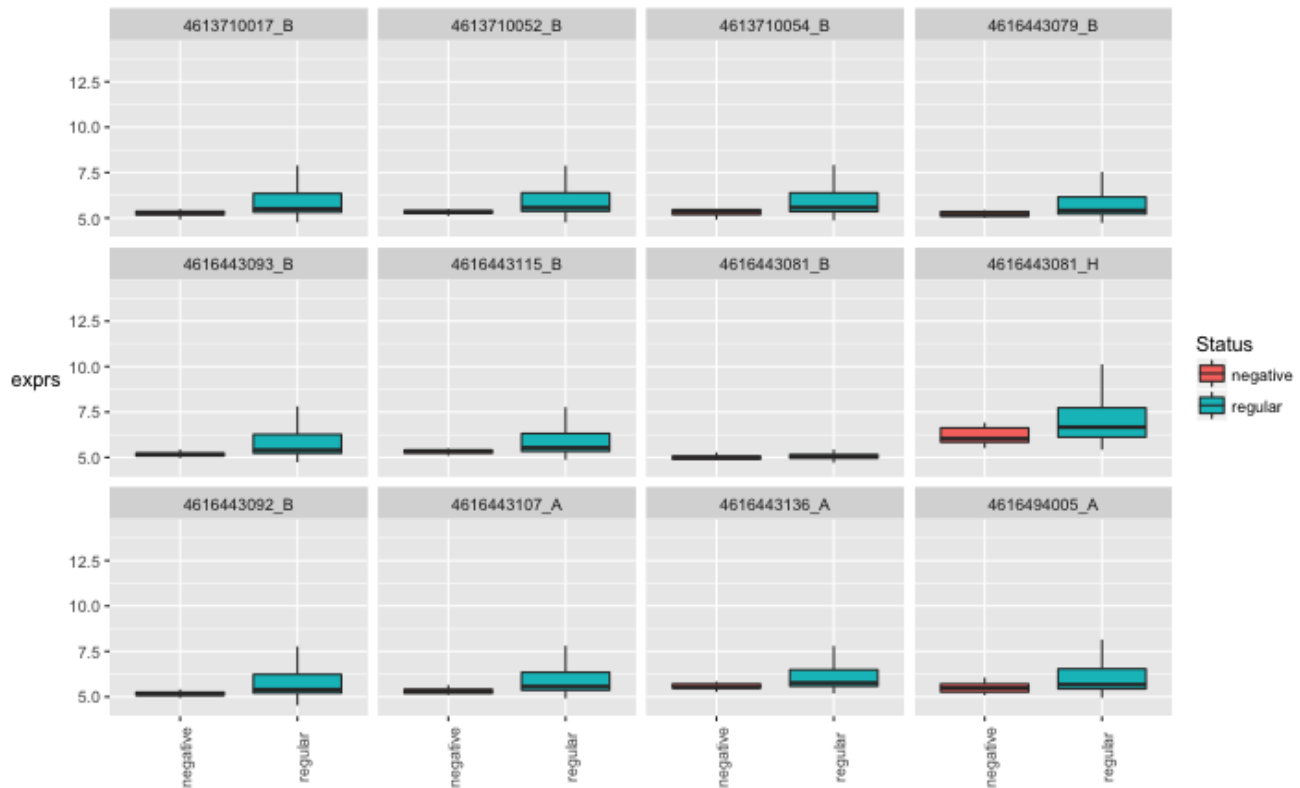
The default boxplot plots a separate box for each array, but often it is beneficial for compare expression levels between different sample groups. If this information is stored in the *phenoData* slot it can be incorporated into the plot. The following compares the overall expression level between UHRR and Brain samples.

```
boxplot(exampleSummaryData.log2[randIDs,], SampleGroup="SampleFac")
```



In a similar manner, we may wish to visualize the differences between sample groups for particular probe groups. As a simple example, we look at the difference between negative controls and regular probes for each array. You should notice that the negative controls are consistently lower (as expected) with the exception of array 4616443081_B.

```
boxplot(exampleSummaryData.log2[randIDs,], probeFactor = "Status")
```



Extra feature annotation is available from annotation packages in Bioconductor, and [beadarray](#) includes functionality to extract these data from the annotation packages. The annotation of the object must be set in order that the correct annotation package can be loaded. For example, the `exampleSummaryData` object was generated from Humanv3 data so the [illuminaHumanv3.db](#) package must be present. The `addFeatureData` function annotates all features of an `ExpressionSetIllumina` object using particular mappings from the [illuminaHumanv3.db](#) package. To see which mappings are available you can use the `illuminaHumanv3()` function, or equivalent from other packages.

```
annotation(exampleSummaryData)
## [1] "Humanv3"

exampleSummaryData.log2 <- addFeatureData(exampleSummaryData.log2,
toAdd = c("SYMBOL", "PROBEQUALITY", "CODINGZONE", "PROBESEQUENCE", "GENOMICLOCATION"))

head(fData(exampleSummaryData.log2))

##           Row.names ArrayAddressID  IlluminaID  Status SYMBOL PROBEQUALITY
## ILMN_1802380 ILMN_1802380      10008 ILMN_1802380 regular  RERE      Perfect
## ILMN_1893287 ILMN_1893287      10010 ILMN_1893287 regular  <NA>      Bad
## ILMN_1736104 ILMN_1736104      10017 ILMN_1736104 regular  <NA>      Bad
## ILMN_1792389 ILMN_1792389      10019 ILMN_1792389 regular  RNF165    Perfect
## ILMN_1854015 ILMN_1854015      10020 ILMN_1854015 regular  <NA>      Bad
## ILMN_1904757 ILMN_1904757      10021 ILMN_1904757 regular  <NA>      Perfect***
##           CODINGZONE
```

```
## ILMN_1802380 Transcriptomic GCCCTGACCTTCATGGTGTCTTTGAAGCCCAACCACTCGGTTTCCTTCGG
## ILMN_1893287 Transcriptomic? GGATTTCTTACACTCTCCACTTCTGAATGCTTGAAACACTTGCCATGCT
## ILMN_1736104 Intergenic TGCCATCTTTGCTCCACTGTGAGAGGCTGCTCACACCACCCCTACATGC
## ILMN_1792389 Transcriptomic CTGTAGCAACGTCTGTCAGGCCCCCTTGTGTTTCATCTCCTGCGCGCGTA
## ILMN_1854015 Intergenic GCAGAAAACCATGAGCTGAAATCTCTACAGGAACCAGTGCTGGGGTAGGG
## ILMN_1904757 Transcriptomic? AGCTGTACCGTG GGGAGGCTTGGTCCTCTTGCCCCATTTGTGTGATGTCT
##
## GENOMICLOCATION
## ILMN_1802380 chr1:8412758:8412807:-
## ILMN_1893287 chr9:42489407:42489456:+
## ILMN_1736104 chr3:134572184:134572223:-
## ILMN_1792389 chr18:44040244:44040293:+
## ILMN_1854015 chr3:160827837:160827885:+
## ILMN_1904757 chr3:197872267:197872316:+
```

```
illuminaHumanv3()
```

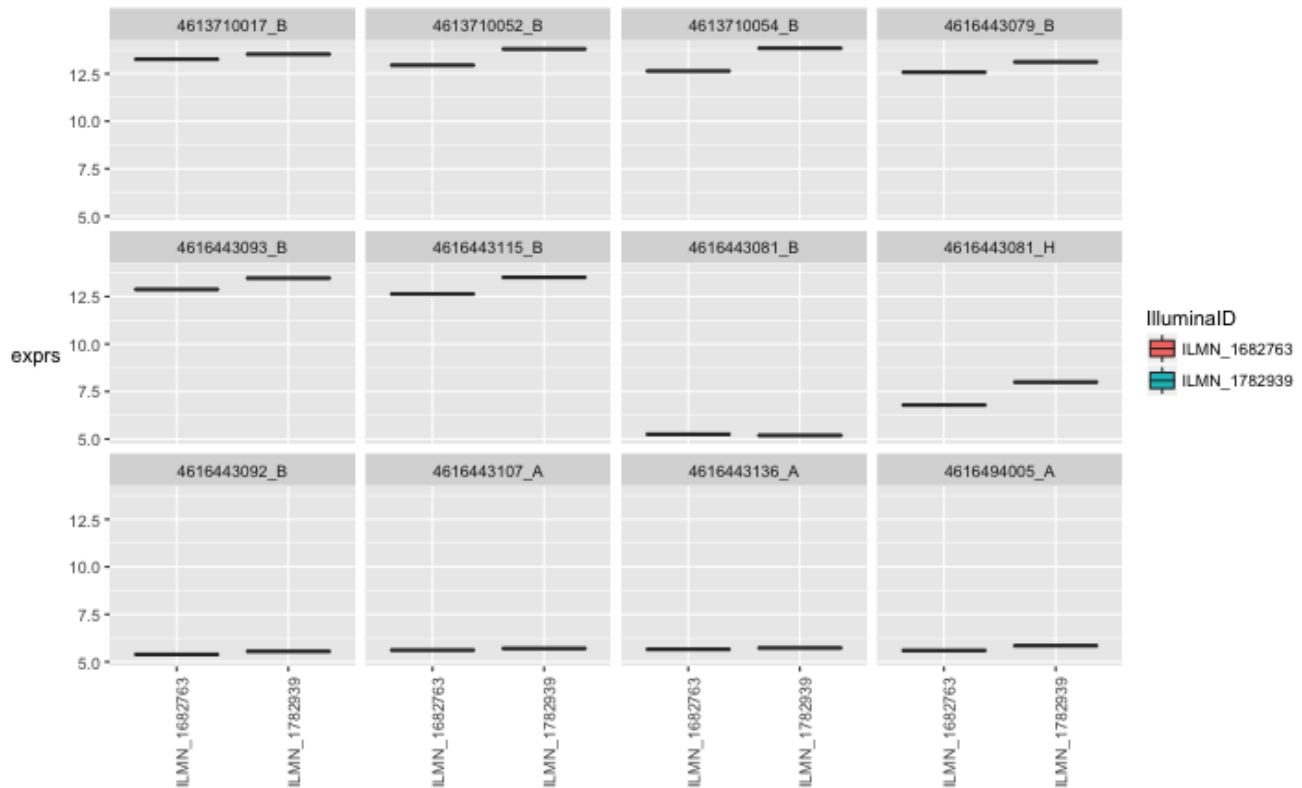
```
## #####Mappings based on RefSeqID####
## Quality control information for illuminaHumanv3:
##
##
## This package has the following mappings:
##
## illuminaHumanv3ACCNUM has 31857 mapped keys (of 49576 keys)
## illuminaHumanv3ALIAS2PROBE has 63289 mapped keys (of 118097 keys)
## illuminaHumanv3CHR has 29550 mapped keys (of 49576 keys)
## illuminaHumanv3CHRLNGTHS has 93 mapped keys (of 93 keys)
## illuminaHumanv3CHRLOC has 29354 mapped keys (of 49576 keys)
## illuminaHumanv3CHRLOCEND has 29354 mapped keys (of 49576 keys)
## illuminaHumanv3ENSEMBL has 29154 mapped keys (of 49576 keys)
## illuminaHumanv3ENSEMBL2PROBE has 20997 mapped keys (of 30486 keys)
## illuminaHumanv3ENTREZID has 29551 mapped keys (of 49576 keys)
## illuminaHumanv3ENZYME has 3526 mapped keys (of 49576 keys)
## illuminaHumanv3ENZYME2PROBE has 967 mapped keys (of 975 keys)
## illuminaHumanv3GENENAME has 29551 mapped keys (of 49576 keys)
## illuminaHumanv3GO has 26989 mapped keys (of 49576 keys)
## illuminaHumanv3GO2ALLPROBES has 19471 mapped keys (of 20246 keys)
## illuminaHumanv3GO2PROBE has 15200 mapped keys (of 15895 keys)
## illuminaHumanv3MAP has 29402 mapped keys (of 49576 keys)
## illuminaHumanv3OMIM has 21943 mapped keys (of 49576 keys)
## illuminaHumanv3PATH has 9180 mapped keys (of 49576 keys)
## illuminaHumanv3PATH2PROBE has 229 mapped keys (of 229 keys)
## illuminaHumanv3PMID has 29329 mapped keys (of 49576 keys)
## illuminaHumanv3PMID2PROBE has 439054 mapped keys (of 473622 keys)
## illuminaHumanv3REFSEQ has 29551 mapped keys (of 49576 keys)
## illuminaHumanv3SYMBOL has 29551 mapped keys (of 49576 keys)
```

```
## illuminaHumanv3UNIGENE has 29430 mapped keys (of 49576 keys)
## illuminaHumanv3UNIPROT has 27704 mapped keys (of 49576 keys)
##
##
## Additional Information about this package:
##
## DB schema: HUMANCHIP_DB
## DB schema version: 2.1
## Organism: Homo sapiens
## Date for NCBI data: 2015-Mar17
## Date for GO data: 20150314
## Date for KEGG data: 2011-Mar15
## Date for Golden Path data: 2010-Mar22
## Date for Ensembl data: 2015-Mar13
## #####Custom Mappings based on probe sequence####
## illuminaHumanv3ARRAYADDRESS()
## illuminaHumanv3NUID()
## illuminaHumanv3PROBEQUALITY()
## illuminaHumanv3CODINGZONE()
## illuminaHumanv3PROBESEQUENCE()
## illuminaHumanv3SECONDMATCHES()
## illuminaHumanv3OTHERGENOMICMATCHES()
## illuminaHumanv3REPEATMASK()
## illuminaHumanv3OVERLAPPINGSNP()
## illuminaHumanv3ENTREZREANNOTATED()
## illuminaHumanv3GENOMICLOCATION()
## illuminaHumanv3SYMBOLREANNOTATED()
## illuminaHumanv3REPORTERGROUENAME()
## illuminaHumanv3REPORTERGROUPEID()
## illuminaHumanv3ENSEMBLREANNOTATED()
```

If we suspect that a particular gene may be differentially expressed between conditions, we can subset the *ExpressionSetIllumina* object to just include probes that target the gene, and plot the response of these probes against the sample groups. Furthermore, the different probes can be distinguished using the `probeFactor` parameter.

```
ids <- which(fData(exampleSummaryData.log2)[,"SYMBOL"] == "ALB")

boxplot(exampleSummaryData.log2[ids,],
        SampleGroup = "SampleFac", probeFactor = "IlluminaID")
```



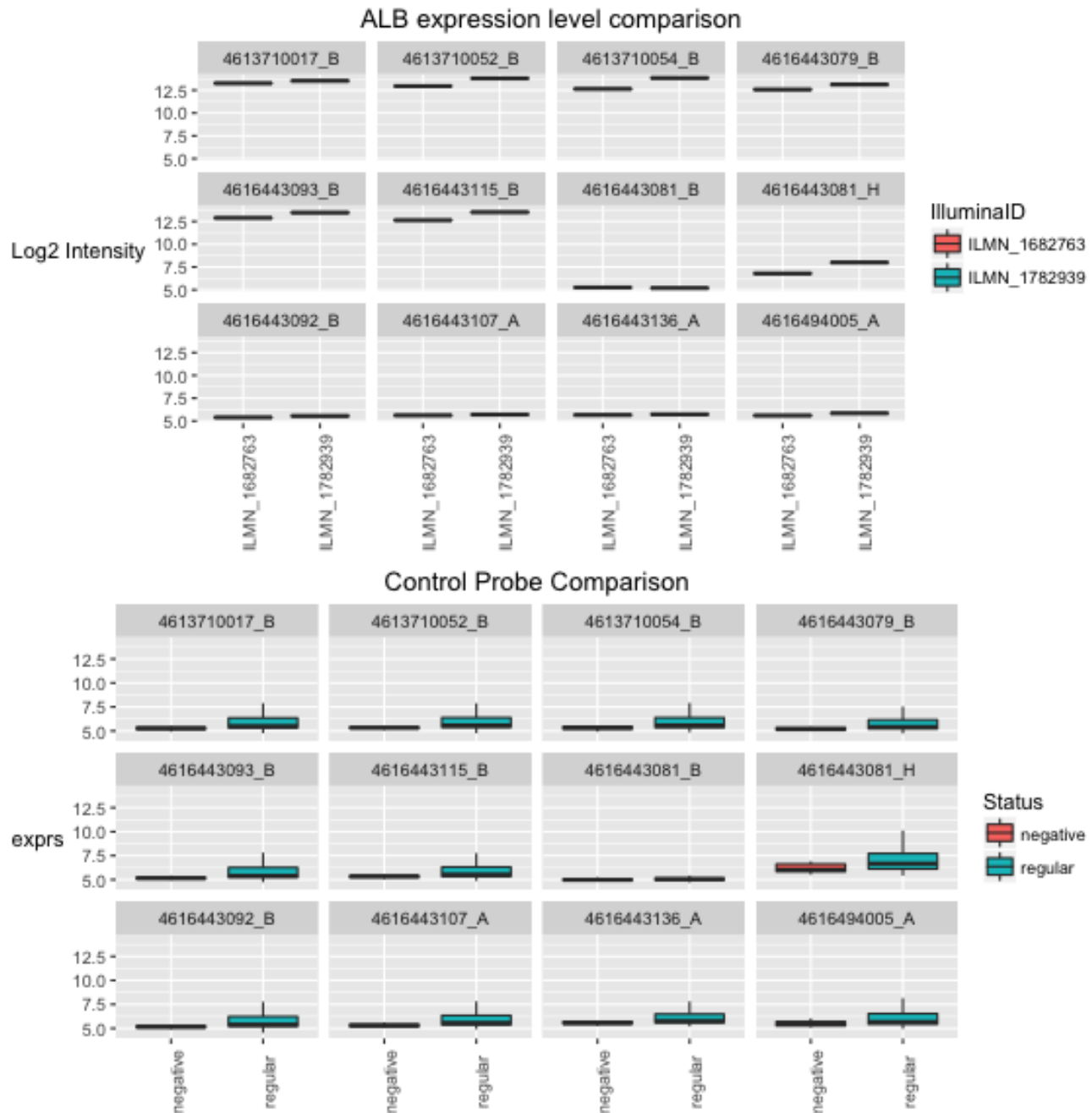
4.1 A note about ggplot2

The `boxplot` function in *beadarray* creates graphics using the *ggplot2* package rather than the R base graphics system. Therefore, the standard way of manipulating graphics using `par` and `mflow` etc will not work with the output of `boxplot`. However, the *ggplot2* package has equivalent functionality and is a more powerful and flexible system. There are numerous tutorials on how to use the *ggplot2* package, which is beyond the scope of this vignette. In the below code, we assign the results of `boxplot` to objects that we combine using the *gridExtra* package. The code also demonstrates how aspects of the plot can be altered programatically.

```
require("gridExtra")
bp1 <- boxplot(exampleSummaryData.log2[ids,],
  SampleGroup = "SampleFac", probeFactor = "IlluminaID")
bp1 <- bp1 + labs(title = "ALB expression level comparison") + xlab("Illumina Probe") + ylab("Expression Level")

bp2 <- boxplot(exampleSummaryData.log2[randIDs,], probeFactor = "Status")
bp2 <- bp2 + labs(title = "Control Probe Comparison")

grid.arrange(bp1, bp2)
```



We can also extract the data that was used to construct the plot.

```
bp1$data
```

```
##          Var1          Var2      value SampleGroup  probeFactor
## 1 ILMN_1782939 4613710017_B 13.528212          UHRR ILMN_1782939
## 2 ILMN_1682763 4613710017_B 13.264742          UHRR ILMN_1682763
## 3 ILMN_1782939 4613710052_B 13.800577          UHRR ILMN_1782939
## 4 ILMN_1682763 4613710052_B 12.947888          UHRR ILMN_1682763
## 5 ILMN_1782939 4613710054_B 13.841128          UHRR ILMN_1782939
```

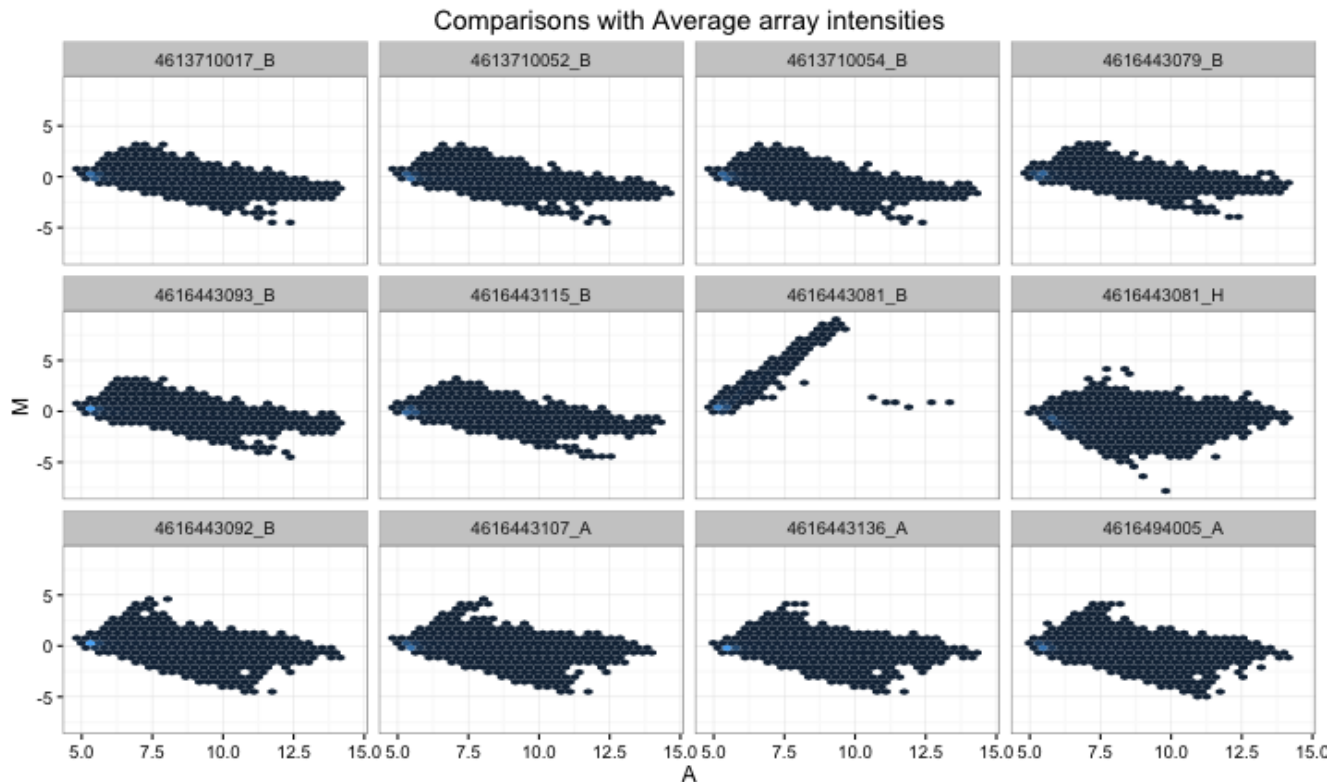
```
## 6  ILMN_1682763 4613710054_B 12.641636      UHRR ILMN_1682763
## 7  ILMN_1782939 4616443079_B 13.119897      UHRR ILMN_1782939
## 8  ILMN_1682763 4616443079_B 12.575922      UHRR ILMN_1682763
## 9  ILMN_1782939 4616443093_B 13.468822      UHRR ILMN_1782939
## 10 ILMN_1682763 4616443093_B 12.878392      UHRR ILMN_1682763
## 11 ILMN_1782939 4616443115_B 13.510831      UHRR ILMN_1782939
## 12 ILMN_1682763 4616443115_B 12.634381      UHRR ILMN_1682763
## 13 ILMN_1782939 4616443081_B  5.190355      Brain ILMN_1782939
## 14 ILMN_1682763 4616443081_B  5.249992      Brain ILMN_1682763
## 15 ILMN_1782939 4616443081_H  7.995407      Brain ILMN_1782939
## 16 ILMN_1682763 4616443081_H  6.788807      Brain ILMN_1682763
## 17 ILMN_1782939 4616443092_B  5.549147      Brain ILMN_1782939
## 18 ILMN_1682763 4616443092_B  5.388535      Brain ILMN_1682763
## 19 ILMN_1782939 4616443107_A  5.704762      Brain ILMN_1782939
## 20 ILMN_1682763 4616443107_A  5.617309      Brain ILMN_1682763
## 21 ILMN_1782939 4616443136_A  5.729863      Brain ILMN_1782939
## 22 ILMN_1682763 4616443136_A  5.658919      Brain ILMN_1682763
## 23 ILMN_1782939 4616494005_A  5.849509      Brain ILMN_1782939
## 24 ILMN_1682763 4616494005_A  5.598482      Brain ILMN_1682763
```

5 Other exploratory analysis

Replicate samples can also be compared using the `plotMA` function.

```
mas <- plotMA(exampleSummaryData.log2, do.log=FALSE)
```

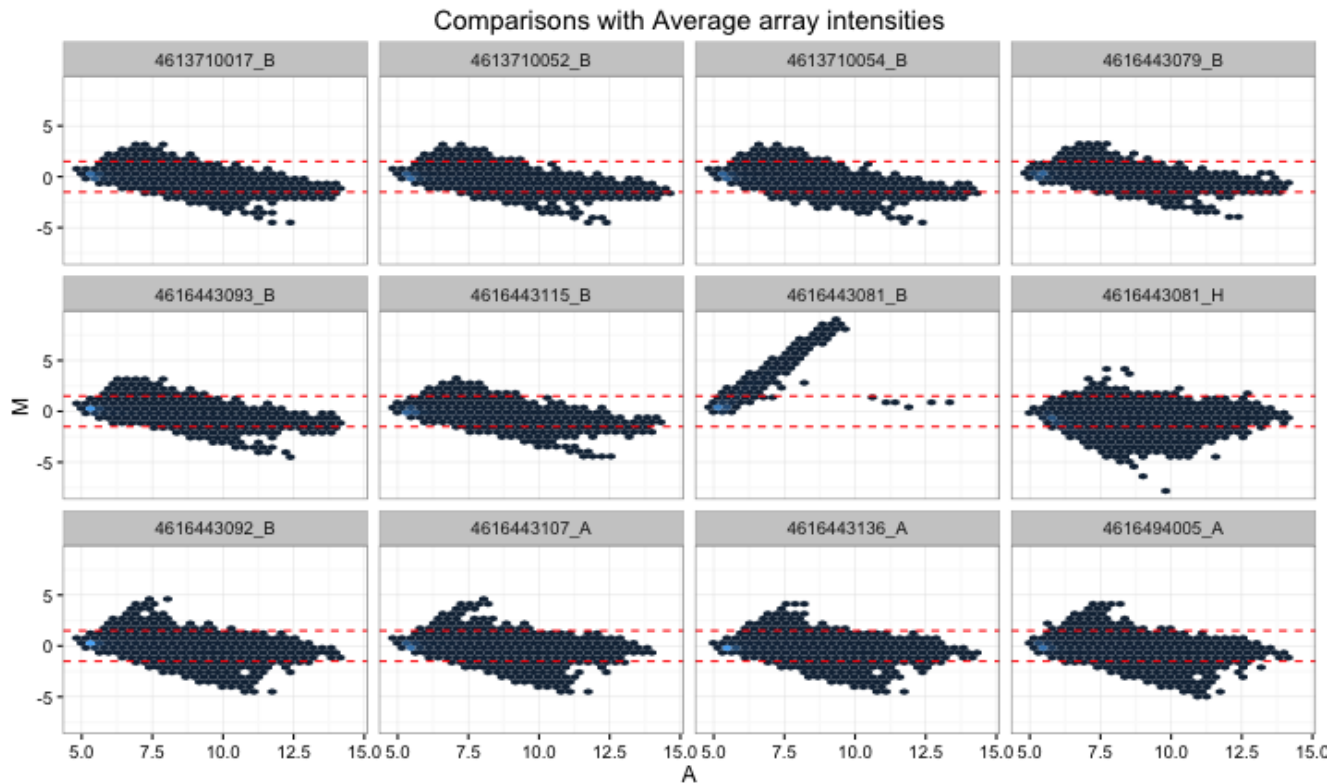
```
mas
```



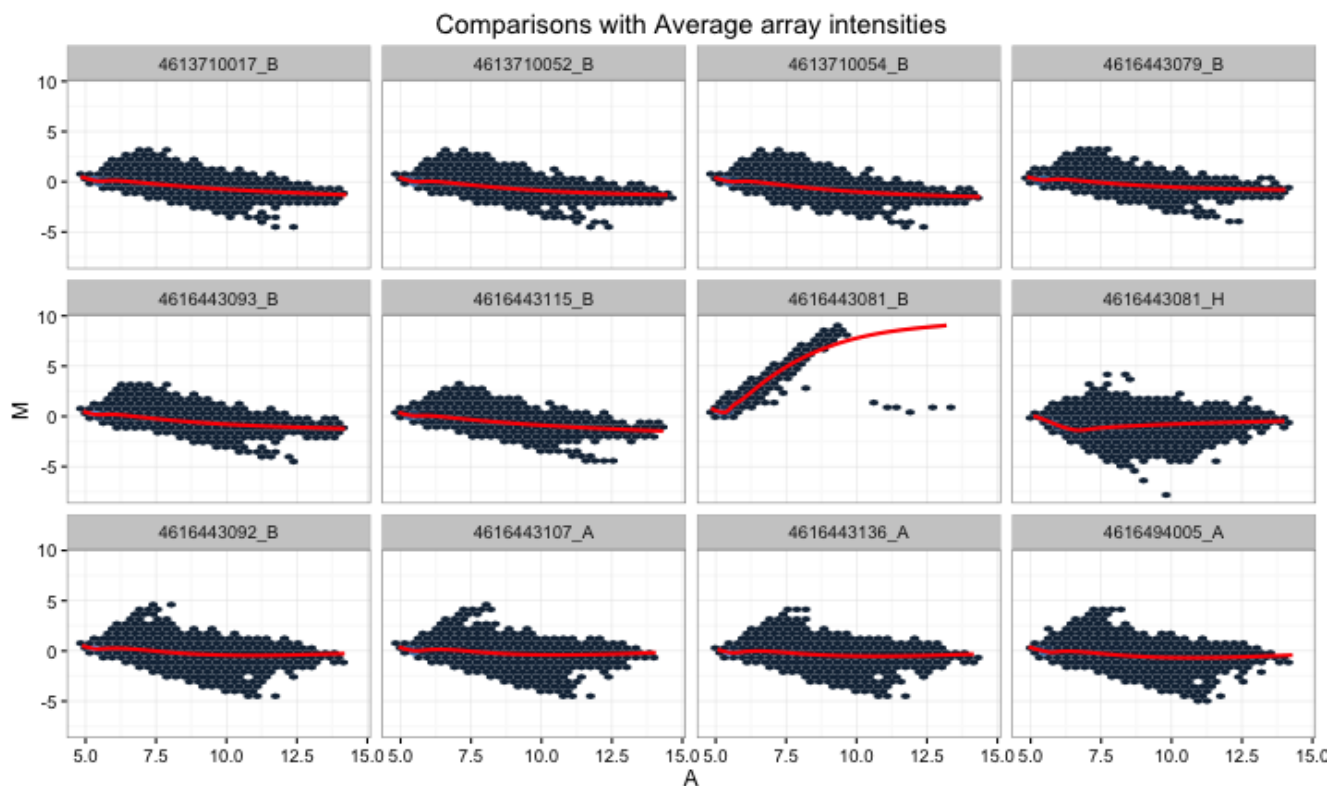
In each panel we see the MA plots for all arrays in the experiment compared to a 'reference' array composed of the average intensities of all probes. On an MA plot, for each probe we plot the average of the log₂ -intensities from the two arrays on the x-axis and the difference in intensities (log -ratios) on the y-axis. We would expect most probes to be and hence most points on the plot should lie along the line $y=0$.

As with boxplot, the object returned is a *ggplot2* object that can be modified by the end-user.

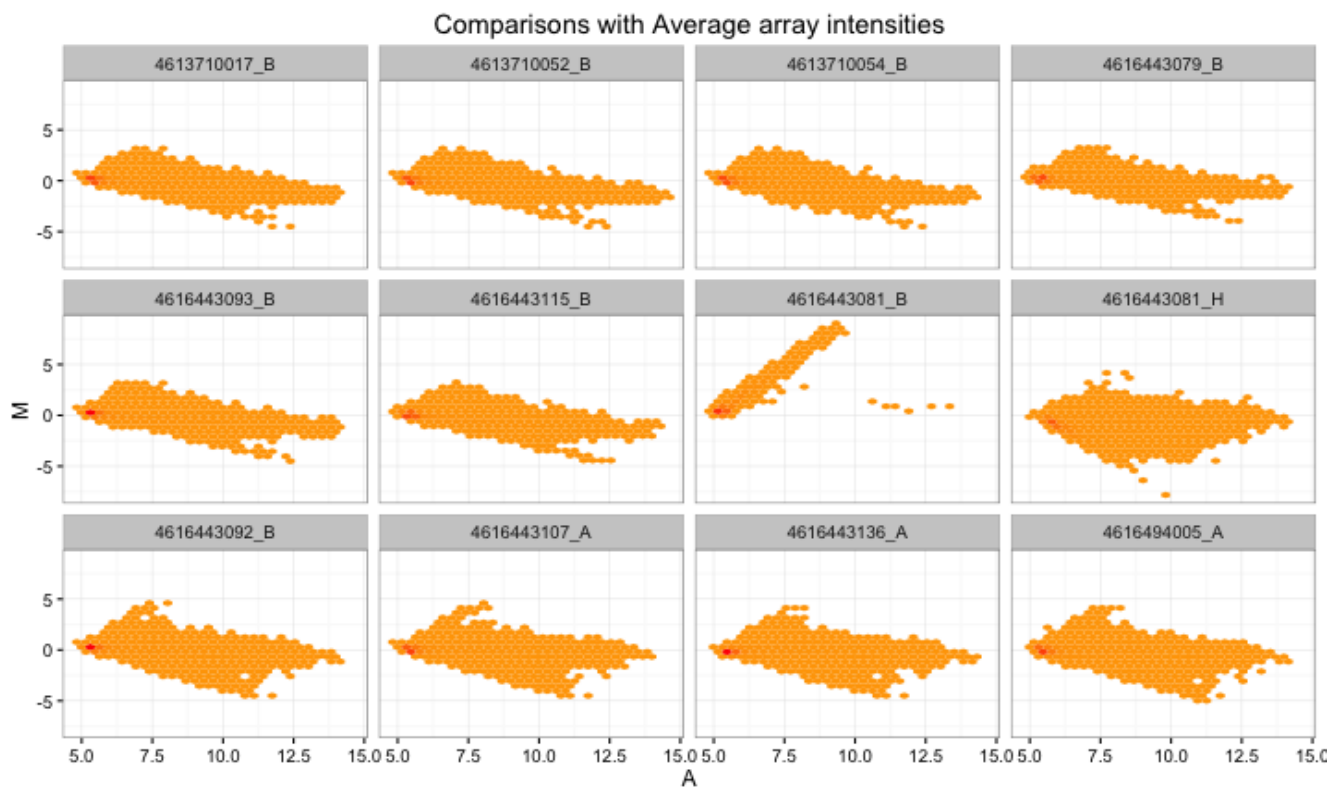
```
##Added lines on the y axis
mas + geom_hline(yintercept=c(-1.5,1.5),col="red",lty=2)
```



```
##Added a smoothed line to each plot
mas+ geom_smooth(col="red")
```

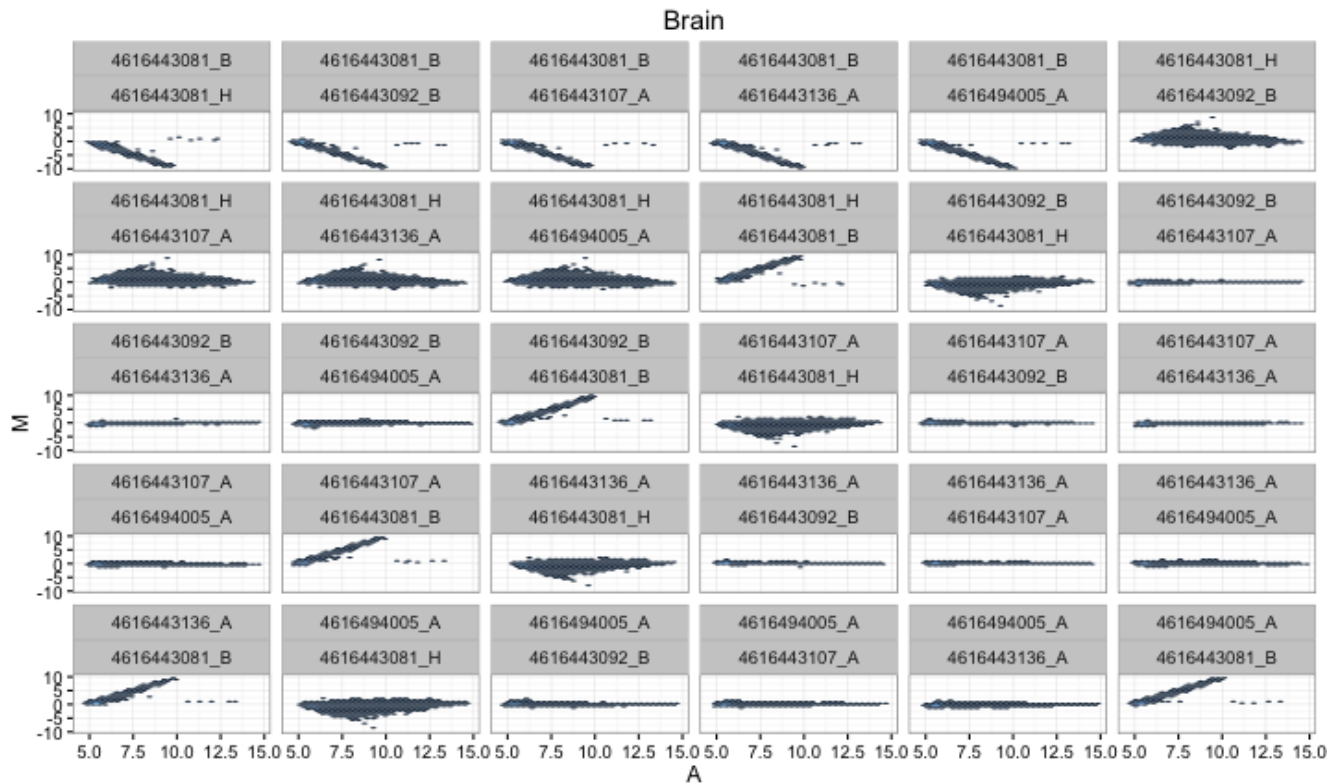


```
##Changing the color scale
mas + scale_fill_gradient2(low="yellow",mid="orange",high="red")
```



We can also specify a sample grouping, which will make all pairwise comparisons

```
mas <- plotMA(exampleSummaryData.log2,do.log=FALSE,SampleGroup="SampleFac")
mas[[1]]
```



6 Normalisation

To correct for differences in expression level across a chip and between chips we need to normalise the signal to make the arrays comparable. The normalisation methods available in the [affy](#) package, or variance-stabilising transformation from the [lumi](#) package may be applied using the `normaliseIllumina` function. Below we quantile normalise the \log_2 transformed data.

```
exampleSummaryData.norm <- normaliseIllumina(exampleSummaryData.log2,
method="quantile", transform="none")
```

An alternative approach is to combine normal-exponential background correction with quantile normalisation as suggested in the [limma](#) package. However, this requires data that have not been log-transformed. Note that the control probes are removed from the output object

```
exampleSummaryData.norm2 <- normaliseIllumina(channel(exampleSummaryData, "G.ul"),
method="neqc", transform="none")
```

7 Filtering

Filtering non-responding probes from further analysis can improve the power to detect differential expression. One way of achieving this is to remove probes whose probe sequence has undesirable properties. Four basic annotation quality categories ('Perfect', 'Good', 'Bad' and 'No match') are defined and have been shown to correlate with expression level and measures of differential expression. We recommend removing probes assigned a 'Bad' or 'No match' quality score after normalization. This approach is similar to the common practice of removing lowly-expressed probes, but with the additional benefit of discarding probes with a high expression level caused by non-specific hybridization.

```
library(illuminaHumanv3.db)

ids <- as.character(featureNames(exampleSummaryData.norm))

qual <- unlist(mget(ids, illuminaHumanv3PROBEQUALITY, ifnotfound=NA))

table(qual)

## qual
##      Bad      Good   Good***  Good****  No match  Perfect  Perfect***
##    13475      925      148       358      1739    24687    6269
## Perfect****
##      1975

rem <- qual == "No match" | qual == "Bad" | is.na(qual)

exampleSummaryData.filt <- exampleSummaryData.norm[!rem,]

dim(exampleSummaryData.filt)

## Features  Samples Channels
##    34362      12         1
```

8 Differential expression

The differential expression methods available in the limma package can be used to identify differentially expressed genes. The functions `lmFit` and `eBayes` can be applied to the normalised data. In the example below, we set up a design matrix for the example experiment and fit a linear model to summaries the data from the UHRR and Brain replicates to give one value per condition. We then define contrasts comparing the Brain sample to the UHRR and calculate moderated t-statistics with empirical Bayes shrinkage of the sample variances. In this particular experiment, the Brain and UHRR samples are very different and we would expect to see many differentially expressed genes.

Empirical array quality weights can be used to measure the relative reliability of each array. A variance

is estimated for each array by the `arrayWeights` function which measures how well the expression values from each array follow the linear model. These variances are converted to relative weights which can then be used in the linear model to down-weight observations from less reliable arrays which improves power to detect differential expression. You should notice that some arrays have very low weight consistent with their poor QC.

We then define a contrast comparing UHRR to Brain Reference and calculate moderated t -statistics with empirical Bayes' shrinkage of the sample variances.

```
rna <- factor(pData(exampleSummaryData)[, "SampleFac"])

design <- model.matrix(~0+rna)
colnames(design) <- levels(rna)
aw <- arrayWeights(exprs(exampleSummaryData.filt), design)
aw
fit <- lmFit(exprs(exampleSummaryData.filt), design, weights=aw)
contrasts <- makeContrasts(UHRR-Brain, levels=design)
contr.fit <- eBayes(contrasts.fit(fit, contrasts))
topTable(contr.fit, coef=1)
```

8.1 Automating the DE analysis

A convenience function has been created to automate the differential expression analysis and repeat the above steps. The requirements to the function are a normalised object and a `SampleGroup`. By default, a design matrix and contrast matrix are derived from the `SampleGroup` by considering all pairwise contrasts. The matrices used, along with the array weights are saved in the output and can be retrieved later.

```
limmaRes <- limmaDE(exampleSummaryData.filt, SampleGroup="SampleFac")
limmaRes

## Results of limma analysis
## Design Matrix used...
##   Brain UHRR
## 1      0    1
## 2      0    1
## 3      0    1
## 4      0    1
## 5      0    1
## 6      0    1
## .....
##
## Array Weights....
## Contrast Matrix used...
##           Contrasts
```

```
## Levels Brain-UHRR
## Brain 1
## UHRR -1
## Array Weights....
## 2.09 2.527 ... 2.086 1.287
## Top Table
## Top 10 probes for contrast Brain-UHRR
## Row.names ArrayAddressID IlluminaID Status SYMBOL PROBEQUALITY
## ILMN_1651358 ILMN_1651358 4830541 ILMN_1651358 regular HBE1 Perfect
## ILMN_1796678 ILMN_1796678 450537 ILMN_1796678 regular HBG1 Perfect
## ILMN_1713458 ILMN_1713458 6980192 ILMN_1713458 regular HBZ Perfect
## ILMN_1783832 ILMN_1783832 7570189 ILMN_1783832 regular GAGE6 Good****
## CODINGZONE PROBESEQUENCE
## ILMN_1651358 Transcriptomic ATTCTGGCTACTCACTTTGGCAAGGAGTTCACCCCTGAAGTGCAGGCTGC
## ILMN_1796678 Transcriptomic AGAATTCACCCCTGAGGTGCAGGCTTCCTGGCAGAAGATGGTGAAGTGCAG
## ILMN_1713458 Transcriptomic GTCCTGGAGGTCCCCAGCCCCACTTACCGCGTAATGCGCCAATAAACCA
## ILMN_1783832 Transcriptomic CCACAGACTGGGTGTGAGTGTGAAGATGGTCCTGATGGGCAGGAGGTGGA
## GENOMICLOCATION LogFC LogOdds pvalue
## ILMN_1651358 chr11:5289754:5289803:- -7.344613 67.02204 5.296571e-34
## ILMN_1796678 chr11:5269621:5269670:- -7.320711 66.49238 9.666297e-34
## ILMN_1713458 chr16:2044444:204493:+ -6.419033 64.32882 1.084133e-32
## ILMN_1783832 chrX:49330136:49330185:+ -5.972782 64.09107 1.408963e-32
##
##
## Significant probes with adjusted p-value < 0.05
## Direction
## -1 0 1
## 4709 25642 4011

DesignMatrix(limmaRes)

## Brain UHRR
## 1 0 1
## 2 0 1
## 3 0 1
## 4 0 1
## 5 0 1
## 6 0 1
## 7 1 0
## 8 1 0
## 9 1 0
## 10 1 0
## 11 1 0
## 12 1 0
## attr(,"assign")
```

```
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$`as.factor(SampleGroup)`
## [1] "contr.treatment"

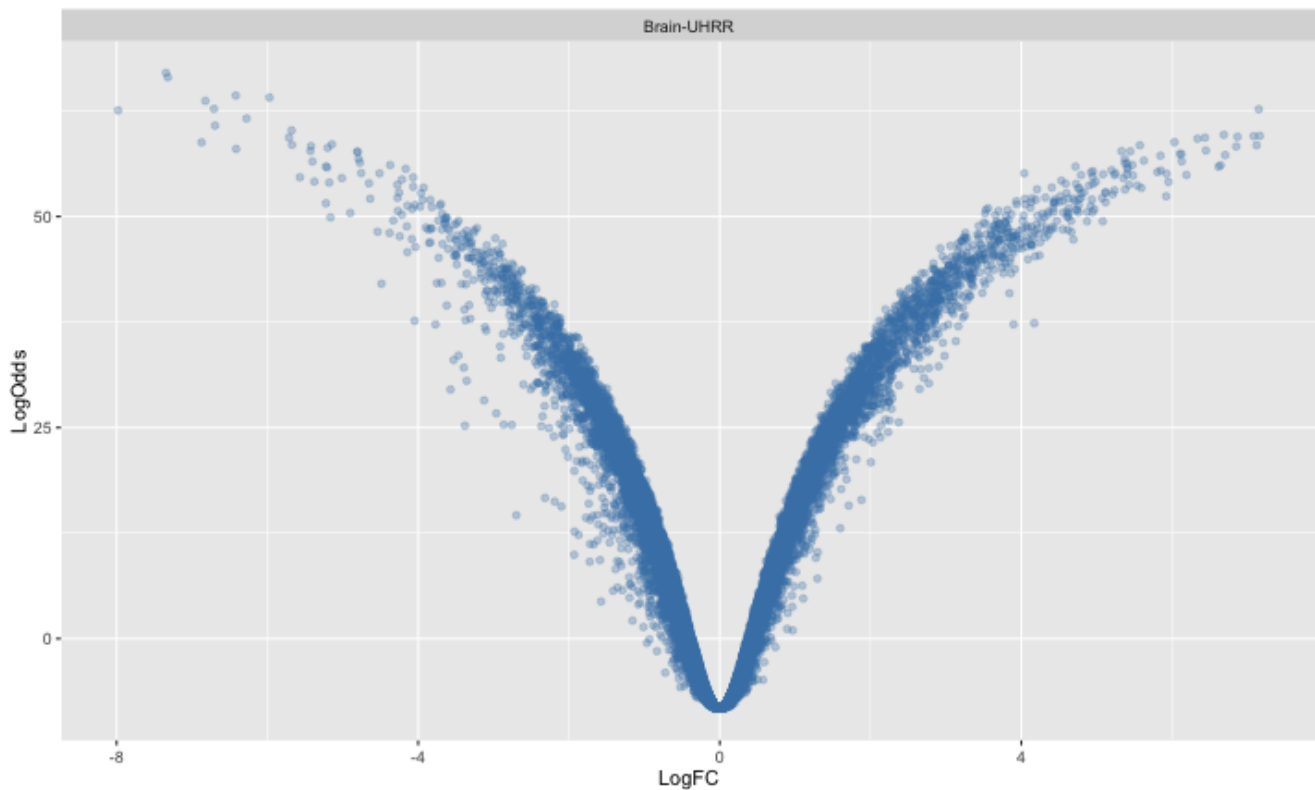
ContrastMatrix(limmaRes)

##           Contrasts
## Levels   Brain-UHRR
##   Brain           1
##   UHRR            -1

ArrayWeights(limmaRes)

##           1           2           3           4           5           6           7           8
## 2.09018696 2.52678943 1.45410355 1.77470959 2.13405097 1.85777235 0.01233139 0.11159911
##           9          10          11          12
## 2.45290622 2.04233539 2.08578184 1.28699294

plot(limmaRes)
```



9 Output as GRanges

To assist with meta-analysis, and integration with other genomic data-types, it is possible to export the normalised values as a *GRanges* object. Therefore it is easy to perform overlaps, counts etc with other data using the *GenomicRanges* and *GenomicFeatures* packages. In order for the ranges to be constructed, the genomic locations of each probe have to be obtained from the appropriate annotation package (*illuminaHumanv3.db* in this example). Provided that this package has been installed, the mapping should occur automatically. The expression values are stored in the *GRanges* object along with the featureData.

```
gr <- as(exampleSummaryData.filt[,1:5], "GRanges")
gr
```

```
## GRanges object with 37013 ranges and 14 metadata columns:
```

##	seqnames	ranges	strand	Row.names
##	<Rle>	<IRanges>	<Rle>	<character>
##	ILMN_1776601	chr1 [69476, 69525]	+	ILMN_1776601
##	ILMN_1665540	chr1 [324468, 324517]	+	ILMN_1665540
##	ILMN_1776483	chr1 [324469, 324518]	+	ILMN_1776483
##	ILMN_1682912	chr1 [324673, 324722]	+	ILMN_1682912
##	ILMN_1889155	chr1 [759949, 759998]	+	ILMN_1889155
##
##	ILMN_1691189	chrUn_g1000211 [23460, 23509]	-	ILMN_1691189
##	ILMN_1722620	chr4_g1000193_random [75089, 75138]	-	ILMN_1722620
##	ILMN_1821517	chrM [8249, 8287]	+	ILMN_1821517
##	ILMN_1660133	chr7_g1000195_random [165531, 165580]	+	ILMN_1660133
##	ILMN_1684166	chr4_g1000194_random [55310, 55359]	-	ILMN_1684166

##	ArrayAddressID	IlluminaID	Status	SYMBOL	PROBEQUALITY	CODINGZONE
##	<numeric>	<factor>	<factor>	<factor>	<factor>	<factor>
##	ILMN_1776601	3610128	ILMN_1776601	regular	OR4F5	Perfect Transcriptomi
##	ILMN_1665540	2570482	ILMN_1665540	regular	<NA>	Perfect**** Transcriptomi
##	ILMN_1776483	6290672	ILMN_1776483	regular	<NA>	Perfect**** Transcriptomi
##	ILMN_1682912	4060014	ILMN_1682912	regular	<NA>	Perfect**** Transcriptomi
##	ILMN_1889155	1780768	ILMN_1889155	regular	<NA>	Perfect*** Transcriptomi
##
##	ILMN_1691189	5310750	ILMN_1691189	regular	<NA>	Perfect**** Transcriptomi
##	ILMN_1722620	5560181	ILMN_1722620	regular	MGC39584	Perfect**** Transcriptomi
##	ILMN_1821517	6550386	ILMN_1821517	regular	<NA>	Good Transcriptomi
##	ILMN_1660133	7510136	ILMN_1660133	regular	<NA>	Perfect*** Transcriptomi
##	ILMN_1684166	7650241	ILMN_1684166	regular	MAFIP	Perfect Transcriptomi

##	PROBESEQUENCE
##	<factor>
##	ILMN_1776601 TGTGTGGCAACGCATGTGTCGGCATTATGGCTGTCACATGGGGAATTGGC
##	ILMN_1665540 CAGAACTTTCTCCAGTCAGCCTCTACAGACCAAGCTCATGACTCACAATG
##	ILMN_1776483 AGAACTTTCTCCAGTCAGCCTCTACAGACCAAGCTCATGACTCACAATGG

```
## ILMN_1682912 GTCGACCTCACCAGGCCCAGCTCATGCTTCTTTGCAGCCTCTCCAGGCCC
## ILMN_1889155 GCCCCAAGTGAGGAACCCCTCAGACATTTGCAGAGGAGTCAGTGTGCTAG
## ...
## ILMN_1691189 GCCTGTCTTCAAACTAAGATTACAAAGCCATGGTAACACTGTGTAAGTG
## ILMN_1722620 CCAGCATCTCCTGGACAGTCAGCCGAGTGTTCATGATACCAGCCATAC
## ILMN_1821517 GCAGGGCCCGTATTTACCCTATAGCACCCCCTCTAACCCTTTTGAGACC
## ILMN_1660133 GCAGACAGCCTGAGGAAGATATAAGTAGAGGGATGGAGAATCCTAGGGCC
## ILMN_1684166 TTTCTCCTCTGTCCCACTTATCCCGAGGGACCCAGAAGCAAGTGTCCAC
## GENOMICLOCATION X4613710017_B X4613710052_B
## <factor> <numeric> <numeric>
## ILMN_1776601 chr1:69476:69525:+ 5.657618 5.322804
## ILMN_1665540 chr1:324468:324517:+ 6.410364 6.197932
## ILMN_1776483 chr1:324469:324518:+ 6.659373 6.230466
## ILMN_1682912 chr1:324673:324722:+ 6.055430 6.035210
## ILMN_1889155 chr1:759949:759998:+ 5.517109 5.535570
## ...
## ILMN_1691189 chrUn_g1000211:23460:23509:- 5.269216 5.119708
## ILMN_1722620 chr4_g1000193_random:75089:75138:- 5.868094 5.881239
## ILMN_1821517 chrM:8249:8287:+ 13.239307 13.530129
## ILMN_1660133 chr7_g1000195_random:165531:165580:+ 5.759365 5.937252
## ILMN_1684166 chr4_g1000194_random:55310:55359:- 5.442914 5.341760
## X4613710054_B X4616443079_B X4616443093_B
## <numeric> <numeric> <numeric>
## ILMN_1776601 5.427858 5.237883 5.362930
## ILMN_1665540 6.277684 6.222829 6.303296
## ILMN_1776483 6.317478 6.086825 6.264725
## ILMN_1682912 5.998791 6.016191 6.016191
## ILMN_1889155 5.518179 5.472247 5.611769
## ...
## ILMN_1691189 5.306670 5.339929 5.462299
## ILMN_1722620 5.692465 5.740337 5.950920
## ILMN_1821517 13.609522 13.095844 13.192104
## ILMN_1660133 5.896739 5.529788 5.917756
## ILMN_1684166 5.253823 5.786705 5.719844
## -----
## seqinfo: 48 sequences from an unspecified genome; no seqlengths
```

The limma analysis results can also be exported as a GRanges object for downstream analysis. The elementMetadata of the output object is set to the statistics from the limma analysis.

```
lgr <- as(limmaRes, "GRanges")
lgr

## GRangesList object of length 1:
## $Brain-UHRR
```

```
## GRanges object with 37013 ranges and 3 metadata columns:
##           seqnames           ranges strand |           LogFC
##           <Rle>           <IRanges> <Rle> |           <numeric>
## ILMN_1776601      chr1 [ 69476,  69525]   + | -0.0417471673184382
## ILMN_1665540      chr1 [324468, 324517]   + | -0.310015481626054
## ILMN_1776483      chr1 [324469, 324518]   + | -0.545248865530192
## ILMN_1682912      chr1 [324673, 324722]   + | -0.280075141270522
## ILMN_1889155      chr1 [759949, 759998]   + |  0.00935410440764795
##           ...           ...           ... |           ...
## ILMN_1691189      chrUn_gl000211 [ 23460,  23509]   - |  0.143264446289193
## ILMN_1722620      chr4_gl000193_random [ 75089,  75138]   - | -0.406766579821191
## ILMN_1821517      chrM [ 8249,  8287]     + |  0.0155974678785551
## ILMN_1660133      chr7_gl000195_random [165531, 165580]   + | -0.371030650489396
## ILMN_1684166      chr4_gl000194_random [ 55310,  55359]   - |  0.000462823768571319
##           LogOdds           PValue
##           <numeric>           <numeric>
## ILMN_1776601 -8.17361791442789  0.644439702761411
## ILMN_1665540 -3.19675200376246  0.00190943267349829
## ILMN_1776483  2.89721369582597  4.15850553120308e-06
## ILMN_1682912 -3.02509723946879  0.0015999301905369
## ILMN_1889155 -8.2796390853818  0.911256304500686
##           ...           ...           ...
## ILMN_1691189 -7.69584585909284  0.290595627860216
## ILMN_1722620  0.296467921866389  5.53352021578778e-05
## ILMN_1821517 -8.27190883294181  0.869373703086076
## ILMN_1660133 -1.75295130978946  0.000436179791791669
## ILMN_1684166 -8.2861943021698  0.996284874188181
## -----
## seqinfo: 48 sequences from an unspecified genome; no seqlengths
```

The data can be manipulated according to the DE stats

```
lgr <- lgr[[1]]
lgr[order(lgr$LogOdds,decreasing=T)]

## GRanges object with 37013 ranges and 3 metadata columns:
##           seqnames           ranges strand |           LogFC
##           <Rle>           <IRanges> <Rle> |           <numeric>
## ILMN_1651358      chr11 [ 5289754,  5289803]   - | -7.34461266210468
## ILMN_1796678      chr11 [ 5269621,  5269670]   - | -7.32071070399014
## ILMN_1713458      chr16 [ 204444,  204493]     + | -6.41903262349249
## ILMN_1783832      chrX [49330136, 49330185]     + | -5.97278193212517
## ILMN_1782939      chr4 [74285311, 74285356]     + | -6.82215124660207
##           ...           ...           ... |           ...
```

```
## ILMN_1795567 chr4 [ 156920, 156969] + | -2.01101921826208e-05
## ILMN_2288639 chrX [ 49369673, 49369722] + | -1.96088181594334e-05
## ILMN_1727040 chr1 [ 44401157, 44401206] + | -5.43947807685186e-06
## ILMN_1813701 chr1 [178443252, 178443301] + | -3.58307724734885e-06
## ILMN_1705568 chr7 [107271045, 107271094] + | -3.15620111468462e-06
##
##          LogOdds          PValue
##          <numeric>          <numeric>
## ILMN_1651358 67.0220419617412 5.29657071699751e-34
## ILMN_1796678 66.4923816114673 9.66629693323439e-34
## ILMN_1713458 64.3288232451041 1.08413328947548e-32
## ILMN_1783832 64.0910654560522 1.40896346875058e-32
## ILMN_1782939 63.7054328795779 2.15233239877643e-32
## ...
## ILMN_1795567 -8.28620574316231 0.99984457243611
## ILMN_2288639 -8.2862057437304 0.999846789087848
## ILMN_1727040 -8.28620576098513 0.999948094879518
## ILMN_1813701 -8.28620576239985 0.999968528746846
## ILMN_1705568 -8.28620576258482 0.999972292910236
## -----
## seqinfo: 48 sequences from an unspecified genome; no seqlengths

lgr[p.adjust(lgr$PValue)<0.05]

## GRanges object with 9420 ranges and 3 metadata columns:
##          seqnames          ranges strand |          LogFC
##          <Rle>          <IRanges> <Rle> |          <numeric>
## ILMN_1709067 chr1 [ 879456, 879505] + | -0.6285449686972
## ILMN_1705602 chr1 [ 900738, 900787] + | -0.613232031371666
## ILMN_1770454 chr1 [ 991196, 991245] + | -1.01510479372037
## ILMN_1780315 chr1 [1246734, 1246783] + | -0.721249818912305
## ILMN_1773026 chr1 [1372636, 1372685] + | 0.669488998888564
## ...
## ILMN_2398587 chr6_apd_hap1 [1269579, 1269628] + | -1.25247098453431
## ILMN_1692486 chr6_apd_hap1 [1270027, 1270031] + | -1.58597035134591
## ILMN_1692486 chr6_apd_hap1 [1270238, 1270282] + | -1.58597035134591
## ILMN_1708006 chr6_apd_hap1 [2793475, 2793524] + | -2.07798043507869
## ILMN_2070300 chr6_apd_hap1 [3079946, 3079995] - | -1.43842772631114
##
##          LogOdds          PValue
##          <numeric>          <numeric>
## ILMN_1709067 7.42471212143437 4.8130494289674e-08
## ILMN_1705602 5.60598100967875 2.87153951468259e-07
## ILMN_1770454 18.3445724383623 1.13472633773764e-12
## ILMN_1780315 8.31031283452404 2.02073133623892e-08
## ILMN_1773026 10.3130628453766 2.8487426766453e-09
## ...
```

```
## ILMN_2398587 19.6370877229845 3.22590423083603e-13
## ILMN_1692486 26.7915783558006 3.05100900018172e-16
## ILMN_1692486 26.7915783558006 3.05100900018172e-16
## ILMN_1708006 33.118354645671 6.36280713220114e-19
## ILMN_2070300 25.1915158013652 1.44926001142217e-15
## -----
## seqinfo: 48 sequences from an unspecified genome; no seqlengths
```

We can do overlaps with other *GRanges* objects

```
library(GenomicRanges)
HBE1 <- GRanges("chr11", IRanges(5289580,5291373),strand="-")

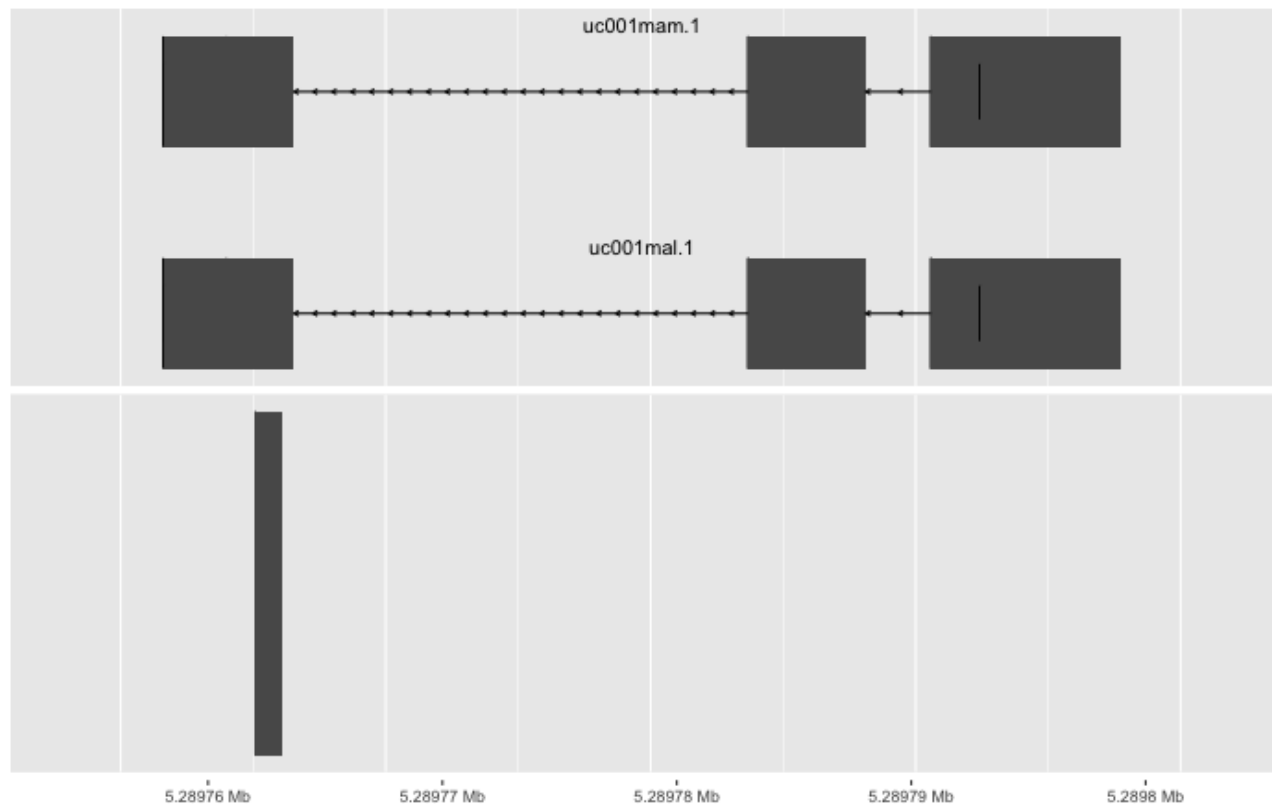
lgr[lgr %over% HBE1]

## GRanges object with 1 range and 3 metadata columns:
##           seqnames           ranges strand |           LogFC           LogOdds
##           <Rle>             <IRanges> <Rle> |           <numeric>           <numeric>
## ILMN_1651358 chr11 [5289754, 5289803]   - | -7.34461266210468 67.0220419617412
##           PValue
##           <numeric>
## ILMN_1651358 5.29657071699751e-34
## -----
## seqinfo: 48 sequences from an unspecified genome; no seqlengths
```

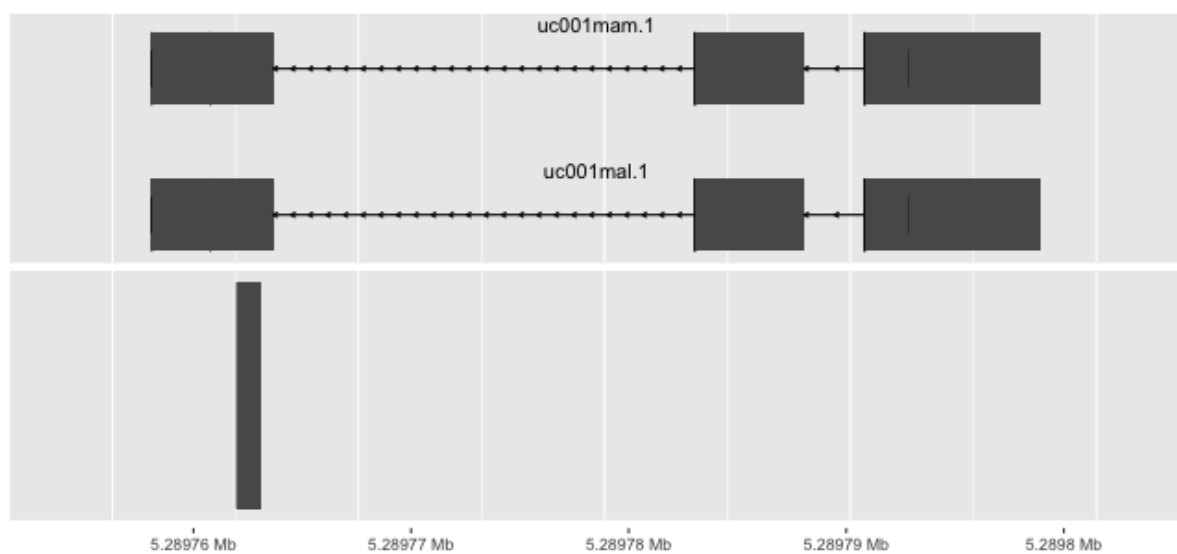
9.1 Visualisation options

Having converted the DE results into a common format such as *GRanges* allows access to common routines, such as those provided by *ggbio*. For example, it is often useful to know where exactly the illumina probes are located with respect to the gene.

```
library(ggbio)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
tx <- TxDb.Hsapiens.UCSC.hg19.knownGene
p1 <- autoplot(tx, which=HBE1)
p2 <- autoplot(lgr[lgr %over% HBE1])
tracks(p1,p2)
```

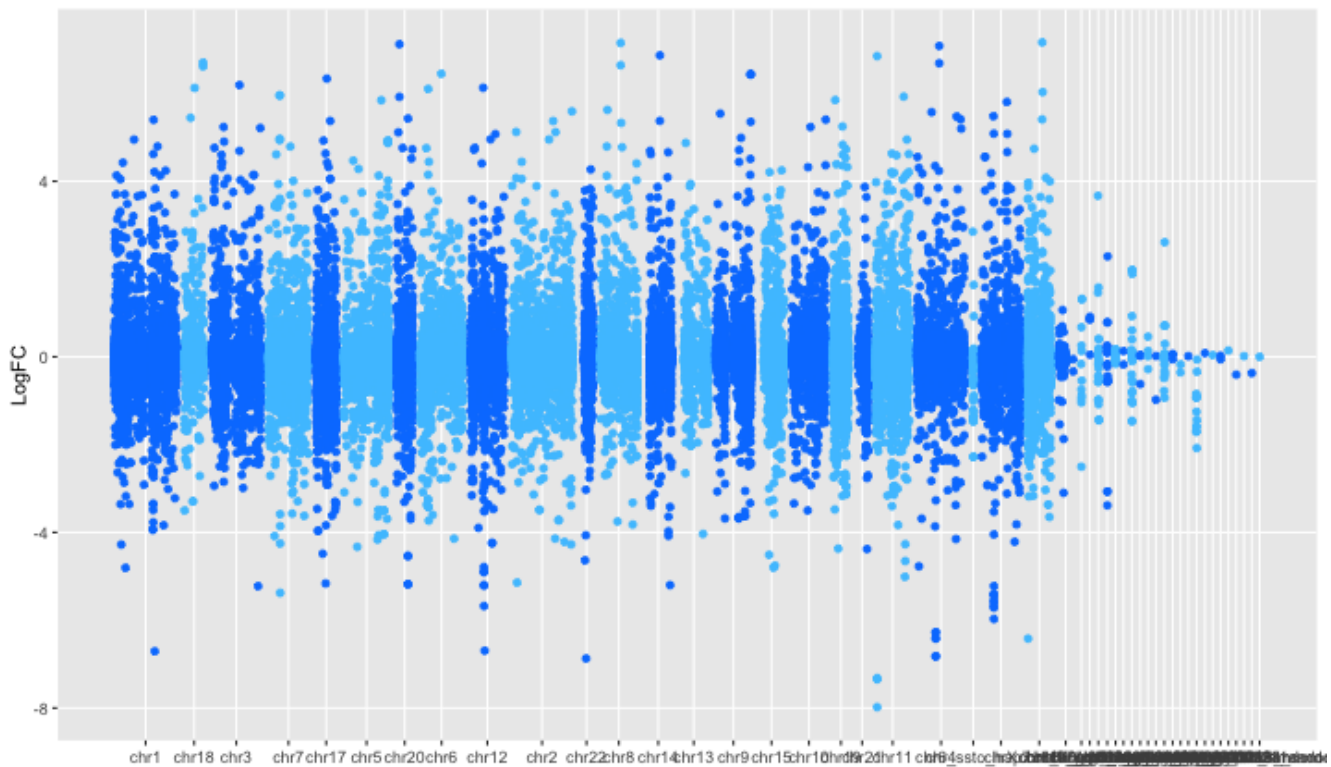


```
id <- plotIdeogram(genome="hg19", subchr="chr11")  
tracks(id,p1,p2)
```



Genome-wide plots are also available

```
plotGrandLinear(lgr, aes(y = LogFC))
```



10 Creating a GEO submission file

Most journals are now requiring that data are deposited in a public repository prior to publication of a manuscript. Formatting the microarray and associated metadata can be time-consuming, so we have provided a function to create a template for a GEO submission. GEO require particular meta data to be recorded regarding the experimental protocols. The output of the `makeGEOSubmissionFiles` includes a spreadsheet with the relevant fields that can be filled in manually. The normalised and raw data are written to tab-delimited files. By default, the annotation package associated with the data is consulted to determine which probes are exported. Any probes that are present in the data, but not in the annotation package are excluded from the submission file.

```
rawdata <- channel(exampleSummaryData, "G")
normdata <- normaliseIllumina(rawdata)

makeGEOSubmissionFiles(normdata, rawdata)
```

Alternatively, GEO's official probe annotation files can be used to decide which probes to include in the submission. You will first have to download the appropriate file from the GEO website.

```
download.file(
  "ftp://ftp.ncbi.nlm.nih.gov/geo/platforms/GPL6nnn/GPL6947/annot/GPL6947.annot.gz",
  destfile="GPL6947.annot.gz"
)

makeGEOSubmissionFiles(normdata,rawdata,softTemplate="GPL6947.annot.gz")
```

11 Analysing data from GEO

beadarray now contains functionality that can assist in the analysis of data available in the GEO (Gene Expression Omnibus) repository. We can download such data using [GEOquery](#):

```
library(GEOquery)
url <- "ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SeriesMatrix/GSE33126/"
filenm <- "GSE33126_series_matrix.txt.gz"
if(!file.exists("GSE33126_series_matrix.txt.gz")) download.file(paste(url, filenm, sep=""))
gse <- getGEO(filename=filenm)
head(exprs(gse))
```

Now we convert this to an *ExpressionSetIllumina*; *beadarray*'s native class for dealing with summarised data. The *annotation* slot stored in the *ExpressionSet* is converted from a GEO identifier (e.g. GPL10558) to one recognised by *beadarray* (e.g. Humanv4). If no conversion is possible, the resulting object will have NULL for the annotation slot. If successful, you should notice that the object is automatically annotated against the latest available annotation package.

```
summaryData <- as(gse, "ExpressionSetIllumina")
summaryData
head(fData(summaryData))
```

As we have annotated using the latest packages, we have imported the probe quality scores. We can calculate Detection scores by using the 'No match' probes as a reference; useful as data in repositories rarely export these data

```
fData(summaryData)$Status <-
  ifelse(fData(summaryData)$PROBEQUALITY=="No match","negative","regular" )

Detection(summaryData) <- calculateDetection(summaryData,
  status=fData(summaryData)$Status)
```

The 'neqc' normalisation method from limma can also be used now.

```
summaryData.norm <- normaliseIllumina(summaryData,method="neqc",
  status=fData(summaryData)$Status)
boxplot(summaryData.norm)
```

We can do differential expression if we know the column in the phenoData that contains sample group information

```
limmaResults <- limmaDE(summaryData.norm, "source_name_ch1")
limmaResults
```

12 Reading bead summary data into beadarray

BeadStudio/GenomeStudio is Illumina's proprietary software for analyzing data output by the scanning system (BeadScan/iScan). It contains different modules for analyzing data from different platforms. For further information on the software and how to export summarized data, refer to the user's manual. In this section we consider how to read in and analyze output from the gene expression module of BeadStudio/GenomeStudio.

The example dataset used in this section consists of an experiment with one Human WG-6 version 2 BeadChip. These arrays were hybridized with the control RNA samples used in the MAQC project (3 replicates of UHRR and 3 replicates of Brain Reference RNA).

The non-normalized data for regular and control probes was output by BeadStudio/GenomeStudio.

The example BeadStudio output used in this section is available as a zip file that can be downloaded from

tt http://compbio.works/data/BeadStudioExample/AsuragenMAQC_BeadStudioOutput.zip.

You will need to download and unzip the contents of this file to the current R working directory. Inside this zip file you will find several files including summarized, non-normalized data and a file containing control information. We give a more detailed description of each of the particular files we will make use of below.

- Sample probe profile (AsuragenMAQC-probe-raw.txt) (*required*) - text file which contains the non-normalized summary values as output by BeadStudio. Inside the file is a data matrix with some 48,000 rows. In newer versions of the software, these data are preceded by several lines of header information. Each row is a different probe in the experiment and the columns give different measurements for the gene. For each array, we record the summarized expression level (AVG_Signal), standard error of the bead replicates (BEAD_STDERR), number of beads (Avg_NBEADS) and a detection p -value (Detection Pval) which estimates the probability of a gene being detected above the background level. When exporting this file from BeadStudio, the user is able to choose which columns to export.
- Control probe profile (AsuragenMAQC-controls.txt) (*recommended*) - text file which contains the summarized data for each of the controls on each array, which may be useful for diagnostic and calibration purposes. Refer to the Illumina documentation for information on what each control measures.
- targets file (*optional*) - text file created by the user specifying which sample is hybridized to each array. No such file is provided for this dataset, however we can extract sample annotation information from the column headings in the sample probe profile.

Files with normalized intensities (those with avg in the name), as well as files with one intensity value per gene (files with gene in the name) instead of separate intensities for different probes targeting the same transcript, are also available in this download. We recommend users work with the non-normalized probe-specific data in their analysis where possible. Illumina's background correction step, which subtracts the intensities of the negative control probes from the intensities of the regular probes, should also be avoided.

```
library(beadarray)
dataFile = "AsuragenMAQC-probe-raw.txt"
qcFile = "AsuragenMAQC-controls.txt"
BSData = readBeadSummaryData(dataFile = dataFile,
qcFile = qcFile, controlID = "ProbeID",
skip = 0, qc.skip = 0, qc.columns = list(exprs = "AVG_Signal",
Detection = "Detection Pval"))
```

The arguments of `readBeadSummaryData` can be modified to suit data from versions 1, 2 or 3 of BeadStudio. The current default settings should work for version 3 output. Users may need to change the argument `sep`, which specifies if the `dataFile` is comma or tab delimited and the `skip` argument which specifies the number of lines of header information at the top of the file. Possible skip arguments of 0, 7 and 8 have been observed, depending on the version of BeadStudio or way in which the data was exported. The `columns` argument is used to specify which column headings to read from `dataFile` and store in various matrices. Note that the naming of the columns containing the standard errors changed between versions of BeadStudio (earlier versions used BEAD STDEV in place of BEAD STDERR - be sure to check that the `columns` argument is appropriate for your data). Equivalent arguments (`qc.sep`, `qc.skip` and `qc.columns`) are used to read the data from `qcFile`. See the help page (`?readBeadSummaryData`) for a complete description of each argument to the function.

12.1 Reading IDAT files

We can also read BeadArray data in the format produced directly by the scanner, the IDAT file. The example below uses the [GEOquery](#) to obtain the four IDAT files stored as supplementary information for GEO series GSE27073. In this case the stored files have been compressed using gzip and need to be decompressed before [beadarray](#) can read them. If you are using IDAT files as they come of the scanner this step will not be necessary.

```
library(beadarray)
library(GEOquery)
downloadDir <- tempdir()
getGEOSuppFiles("GSE27073", makeDirectory = FALSE, baseDir = downloadDir)
idatFiles <- list.files(path = downloadDir, pattern = ".idat.gz", full.names=TRUE)
sapply(idatFiles, gunzip)
idatFiles <- list.files(path = downloadDir, pattern = ".idat", full.names=TRUE)
BSData <- readIdatFiles(idatFiles)
```

The output from `readIdatFiles()` is an object of class *ExpressionSetIllumina*, as described earlier.

13 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

14 Asking for help on beadarray

Wherever possible, questions about *beadarray* should be sent to the Bioconductor mailing list¹. 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 using², and try to provide a reproducible example of your problem. This will help us to diagnose the problem.

This vignette was built with the following versions of R and

```
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] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
## [2] GenomicFeatures_1.22.8
## [3] ggbio_1.18.3
## [4] GenomicRanges_1.22.3
## [5] GenomeInfoDb_1.6.1
## [6] gridExtra_2.0.0
## [7] illuminaHumanv3.db_1.26.0
## [8] org.Hs.eg.db_3.2.3
## [9] RSQLite_1.0.0
## [10] DBI_0.3.1
## [11] AnnotationDbi_1.32.3
```

¹<http://www.bioconductor.org>

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

```
## [12] IRanges_2.4.6
## [13] S4Vectors_0.8.9
## [14] beadarrayExampleData_1.8.0
## [15] beadarray_2.20.1
## [16] ggplot2_2.0.0
## [17] Biobase_2.30.0
## [18] BiocGenerics_0.16.1
## [19] knitr_1.12
##
## loaded via a namespace (and not attached):
## [1] base64_1.1 Rcpp_0.12.3
## [3] biovizBase_1.18.0 lattice_0.20-33
## [5] Rsamtools_1.22.0 Biostrings_2.38.3
## [7] digest_0.6.9 plyr_1.8.3
## [9] futile.options_1.0.0 acepack_1.3-3.3
## [11] evaluate_0.8 BiocInstaller_1.20.1
## [13] highr_0.5.1 zlibbioc_1.16.0
## [15] hexbin_1.27.1 rpart_4.1-10
## [17] Matrix_1.2-3 labeling_0.3
## [19] splines_3.2.3 BiocParallel_1.4.3
## [21] stringr_1.0.0 foreign_0.8-66
## [23] biomaRt_2.26.1 RCurl_1.95-4.7
## [25] munsell_0.4.2 rtracklayer_1.30.1
## [27] mgcv_1.8-10 nnet_7.3-11
## [29] SummarizedExperiment_1.0.2 Hmisc_3.17-1
## [31] XML_3.98-1.3 reshape_0.8.5
## [33] GenomicAlignments_1.6.3 bitops_1.0-6
## [35] RBGL_1.46.0 grid_3.2.3
## [37] nlme_3.1-122 GGally_1.0.1
## [39] gtable_0.1.2 magrittr_1.5
## [41] formatR_1.2.1 scales_0.3.0
## [43] graph_1.48.0 stringi_1.0-1
## [45] XVector_0.10.0 reshape2_1.4.1
## [47] limma_3.26.5 latticeExtra_0.6-26
## [49] futile.logger_1.4.1 Formula_1.2-1
## [51] BiocStyle_1.8.0 lambda.r_1.1.7
## [53] RColorBrewer_1.1-2 BeadDataPackR_1.22.0
## [55] tools_3.2.3 illuminaio_0.12.0
## [57] dichromat_2.0-0 OrganismDbi_1.12.1
## [59] BSgenome_1.38.0 survival_2.38-3
## [61] colorspace_1.2-6 cluster_2.0.3
## [63] VariantAnnotation_1.16.4
```