ARRm: Adaptive Robust Regression method for normalization of methylation data

Jean-Philippe Fortin, Aurélie Labbe and Celia M.T. Greenwood

May 19, 2021

1 Introduction

With the adaptation of microarray hybridization techniques developed for gene expression and genomics studies to methylation data, there has been a revolution in the development of DNA methylation profiling techniques [1]. Illumina recently released the Infinium HumanMethylation450 BeadChip, a single CpG site resolution array using bisulfite-converted DNA. The Infinium 450k methylation array comprises two different chemistry technologies (Infinium I and II). Microarray methylation data can be affected by technical artifacts, like any microarray-based assay. Signals can be affected by the position of the probes on the chip, probe design, inter-batch differences in chips, laboratory conditions or other unknown factors. In gene-expression studies, quantile normalization is one of the most popular approaches for between-array correction. It forces the distributions of gene expressions to be essentially identical across samples; this assumption is justified in gene expression studies where only a few genes are expected to be differentially expressed across the samples. Various versions of quantile normalization procedures are currently being used to preprocess DNA methylation data and have been previously compared for the 27k platform [2]; here we argue, as have others [3], that these methods are not appropriate for these data. Indeed, it is well-known that overall DNA methylation levels can be significantly different when tumor cells are compared to normal cells [4, 5]. Furthermore, even when all samples are from the same cell type, genome-wide methylation alterations can be seen. For example, global patterns of hypomethylation with age have been seen in blood cells [6]. We propose a new normalization procedure that does not assume similarity of statistical distributions across samples. Our model corrects explicitly for background intensity, dye bias and spatial effects, and allows these corrections to vary by the quantile of the signals. We use a robust regression model to reduce the influence of outliers. To take into account the difference between Type I and Type II probes, we consider the two probe designs as two different arrays and we apply our normalization procedure separately.

2 Example dataset: ARRm package

To illustrate how our method works, we use an example dataset included in the package ARRmData. The dataset contains 36 methylation profiles from the Illumina Infinium HumanMethylation 450k array. For each sample, methylation levels are provided by a vector of Beta values. In addition, two vectors of negative probes are provided: one for the intensities in the green channel, and one for the intensities in the red channel. The matrix of Beta values for the 36 samples are contained in"betaMatrix", the negative control probes are contained in the two matrices"greenControlMatrix" and "redControlMatrix" and the names of the samples are contained in the vector "sampleNames". We first load these data into the R environment using the following commands:

```
> library(ARRmData)
> data(greenControlMatrix)
> data(redControlMatrix)
> data(betaMatrix)
> data(sampleNames)
```

For the beta matrix, columns are samples (36 in total) and rows are probes (485,577 in total). Names of the columns are the names of the samples.

```
> betaMatrix[1:5,1:3]
```

```
  5621146023_R01C01 5621146023_R01C02 5621146023_R02C01
1         0.6231034         0.7419837         0.7163057
2         0.9401186         0.9336658         0.9248899
3         0.9148064         0.9055801         0.9315221
4         0.2965243         0.3161751         0.3002749
5         0.8469906         0.8736507         0.8857006
```

For the negative control matrices, columns are samples (36 in total) and rows are negative control probes (600 in total). Names of the columns are the names of the samples. The order of the samples in the beta matrix must be identical to that of the samples in the negative control probe matrices.

```
> greenControlMatrix[1:5,1:3]
```

```
   5621146023_R01C01 5621146023_R01C02 5621146023_R02C01
24               316               359               277
25               280               417               283
26               248               425               276
27               259               400               341
28               320               364               269
```

```
> redControlMatrix[1:5,1:3]
```

```
   5621146023_R01C01 5621146023_R01C02 5621146023_R02C01
24               273               432               254
25               208               435               287
26               198               343               276
27               229               339               234
28               241               241               216
```

The names of the probes for the beta matrix (following Illumina"s notation, e.g. "cg00000029" for the first probe), is given in the file "ProbesType.rda", located in the ARRmNormalization package:

```
> library(ARRmNormalization)
> data(ProbesType)
> head(ProbesType)

  Probe_Name Design_Type
1 cg00000029          II
2 cg00000108          II
3 cg00000109          II
4 cg00000165          II
5 cg00000236          II
6 cg00000289          II
```

Names of the samples are of the form:

```
> head(sampleNames)

        sampleNames
1 5621146023_R01C01
2 5621146023_R01C02
3 5621146023_R02C01
4 5621146023_R02C02
5 5621146023_R03C01
6 5621146023_R03C02
```

3 Preparing the workspace

In order to normalize your methylation profiles, you need to format the data in the same fashion as the example dataset presented above. Then you load the package ARRmNormalization:

```
> library(ARRmNormalization)
```

We load here also the additional package ARRmData in order to use the example dataset:

```
> library(ARRmData)
```

We load the data into the workspace:

```
> data(betaMatrix)
> data(greenControlMatrix)
> data(redControlMatrix)
> data(sampleNames)
```

4 Pre-normalization steps

Before normalizing the methylation profiles, we need to extract the background intensities and the physical chip information of the samples. For the background extraction, we use the following command:

```
> backgroundInfo <- getBackground(greenControlMatrix, redControlMatrix)
```

We get a data frame whose two columns are the medians of the background intensity of the green and red channels, respectively, for each sample.

```
> head(backgroundInfo)
```

```
                  green   red
5621146023_R01C01 275.0 249.0
5621146023_R01C02 390.5 365.0
5621146023_R02C01 281.0 239.0
5621146023_R02C02 324.0 268.5
5621146023_R03C01 277.0 231.5
5621146023_R03C02 284.0 238.0
```

For the chip and position information of the samples, there are two ways to process it. The first way is to extract it from the sample names; these must be in the following format to use the extraction function:

```
> head(sampleNames)
```

```
        sampleNames
1 5621146023_R01C01
2 5621146023_R01C02
3 5621146023_R02C01
4 5621146023_R02C02
5 5621146023_R03C01
6 5621146023_R03C02
```

Then use the command getDesignInfo to extract the physical chip information:

```
> designInfo <- getDesignInfo(sampleNames)
> head(designInfo)
```

```
  chipInfo positionInfo       sampleNames
1        1            1 5621146023_R01C01
2        1            2 5621146023_R01C02
3        1            3 5621146023_R02C01
4        1            4 5621146023_R02C02
5        1            5 5621146023_R03C01
6        1            6 5621146023_R03C02
```

The first column gives a chip index, and the second column gives a position index (between 1 and 12).
The second way to build the design information matrix is by providing explicit chip and position indices. For example, let's create these artificial chip and position index vectors:

```
> myChipVector <- c(rep(1,12), rep(2, 12), rep(3, 12))
> myPositionVector <- rep(seq(1:12), 3)
```

Then the design matrix can be created with the same command as above:

```
> designInfo <- getDesignInfo(sampleNames=NULL,
+                   positionVector=myPositionVector, chipVector=myChipVector)
> head(designInfo)
```

```
  chipInfo positionInfo
1        1            1
2        1            2
3        1            3
4        1            4
5        1            5
6        1            6
```

5 ARRm Normalization

We are now ready to normalize the methylation profiles. First, the normalization function needs three data frames similar to those of the example dataset:

1 a matrix of beta values where columns are samples and rows are probes

2 a designInfo data frame obtained with the getDesignInfo function

3 a backgroundInfo data frame obtained with the function getBackground-Info

By default, the percentage of outlier samples to be removed in the estimation of the bias effects is set to 2%; the user can choose his or her own desired percentage of outlier samples to be trimmed in the robust regression, by setting the parameter *outliers.perc* to a number between 0 and 1. For instance, here we normalize all the data from the dataset example with 10% of the outliers removed:

```
> normMatrix <-
+     normalizeARRm(betaMatrix=betaMatrix, designInfo=designInfo,
+         backgroundInfo=backgroundInfo, outliers.perc=0.1)
```

Two more options are offered to the user. The first one allows the normalization of a subset of the probes. This can be very handy since we expect that some of the probes may need to be excluded after quality control, or if the user wants to exclude probes containing SNPs, of probes mapped to sex chromosomes. In that case, the user has to build a vector of all probes names (e.g. "cg00000029" for the first probe) to be normalized and has to pass it to the parameter *goodProbes* in the function *normalizeARRm*. For instance, suppose we want to normalize only half of the probes, say the even numbered probes (for example purposes only). We first construct the list of these probes:

```
> data(ProbesType)
> goodProbes <- as.character(unlist(ProbesType$Probe_Name))[seq(1,485577,2)]
```

and then normalize the data with these probes only:

```
> normMatrix <-
+     normalizeARRm(betaMatrix=betaMatrix, designInfo=designInfo,
+         backgroundInfo=backgroundInfo, outliers.perc=0, goodProbes=goodProbes)
```

Note that normalized Beta values for the non-normalized probes are set to "*NA*". The second option is to decide whether or not chip correction should be performed, by setting *chipCorrection* to be true or false (by default, set to true). This can be useful if chips are confounded with important sample characteristics. For instance, if samples were allocated to chips according to gender or case-control status, chip means are confounded with these sample characteristics, and chip correction should not be performed since it would remove important biological variation. Here is an example of normalization without chip correction, on all probes:

```
> normMatrix <-
+     normalizeARRm(betaMatrix=betaMatrix, designInfo=designInfo,
+         backgroundInfo=backgroundInfo, outliers.perc=0, chipCorrection=F)
```

6 Visualization

We have included visualization tools in order to investigate background, dye bias and on-chip position effects. In this section, we investigate only the non-normalized data (*betaMatrix*), but visualization tools can be applied on normalized values as well, by switching *betaMatrix* to *normMatrix* in the examples below. To avoid long computations in plotting functions, the first step is to extract Beta value distribution quantiles for each sample, separately by probe type. This is achieved by the *getQuantiles* function:

```
> quantiles=getQuantiles(betaMatrix)
> attributes(quantiles)

$names
[1] "green" "red"   "II"
```

In the case only a subset of probes were normalized, the option *goodProbes* is available as before for the function *getQuantiles* so that quantiles are only computed for these given probes:

```
> quantiles=getQuantiles(betaMatrix, goodProbes=goodProbes)
> attributes(quantiles)

$names
[1] "green" "red"   "II"
```

In both cases, the function *getQuantiles* returns a list of three matrices that can be accessed by $\$green$, $\$red$ and $\$II$, corresponding respectively to the quantiles of the Type I green probes, Type I red probes and Type II probes. Each matrix has 100 rows, corresponding to percentiles; columns are samples. For instance , we can look at the matrix for Type II probes:

```
> quantiles$II[1:5,1:4]

   5621146023_R01C01 5621146023_R01C02 5621146023_R02C01 5621146023_R02C02
1%        0.05467959        0.05521203        0.06418919        0.05920067
2%        0.06155891        0.06188955        0.07151738        0.06605582
3%        0.06620405        0.06665925        0.07713512        0.07116320
4%        0.07023557        0.07058765        0.08170022        0.07559437
5%        0.07374357        0.07417378        0.08591799        0.07964716
```

To investigate background effects and dye bias effects on percentiles, the function *quantilePlots* can be applied to the quantiles extracted with the function *getQuantiles*:

```
> quantilePlots(quantiles, backgroundInfo, designInfo)

null device
          1
```

The function has no return value, but instead makes a *pdf* file ("quantilePlots.pdf") of several plots used to visualize background and dye bias effects. For each probe type, a plot of percentiles against background intensity is produced. For Type II probes, there is also a plot of percentiles against dye bias. One can specify which percentiles are plotted with the parameters *percentilesI* and *percentilesII* for Type I and Type II probes respectively. By default, for Type I probes, k-th percentiles are drawn for $k \in \{5, 10, \ldots, 100\}$; for Type II

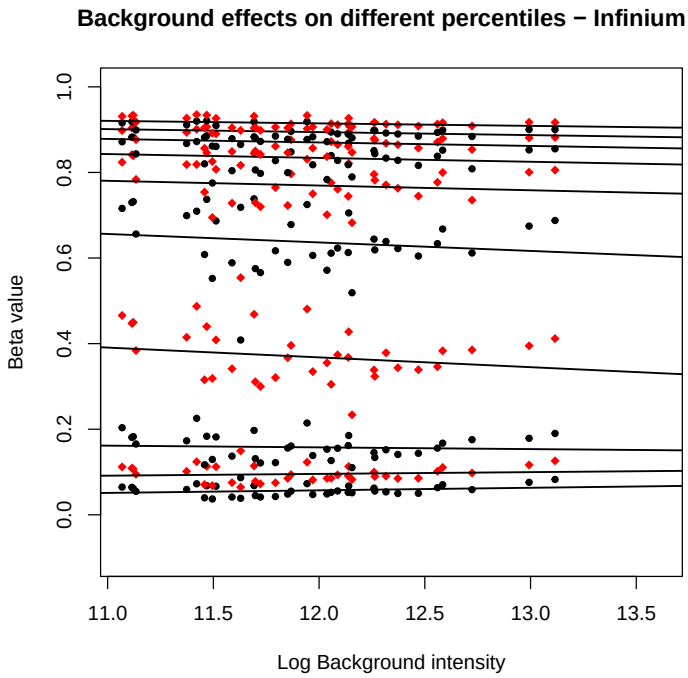

Figure 1: Background effects on Type II probes

probes, k-th percentiles are drawn for $k \in \{10, 20, \ldots, 100\}$. As an example, the corresponding plots for Type II probes, with the default list of percentiles, are presented in Figure 1 and Figure 2. Two colors are used to separate consecutive percentiles. Fitted lines with non-zero slope indicate potential bias in the corresponding percentiles.

To investigate on-chip position effects, the function *positionPlots* can be applied to the quantiles previously extracted with *getQuantiles*:

```
> positionPlots(quantiles, designInfo, percentiles=c(25, 50, 75))

pdf
  2
```

The function has no return value, but instead makes a *pdf* file ("positionPlots.pdf") of several plots used to visualize position effects. For each probe type, and for each sample, deviations from the chip mean are computed for every percentile specified by the function parameter *percentiles*. For these percentiles, deviations are plotted against the position index. If spatial position has no effect on beta value distribution, points are expected to create a uniform

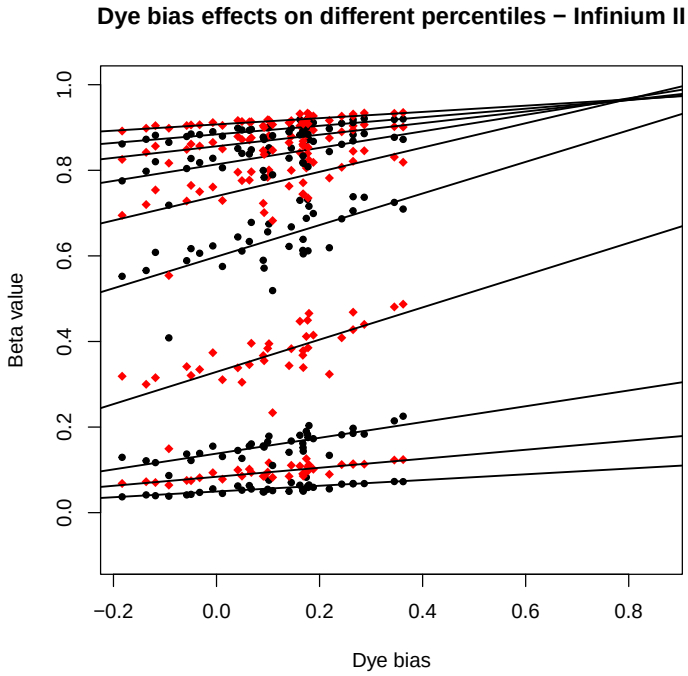

Figure 2: Dye bias effects on Type II probes

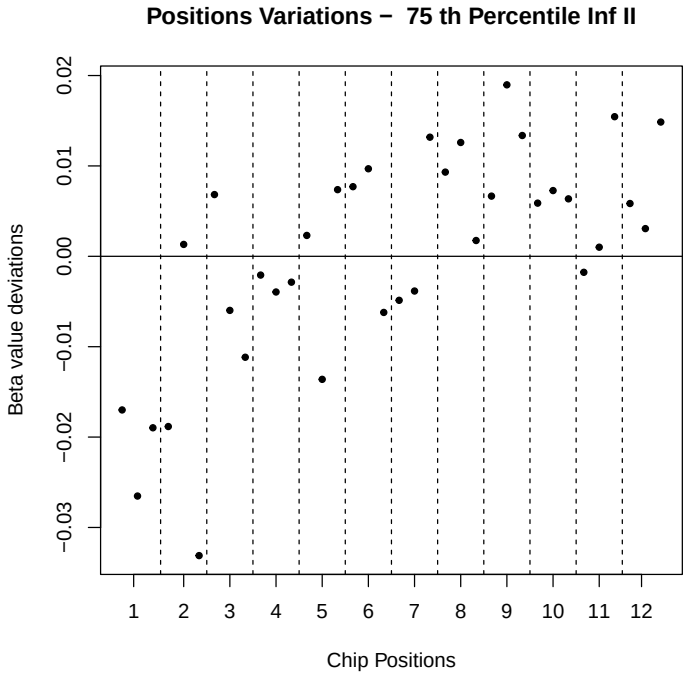

Figure 3: Position effects for for Type II probes - 75^{th} percentile

cloud centered around zero. An example plot is shown for the 75^{th} percentile of Type II probes in Figure 3.

For investigation of the coefficients estimated by the robust regression model used in the ARRm normalization, the function *getCoefficients* can be used to extract the coefficients:

```
> coefficients <- getCoefficients(quantiles, designInfo, backgroundInfo, outliers.perc=0.02)
> attributes(coefficients)

$names
[1] "green" "red"   "II"
```

The function returns a list of three lists, one corresponding for each probe type, that can be accessed by $\$green$, $\$red$ and $\$II$. Each sublist contains different subfields corresponding to different effect estimates:

```
> attributes(coefficients$II)

$names
[1] "res"                "background.vector"   "dyebias.vector"
[4] "chip.variations"    "position.variations"
```

$*res* contains a vector of summary statistics for residuals. For each sample S_i, the statistic $\sum_{k=1}^{100} e_{i,k}^2$ is computed, where $e_{i,k}$ is the residual for the sample S_i in the regression model fitted to the k-th percentile.
$*background.vector* contains the regression coefficient corresponding to background effect for each percentile, and $*dyebias.vector* contains the regression coefficient corresponding to dye bias for each percentile. For Type I probes, $*dyebias.vector* contains only missing values, since no dye bias doesn't apply for these probes. $*chip.variations* is a matrix where rows are percentiles, and columns are chip indices, and entries represent the estimated chip variations from the grand mean, for each percentile. Similarly, $*position.variations* is a matrix where rows are percentiles, and columns are position indices (from 1 to 12), and entries represent the estimated position variations from the corresponding chip mean, for each percentile.

References

[1] Laird, P. W. *Principles and challenges of genomewide DNA methylation analysis*, Nat Rev Genet,11(3),191-203, 2010

[2] Sun, Z. and Chai, H. S. and Wu, Y. and White, W. M. and Donkena, K. V. and Klein, C. J. and Garovic, V. D. and Therneau, T. M. and Kocher, J. P. *Batch effect correction for genome-wide methylation data with Illumina Infinium platform*, BMC Med Genomics, 4, 84, 2011.

[3] Siegmund, K. D., *Statistical approaches for the analysis of DNA methylation microarray data*, Hum Genet, 129(6),585-95, 2011.

[4] Jones, P. A. and Baylin, S. B., *The epigenomics of cancer*, Cell, 128(4), 683-92, 2007.

[5] Lister, R. and Pelizzola, M. and Dowen, R. H. and Hawkins, R. D. and Hon, G. and Tonti-Filippini, J. and Nery, J. R. and Lee, L. and Ye, Z. and Ngo, Q. M. and Edsall, L. and Antosiewicz-Bourget, J. and Stewart, R. and Ruotti, V. and Millar, A. H. and Thomson, J. A. and Ren, B. and Ecker, J. R., *Human DNA methylomes at base resolution show widespread epigenomic differences*, Nature, 462(7271), 315-22, 2009.

[6] Fuke, C. and Shimabukuro, M. and Petronis, A. and Sugimoto, J. and Oda, T. and Miura, K. and Miyazaki, T. and Ogura, C. and Okazaki, Y. and Jinno, Y., *Age related changes in 5-methylcytosine content in human peripheral leukocytes and placentas: an HPLC-based study*, Ann Hum Genet, 68(3), 196-204, 2004.