The minfi User's Guide
Analyzing Illumina 450k Methylation Arrays

Modified by Sean Davis

Original by Kasper D. Hansen and Martin J. Aryee

Compiled: February 6, 2014

1 Introduction

The *minfi* package provides tools for analyzing Illumina's Methylation arrays, with a special focus on the new 450k array for humans. At the moment Illumina's 27k methylation arrays are not supported.

The tasks addressed in this package include preprocessing, QC assessments, identification of interesting methylation loci and plotting functionality. Analyzing these types of arrays is ongoing research in ours and others groups. In general, the analysis of 450k data is not straightforward and we anticipate many advances in this area in the near future.

The input data to this package are IDAT files, representing two different color channels prior to normalization. It is possible to use Genome Studio files together with the data structures contained in this package, but in general Genome Studio files are already normalized and we do not recommend this.

Chip design and terminology

The 450k array has a complicated design. What follows is a quick overview.

Each sample is measured on a single array, in two different color channels (red and green). Each array measures roughly 450,000 CpG positions. Each CpG is associated with two measurements: a methylated measurement and an "un"-methylated measurement. These two values can be measured in one of two ways: using a "Type I" design or a "Type II design". CpGs measured using a Type I design are measured using a single color, with two different probes in the same color channel providing the methylated and the unmethylated measurements. CpGs measured using a Type II design are measured using a single probe, and two different colors provide the methylated and the unmethylated measurements. Practically, this implies that on this array there is *not* a one-to-one correspondence between probes and CpG positions. We have therefore tried to be precise about this and we refer to a "methylation position" (or "CpG") when we refer to a single-base genomic locus. The previous generation 27k methlation array uses only the Type I design.

In this package we refer to differentially methylated positions (DMPs) by which we mean a single genomic position that has a different methylation level in two different groups of samples (or conditions). This is different from differentially methylated regions (DMRs) which imply more that more than one methylation positions are different between conditions.

Physically, each sample is measured on a single "array". There are 12 arrays on a single physical "slide" (organized in a 6 by 2 grid). Slides are organized into "plates" containing at most 8 slides (96 arrays).

Workflow and R data classes

A set of 450k data files will initially be read into an `RGChannelSet`, representing the raw intensities as two matrices: one being the green channel and one being the red channel. This is a class which is very similar to an `ExpressionSet` or an `NChannelSet`.

The `RGChannelSet` is, together with a `IlluminaMethylationManifest` object, preprocessed into a `MethylSet`. The `IlluminaMethylationManifest` object contains the array design, and describes how probes and color channels are paired together to measure the methylation level at a specific CpG. The object also contains information about control probes (also known as QC probes). The `MethylSet` contains normalized data and essentially consists of two matrices containing the methylated and the unmethylated evidence for each CpG. Only the `RGChannelSet` contains information about the control probes.

The process described in the previous paragraph is very similar to the paradigm for analyzing Affymetrix expression arrays using the *affy* package (an `AffyBatch` is preprocessed into an `ExpressionSet` using array design information stored in a CDF environment (package)).

A `MethylSet` is the starting point for any post-normalization analysis, such as searching for DMPs or DMRs.

Getting Started

```
require(minfi)
require(minfiData)
```

2 Reading Data

This package supports analysis of IDAT files, containing the summarized bead information.

In our experience, most labs use a "Sample Sheet" CSV file to describe the layout of the experiment. This is based on a sample sheet file provided by Illumina. Our pipeline assumes the existence of such a file(s), but it is relatively easy to create such a file using for example Excel, if it is not available.

We use an example dataset with 6 samples, spread across two slides. First we obtain the system path to the IDAT files; this requires a bit since the data comes from an installed package

```
baseDir <- system.file("extdata", package = "minfiData")
list.files(baseDir)


## [1] "5723646052"      "5723646053"      "SampleSheet.csv"
```

This shows the typical layout of 450k data: each "slide" (containing 12 arrays) is stored in a separate directory, with a numeric name. The top level directory contains the sample sheet file. Inside the slide directories we find the IDAT files (and possible a number of JPG images or other files):

```
list.files(file.path(baseDir, "5723646052"))


## [1] "5723646052_R02C02_Grn.idat" "5723646052_R02C02_Red.idat"
## [3] "5723646052_R04C01_Grn.idat" "5723646052_R04C01_Red.idat"
## [5] "5723646052_R05C02_Grn.idat" "5723646052_R05C02_Red.idat"
```

The files for each array has another numeric number and consists of a Red and a Grn (Green) IDAT file. Note that for this example data, each slide contains only 3 arrays and not 12. This was done because of file size limitations and because we only need 6 arrays to illustrate the package's functionality.

First we read the sample sheet. We provide a convenience function for reading in this file `read.450k.sheet`. This function has a couple of attractive bells and whistles. Let us look at the output

```
targets <- read.450k.sheet(baseDir)


## [read.450k.sheet] Found the following CSV files:
## [1] "/Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/SampleSheet.csv"


targets


##   Sample_Name Sample_Well Sample_Plate Sample_Group Pool_ID person
## 1    GroupA_3          H5           NA       GroupA      NA    id3
## 2    GroupA_2          D5           NA       GroupA      NA    id2
## 3    GroupB_3          C6           NA       GroupB      NA    id3
## 4    GroupB_1          F7           NA       GroupB      NA    id1
## 5    GroupA_1          G7           NA       GroupA      NA    id1
## 6    GroupB_2          H7           NA       GroupB      NA    id2
##   age sex status  Array     Slide
## 1  83   M normal R02C02 5.724e+09
## 2  58   F normal R04C01 5.724e+09
## 3  83   M cancer R05C02 5.724e+09
## 4  75   F cancer R04C02 5.724e+09
## 5  75   F normal R05C02 5.724e+09
## 6  58   F cancer R06C02 5.724e+09
##
## 1 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646052/572364605
## 2 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646052/572364605
## 3 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646052/572364605
## 4 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646053/572364605
## 5 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646053/572364605
## 6 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723646053/572364605
```

First the output: this is just a `data.frame`. It contains a column `Basename` that describes the location of the IDAT file corresponding to the sample, as well as two columns `Array` and `Slide`. In the sample sheet provided by Illumina, these two columns are named `Sentrix_Position` and `Sentrix_ID`, but we rename them. We provide more detail on the use of this function below. The `Basename` column tend to be too large for display, here it is simplified relative to `baseDir`:

```
sub(baseDir, "", targets$Basename)


## [1] "/5723646052/5723646052_R02C02" "/5723646052/5723646052_R04C01"
## [3] "/5723646052/5723646052_R05C02" "/5723646053/5723646053_R04C02"
## [5] "/5723646053/5723646053_R05C02" "/5723646053/5723646053_R06C02"
```

(This is just for display purposes).

With this `data.frame`, it is easy to read in the data

```
RGset <- read.450k.exp(base = baseDir, targets = targets)
```

Let us look at the associated pheno data, which is really just the information contained in the targets object above.

```
show(RGset)


## RGChannelSet (storageMode: lockedEnvironment)
## assayData: 622399 features, 6 samples
##   element names: Green, Red
## phenoData
##   sampleNames: 5723646052_R02C02 5723646052_R04C01 ...
##     5723646053_R06C02 (6 total)
##   varLabels: Sample_Name Sample_Well ... filenames (13 total)
##   varMetadata: labelDescription
## Annotation
##   array: IlluminaHumanMethylation450k
##   annotation: ilmn12.hg19
## [[1]]
## NULL
##
## [[2]]
## NULL


pd <- pData(RGset)
pd


##                    Sample_Name Sample_Well Sample_Plate Sample_Group
## 5723646052_R02C02    GroupA_3          H5           NA       GroupA
## 5723646052_R04C01    GroupA_2          D5           NA       GroupA
## 5723646052_R05C02    GroupB_3          C6           NA       GroupB
## 5723646053_R04C02    GroupB_1          F7           NA       GroupB
## 5723646053_R05C02    GroupA_1          G7           NA       GroupA
## 5723646053_R06C02    GroupB_2          H7           NA       GroupB
##                      Pool_ID person age sex status  Array     Slide
## 5723646052_R02C02      NA    id3  83   M normal R02C02 5.724e+09
## 5723646052_R04C01      NA    id2  58   F normal R04C01 5.724e+09
## 5723646052_R05C02      NA    id3  83   M cancer R05C02 5.724e+09
## 5723646053_R04C02      NA    id1  75   F cancer R04C02 5.724e+09
## 5723646053_R05C02      NA    id1  75   F normal R05C02 5.724e+09
## 5723646053_R06C02      NA    id2  58   F cancer R06C02 5.724e+09
##
## 5723646052_R02C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646052_R04C01 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646052_R05C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R04C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R05C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R06C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
##
## 5723646052_R02C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646052_R04C01 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646052_R05C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R04C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R05C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
## 5723646053_R06C02 /Library/Frameworks/R.framework/Versions/3.0/Resources/library/minfiData/extdata/5723
```

The `read.450k.exp` also makes it possible to read in an entire directory or directory tree (with `recursive` set to `TRUE`) by using the function just with the argument `base` and `targets=NULL`, like

```
RGset2 = read.450k.exp(file.path(baseDir, "5723646052"))
RGset3 = read.450k.exp(baseDir, recursive = TRUE)
```

3 Quality Control

`minfi` provides several plots that can be useful for identifying samples with data quality problems. These functions can display summaries of signal from the array (e.g. density plots) as well as the values of several types of control probes included on the array. Our understanding of the expected sample behavior in the QC plots is still evolving and will improve as the number of available samples from the array increases. A good rule of thumb is to be wary of samples whose behavior deviates from that of others in the same or similar experiments.

Detection P-values

minfi provides several functions and diagnostic plots to assess quality of the methylation samples. As a starting point, we suggest to look at the function `detectionP()` which identies failed positions defined as both the methylated and unmethylated channel reporting background (noise) signal levels:

```
detP <- detectionP(RGset)
failed <- detP > 0.01
head(failed, n = 3)
```

```
##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873             FALSE              TRUE             FALSE
## cg00212031             FALSE              TRUE             FALSE
## cg00213748             FALSE              TRUE              TRUE
##            5723646053_R04C02 5723646053_R05C02 5723646053_R06C02
## cg00050873              TRUE              TRUE              TRUE
## cg00212031              TRUE              TRUE              TRUE
## cg00213748              TRUE              TRUE              TRUE
```

To see the fraction of failed positions per sample:

```
colMeans(failed)
```

```
## 5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
##         0.0009928         0.0032770         0.0092397
## 5723646053_R04C02 5723646053_R05C02 5723646053_R06C02
##         0.0042244         0.0034561         0.0348374
```

and to see how many positions failed in ¿ 50% of the samples:

```
sum(rowMeans(failed) > 0.5)
```

```
## [1] 1047
```

A simple way to quickly check if a sample failed is to look at the log median intensity in both the methylated and unmethylated channels. When plotting the U channel against the M channel, it has been observed that good samples cluster together, while failed samples tend to separate and to have lower median intensities.

But wait, so far we only have green and red intensities. We need the methylated and unmethylated signals; the function `preprocessRaw` is done for that. It takes as input a *RGChannelSet*, convert the red and green intensities to methylated and unmethylated signals according to the probe design stored in the manifest object, and returns the converted signals in a new object of class *MethylSet*.

```
MSet <- preprocessRaw(RGset)
MSet
```

```
## MethylSet (storageMode: lockedEnvironment)
## assayData: 485512 features, 6 samples
##   element names: Meth, Unmeth
## phenoData
##   sampleNames: 5723646052_R02C02 5723646052_R04C01 ...
##     5723646053_R06C02 (6 total)
##   varLabels: Sample_Name Sample_Well ... filenames (13 total)
##   varMetadata: labelDescription
## Annotation
##   array: IlluminaHumanMethylation450k
##   annotation: ilmn12.hg19
## Preprocessing
##   Method: Raw (no normalization or bg correction)
##   minfi version: 1.8.9
##   Manifest version: 0.4.0
```

To access the methylated and unmethylated intensities:

```
head(getMeth(MSet), n = 3)
```

```
##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873             22041               588             20505
## cg00212031               679               569               439
## cg00213748              1620               421               707
##            5723646053_R04C02 5723646053_R05C02 5723646053_R06C02
## cg00050873               404               464               381
## cg00212031               401               468               639
## cg00213748               389               232               695
```

```
head(getUnmeth(MSet), n = 3)
```

```
##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873              1945               433              1012
## cg00212031              6567               300              2689
## cg00213748               384               461               295
##            5723646053_R04C02 5723646053_R05C02 5723646053_R06C02
## cg00050873               341               580               315
## cg00212031               464               753               704
## cg00213748               836               734               759
```

We will talk more about the MethylSet later. The functions `getQC` and `plotQC` are designed to extract the quality control information from the MethylSet:

```
qc <- getQC(MSet)
head(qc)
```

```
## DataFrame with 6 rows and 2 columns
##                        mMed      uMed
##                   <numeric> <numeric>
## 5723646052_R02C02     11.70     11.82
## 5723646052_R04C01     11.99     11.95
## 5723646052_R05C02     11.56     12.05
## 5723646053_R04C02     12.07     12.09
## 5723646053_R05C02     12.23     12.08
## 5723646053_R06C02     11.37     11.61
```

```
plotQC(qc)
```

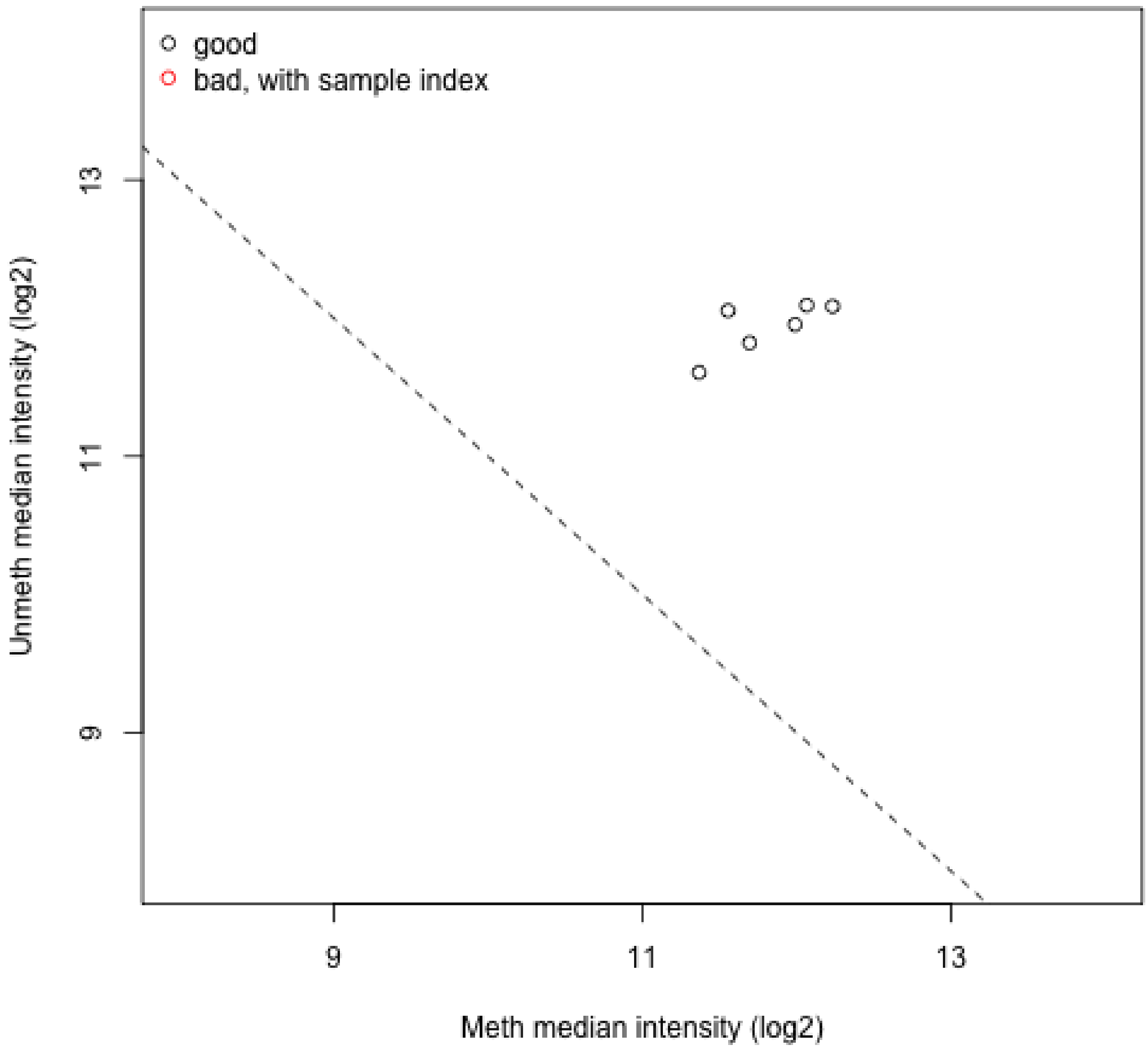


Figure 1: A QC plot can show outlier samples that may be of poor quality

QC Report

The wrapper function `qcReport` function can be used to produce a PDF QC report of the most common plots. If provided, the optional sample name and group options will be used to label and color plots. Samples within a group are assigned the same color. The sample group option can also be used as a very cursory way to check for batch effects (e.g. by setting it to a processing day variable.)

```r
qcReport(RGset, sampNames = pd$Sample_Name, sampGroups = pd$Sample_Group,
    pdf = "qcReport.pdf")
```

The components of the QC report can also be customized and produced individually as detailed below.

Density plots

The `densityPlot` function produces density plots of the methylation Beta values for all samples, typically colored by sample group. While the density plots in Figure 2 are useful for identifying deviant samples, it is not easy to identify the specific problem sample. If there is a concern about outlier samples, a useful follow-up is the "bean" plot (Figure 3) that shows each sample in its own section. While the shape of the distribution for "good" samples will differ from experiment to experiment, many conditions have methylation profiles characterized by two modes - one with close to 0% methylation, and a second at close to 100% methylation.

```r
densityPlot(RGset, sampGroups = pd$Sample_Group, main = "Beta", xlab = "Beta")
```

```r
par(oma = c(2, 10, 1, 1))
densityBeanPlot(RGset, sampGroups = pd$Sample_Group, sampNames = pd$Sample_Name)
```

Control probe plots

The `controlStripPlot` function allows plotting of individual control probe types (Figure 4). The following control probes are available on the array:

```
BISULFITE CONVERSION I     12
BISULFITE CONVERSION II     4
EXTENSION                   4
HYBRIDIZATION               3
NEGATIVE                  614
NON-POLYMORPHIC             4
NORM_A                     32
NORM_C                     61
NORM_G                     32
NORM_T                     61
SPECIFICITY I              12
SPECIFICITY II              3
STAINING                    6
TARGET REMOVAL              2
```

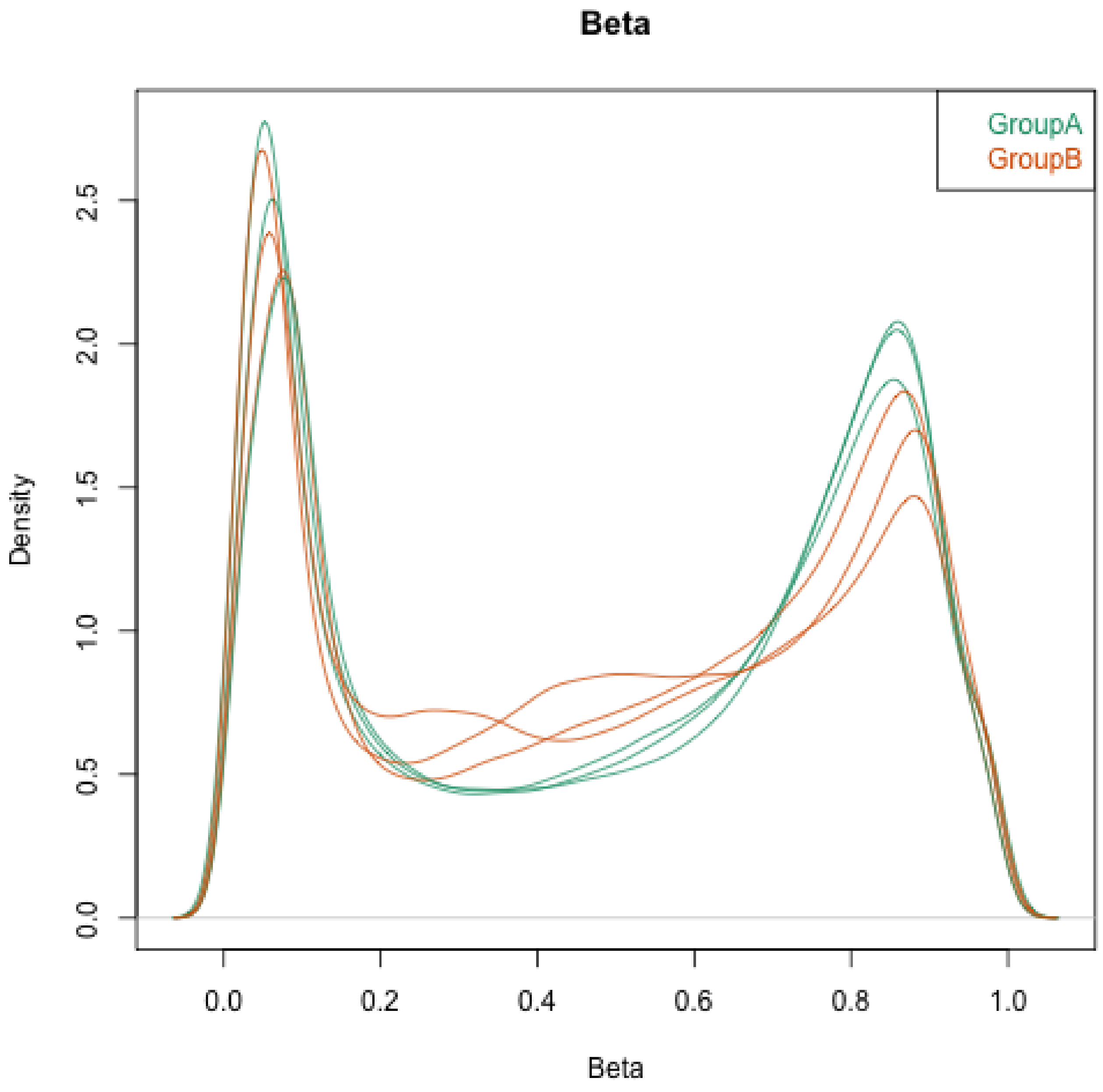


Figure 2: Beta density plots

```
controlStripPlot(RGset, controls = "BISULFITE CONVERSION II", sampNames = pd$Sample_Name)
```

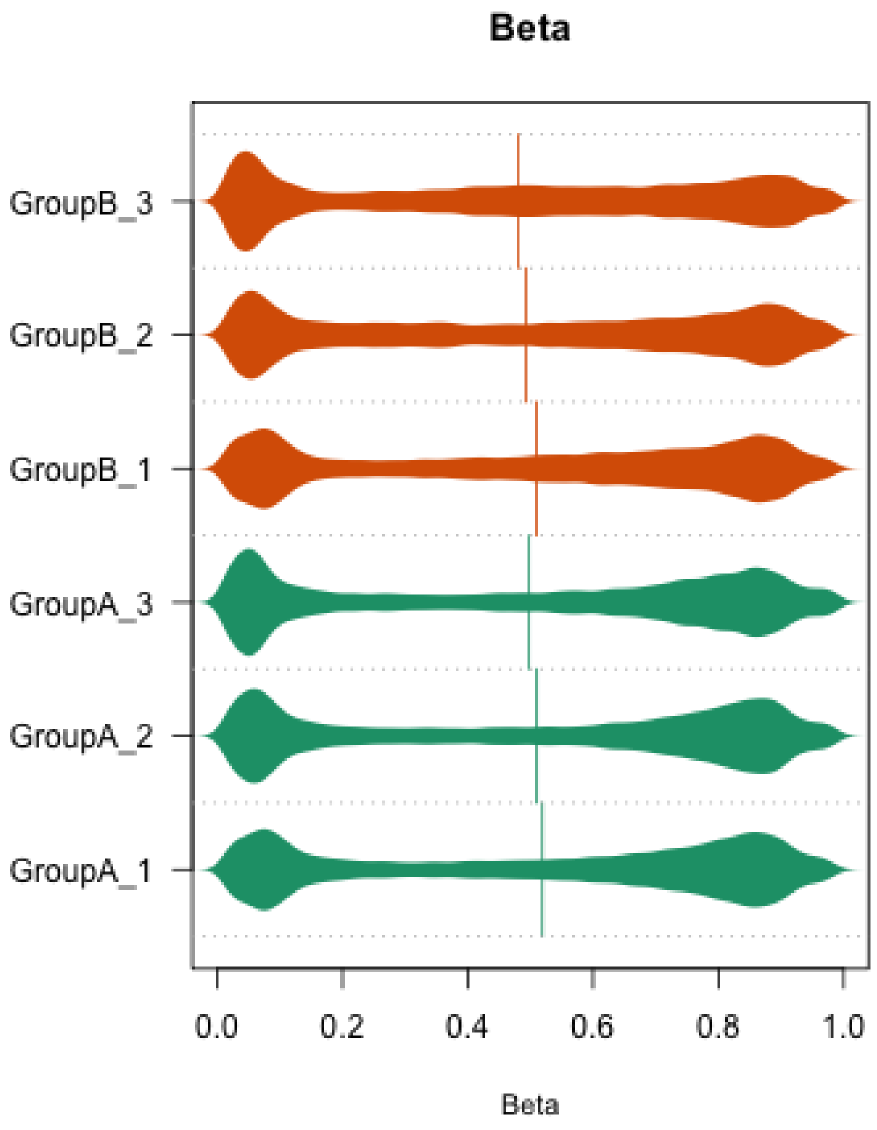


Figure 3: Bean plot of beta values

4 Preprocessing (normalization)

Preprocessing (normalization) takes as input a `RGChannelSet` and returns a `MethylSet`.

A number of preprocessing options are available (and we are working on more methods). Each set of methods are implemented as a function `preprocessXXX` with `XXX` being the name of the method. Each method may have a number of tuning parameters.

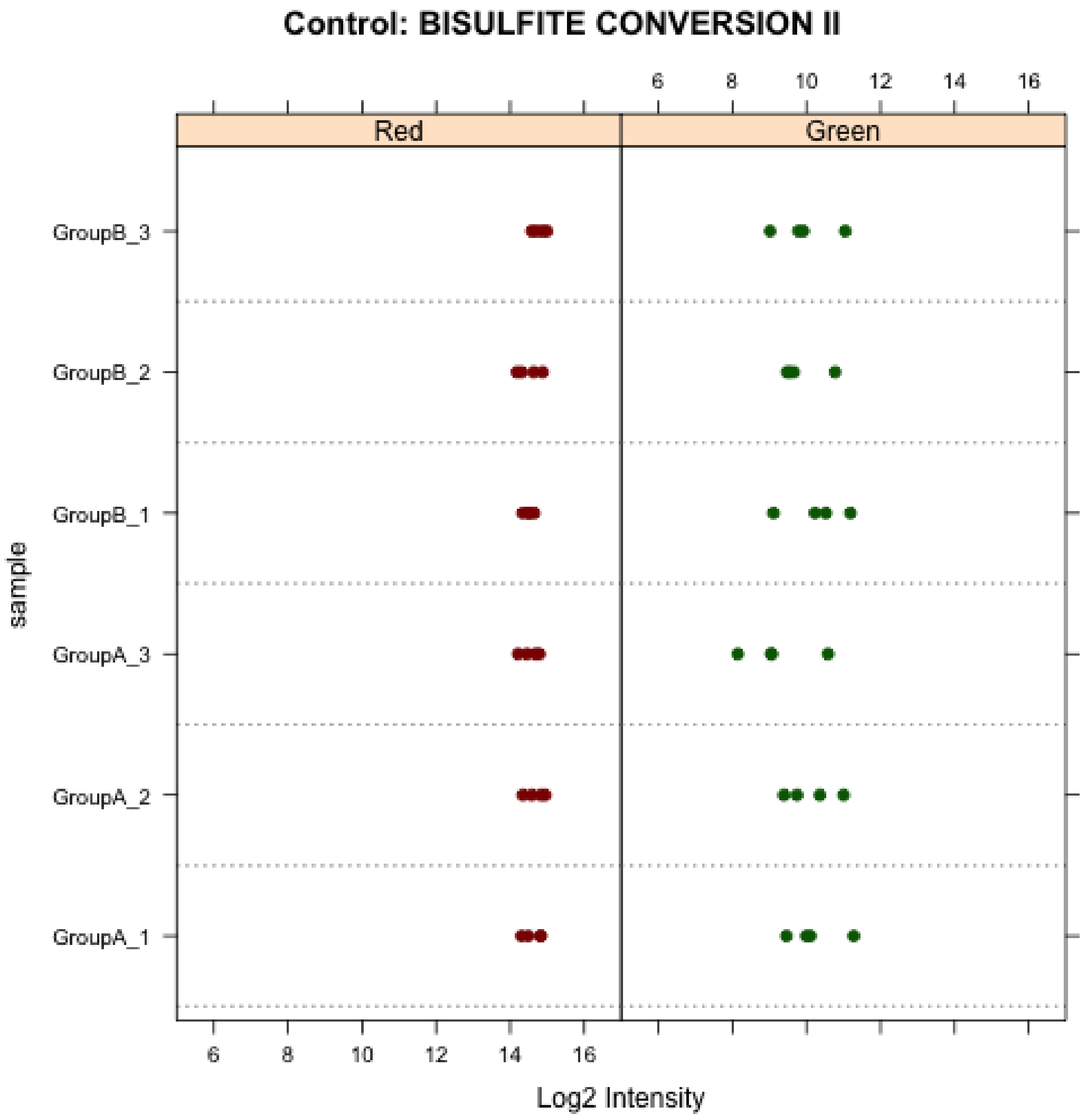


Figure 4: Beta stripplot

"Raw" preprocessing means simply converting the Red and the Green channel into a Methylated and Unmethylated signal

```
MSet.raw <- preprocessRaw(RGset)
```

We have also implemented preprocessing choices as available in Genome Studio. These choices follow the description provided in the Illumina documentation and has been validated by comparing the output of Genome Studio to the output of these algorithms, and this shows the two approaches to be roughly equivalent (for a precise statement, see the manual pages).

Genome studio allows for background subtraction (also called background normalization) as well as something they term control normalization. Both of these are optional and turning both of them off is equivalent to raw preprocessing (`preprocessRaw`).

```
MSet.norm <- preprocessIllumina(RGset, bg.correct = TRUE, normalize = "controls",
    reference = 2)
```

The `reference = 2` selects which array to use as "reference" which is an arbitrary array (we are not sure how Genome Studio makes its choice of reference).

Operating on a MethylSet

Once a `MethylSet` has been generated, we have a various ways of getting access to the methylation data. The most basic functions are `getMeth` and `getUnmeth`, which returns unlogged methylation channels. The function `getBeta` gets "beta"-values which are values between 0 and 1 with 1 interpreted as very high methylation. If `type = "Illumina"` (not the default) these are computed using Illumina's formula

$$\beta = \frac{M}{M + U + 100}$$

Finally, we have the "M-values" (not to be confused with the methylation channel obtained by `getMeth`). M-values are perhaps an unfortunate terminology, but it seems to be standard in the methylation array world. These are computed as $\mathrm{logit}(\beta)$ and are obtained by `getM`.

```
getMeth(MSet.raw)[1:4, 1:3]

##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873             22041               588             20505
## cg00212031               679               569               439
## cg00213748              1620               421               707
## cg00214611               449               614               343

getUnmeth(MSet.raw)[1:4, 1:3]

##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873              1945               433              1012
## cg00212031              6567               300              2689
## cg00213748               384               461               295
## cg00214611              4869               183              1655

getBeta(MSet.raw, type = "Illumina")[1:4, 1:3]

##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873           0.91510            0.5245            0.9486
## cg00212031           0.09243            0.5872            0.1360
## cg00213748           0.76996            0.4287            0.6416
## cg00214611           0.08287            0.6845            0.1635
```

```
getM(MSet.raw)[1:4, 1:3]
```

```
##            5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
## cg00050873             3.502            0.4414             4.341
## cg00212031            -3.274            0.9235            -2.615
## cg00213748             2.077           -0.1309             1.261
## cg00214611            -3.439            1.7464            -2.271
```

MDS plots

After preprocessing the raw data to obtain methylation estimates, Multi-dimensional scaling (MDS) plots provide a quick way to get a first sense of the relationship between samples. They are similar to the more familiar PCA plots and display a two-dimensional approximation of sample-to-sample Euclidean distance. Note that while the plot visualizes the distance in epigenomic profiles between samples, the absolute positions of the points is not meaningful. One often expects to see greater between-group than within-group distances (although this clearly depends on the particular experiment). The most variable locations are used when calculating sample distances, with the number specified by the `numPositions` option. Adding sample labels to the MDS plot is a useful way of identifying outliers (figure 5) that behave differently from their peers.

```
mdsPlot(MSet.norm, numPositions = 1000, sampGroups = pd$Sample_Group, sampNames = pd$Sample_Name)
```

The validation of `preprocessIllumina`

By validation we mean "yielding output that is equivalent to Genome Studio".

Illumina offers two steps: control normalization and background subtraction (normalization). Using output from Genome Studio we are certain that the control normalization step is validated, with the following caveat: control normalization requires the selection of one array among the 12 arrays on a chip as a reference array. It is currently unclear how Genome Studio selects the reference; if you know the reference array we can recreate Genome Studio exactly. Background subtraction (normalization) is almost correct: for 18 out of 24 arrays we see exact equivalence and for the remaining 6 out of 24 arrays we only see small discrepancies (a per-array max difference of 1-4 for unlogged intensities). A script for doing this is in `scripts/GenomeStudio.R`.

Subset-quantile within array normalisation (SWAN)

SWAN (subset-quantile within array normalisation) is a new normalization method for Illumina 450k arrays. What follows is a brief description of the methodology (written by the authors of SWAN):

Technical differences have been demonstrated to exist between the Type I and Type II assay designs within a single 450K array[**? ?**]. Using the SWAN method substantially reduces the technical variability between the assay designs whilst maintaining the important biological differences. The SWAN method

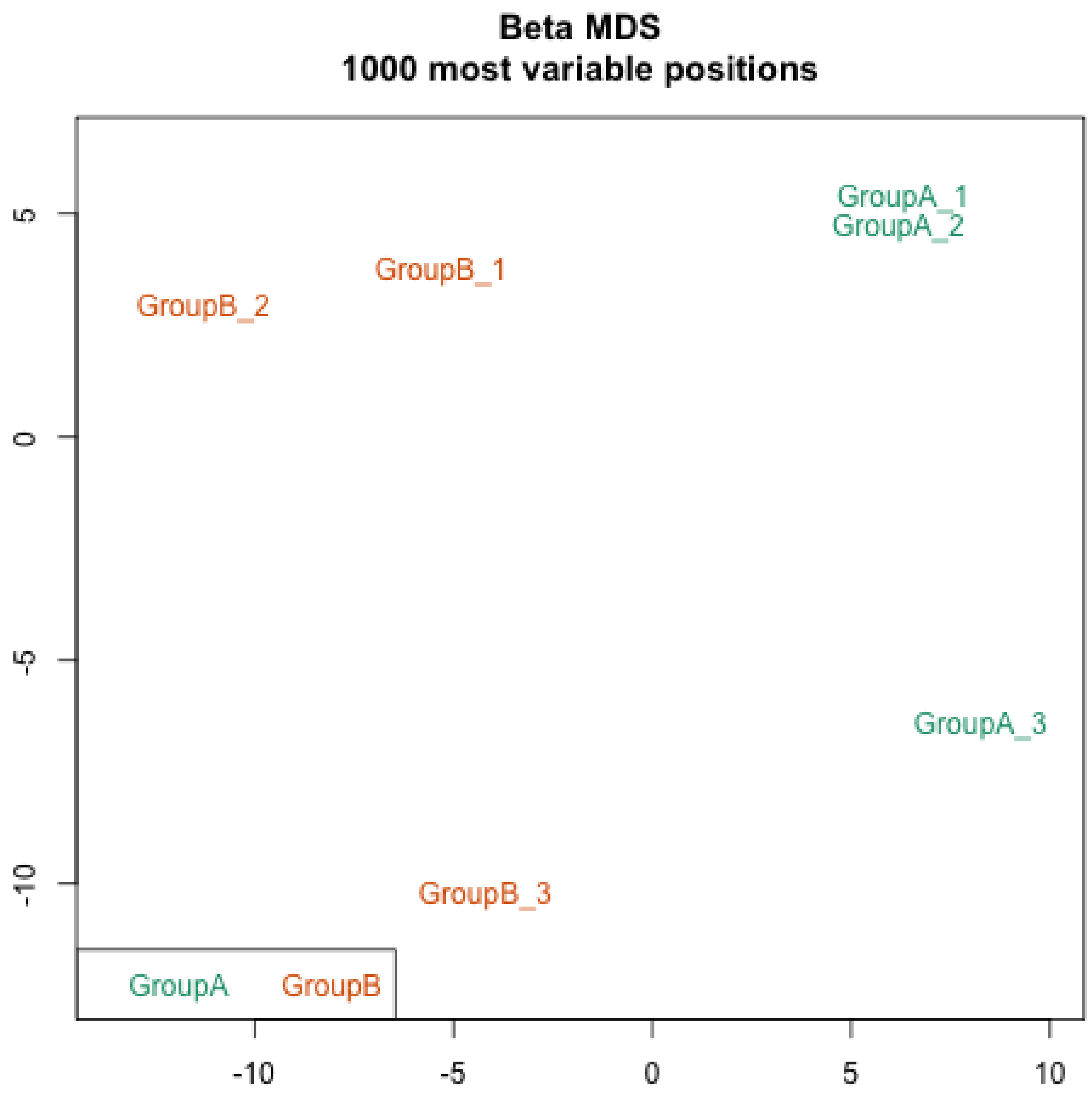


Figure 5: Multi-dimensional scaling plot

makes the assumption that the number of CpGs within the 50bp probe sequence reflects the underlying biology of the region being interrogated. Hence, the overall distribution of intensities of probes with the same number of CpGs in the probe body should be the same regardless of design type. The method then uses a subset quantile normalization approach to adjust the intensities of each array [**?**]. SWAN takes a `MethylSet` as input. This can be generated by either `preprocessRaw` or `preprocessIllumina`. Calling the function without specifying a `MethylSet` uses `preprocessRaw`. It should be noted that, in order to create the normalization subset, SWAN randomly selects Infinium I and II probes that have one, two and three underlying CpGs; as such, we recommend setting a seed (using set.seed)before using preprocessSWAN to ensure that the normalized intensities will be identical, if the normalization is repeated.

```
Mset.swan <- preprocessSWAN(RGsetEx, MsetEx)
```

The technical differences between Infinium I and II assay designs can result in aberrant beta value distributions (Figure 6, panel "Raw"). Using SWAN corrects for the technical differences between the Infinium I and II assay designs and produces a smoother overall beta value distribution (Figure 6, panel "SWAN").

```
par(mfrow = c(1, 2))
plotBetasByType(MsetEx[, 1], main = "Raw")
plotBetasByType(Mset.swan[, 1], main = "SWAN")
```

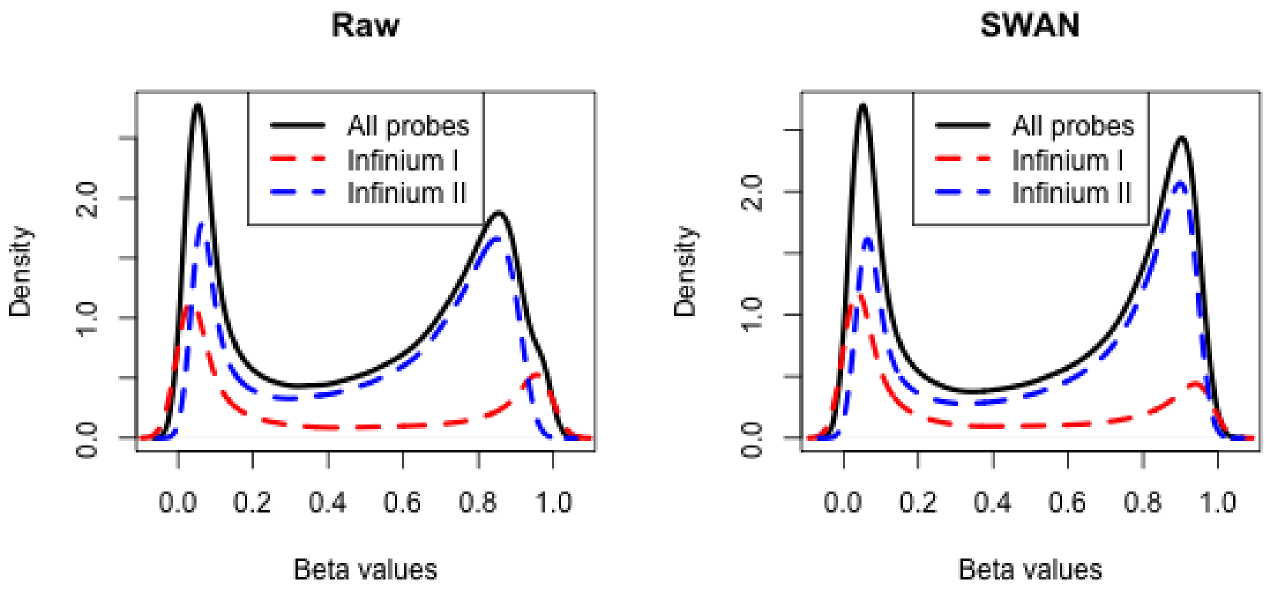


Figure 6: The effect of normalizing using SWAN.

5 Finding differentially methylated positions (DMPs)

We are now ready to use the normalized data to identify DMPs, defined as CpG positions where the methylation level correlates with a phenotype of interest. The phenotype may be categorical (e.g. cancer vs. normal) or continuous (e.g. blood pressure).

We will create a 20,000 CpG subset of our dataset to speed up the demo:

```
mset <- MSet.norm[1:20000, ]
```

Categorical phenotypes

The `dmpFinder` function uses an F-test to identify positions that are differentially methylated between (two or more) groups. Tests are performed on logit transformed Beta values as recommended in Pan et al. Care should be taken if you have zeroes in either the Meth or the Unmeth matrix. One possibility is to threshold the beta values, so they are always in the interval $[\epsilon, 1-\epsilon]$. We call ϵ the betaThreshold

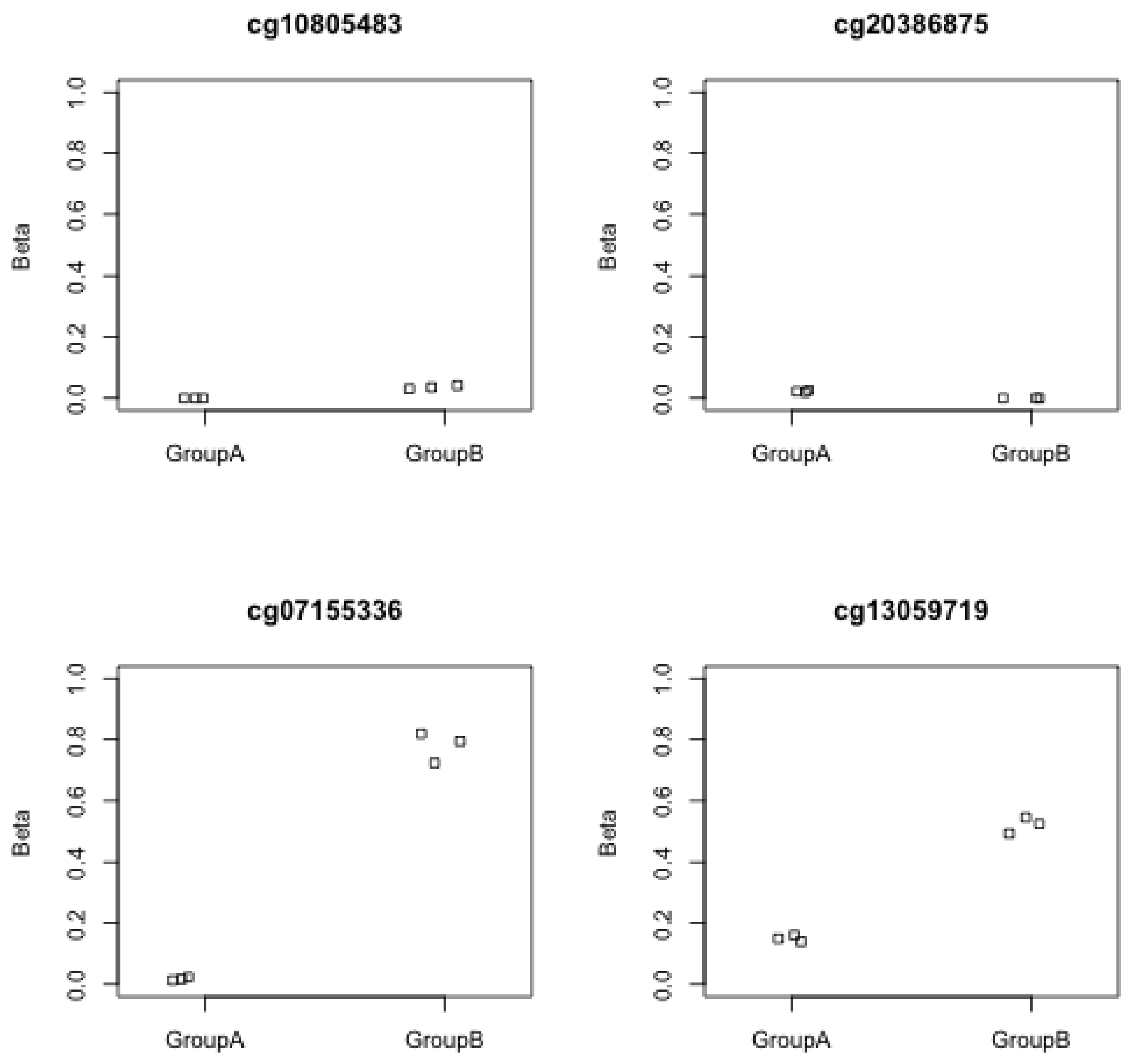
Here we find the differences between GroupA and GroupB.

```
table(pd$Sample_Group)

##
## GroupA GroupB
##      3      3

M <- getM(mset, type = "beta", betaThreshold = 0.001)
dmp <- dmpFinder(M, pheno = pd$Sample_Group, type = "categorical")
head(dmp)

##             intercept      f      pval    qval
## cg10805483     -9.964 1706.1 2.053e-06 0.02640
## cg20386875     -5.434 1445.1 2.860e-06 0.02640
## cg07155336     -5.800  551.0 1.953e-05 0.05148
## cg13059719     -2.506  549.7 1.962e-05 0.05148
## cg08343042     -3.565  506.2 2.311e-05 0.05148
## cg23098069      1.532  497.6 2.391e-05 0.05148
```

`dmpFinder` returns a table of CpG positions sorted by differential methylation p-value.

We can use the `plotCpG` function to plot methylation levels at individual positions:

```
cpgs <- rownames(dmp)[1:4]
par(mfrow = c(2, 2))
plotCpg(mset, cpg = cpgs, pheno = pd$Sample_Group)
```



Continuous phenotypes

We can also identify DMPs where the mean methylation level varies with a continuous covariate using linear regression. Since the sample dataset does not contain any continuous phenotypes we will simulate one for demonstration purposes:

```
continuousPheno <- rnorm(nrow(pd))
```

We now search for DMPs associated with this fake random phenotype.

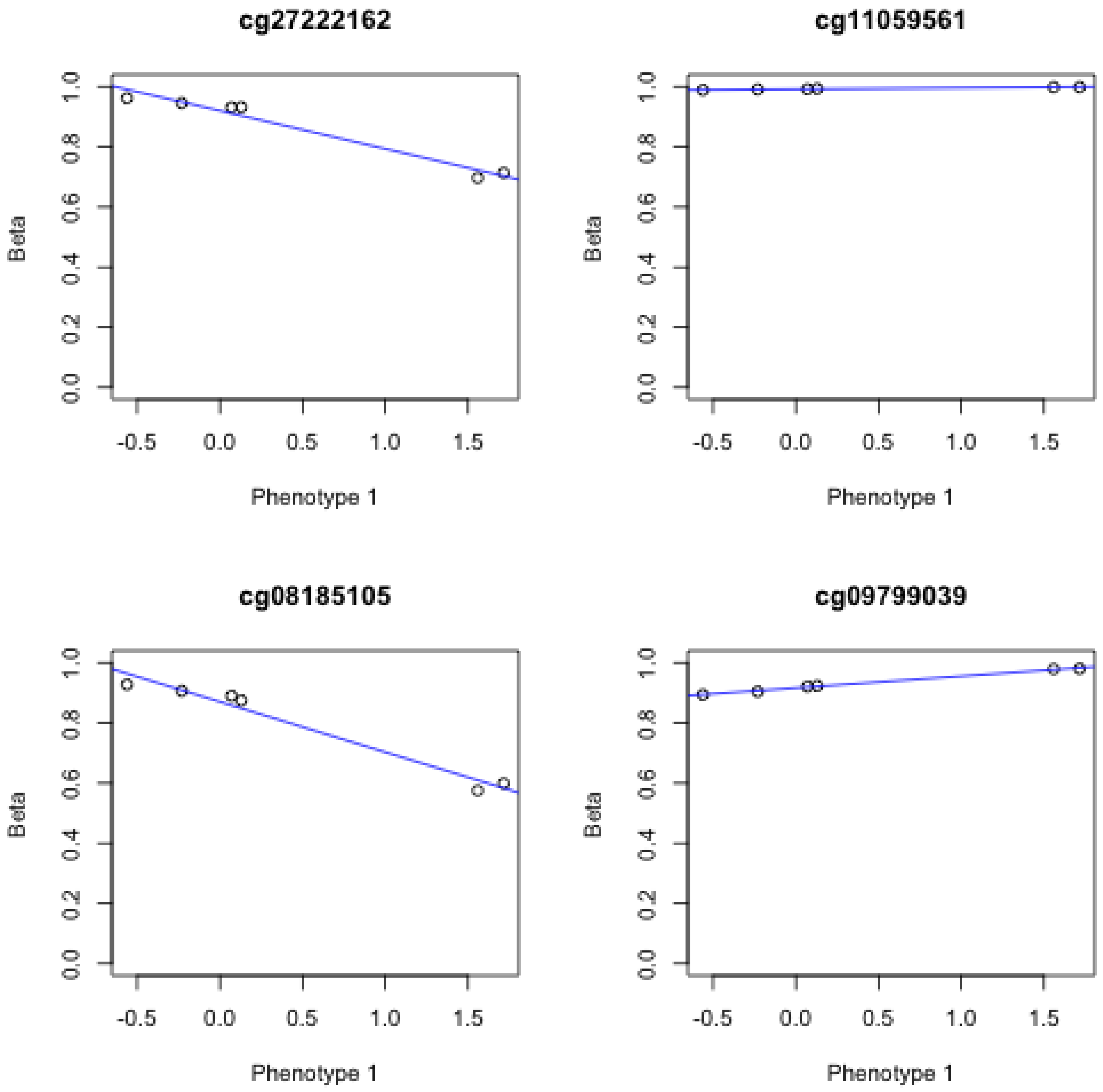
```
dmp <- dmpFinder(mset, pheno = continuousPheno, type = "continuous")

## Warning:  Partial NA coefficients for 7 probe(s)

dmp[1:3, ]

##            intercept    beta      t      pval    qval
## cg24397815    -3.351 -0.7331 -25.81 1.338e-05 0.08318
## cg24683414    -3.212 -0.9242 -25.46 1.414e-05 0.08318
## cg09787089     1.535  0.9960  22.36 2.369e-05 0.08318
```

The `beta` column gives the change in mean phenotype for each unit increase of methylation. We can filter the DMP list to exclude positions with a small effect size:

```
dmp <- subset(dmp, abs(beta) > 1)
```

The `plotCpg` function can be used to visualise these continuous DMPs:

```
cpgs <- rownames(dmp)[1:4]
par(mfrow = c(2, 2))
plotCpg(mset, cpg = cpgs, type = "continuous", pheno = continuousPheno,
    xlab = "Phenotype 1")
```



6 SessionInfo

- R version 3.0.2 Patched (2014-01-22 r64855), `x86_64-apple-darwin10.8.0`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils

- Other packages: ALL 1.4.14, AnnotationDbi 1.24.0, BSgenome 1.29.1, BSgenome.Hsapiens.UCSC.hg19 1.3.19, Biobase 2.21.7, BiocBrazil2014 1.0, BiocGenerics 0.7.5, BiocInstaller 1.12.0, Biostrings 2.29.19, DBI 0.2-7, DESeq2 1.2.9, GEOquery 2.28.0, GenomicFeatures 1.13.43, GenomicRanges 1.13.45, Gviz 1.5.15, IRanges 1.19.38, IlluminaHumanMethylation450kanno.ilmn12.hg19 0.2.1, IlluminaHumanMethylation450kmanifest 0.4.0, KernSmooth 2.23-10, MASS 7.3-29, RColorBrewer 1.0-5, RCurl 1.95-4.1, RSQLite 0.11.4, Rcpp 0.10.4, RcppArmadillo 0.3.910.0, Rsamtools 1.13.46, SNPlocs.Hsapiens.dbSNP.20120608 0.99.9, SRAdb 1.15.0, ShortRead 1.19.13, TxDb.Hsapiens.UCSC.hg19.knownGene 2.10.1, VariantAnnotation 1.7.47, XVector 0.1.4, bitops 1.0-6, bumphunter 1.1.17, caTools 1.14, devtools 1.3, foreach 1.4.1, gdata 2.13.2, genefilter 1.43.0, ggplot2 0.9.3.1, gplots 2.11.3, graph 1.39.3, gtools 3.1.0, iterators 1.0.6, knitr 1.5, lattice 0.20-24, limma 3.17.25, locfit 1.5-9.1, minfi 1.8.9, minfiData 0.4.2, org.Hs.eg.db 2.10.1, parathyroidSE 1.0.4, pheatmap 0.7.7, plyr 1.8, randomForest 4.6-7, reactome.db 1.44.0, reshape 0.8.4, rtracklayer 1.21.12

- Loaded via a namespace (and not attached): Hmisc 3.12-2, R.methodsS3 1.5.1, XML 3.95-0.2, annotate 1.39.0, base64 1.1, beanplot 1.1, biomaRt 2.17.3, biovizBase 1.9.4, cluster 1.14.4, codetools 0.2-8, colorspace 1.2-3, compiler 3.0.2, dichromat 2.0-0, digest 0.6.3, doRNG 1.5.5, evaluate 0.5, formatR 0.9, gtable 0.1.2, highr 0.2.1, httr 0.2, hwriter 1.3, illuminaio 0.3.11, itertools 0.1-1, labeling 0.2, latticeExtra 0.6-26, matrixStats 0.8.12, mclust 4.2, memoise 0.1, multtest 2.17.0, munsell 0.4.2, nlme 3.1-113, nor1mix 1.1-4, pkgmaker 0.17.4, preprocessCore 1.23.0, proto 0.3-10, registry 0.2, reshape2 1.2.2, rngtools 1.2.3, rpart 4.1-4, scales 0.2.3, siggenes 1.35.0, splines 3.0.2, stats4 3.0.2, stringr 0.6.2, survival 2.37-7, tools 3.0.2, whisker 0.3-2, xtable 1.7-1, zlibbioc 1.7.0