Analysing Illumina bead-based data using beadarray

Matt Ritchie and Mark Dunning

September 30, 2008

Introduction

ℹ Illumina have created an alternative microarray technology (BeadArray) based on randomly arranged beads. A specific oligonucleotide sequence is assigned to each *bead type*, which is replicated about 30 times on an array. A series of decoding hybridisations is used to identify each bead on the array. The high degree of replication makes robust measurements for each bead type possible.
BeadArrays are used in many applications, including gene expression studies, SNP genotyping and methylation profiling and are processed in parallel as a Sentrix Array Matrix (SAM) or BeadChip. A SAM is a plate of 96 uniquely prepared hexagonal BeadArrays, each of which contains around 1,500 bead types. A BeadChip comprises of a series of rectangular strips on a slide with each strip containing about 24,000 bead types. For example, there are six pairs of strips on each Human-6 BeadChip. Depending on the particular assay used, the data from a BeadArray may be single channel or two-colour.
This practical describes how to process Illumina BeadArray data using the beadarray package. We demonstrate how to read in the raw files produced by the scanning software, produce diagnostic plots to assess data quality, summarise bead-level data and search for differentially expressed genes. Summarised BeadStudio output can also be processed by our package, and we refer the interested user to the package vignette for further details.

Expression analysis

ℹ The raw data you will need for this exercise is available in `SAMExample.zip` (46 Mb) which can be downloaded from
`http://www.compbio.group.cam.ac.uk/Resources/illumina/`.

Exercise 1: Unzip the contents of `SAMExample.zip` to the current working directory. This file contains tiffs and text files from 10 hexagonal BeadArrays from several SAMs in an experiment which aimed to measure gene expression differences between cell lines.

Description of files

ℹ Reading bead level data into the beadarray requires several files produced by Illumina's BeadScan software. We briefly describe these files below.

- text files (*required*) - a `.txt` or `.csv` file for each strip or hexagon which stores the position, identity and intensity of each bead. These files are usually named

  `chipID_array_strip.csv` for arrays from BeadChips (e.g. `1475542113_A_1.csv` or `samID_Row_Col.csv` for arrays from a SAM (e.g. `1318791_R001_C007.csv`) and are required because of the random arrangement of probes on the array surface, which is unique for each BeadArray.

- tiffs (*optional*) - 1 (single channel) or 2 (two-colour) for each strip on a BeadChip, or hexagon on a SAM. These are usually named using the convention

  `chipID_array_strip_channel.tif`, e.g. `1475542113_A_1_Grn.tif`, is the Cy3 (green) image for strip 1 from array A on BeadChip 1475542113. On a SAM, the naming convention is similar: `samID_Row_Col_Channel.tif`, e.g.

  `1318791_R001_C007_Grn.tif` for the Cy3 image from the array in row 1, column 7 on SAM 1318791. Cy5 (red) images end with the extension `_Red.tif`. Having the images allows the user the choice of different image processing methods, and access to the background intensities (the intensities in the text files have been background corrected).

- bead manifest file (*required*) - contains information about the non-control bead types on the array. See the file `genes.csv` for an example. Different arrays have different manifest files and use different annotation packages. Version 2 BeadChips store this information in a slightly different `.bgx` format, which is required in place of the manifest file for version 2 BeadChips. Other platforms have analogous files, such as the `.opa` for SNP BeadArrays. Annotation packages for some Illumina arrays are available in Bioconductor, for example the illuminaHumanv1 package can be used to obtain further information about the probes each bead type from a Human version 1 BeadChip targets. The manifest files needed to map the probe IDs from the arrays to the Illumina IDs used in the annotation package.

- targets file (*recommended*) - contains sample information for each array. See the file `targets.txt` for an example.

- metrics file (*optional*) - one for each each BeadChip or SAM, usually named

  `Metrics.txt` which contains summary information about intensity, the amount of saturation, focus and registration on the image(s) from each strip or hexagon. In this example, the metrics file is not available.

♠ *To obtain the tiffs and text files from BeadScan version 3.1 you will need to modify the* `settings.xml` *file used by the software. For further details see the Scanning Settings section of* `http://www.compbio.group.cam.ac.uk/Resources/illumina/`.

Exercise 2: Read the bead level data for the example BeadArrays into R using the `readIllumina` function. Look at the help page for this function to understand the arguments it accepts and the default settings. Use the file `targets.txt` to find which samples are hybridised to each array.

```
> library(beadarray)
> targets = read.table("targets.txt", sep = "\t", header = TRUE,
+     as.is = TRUE)
> targets
```

```
> beadInfo = read.csv("genes.csv", as.is = TRUE)
> BLData = readIllumina(arrayNames = targets$arrayID, textType = ".csv",
+     targets = targets, backgroundMethod = "none")
```

ℹ The arrays in this example are part of a study comparing expression in lymphoblastoid cell lines in different individuals using custom BeadArrays. The samples on these arrays are 5 replicates of cell lines from 2 individuals (A and B).
The function `readIllumina` implements the image processing steps used by Illumina when `useImages=TRUE`. However, both the sharpening and background correction steps used by default in BeadScan are optional, and can be controlled using `imageManipulation` and `backgroundMethod` arguments respectively. Refer to the `readIllumina` help page for further details. When the tiffs are ignored by setting `useImages=FALSE`, the background corrected values from the text files are stored in R, which can save on memory usage.

♠ *Storing and reading bead level data requires a considerable amount of disk space and RAM. If your computer has less than 2Gb of RAM, we recommend that you run* `readIllumina` *with* `useImages=FALSE`*, especially when reading raw BeadChip data, which has around 1 million beads per strip.*

The BLData object

Exercise 3: How is bead level data stored within beadarray and how is it accessed? Which hexagonal array on this SAM has the most beads? What is the ProbeID of the 100th bead on array 4?

```
> slotNames(BLData)
```

```
> an = arrayNames(BLData)
> an
```

```
> numBeads(BLData)
```

```
> pData(BLData)
```

```
> colnames(BLData[[1]])
```

```
> BLData[[1]]$G[1:5]
```

```
> BLData[[1]]$Gb[1:5]
```

```
> getArrayData(BLData, array = 1, what = "G", log = FALSE)[1:5]
```

```
> getArrayData(BLData, array = 1, what = "Gb", log = FALSE)[1:5]
```

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

Exploring the data

ℹ Boxplots can be used to compare foreground or background intensities between arrays. Background correction can be performed by the `backgroundCorrect` function if it has not already been carried out when the data was read into R by `readIllumina`. The default setting in both functions is to subtract the local background estimate from the foreground of each bead, as per BeadScan. Other options are available by changing the `method` argument in `backgroundCorrect` or the `backgroundMethod` argument in `readIllumina`.

Exercise 4: Using the plotting features in beadarray, examine the foreground and background intensities from the example BeadArrays. How many negative corrected values are there on each array? Are there any unusual arrays, or trends you notice? What do you notice about the background signal?

```
> BLData.bc = backgroundCorrect(BLData)
> negs = NULL
> for (i in 1:10) {
+     negs[i] = sum(getArrayData(BLData.bc, array = i,
+         log = FALSE) < 0)
+ }
> negs

> ylim = c(4, 16)
> par(mfrow = c(1, 3), mai = c(1.5, 0.6, 0.2, 0.1))
> boxplotBeads(BLData, las = 2, outline = FALSE, ylim = ylim,
+     main = "fg", ylab = expression(log[2](intensity)))
> boxplotBeads(BLData, las = 2, whatToPlot = "Gb", outline = FALSE,
+     ylim = ylim, main = "bg")
> boxplotBeads(BLData.bc, las = 2, whatToPlot = "G", outline = FALSE,
+     ylim = ylim, main = "fg-bg")
```

ℹ The `whatToPlot` argument of `boxplotBeads` controls which intensities are plotted for each bead. Options are `G`, `Gb` and `residG` (Cy3 residuals) for single channel data and `R`, `Rb`, `residR`, `M` (log-ratios) `residM` or `A` (average log-intensities) for two-colour data.

ℹ Spatial artefacts on the array surface can occur from mis-handling or scanning problems. Imageplots can be used to identify these artefacts, and with the raw bead-level data, we can plot false images of each array. This kind of visualisation is not possible when using the summarised BeadStudio output, as the summary values are averaged over spatial positions. Imageplots in R are also more convenient than scrutinising the original tiffs, as multiple arrays can be visualised on the one page.

Exercise 5: Use the `imageplot` function to look for spatial trends on arrays. Are there any arrays that you would consider removing from the analysis? Read the help page for this function to find out more about the arguments it accepts.

```
> par(mfrow = c(2, 5))
> zlim = c(6, 16)
> for (i in 1:10) {
+     imageplot(BLData.bc, array = i, nrow = 50, ncol = 50,
```

```
+         high = "red", low = "yellow", zlim = zlim, main = an[i])
+ }

> x11()
> par(mfrow = c(2, 5))
> zlim = c(-1, 1)
> for (i in 1:10) {
+     imageplot(BLData.bc, whatToPlot = "residG", array = i,
+         nrow = 50, ncol = 50, high = "red", low = "yellow",
+         zlim = zlim, main = an[i])
+ }
```

ℹ In the `imageplot` function, the argument `whatToPlot` is used to choose the quantity to display. For single channel data, `whatToPlot` can be set to `G`, `Gb` or `residG` to plot the Cy3 foreground, background or foreground residuals respectively. For two-colour data, the Cy5 foreground (`R`), background (`Rb`), log-ratios (`M`), average log-intensities (`A`) and residuals (`residR`, `residM`) can be plotted by changing `whatToPlot`. Because of the high number of beads on each array, the `imageplot` function maps a grid of size specified by the `nrow` and `ncol` arguments onto the array surface and averages the intensities of the beads within each section of the grid.

ℹ Recall that the BeadArray technology includes around 30 replicates of each bead type on every array and by default, BeadStudio removes outliers greater than 3 median absolute deviations (MADs) from the median prior to calculating the bead summary values.

Exercise 6: Find out where the outliers occur on each array and plot their location. Do the locations correspond with the trends you observed in the imageplots in the previous exercise? Calculate and plot the number of outliers on each array.

```
> par(mfrow = c(2, 5))
> for (i in 1:10) {
+     o = findAllOutliers(BLData.bc, array = i)
+     plotBeadLocations(BLData.bc, array = i, BeadIDs = o,
+         main = an[i], SAM = TRUE, pch = ".")
+ }
> outliers = NULL
> for (i in 1:10) {
+     outliers[i] = length(findAllOutliers(BLData.bc, array = i))
+ }
> x11()
> par(mai = c(2, 1, 0.2, 0.1))
> barplot(outliers/numBeads(BLData.bc) * 100, main = "Outliers per array",
+     ylab = "%", las = 2, names = an)
```

☞ In this example, the regions of the arrays which appear to be spatial artefacts are also flagged as outliers, which would be removed by BeadStudio before creating summarised values for each bead type. It is important to note that if a very large fraction of the array is affected, the converse may occur, i.e. the 'good' section of the array may appear to be the outlier region, which users need to be aware of.

The number of outliers could be used as an ad-hoc criterion for removing poor quality arrays.

Summarising the data

ℹ The replicate values on each array can be summarised using the default method in BeadStudio, which removes outliers greater than 3 median absolute deviations (MADs) from the median and calculates the mean and standard error of the intensities from the remaining beads.

Exercise 7: Create bead summary data for the example BeadArrays. What does the `imagesPerArray` argument control and why do we need to set it to `1` in this example? How is information stored in the `BSData` object? How many different bead types are on these arrays?

```
> BSData = createBeadSummaryData(BLData.bc, imagesPerArray = 1,
+     method = "illumina")
> slotNames(BSData)

> names(assayData(BSData))

> dim(exprs(BSData))

> dim(se.exprs(BSData))

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

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

> pData(BSData)

> par(mai = c(2, 1, 0.2, 0.1), mfrow = c(1, 2))
> boxplot(as.data.frame(log2(exprs((BSData)))), ylab = expression(log[2](intensity)),
+     las = 2)
> boxplot(as.data.frame(NoBeads(BSData)), ylab = "number of beads",
+     ylim = c(0, 60), las = 2)
```

☞ Other summarisation options are available by changing the `method` argument in `createBeadSummaryData`.

ℹ `BSData` is an object of type *ExpressionSetIllumina* which is an extension of the *ExpressionSet* class developed by the Biocore team used as a container for 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 now refer to the expression values as the `exprs` matrix which can be accessed using `exprs` and subset in the usual manner. The `se.exprs` matrix can be accessed using `se.exprs`, and phenotypic data for the experiment can be accessed using `pData`.

Visualising and normalising summary data

ℹ beadarray provides a way of displaying MA and XY plots for a set of arrays. We call this a *MAXY plot*. On an MA plot, for each gene we plot the average of the expression levels on the two arrays on the x axis and the difference in the measurements on the y axis. For replicate arrays we would expect all genes to be unchanged between the two samples and hence most points on the plot to lie along the line y=0. On the XY plot for replicate arrays we would expect to see most points along the diagonal $y = x$. The MAXY plot shows the MA plots for the arrays in the upper right and XY plots in the lower left.
Both XY and MA plots for a particular comparison of arrays are available separately using `plotXY` and `plotMA`.

Exercise 8: Make MA and XY plots of replicates of the two samples in the experiment. Are the results consistent with the boxplots from the previous exercise?

```
> plotMAXY(exprs(BSData), arrays = 1:5)
> x11()
> plotMAXY(exprs(BSData), arrays = 6:10)
```

Exercise 9: Log_2 transform the data and apply quantile normalisation. Plot the results.

```
> plotDensity(log(exprs(BSData)))
> BSData.log2.quantile = normaliseIllumina(BSData, method = "quantile",
+     transform = "log2")
> plotMAXY(exprs(BSData.log2.quantile), arrays = 1:5, log = FALSE)
```

♠ *If you do a MAXY plot of the normalised data or if you are plotting the non-normalised data obtained from* `createBeadSummaryData` *with* `log=TRUE` *you will need to set* `log=FALSE` *in* `plotMAXY` *to avoid taking the log twice. Read the help page for* `plotMAXY` *for further information.*

☞ After examining a range of diagnostic plots, we are suspicious about the quality of the first array. In the next section, we will look at whether removing this outlier array from further analysis is beneficial.

Differential expression analysis

ℹ Further analysis can be carried out on the summarised data to search for differentially expressed genes using other Bioconductor packages, such as limma. As we are dealing with single-channel expression data, it is necessary to set up a contrast (A-B) to assess differential expression between the samples.

Exercise 10: Fit a linear model to the quantile normalised data and set up a contrast to identify candidate differentially expressed genes between samples A and B. What are the top 10 genes from this analysis? Compare the results to those obtained using only the good quality arrays identified previously. What effect does removing bad data have on the analysis?

```
> design = cbind(A = rep(c(1, 0), each = 5), B = rep(c(0,
+     1), each = 5))
> design

> fit1 = lmFit(exprs(BSData.log2.quantile), design)
> contr.fit1 = contrasts.fit(fit1, contrasts = c(1, -1))
> contr.fit1 = eBayes(contr.fit1)
> topTable(contr.fit1, coef = 1)

> badarray = 1
> fit2 = lmFit(exprs(BSData.log2.quantile)[, -badarray],
+     design[-badarray, ])
> contr.fit2 = contrasts.fit(fit2, contrasts = c(1, -1))
> contr.fit2 = eBayes(contr.fit2)
> topTable(contr.fit2, coef = 1)

> xlim = c(-1.1, 1.7)
> ylim = c(-7, 20)
> par(mfrow = c(1, 2))
> volcanoplot(contr.fit1, main = "Results from full data set",
+     xlim = xlim, ylim = ylim)
> volcanoplot(contr.fit2, main = "Results after removing poor quality data",
+     xlim = xlim, ylim = ylim)
```

Annotation

ℹ Annotation information can be obtained from the bead manifest file, or from annotation packages available from Bioconductor. For the custom arrays used in this experiment, we will use the information from the manifest file, `genes.csv` which is stored in `beadInfo`. To match the ordering of 'ProbeIDs' in our `BSData` (or linear model output `contr.fit2` - the probes are in the same order in each object) with those in the 'ProbeID' column of the manifest file, we use the `match` function.

Exercise 11: Add transcript annotation information to the results and write them out to file.

```
> rownames(exprs(BSData))[1:5]

> contr.fit2$genes[1:5, 1]

> beadIDs = rownames(exprs(BSData))
> index = match(beadIDs, beadInfo$ProbeID)
> anno = beadInfo[index, ]
> contr.fit2$genes = anno
> topTable(contr.fit2, coef = 1)

> write.fit(contr.fit2, file = "results.txt")
```

♠ *Note that in this example, the control ProbeIDs do not have a match in the bead manifest file.*

SNP analysis

ℹ BeadArrays can also be used for SNP genotyping. In this example, we explore the raw data from 4 BeadArrays, each of which types 1,500 SNPs in a different sample using Illumina's GoldenGate assay. This assay produces two-colour data, with the Cy3 and Cy5 channels measuring allele A and allele B hybridisation respectively. This data set is provided as one of the example data sets in beadarray.

Exercise 12: Load the raw SNP data and confirm that it is two-colour. What is the Cy5 background intensity for the 10th bead on array 4?

```
> data(BLData)
> numBeads(BLData)

> colnames(BLData[[1]])

> getArrayData(BLData, array = 1, what = "G", log = FALSE)[1:5]

> getArrayData(BLData, array = 1, what = "R", log = FALSE)[1:5]
```

Plotting two-colour data

ℹ Various plotting functions for two-colour data are offered in beadarray. Boxplots of the bead level data can be generated using `boxplotBeads` as before and changing the `whatToPlot` argument (`R`, `residR`, `M` `residM` and `A` can be specified for two-colour data). Plots of the Cy5 versus Cy3 intensities for all probes on a given array, or for particular SNPs between arrays, and density plots of the signal from each channel are also possible.

Exercise 13: Make a plot of the Cy5 versus Cy3 intensities for arrays 1 and 2. Notice the distinct clouds associated with the AA, AB and BB genotypes for each sample. What do you think the fourth cloud represents? Plot the Cy3 and Cy5 densities for each array. In light of what you see in this plot, how might you normalise this data?

```
> x11()
> par(mfrow = c(1, 2))
> plotRG(BLData, arrays = 1)
> plotRG(BLData, arrays = 2)
> x11()
> plotBeadDensities(BLData, whatToPlot = c("G", "R"), col = rep(c(4,
+     7), times = 4), lwd = 2)
> legend(13, 1.1, legend = c("Cy3", "Cy5"), col = c(4,
+     7), lwd = 2)
```

Exercise 14: Plot the raw bead intensities for the SNPs with IDs 1037 and 913. Can you distinguish between the 3 genotypes for these SNPs?

```
> x11()
> par(mfrow = c(1, 2))
> plotRG(BLData, ProbeIDs = "1037", arrays = 1:4, main = "SNP 1037",
+     pch = 16)
> plotRG(BLData, ProbeIDs = "913", arrays = 1:4, main = "SNP 913",
+     pch = 16)
```

ℹ Having access to the raw bead-level data means that we can calculate variances and covariances for each SNP on each array on the original or log-scale. These quantities may be useful in genotype calling algorithms.

Exercise 15: Summarise the bead-level data to obtain average RG, A and M values for each SNP on the log_2 scale. Make boxplots of the summarised log-ratios from each array.

```
> RG = createBeadSummaryData(BLData, what = "RG", log = TRUE)

> class(RG)

> slotNames(RG)

> names(RG@assayData)

> A = createBeadSummaryData(BLData, what = "A", log = TRUE)
> M = createBeadSummaryData(BLData, what = "M", log = TRUE)
> dim(exprs(M))

> x11()
> boxplot(as.data.frame(exprs(M)), ylab = expression(log[2](R/G)))
```

ℹ The `RG` data is stored in an object of class *SnpSetIllumina* from the beadarraySNP package. See the *SnpSetIllumina* help page for further details about this class.

ℹ The version of R and the packages used to complete this tutorial are listed below. The beadarray package is being actively developed to accommodate new applications of BeadArrays, and to suit changing data formats, which means from time-to-time, some of the commands may be slightly different to the ones listed in this tutorial. Refer to the help page of individual functions to check the current arguments if you are using a more recent version of beadarray. If you have further questions about using beadarray, please email the Bioconductor mailing list (`bioconductor@stat.math.ethz.ch`).

```
> sessionInfo()
```

Acknowledgements

We are grateful to Barbara Stranger and Matthew Forrest for allowing us to use their data in this tutorial and Inma Spiteri for providing the BeadChip data set distributed with the beadarray package. We also thank Julie Addison, Tom Hardcastle, John Marioni, Inma Spiteri and Anna Git for their helpful feedback on the exercises in this tutorial.