Analysis of Bead-summary Data using beadarray

Mark Dunning

October 8, 2012

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

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

```
             G:4613710017_B G:4613710052_B G:4613710054_B G:4616443079_B
ILMN_1802380      0.2833023      0.3367157      0.2750020      0.4141796
ILMN_1893287      0.3963681      0.3882834      0.5516421      0.6761106
ILMN_1736104      0.4704854      0.4951260      0.4031143      0.5276266
ILMN_1792389      0.4038533      0.4728013      0.5032908      0.3447242
ILMN_1854015      0.5663066      0.3783570      0.5511991      0.5358812
             G:4616443093_B
ILMN_1802380      0.3581862
ILMN_1893287      0.4448673
ILMN_1736104      0.4864355
ILMN_1792389      0.3951935
ILMN_1854015      0.6748219
```

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
                         2                           2
cy3_hyb,low_stringency_hyb              housekeeping
                         4                           7
                  labeling         low_stringency_hyb
                         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
```

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.

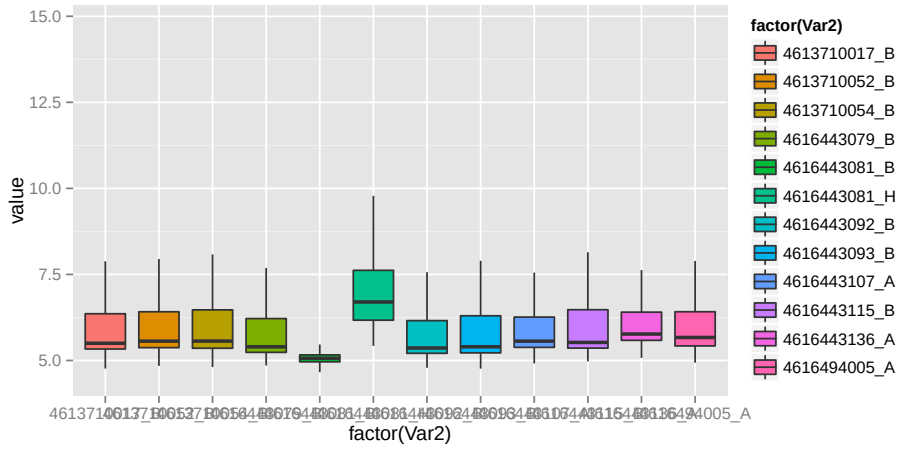
```
> randIDs <- sample(featureNames(exampleSummaryData), 1000)
> exampleSummaryData[randIDs, ]

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_1738468 ILMN_1676588 ... ILMN_2221769 (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
```

Exploratory analysis using boxplots

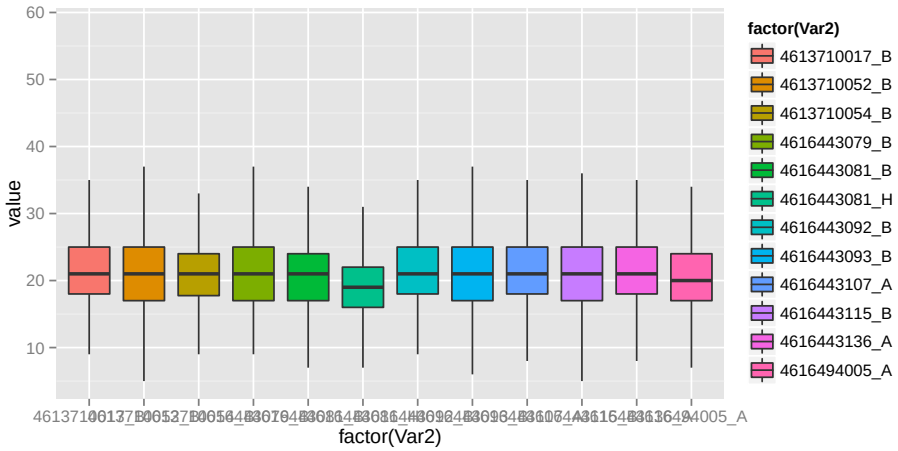
Boxplots of intensity levels and the number of beads are useful for quality assessment purposes. beadarray includes a modified version of the `boxplot` function that can take any valid *ExpressionSetIllumina* object and plot the expression matrix by default. For these examples we plot just a subset of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

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



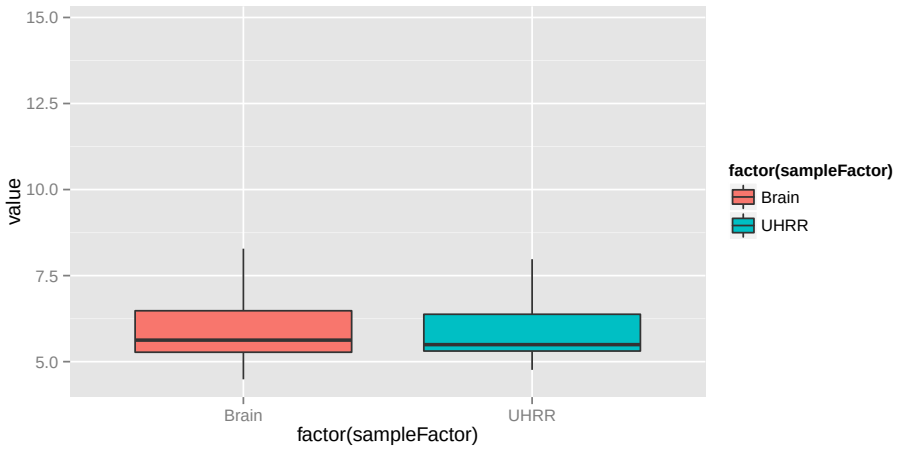
The function can also plot other `assayData` items, such as the number of observations.

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



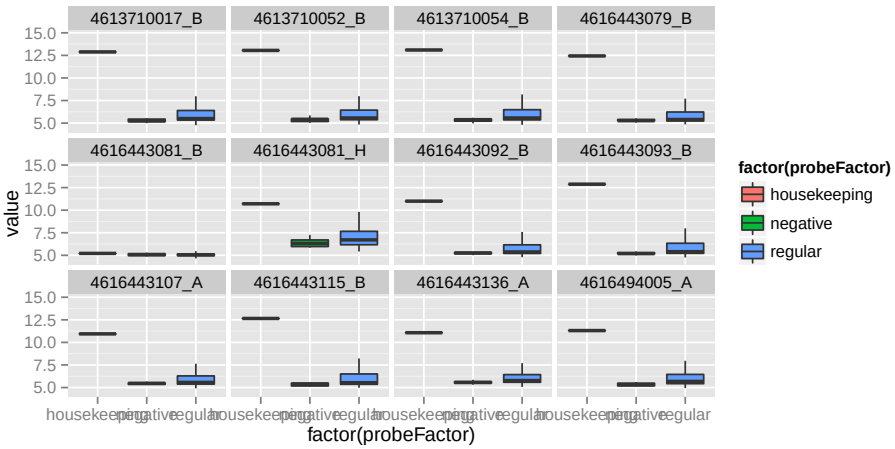
The default boxplot plots a separate box for each array, but often it is beneficial for compare expression levels between different sample groups. If this information is stored in the *phenoData* slot it can be incorporated into the plot. The following compares the overall expression level between UHRR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

```
> boxplot(exampleSummaryData.log2[randIDs, ], sampleFactor = "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 as 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
ILMN_1802380 ILMN_1802380          10008 ILMN_1802380 regular   RERE
ILMN_1893287 ILMN_1893287          10010 ILMN_1893287 regular   <NA>
ILMN_1736104 ILMN_1736104          10017 ILMN_1736104 regular   <NA>
ILMN_1792389 ILMN_1792389          10019 ILMN_1792389 regular RNF165
ILMN_1854015 ILMN_1854015          10020 ILMN_1854015 regular   <NA>
ILMN_1904757 ILMN_1904757          10021 ILMN_1904757 regular   <NA>
             PROBEQUALITY      CODINGZONE
ILMN_1802380      Perfect  Transcriptomic
ILMN_1893287          Bad Transcriptomic?
ILMN_1736104          Bad      Intergenic
ILMN_1792389      Perfect  Transcriptomic
ILMN_1854015          Bad      Intergenic
ILMN_1904757   Perfect*** Transcriptomic?
                                                    PROBESEQUENCE
ILMN_1802380 GCCCTGACCTTCATGGTGTCTTTGAAGCCCAACCACTCGGTTTCCTTCGG
ILMN_1893287 GGATTTCCTACACTCTCCACTTCTGAATGCTTGGAAACACTTGCCATGCT
ILMN_1736104 TGCCATCTTTGCTCCACTGTGAGAGGCTGCTCACACCACCCCCTACATGC
ILMN_1792389 CTGTAGCAACGTCTGTCAGGCCCCCTTGTGTTTCATCTCCTGCGCGCGTA
ILMN_1854015 GCAGAAAACCATGAGCTGAAATCTCTACAGGAACCAGTGCTGGGGTAGGG
```

```
ILMN_1904757 AGCTGTACCGTGGGGAGGCTTGGTCCTCTTGCCCCATTTGTGTGATGTCT
                       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 60149 mapped keys (of 94889 keys)
illuminaHumanv3CHR has 29867 mapped keys (of 49576 keys)
illuminaHumanv3CHRLENGTHS has 93 mapped keys (of 93 keys)
illuminaHumanv3CHRLOC has 29299 mapped keys (of 49576 keys)
illuminaHumanv3CHRLOCEND has 29299 mapped keys (of 49576 keys)
illuminaHumanv3ENSEMBL has 29041 mapped keys (of 49576 keys)
illuminaHumanv3ENSEMBL2PROBE has 20831 mapped keys (of 25527 keys)
illuminaHumanv3ENTREZID has 29868 mapped keys (of 49576 keys)
illuminaHumanv3ENZYME has 3521 mapped keys (of 49576 keys)
illuminaHumanv3ENZYME2PROBE has 967 mapped keys (of 975 keys)
illuminaHumanv3GENENAME has 29868 mapped keys (of 49576 keys)
illuminaHumanv3GO has 26277 mapped keys (of 49576 keys)
illuminaHumanv3GO2ALLPROBES has 16472 mapped keys (of 16527 keys)
illuminaHumanv3GO2PROBE has 12743 mapped keys (of 12818 keys)
illuminaHumanv3MAP has 29559 mapped keys (of 49576 keys)
illuminaHumanv3OMIM has 20993 mapped keys (of 49576 keys)
illuminaHumanv3PATH has 9174 mapped keys (of 49576 keys)
illuminaHumanv3PATH2PROBE has 229 mapped keys (of 229 keys)
illuminaHumanv3PFAM has 27593 mapped keys (of 49576 keys)
illuminaHumanv3PMID has 29400 mapped keys (of 49576 keys)
illuminaHumanv3PMID2PROBE has 344773 mapped keys (of 354292 keys)
illuminaHumanv3PROSITE has 27593 mapped keys (of 49576 keys)
illuminaHumanv3REFSEQ has 29868 mapped keys (of 49576 keys)
illuminaHumanv3SYMBOL has 29868 mapped keys (of 49576 keys)
illuminaHumanv3UNIGENE has 29527 mapped keys (of 49576 keys)
illuminaHumanv3UNIPROT has 27646 mapped keys (of 49576 keys)

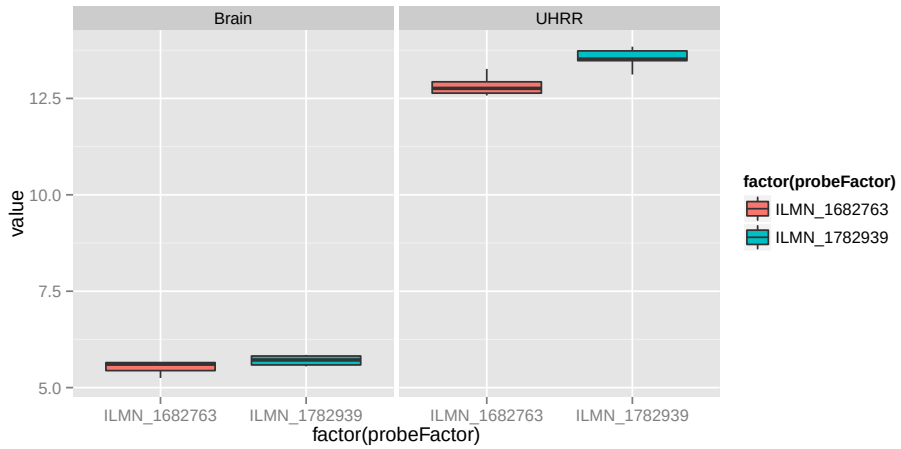
Additional Information about this package:

DB schema: HUMANCHIP_DB
DB schema version: 2.1

```
Organism: Homo sapiens
Date for NCBI data: 2012-Sep4
Date for GO data: 20120901
Date for KEGG data: 2011-Mar15
Date for Golden Path data: 2010-Mar22
Date for Ensembl data: 2012-Jul31


####Custom Mappings based on probe sequence####


illuminaHumanv3ARRAYADDRESS()
illuminaHumanv3NUID()
illuminaHumanv3PROBEQUALITY()
illuminaHumanv3CODINGZONE()
illuminaHumanv3PROBESEQUENCE()
illuminaHumanv3SECONDMATCHES()
illuminaHumanv3OTHERGENOMICMATCHES()
illuminaHumanv3REPEATMASK()
illuminaHumanv3OVERLAPPINGSNP()
illuminaHumanv3ENTREZREANNOTATED()
illuminaHumanv3GENOMICLOCATION()
illuminaHumanv3SYMBOLREANNOTATED()
illuminaHumanv3REPORTERGROUPNAME()
illuminaHumanv3REPORTERGROUPID()
```

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, ], sampleFactor = "SampleFac",
+     probeFactor = "IlluminaID")
```



A note about ggplot2

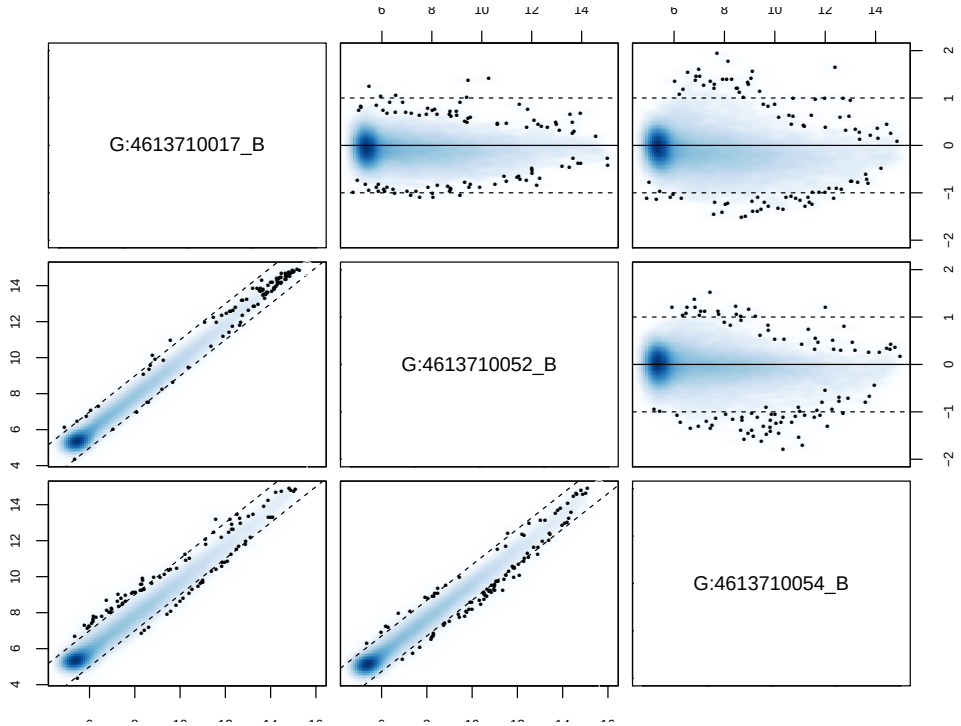
The `boxplot` function in beadarray creates graphics using the ggplot2 package rather than the `R` base graphics system. Therefore, the standard way of manipulating graphics using `par` and `mfrow` etc will not work with the output of `boxplot`. However, the ggplot2 package has equivalent functionality 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 viewports (a concept from the `grid` graphics system). The code also demonstrates how aspects of the plot can be altered programatically.

```
> require("gridExtra")
> bp1 <- boxplot(exampleSummaryData.log2[ids, ], sampleFactor = "SampleFac",
+     probeFactor = "IlluminaID") + labs(title = "ALB expression level comparison") +
+     xlab("Illumina Probe") + ylab("Log2 Intensity")
> bp2 <- boxplot(exampleSummaryData.log2[randIDs, ], probeFactor = "Status") +
+     labs(title = "Control Probe Comparison")
> print(bp1, vp = viewport(width = 0.5, height = 1, x = 0.25, y = 0.5))
> print(bp2, vp = viewport(width = 0.5, height = 1, x = 0.75, y = 0.5))
```

Other exploratory analysis

Replicate samples can also be compared using the `plotMAXY`.

```
> plotMAXY(exprs(exampleSummaryData), arrays = 1:3, pch = 16, log = FALSE)
```



In the top right corner we see the MA plots for all pairwise comparisons involving the 3 arrays. 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. For replicate arrays we would expect all probes to be unchanged between the two samples and hence most points on the plot should 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 second array is systematically different to the other replicates and may benefit from normalisation. Both XY and MA plots are available separately for a particular comparison of arrays using plotXY and plotMA.

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")
```

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. You can verify the relationship between probe quality and intensity by using the boxplot function.

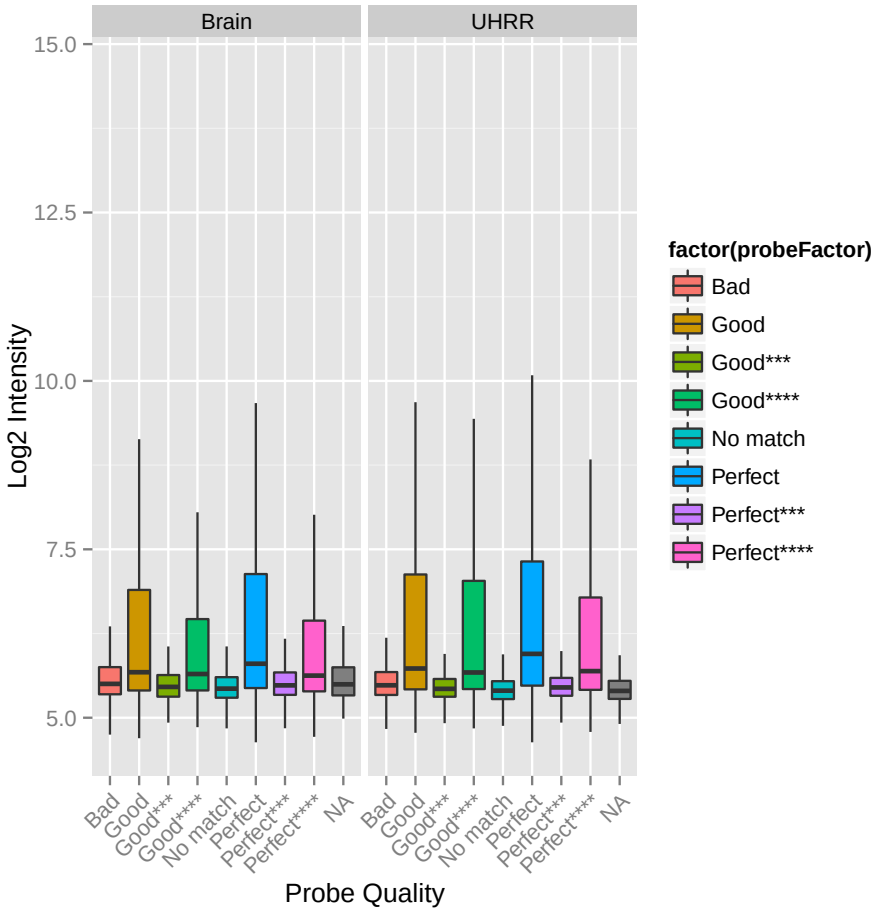
```
> library(illuminaHumanv3.db)
> ids <- as.character(featureNames(exampleSummaryData.norm))
> qual <- unlist(mget(ids, illuminaHumanv3PROBEQUALITY, ifnotfound = NA))
> table(qual)
```

```
qual
        Bad        Good     Good***    Good****    No match     Perfect
      13472         924         148         358        1739       24654
 Perfect*** Perfect****
       6269        1974
```

```
> rem <- qual == "No match" | qual == "Bad" | is.na(qual)
> exampleSummaryData.filt <- exampleSummaryData.norm[!rem, ]
> dim(exampleSummaryData.filt)
```

```
Features  Samples Channels
   34327       12        1
```

```
> boxplot(exampleSummaryData.norm, probeFactor = "PROBEQUALITY",
+     sampleFactor = "SampleFac") + opts(axis.text.x = theme_text(angle = 45,
+     hjust = 1)) + xlab("Probe Quality") + ylab("Log2 Intensity")
```



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.

```
> library(limma)
> rna <- factor(pData(exampleSummaryData)[, "SampleFac"])
> design <- model.matrix(~0 + rna)
> colnames(design) <- levels(rna)
> aw <- arrayWeights(exprs(exampleSummaryData.filt), design)
> aw

        1          2          3          4          5          6          7
2.08980504 2.52705232 1.45341722 1.77516015 2.13370848 1.85736443 0.01232656
        8          9         10         11         12
0.11157875 2.45025440 2.04457347 2.08860721 1.28684347

> 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)

                ID     logFC  AveExpr         t      P.Value    adj.P.Val
22033 ILMN_1651358  7.344616 9.202611  87.76608 5.232909e-34 1.638191e-29
2043  ILMN_1796678  7.320723 9.608215  85.77754 9.544621e-34 1.638191e-29
31831 ILMN_1713458  6.418935 8.954098  78.21252 1.073627e-32 1.196177e-28
33986 ILMN_1783832  5.972720 8.323385  77.43724 1.393862e-32 1.196177e-28
3174  ILMN_1782939  6.822236 9.310045  76.20004 2.125596e-32 1.459307e-28
28936 ILMN_1688543  6.708263 9.032077  73.23584 6.009507e-32 3.074085e-28
3879  ILMN_1795679 -7.150692 9.069547 -73.11785 6.268709e-32 3.074085e-28
17887 ILMN_2084825  7.980276 9.899318  72.68451 7.324581e-32 3.142886e-28
9196  ILMN_1782204  6.275295 8.666149  69.87509 2.055919e-31 7.841505e-28
10390 ILMN_1665994  6.693550 8.725046  67.50368 5.075224e-31 1.742172e-27
            B
22033 67.03462
2043  66.50542
31831 64.33905
33986 64.10221
3174  63.71812
28936 62.76583
3879  62.72696
17887 62.58353
9196  61.62786
10390 60.78467
```

Annotation of results

The `topTable` function displays the results of the empirical Bayes analysis alongside the annotation assigned by Illumina to each probe in the linear model fit. Often this will not provide sufficient information to infer biological meaning from the results. Within Bioconductor, annotation packages are available for each of the major Illumina expression array platforms that map the probe sequences designed by Illumina to functional information useful for downstream analysis. As before, the illuminaHumanv3.db package can be used for the arrays in this example dataset.

```
> ids2 <- featureNames(exampleSummaryData.filt)
> chr <- mget(ids2, illuminaHumanv3CHR, ifnotfound = NA)
> cytoband <- mget(ids2, illuminaHumanv3MAP, ifnotfound = NA)
> refseq <- mget(ids2, illuminaHumanv3REFSEQ, ifnotfound = NA)
> entrezid <- mget(ids2, illuminaHumanv3ENTREZID, ifnotfound = NA)
> symbol <- mget(ids2, illuminaHumanv3SYMBOL, ifnotfound = NA)
> genename <- mget(ids2, illuminaHumanv3GENENAME, ifnotfound = NA)
> anno <- data.frame(Ill_ID = ids2, Chr = as.character(chr), Cytoband = as.character(cytoband),
+     RefSeq = as.character(refseq), EntrezID = as.numeric(entrezid),
+     Symbol = as.character(symbol), Name = as.character(genename))
> contr.fit$genes <- anno
> topTable(contr.fit)

           Ill_ID Chr     Cytoband
22033 ILMN_1651358  11      11p15.5
2043  ILMN_1796678  11      11p15.5
31831 ILMN_1713458  16      16p13.3
33986 ILMN_1783832   X Xp11.4-p11.2
3174  ILMN_1782939   4       4q13.3
28936 ILMN_1688543   1     1q21-q23
3879  ILMN_1795679   8      8q21.13
17887 ILMN_2084825  11      11p15.5
9196  ILMN_1782204   4       4q13.3
10390 ILMN_1665994  12    12q13-q14
                                                                                           RefSeq
22033                                                                c("NM_005330", "NP_005321")
2043                                                                 c("NM_000559", "NP_000550")
31831                                                                c("NM_005332", "NP_005323")
33986                                                                c("NM_001476", "NP_001467")
3174                                                                 c("NM_000477", "NP_000468")
28936                                                                c("NM_001643", "NP_001634")
3879                              c("NM_001199214", "NM_007029", "NP_001186143", "NP_008960")
17887                                                                c("NM_000184", "NP_000175")
9196                                                                 c("NM_001134", "NP_001125")
10390 c("NM_001200053", "NM_001200054", "NM_006928", "NP_001186982", "NP_001186983", "NP_008859")
      EntrezID Symbol                  Name     logFC  AveExpr         t
22033     3046   HBE1 hemoglobin, epsilon 1  7.344616 9.202611  87.76608
2043      3047   HBG1   hemoglobin, gamma A  7.320723 9.608215  85.77754
31831     3050    HBZ      hemoglobin, zeta  6.418935 8.954098  78.21252
33986     2578  GAGE6           G antigen 6  5.972720 8.323385  77.43724
3174       213    ALB               albumin  6.822236 9.310045  76.20004
28936      336  APOA2   apolipoprotein A-II  6.708263 9.032077  73.23584
3879     11075  STMN2       stathmin-like 2 -7.150692 9.069547 -73.11785
17887     3048   HBG2   hemoglobin, gamma G  7.980276 9.899318  72.68451
9196       174    AFP     alpha-fetoprotein  6.275295 8.666149  69.87509
10390     6490   PMEL premelanosome protein  6.693550 8.725046  67.50368
           P.Value    adj.P.Val        B
22033 5.232909e-34 1.638191e-29 67.03462
2043  9.544621e-34 1.638191e-29 66.50542
31831 1.073627e-32 1.196177e-28 64.33905
33986 1.393862e-32 1.196177e-28 64.10221
```

```
3174  2.125596e-32 1.459307e-28 63.71812
28936 6.009507e-32 3.074085e-28 62.76583
3879  6.268709e-32 3.074085e-28 62.72696
17887 7.324581e-32 3.142886e-28 62.58353
9196  2.055919e-31 7.841505e-28 61.62786
10390 5.075224e-31 1.742172e-27 60.78467
```

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 in the file `AsuragenMAQC_BeadStudioOutput.zip` which can be downloaded from
tt http://www.switchtoi.com/datasets.ilmn.

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.

Citing beadarray

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

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

1 Asking for help on beadarray

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

This vignette was built with the following versions of R and

```
> sessionInfo()

R version 2.15.1 (2012-06-22)
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

locale:
[1] C/en_US.US-ASCII/en_US.US-ASCII/C/en_US.US-ASCII/en_US.US-ASCII

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] limma_3.14.0              gridExtra_0.9.1
```

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

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

```
 [3] illuminaHumanv3.db_1.16.0  org.Hs.eg.db_2.8.0
 [5] RSQLite_0.11.2             DBI_0.2-5
 [7] AnnotationDbi_1.20.0       beadarrayExampleData_1.0.5
 [9] beadarray_2.8.1            ggplot2_0.9.2.1
[11] Biobase_2.18.0             BiocGenerics_0.4.0

loaded via a namespace (and not attached):
 [1] AnnotationForge_1.0.0 BeadDataPackR_1.10.0  IRanges_1.16.2
 [4] KernSmooth_2.23-8     MASS_7.3-22           RColorBrewer_1.0-5
 [7] colorspace_1.1-1      dichromat_1.2-4       digest_0.5.2
[10] gtable_0.1.1          labeling_0.1          memoise_0.1
[13] munsell_0.4           parallel_2.15.1       plyr_1.7.1
[16] proto_0.3-9.2         reshape2_1.2.1        scales_0.2.2
[19] stats4_2.15.1         stringr_0.6.1         tools_2.15.1
```