Minfi tutorial
BioC2014

Jean-Philippe Fortin Kasper D. Hansen

July 2014

1 Introduction

The goal of the tutorial is to present a standard analysis workflow of 450K data with the package minfi, incorporating the functions recently added to the package. We invite you to read the software paper recently published [1] and the online package vignette on the Bioconductor project [2] for more details.

We will start from the very beginning by reading input raw data (IDAT files) from an example dataset, and ending with a list of candidate genes for differential methylation. Among others, we will cover quality control assessments, different within-array and between-array normalizations, SNPs removal, sex prediction, differentially methylated positions (DMPs) analysis and bump hunting for differentially methylated regions (DMRs).

If time permits, we will introduce a complementary interactive visualization tool package, shinyMethyl that allows interactive quality control assessment. [3] You can download the package online from `gitHub` [4] and try it online at `http://spark.rstudio.com/jfortin/shinyMethyl/`

450k Array design and terminology

In this section, we introduce briefly the 450K array as well as the terminology used throughout the minfi package. Each sample is measured on a single array, in two different color channels (red and green). As the name of the platform indicates, each array measures more than 450,000 CpG positions. For each CpG, we have two measurements: a methylated intensity and an unmethylated intensity. Depending on the probe design, the signals are reported in different colors:

For **Type I** design, both signals are measured in the same color: one probe for the methylated signal and one probe for the unmethylated signal.

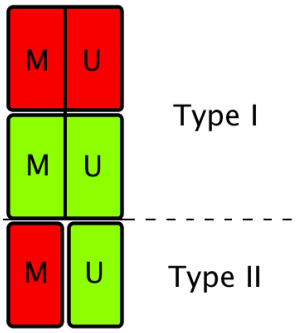


Figure 1: **Probe design of the 450k array.** For **Type I** design, both signals are measured in the same color: one probe for the methylated signal and one probe for the unmethylated signal. For **Type II** design, only one probe is used. The **Green** intensity measures the methylated signal, and the **Red** intensity measures the unmethylated signal.

For **Type II** design, only one probe is used. The **Green** intensity measures the methylated signal, and the **Red** intensity measures the unmethylated signal.

Some definitions

Two commonly measures are used to report the methylation levels: Beta values and M values.

Beta value:

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

where M and U denote the methylated and unmethylated signals respectively. In minfi the constant 100 can easily be changed, if needed.

MValue:

$$Mval = \log\left(\frac{M}{U}\right)$$

DMP: Differentially methylated position: single genomic position that has a different methylated level in two different groups of samples (or conditions)
DMR: Differentially methylated region: when consecutive genomic locations are differentially methylated in the same direction.
Array: One sample
Slide: Physical slide containing 12 arrays (6×2 grid)
Plate: Physical plate containing at most 8 slides (96 arrays). For this tutorial, we use **batch** and plate interchangeably.

2 Reading Data

The starting point of minfi is reading the .IDAT files with the built-in function `read.450k.exp`. Several options are available: the user can specify the sample filenames to be read in along with the directory path, or can specify the directory that contains the files. In the latter case, all the files with the extension .IDAT located in the directory will be loaded into R. The user can also read in a sample sheet, and then use the sample sheet to load the data into a `RGChannelSet`. For more information, see the `minfi` vignette. Here, we will load the dataset containing 6 samples from the `minfiData` package using the sample sheet provided within the package:

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

```
> baseDir <- system.file("extdata",package="minfiData")
> targets <- read.450k.sheet(baseDir)
```

```
[read.450k.sheet] Found the following CSV files:
[1] "/usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/SampleSheet.csv"
```

```
> targets
```

```
  Sample_Name Sample_Well Sample_Plate Sample_Group Pool_ID person age sex
1    GroupA_3          H5           NA       GroupA      NA    id3  83   M
2    GroupA_2          D5           NA       GroupA      NA    id2  58   F
3    GroupB_3          C6           NA       GroupB      NA    id3  83   M
4    GroupB_1          F7           NA       GroupB      NA    id1  75   F
5    GroupA_1          G7           NA       GroupA      NA    id1  75   F
6    GroupB_2          H7           NA       GroupB      NA    id2  58   F
  status  Array      Slide
1 normal R02C02 5723646052
2 normal R04C01 5723646052
3 cancer R05C02 5723646052
4 cancer R04C02 5723646053
5 normal R05C02 5723646053
6 cancer R06C02 5723646053
                                                                                                  Basename
1 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646052/5723646052_R02C02
2 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646052/5723646052_R04C01
3 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646052/5723646052_R05C02
4 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646053/5723646053_R04C02
5 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646053/5723646053_R05C02
6 /usr/local/R/R-3.1/lib/R/site-library/minfiData/extdata/5723646053/5723646053_R06C02
```

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

The class of RGSet is a `RGChannelSet` object. This is the initial object of a minfi analysis that contains the raw intensities in the green and red channels. Note that this object contains the intensities of the internal control probes as well. Because we read the data from a data sheet experiment, the phenotype data is also stored in the `RGChannelSet` and can be accessed via the accessor command `pData`:

```
> phenoData <- pData(RGSet)
> phenoData[,1:6]
```

```
                 Sample_Name Sample_Well Sample_Plate Sample_Group Pool_ID
5723646052_R02C02    GroupA_3          H5           NA       GroupA      NA
5723646052_R04C01    GroupA_2          D5           NA       GroupA      NA
5723646052_R05C02    GroupB_3          C6           NA       GroupB      NA
5723646053_R04C02    GroupB_1          F7           NA       GroupB      NA
5723646053_R05C02    GroupA_1          G7           NA       GroupA      NA
5723646053_R06C02    GroupB_2          H7           NA       GroupB      NA
                 person
5723646052_R02C02    id3
5723646052_R04C01    id2
5723646052_R05C02    id3
5723646053_R04C02    id1
5723646053_R05C02    id1
5723646053_R06C02    id2
```

The `RGChannelSet` stores also a manifest object that contains the probe design information of the array. This object is mainly of interest for developers; the package makes use of this behind the scenes.

```
> manifest <- getManifest(RGSet)
> manifest
```

```
IlluminaMethylationManifest object
Annotation
  array: IlluminaHumanMethylation450k
Number of type I probes: 135476
Number of type II probes: 350036
Number of control probes: 850
Number of SNP type I probes: 25
Number of SNP type II probes: 40
```

```
> head(getProbeInfo(manifest))
```

```
DataFrame with 6 rows and 8 columns
         Name    AddressA    AddressB       Color       NextBase
  <character> <character> <character> <character> <DNAStringSet>
1  cg00050873    32735311    31717405         Red               A
```

```
2  cg00212031    29674443    38703326         Red              T
3  cg00213748    30703409    36767301         Red              A
4  cg00214611    69792329    46723459         Red              A
5  cg00455876    27653438    69732350         Red              A
6  cg01707559    45652402    64689504         Red              A
                                          ProbeSeqA
                                       <DNAStringSet>
1 ACAAAAAAACAACACACAACTATAATAATTTTTAAAATAAATAAACCCCA
2 CCCAATTAACCACAAAAACTAAACAAATTATACAATCAAAAAAACATACA
3 TTTTAACACCTAACACCATTTTAACAATAAAAATTCTACAAAAAAAAACA
4 CTAACTTCCAAACCACACTTTATATACTAAACTACAATATAACACAAACA
5 AACTCTAAACTACCCAACACAAACTCCAAAAACTTCTCAAAAAAAACTCA
6 ACAAATTAAAAACACTAAAACAAACACAACAACTACAACAACAAAAAACA
                                          ProbeSeqB      nCpG
                                       <DNAStringSet> <integer>
1 ACGAAAAAACAACGCACAACTATAATAATTTTTAAAATAAATAAACCCCG         2
2 CCCAATTAACCGCAAAAACTAAACAAATTATACGATCGAAAAAACGTACG         4
3 TTTTAACGCCTAACACCGTTTTAACGATAAAAATTCTACAAAAAAAAACG         3
4 CTAACTTCCGAACCGCGCTTTATATACTAAACTACAATATAACGCGAACG         5
5 AACTCTAAACTACCCGACACAAACTCCAAAAACTTCTCGAAAAAAACTCG         2
6 GCGAATTAAAAACACTAAAACGAACGCGACGACTACAACGACAAAAAACG         6
```

See the minfi vignette for more information.

3 Class structure

minfi contains a number of classes corresponding to various transformations of the raw data. It is important to understand how these classes relate to each other. Figure 2 provides a useful overview.

MethylSet and RatioSet

A `MethylSet` objects contains the methylated and unmethylated signals. The most basic way to construct a `MethylSet` is to using the function `preprocessRaw` which uses the array design to match up the different probes and color channels to construct the methylated and unmethylated signals. This function does not do any normalization (see later).

```
> MSet <- preprocessRaw(RGSet)
> MSet
```

```
MethylSet (storageMode: lockedEnvironment)
assayData: 485512 features, 6 samples
```

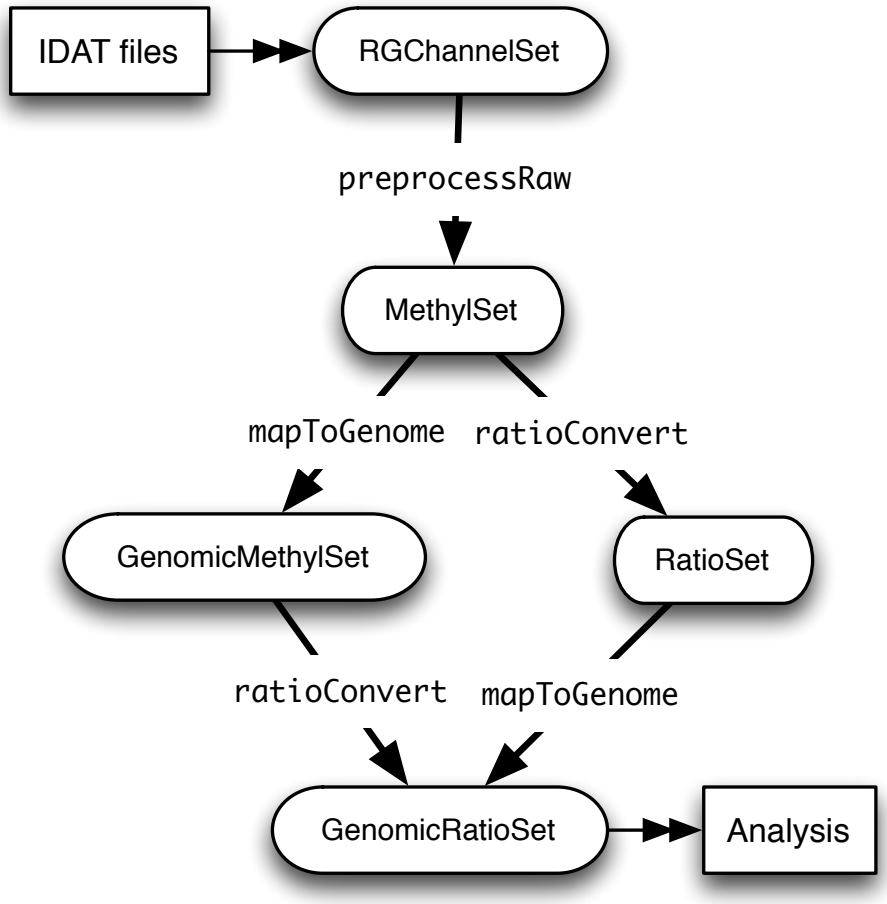


Figure 2: Flow chart of the `minfi`'s functions

```
  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: 1.10.2
  Manifest version: 0.4.0
```

The accessors `getMeth` and `getUnmeth` can be used to get the methylated and unmethylated intensities matrices:

```
> head(getMeth(MSet)[,1:3])
```

```
           5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
cg00050873             22041               588             20505
cg00212031               679               569               439
cg00213748              1620               421               707
cg00214611               449               614               343
cg00455876              5921               398              3257
cg01707559              1238               646               637
```

```
> head(getUnmeth(MSet)[,1:3])
```

```
           5723646052_R02C02 5723646052_R04C01 5723646052_R05C02
cg00050873              1945               433              1012
cg00212031              6567               300              2689
cg00213748               384               461               295
cg00214611              4869               183              1655
cg00455876              1655               792              1060
cg01707559             12227              1009              7414
```

A `RatioSet` object is a class designed to store Beta values and/or M values instead of the methylated and unmethylated signals. An optional copy number matrix, `CN`, the sum of the methylated and unmethylated signals, can be also stored. Mapping a `MethylSet` to a `RatioSet` is irreversible, i.e. one cannot technically retrieve the methylated and unmethylated signals from a `RatioSet`. A `RatioSet` can be created with the function `ratioConvert`:

```
> ratioSet <- ratioConvert(MSet, what = "both", keepCN = TRUE)
> ratioSet
```

```
RatioSet (storageMode: lockedEnvironment)
assayData: 485512 features, 6 samples
  element names: Beta, CN, M
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: 1.10.2
  Manifest version: 0.4.0
```

The functions `getBeta`, `getM` and `getCN` return respectively the Beta value matrix, M value matrix and a the Copy Number matrix.

```
> beta <- getBeta(ratioSet)
```

Why do we have these two classes? This is to allow methods development where normalization is done directly on the beta and/or M-values, such as quantile normalization of the Beta matrix (which we btw. do not recommend).

Mapping to the Genome

The function `mapToGenome` applied to a `RatioSet` object will add genomic coordinates to each probe together with some additional annotation information. The output object is a `GenomicRatioSet` (class holding M or/and Beta values together with associated genomic coordinates). It is possible to merge the manifest object with the genomic locations by setting the option `mergeManifest` to `TRUE`.

```
> gset <- mapToGenome(ratioSet)
> gset
```

```
class: GenomicRatioSet
dim: 485512 6
exptData(0):
assays(3): Beta M CN
rownames(485512): cg13869341 cg14008030 ... cg08265308 cg14273923
rowData metadata column names(0):
colnames(6): 5723646052_R02C02 5723646052_R04C01 ... 5723646053_R05C02
  5723646053_R06C02
```

```
colData names(13): Sample_Name Sample_Well ... Basename filenames
Annotation
  array: IlluminaHumanMethylation450k
  annotation: ilmn12.hg19
Preprocessing
  Method: Raw (no normalization or bg correction)
  minfi version: 1.10.2
  Manifest version: 0.4.0
```

Note that the `GenomicRatioSet` extends the class `SummarizedExperiment`. Here are the main accessors functions to access the data:

```
> beta <- getBeta(gset)
> m <- getM(gset)
> cn <- getCN(gset)

> sampleNames <- sampleNames(gset)
> probeNames <- featureNames(gset)
> pheno <- pData(gset)
```

To return the probe locations as a `GenomicRanges` objects, one can use the accessor `granges`:

```
> gr <- granges(gset)
> head(gr, n= 3)

GRanges with 3 ranges and 0 metadata columns:
             seqnames         ranges strand
                <Rle>      <IRanges>  <Rle>
  cg13869341     chr1 [15865, 15865]      *
  cg14008030     chr1 [18827, 18827]      *
  cg12045430     chr1 [29407, 29407]      *
  ---
  seqlengths:
    chr1  chr2  chr3  chr4  chr5  chr6 ... chr19 chr20 chr21 chr22  chrX  chrY
      NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA    NA
```

We can similary use `mapToGenome` on a `MethylSet` to get a `GenomicMethylSet`.

Annotation

To access the full annotation, one can use the command `getAnnotation`:

```
> annotation <- getAnnotation(gset)
> names(annotation)
```

```
 [1] "chr"                       "pos"
 [3] "strand"                    "Name"
 [5] "AddressA"                  "AddressB"
 [7] "ProbeSeqA"                 "ProbeSeqB"
 [9] "Type"                      "NextBase"
[11] "Color"                     "Probe_rs"
[13] "Probe_maf"                 "CpG_rs"
[15] "CpG_maf"                   "SBE_rs"
[17] "SBE_maf"                   "Islands_Name"
[19] "Relation_to_Island"        "Forward_Sequence"
[21] "SourceSeq"                 "Random_Loci"
[23] "Methyl27_Loci"             "UCSC_RefGene_Name"
[25] "UCSC_RefGene_Accession"    "UCSC_RefGene_Group"
[27] "Phantom"                   "DMR"
[29] "Enhancer"                  "HMM_Island"
[31] "Regulatory_Feature_Name"   "Regulatory_Feature_Group"
[33] "DHS"
```

The order and content of the annotation `DataFrame` is in the same order as the main object (here: `gset`). There are a number of convenience functions to get parts of the annotation, like

```
> islands <- getIslandStatus(gset)
> head(islands)
```

```
[1] "OpenSea" "OpenSea" "Island"  "Island"  "Island"  "OpenSea"
```

```
> probeType <- getProbeType(gset)
> head(probeType)
```

```
[1] "I"  "II" "I"  "I"  "I"  "II"
```

(see later regarding SNPs). Also, you can get different subsets of the annotation by using the `what` argument.

4 Quality control

Efficient and reliable quality control is important. Our view on this has evolved over time and currently we recommend using the qc plot described below as well as shinyMethyl for interactive visualization. We conclude this section with some comments on what we don't recommend using.

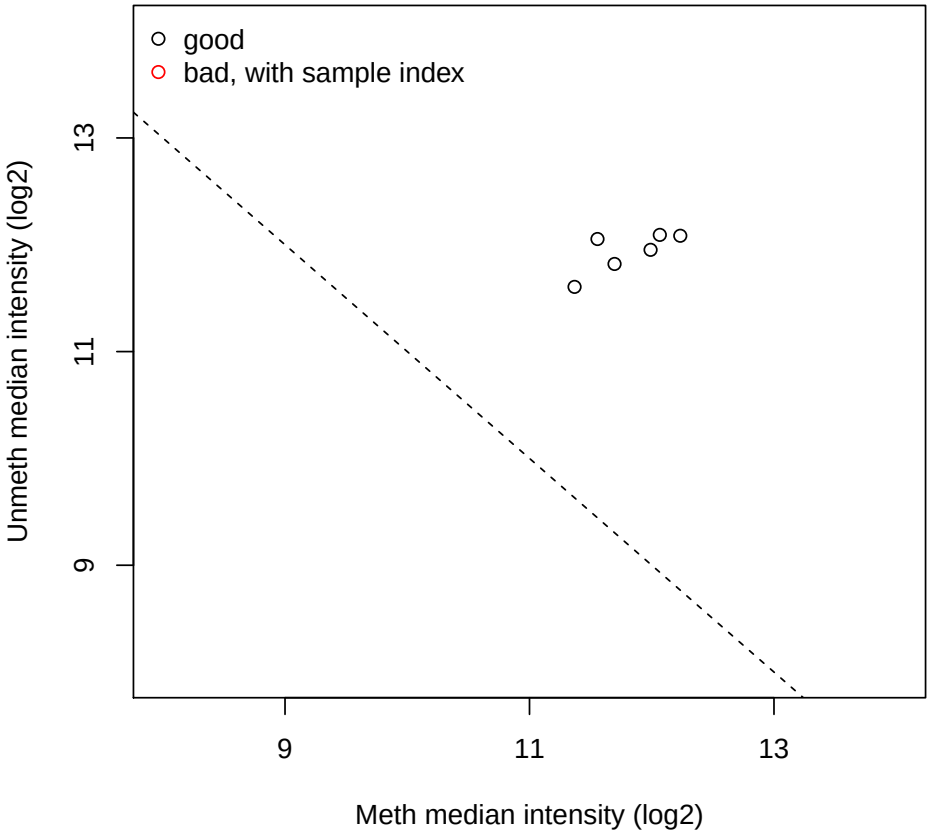
QC plot

minfi provides a simple quality control plot that uses the log median intensity in both the methylated (M) and unmethylated (U) channels. When plotting these two medians against each other, it has been observed that good samples cluster together, while failed samples tend to separate and have lower median intensities [1]. In general, we advice users to make the plot and make a judgement. The line separating "bad" from "good" samples represent a useful cutoff, which may have to be adapted to a specific dataset. The functions `getQC` and `plotQC` are designed to extract and plot 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  11.69566  11.82058
5723646052_R04C01  11.99046  11.95274
5723646052_R05C02  11.55603  12.05393
5723646053_R04C02  12.06609  12.09276
5723646053_R05C02  12.23332  12.08448
5723646053_R06C02  11.36851  11.60594
```

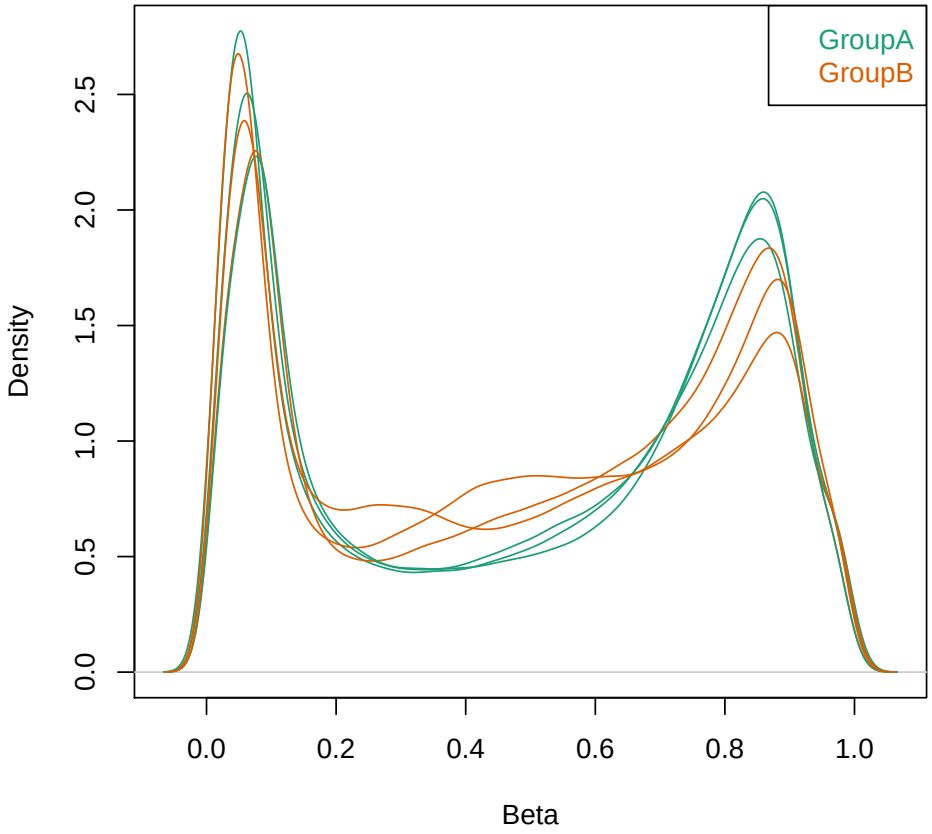
```
> plotQC(qc)
```



Moreover, the function `addQC` applied to the `MethylSet` will add the QC information to the phenotype data.

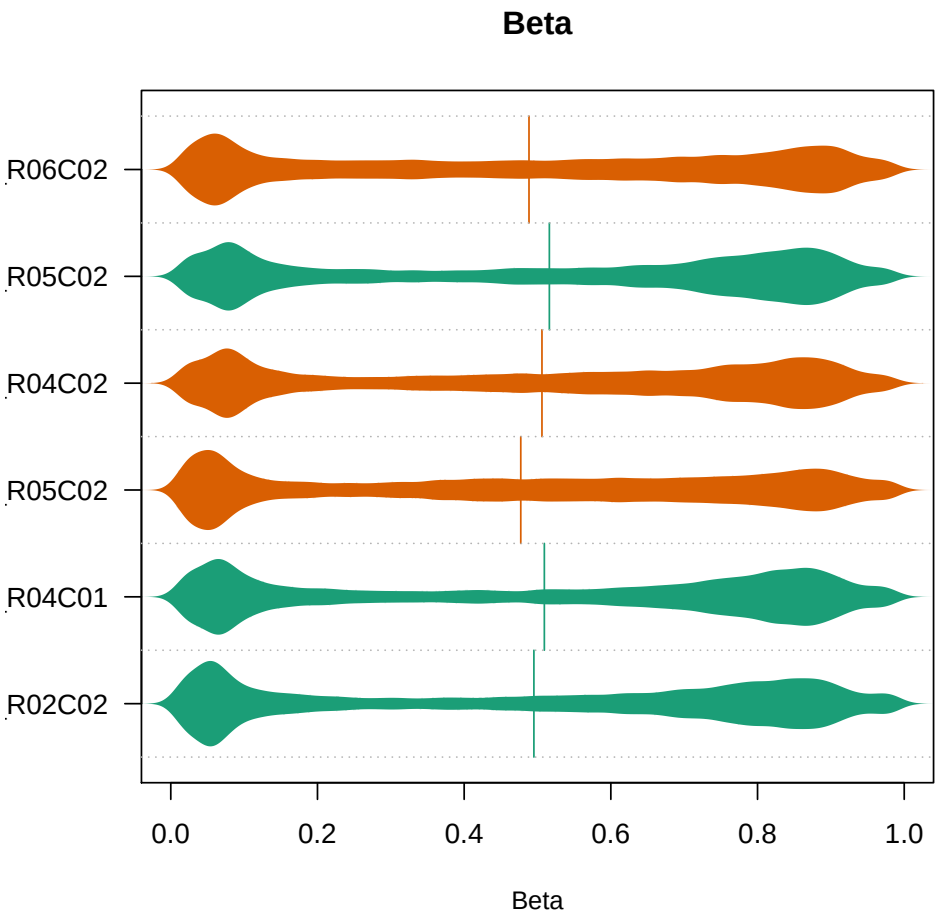
To further explore the quality of the samples, it is useful to look at the Beta value densities of the samples, with the option to color the densities by group:

```
> densityPlot(MSet, sampGroups = phenoData$Sample_Group)
```



or density bean plots:

```
> densityBeanPlot(MSet, sampGroups = phenoData$Sample_Group)
```



shinyMethyl [3] is particularly useful to visualize all plots at the same time in an interactive fashion.

Control probes plot

The 450k array contains several internal control probes that can be used to assess the quality control of different sample preparation steps (bisulfite conversion, hybridization, etc.). The values of these control probes are stored in the initial `RGChannelSet` and can be plotted by using the function `controlStripPlot` and by specifying the control probe type:

```
> controlStripPlot(RGSet, controls="BISULFITE CONVERSION II")
```

All the plots above can be exported into a pdf file in one step using the function `qcReport`:

```
> qcReport(RGSet, pdf= "qcReport.pdf")
```

In practice, we use the QC plot presented above as well as inspection of the bisulfite conversion probes and marginal densities to do QC. We don't really use the `qcReport` anymore.

5 SNPs

Because the presence of SNPs inside the probe body or at the nucleotide extension can have important consequences on the downstream analysis, minfi offers the possibility to remove such probes. The function `getSnpInfo`, applied to a `GenomicRatioSet`, returns a data frame with 6 columns containing the SNP information of the probes:

```
> snps <- getSnpInfo(gset)
> head(snps,10)
```

```
DataFrame with 10 rows and 6 columns
              Probe_rs Probe_maf      CpG_rs   CpG_maf      SBE_rs   SBE_maf
            <character> <numeric> <character> <numeric> <character> <numeric>
cg13869341           NA        NA          NA        NA          NA        NA
cg14008030           NA        NA          NA        NA          NA        NA
cg12045430           NA        NA          NA        NA          NA        NA
cg20826792           NA        NA          NA        NA          NA        NA
cg00381604           NA        NA          NA        NA          NA        NA
cg20253340           NA        NA          NA        NA          NA        NA
cg21870274           NA        NA          NA        NA          NA        NA
cg03130891   rs77418980  0.305556          NA        NA          NA        NA
cg24335620 rs147502335  0.012800          NA        NA          NA        NA
cg16162899           NA        NA          NA        NA          NA        NA
```

`Probe`, `CpG` and `SBE` correspond the SNPs present inside the probe body, at the CpG interrogation and at the single nucleotide extension respectively. The columns with `rs` give the names of the SNPs while the columns with `maf` gives the minor allele frequency of the SNPs based on the dbSnp database. The function `addSnpInfo` will add to the `GenomicRanges` of the `GenomicRatioSet` the 6 columns.:

```
> gset <- addSnpInfo(gset)
> head(granges(gset))
```

```
GRanges with 6 ranges and 6 metadata columns:
             seqnames         ranges strand |    Probe_rs Probe_maf      CpG_rs
                <Rle>      <IRanges>  <Rle> | <character> <numeric> <character>
  cg13869341     chr1 [15865, 15865]      * |        <NA>      <NA>        <NA>
  cg14008030     chr1 [18827, 18827]      * |        <NA>      <NA>        <NA>
  cg12045430     chr1 [29407, 29407]      * |        <NA>      <NA>        <NA>
  cg20826792     chr1 [29425, 29425]      * |        <NA>      <NA>        <NA>
```

```
  cg00381604     chr1 [29435, 29435]      * |        <NA>      <NA>        <NA>
  cg20253340     chr1 [68849, 68849]      * |        <NA>      <NA>        <NA>
              CpG_maf      SBE_rs   SBE_maf
            <numeric> <character> <numeric>
  cg13869341      <NA>        <NA>      <NA>
  cg14008030      <NA>        <NA>      <NA>
  cg12045430      <NA>        <NA>      <NA>
  cg20826792      <NA>        <NA>      <NA>
  cg00381604      <NA>        <NA>      <NA>
  cg20253340      <NA>        <NA>      <NA>
  ---
  seqlengths:
    chr1  chr2  chr3  chr4  chr5  chr6 ... chr19 chr20 chr21 chr22  chrX  chrY
      NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA    NA
```

We strongly recommend to drop the probes that contain either a SNP at the CpG interrogation or at the single nucleotide extension. The function `dropLociWithSnps` allows to drop the corresponding probes (introduced in minfi 1.11.9). Here is an example where we drop the probes containing a SNP at the CpG interrogation and/or at the single nucleotide extension, for any minor allele frequency:

```
> gset <- dropLociWithSnps(gset, snps=c("SBE","CpG"), maf=0)
```

There are several options for SNP databases. These are contained inside the annotation object. A list of databases can be had by printing the annotation object. In mifni 1.11.8 we made it possible to do the following

```
> getAnnotationObject(gset)
```

In earlier versions of minfi, you can get the same outcome by first getting the name of the annotation object and then printing it, by

```
> annoStr <- paste(annotation(MsetEx), collapse = "anno.")
> annoStr
```

```
[1] "IlluminaHumanMethylation450kanno.ilmn12.hg19"
```

```
> get(annoStr)
```

```
IlluminaMethylationAnnotation object
Annotation
  array: IlluminaHumanMethylation450k
  annotation: ilmn12
  genomeBuild: hg19
Available annotation
  Islands.UCSC
  Locations
```

```
  Manifest
  Other
  SNPs.132CommonSingle
  SNPs.135CommonSingle
  SNPs.137CommonSingle
  SNPs.Illumina
Defaults
  Locations
  Manifest
  SNPs.137CommonSingle
  Islands.UCSC
  Other
```

Cross-reactive probes

It has been previously reported than about 6% of the probes on the 450K array co-hybridize to alternate genomic sequences, therefore potentially generating spurious signals [5]. We are planning to include a function in minfi that drops these cross-reactive probes. The function can be either applied to a `[Genomic]MethylSet` or a `[Genomic]RatioSet`:

```
> gset <- dropCrossReactiveProbes(gset)
```

This functionality is currently being tested.

6 Preprocessing and normalization

So far, we did not use any normalization to process the data. Different normalization procedures are available in minfi.

preprocessRaw

As seen before, it converts a `RGChannelSet` to a `MethylSet` by converting the Red and Green channels into a matrix of methylated signals and a matrix of unmethylated signals. No normalization is performed.

Input: `RGChannelSet`
Output: `MethylSet`

preprocessIllumina

Convert a `RGChannelSet` to a `MethylSet` by implementing the preprocessing choices as available in Genome Studio: background subtraction and control normalization. Both of them are optional and turning them off is equivalent to raw preprocessing (`preprocessRaw`):

```
> MSet.illumina <- preprocessIllumina(RGSet, bg.correct = TRUE,
+                                     normalize = "controls")
```

Input: `RGChannelSet`
Output: `MethylSet`

preprocessSWAN

Perform Subset-quantile within array normalization (SWAN) [6], a within-array normalization correction for the technical differences between the Type I and Type II array designs. The algorithm matches the Beta-value distributions of the Type I and Type II probes by applying a within-array quantile normalization separately for different subsets of probes (divided by CpG content). The input of SWAN is a `MethylSet`, and the function returns a `MethylSet` as well. If an `RGChannelSet` is provided instead, the function will first call `preprocessRaw` on the `RGChannelSet`, and then apply the SWAN normalization. We recommend setting a seed (using `set.seed`) before using `preprocessSWAN` to ensure that the normalized intensities will be reproducible.

```
> MSet.swan <- preprocessSWAN(RGSet)
```

Input: `RGChannelSet` or `MethylSet`
Output: `MethylSet`

preprocessQuantile

This function implements stratified quantile normalization preprocessing. The normalization procedure is applied to the Meth and Unmeth intensities separately. The distribution of type I and type II signals is forced to be the same by first quantile normalizing the type II probes across samples and then interpolating a reference distribution to which we normalize the type I probes. Since probe types and probe regions are confounded and we know that DNAm distributions vary across regions we stratify the probes by region before applying this interpolation. Note that this algorithm relies on the assumptions necessary for quantile normalization to be applicable and thus is not recommended for cases where global changes are expected such as in cancer-normal comparisons. Note that this normalization procedure is essentially similar to one previously presented [7]. The different options can be summarized into the following list:

1) If `fixMethOutlier` is `TRUE`, the functions fixes outliers of both the methylated and unmethylated channels when small intensities are close to zero.

2) If `removeBadSamples` is `TRUE`, it removes bad samples using the QC criterion discussed previously

3) Performs stratified subset quantile normalization if `quantileNormalize=TRUE` and `stratified=TRUE`

4) Predicts the sex (if not provided in the `sex` argument) using the function `getSex` and normalizes males and females separately for the probes on the X and Y chromosomes

```
> gset.quantile <- preprocessQuantile(RGSet, fixOutliers = TRUE,
+ removeBadSamples = TRUE, badSampleCutoff = 10.5,
+ quantileNormalize = TRUE, stratified = TRUE,
+ mergeManifest = FALSE, sex = NULL)
```

```
[preprocessQuantile] Mapping to genome.
[preprocessQuantile] Fixing outliers.
[preprocessQuantile] Quantile normalizing.
```

Input: `RGChannelSet`
Output: `GenomicRatioSet`

Note that the function returns a `GenomicRatioSet` object ready for downstream analysis.

preprocessFunnorm

The function `preprocessFunnorm` implements the functional normalization algorithm developed in [8]. Briefly, it uses the internal control probes present on the array to infer between-array technical variation. It is particularly useful for studies comparing conditions with known large-scale differences, such as cancer/normal studies, or between-tissue studies. It has been shown that for such studies, functional normalization outperforms other existing approaches [8]. By default, is uses the first two principal components of the control probes to infer the unwanted variation.

```
> gset.funnorm <- preprocessFunnorm(RGSet)
```

```
[preprocessFunnorm] Mapping to genome
[preprocessFunnorm] Quantile extraction
[preprocessFunnorm] Normalization
```

Input: `RGChannelSet`
Output: `GenomicRatioSet`

As the `preprocessQuantile` function, it returns a `GenomicRatioSet` object.

7 dmpFinder: to find differentially methylated positions (DMPs)

While we do not encourage particularly a single position differential methylation analysis, minfi implements a simple algorithm called `dmpFinder` to find differentially methylated positions with respect to a phenotype covariate. The phenotype may be categorical (e.g. cancer vs. normal) or continuous (e.g. blood pressure). Below is an example of a DMP analysis for age using the `gset.funnorm` object created above:

```
> beta <- getBeta(gset.funnorm)
> age  <- pData(gset.funnorm)$age
> dmp <- dmpFinder(beta, pheno = age  , type = "continuous")
> head(dmp)
```

```
                 intercept          beta         t         pval      qval
cg05764847       0.9984524 -0.0007670769 -51.86582 8.270863e-07 0.4015603
cg10467968       0.3434299  0.0061458131  33.35419 4.818942e-06 0.9999983
cg00135841      -0.6673803  0.0189247545  28.63051 8.857500e-06 0.9999983
ch.17.28266240F  0.1352752 -0.0008316485 -26.87454 1.139697e-05 0.9999983
cg05389038       0.5306783 -0.0056651940 -22.63167 2.257633e-05 0.9999983
cg21498547      -1.8114330  0.0323934580  21.46843 2.784158e-05 0.9999983
```

8 Bumphunter: to find differentially methylated regions (DMRs)

The `bumphunter` function in minfi is a version of the bump hunting algorithm [9] adapted to the 450k array, relying on the `bumphunter` function implemented in the eponym package bumphunter [10].

Instead of looking for association between a single genomic location and a phenotype of interest, `bumphunter` looks for genomic regions that are differentially methylated between two conditions. In the context of the 450k array, the algorithm first defines *clusters* of probes. Clusters are simply groups of probes such that two consecutive probe locations in the cluster are not separated by more than some distance `mapGap`. Briefly, the algorithm first computes a t-statistic at each genomic location, with optional smoothing. Then, it defines a candidate region to be a cluster of probes for which all the t-statistics exceed a predefined threshold. To test for significance of the candidate regions, the algorithm uses permutations (defined by the parameter B). The permutation scheme is expensive, and can take a few days when the number of candidate bumps is large. To avoid wasting time, we propose the following guideline:

- Define your phenotype of interest

```
> pheno <- pData(gset.funnorm)$status
> designMatrix <- model.matrix(~ pheno)
```

- Run the algorithm with $B = 0$ permutation on the Beta-values, with a medium difference cutoff, say 0.2 (which corresponds to 20% difference on the Beta-values):

```
> dmrs <- bumphunter(gset.funnorm, design = designMatrix,
+                   cutoff = 0.2, B=0, type="Beta")
```

- If the number of candidate bumps is large, say > 30000, increase the cutoff to reduce the number of candidate bumps. The rationale behind this is that the most of the additional candidate regions found by lowering the cutoff will be found to be non-significant after the permutation scheme, and therefore time can be saved by being more stringent on the cutoff (high cutoff).

- Once you have decided on the cutoff, run the algorithm with a large number of permutations, say $B = 1000$:

```
> dmrs <- bumphunter(gset.funnorm, design = designMatrix,
+                   cutoff = 0.2, B=1000, type="Beta")
```

Since the permutation scheme can be expensive, parallel computation is implemented in the `bumphunter` function. The `foreach` package allows different parallel "back-ends" that will distribute the computation across multiple cores in a single machine, or across machines in a cluster. For instance, if one wished to use 3 cores, the two following commands have to be run before running bumphunter:

```
> library(doParallel)
> registerDoParallel(cores = 3)
```

The results of `bumphunter` are stored in a data frame with the rows being the different differentially methylated regions (DMRs):

```
> names(dmrs)
> head(dmrs$table, n=3)
```

As an example, we have run the bump hunting algorithm to find DMRs between colon and kidney (20 samples each from TCGA), with $B = 1000$ permutations, and a cutoff of 0.2 on the Beta values:

```
        chr     start       end      value     area cluster indexStart indexEnd
15861  chr8 145103393 145107199  0.3767581 6.404887  194325     238277   238293
4810  chr13 113425756 113428172  0.4257673 5.960743   57562     337302   337315
4064  chr12  54446019  54447349  0.3278039 5.900470   46543     311839   311856
17813 chr10  11206772  11208339 -0.4148710 5.393322   21466     251989   252001
18360 chr10 130844121 130844899 -0.5869309 5.282378   29724     269921   269929
4054  chr12  54409207  54409770  0.4370705 5.244846   46529     311742   311753
       L clusterL p.value fwer p.valueArea fwerArea
```

```
15861 17       17       0    0          0        0
4810  14       24       0    0          0        0
4064  18       28       0    0          0        0
17813 13       15       0    0          0        0
18360  9        9       0    0          0        0
4054  12       42       0    0          0        0
```

The `start` and `end` columns indicate the limiting genomic locations of the DMR; the `value` column indicates the average difference in methylation in the bump, and the `area` column indicates the area of the bump with respect to the 0 line. The `fwer` column returns the family-wise error rate (FWER) of the regions estimated by the permeation scheme. One can filter the results by picking a cutoff on the FWER.

9 Other important topics

Batch effects correction with SVA

Surrogate variable analysis (SVA) [11, 12] is a useful tool to identified surrogate variables for unwanted variation while protecting for a phenotype of interest. In our experience, running SVA after normalizing the 450K data with `preprocessFunnorm` or `preprocessQuantile` increases the statistical power of the downstream analysis. For instance, to run SVA on the M-values, protecting for case-control status, the following code can be used to estimate the surrogate variables (this can take a few hours to run):

```
> require(sva)
> mval <- getM(gset)
> pheno <- pData(gset)
> mod <- model.matrix(~as.factor(status), data=pheno)
> mod0 <- model.matrix(~1, data=pheno)
> sva.results <- sva(mval, mod, mod0)
```

Once the surrogate variables are computed, one can include them in the downstream analysis to adjust for unknown unwanted variation. See sea package vignette for a more comprehensive use of `sva`.

Cell Type Composition

As shown in [13], biological findings in blood samples can often be confounded with cell type composition. In order to estimate the confounding levels between phenotype and cell type composition, the function `estimateCellCounts` depending on the package FlowSorted.Blood.450k [14] estimates the cell type composition of blood samples by using a mod-

ified version of the algorithm described in [15]. The function takes as input a `RGChannelSet` and returns a cell counts vector for each samples:

```
> require(FlowSorted.Blood.450k)
> cellCounts <- estimateCellCounts(RGSet)
```

Block finder

The approximately 170,000 open sea probes on the 450K array can be used to detect long-range changes in methylation status. These large scale changes that can range up to several Mb have typically been identified only through whole-genome bisulfite sequencing. The function `blockFinder` groups the average methylation values in open-sea probe cluster (via `cpgCollapse`) into large regions, and then run the `bumphunter` algorithm with a large (250KB+) smoothing window (see the bump hunting section for DMRs above).

Sex prediction

By looking at the median total intensity of the X chromosome-mapped probes, denoted $med(X)$, and the median total intensity of the Y-chromosome-mapped probes, denoted $med(Y)$, one can observe two different clusters of points corresponding to which gender the samples belong to. To predict the gender, `minfi` separate the points by using a cutoff on $\log_2 med(Y) - \log_2 med(Y)$. The default cutoff is -2. Since the algorithm needs to map probes to the X-chr and to the Y-chr, the input of the function `getSex()` needs to be a `GenomicMethylSet` or a `GenomicRatioSet`.

```
> predictedSex <- getSex(gset, cutoff = -2)$predictedSex
> head(predictedSex)
```

```
[1] "M" "F" "M" "F" "F" "F"
```

To choose the cutoff to separate the two gender clusters, one can plot $med(Y)$ against $med(Y)$ with the function `plotSex`:

```
> plotSex(getSex(gset, cutoff = -2))
```

Finally, the function `addSex` applied to the `GenomicRatioSet` will add the predicted sex to the phenotype data. *Remark*: the function does not handle datasets with only females or only males

10 Advanced functions

getSnpBeta

The array contains by design 65 probes that are not meant to interrogate methylation status, but instead are designed to interrogate SNPs. By default, minfi drops these probes. The function `getSnpBeta` devel version allows the user to extract the Beta values for those probes from an `RGChannelSet`. The return object is a matrix with the columns being the samples and the rows being the different SNP probes:

```
> snps <- getSnpBeta(rgset)
> head(snaps)
```

These SNP probes are intended to be used for sample tracking and sample mixups. Each SNP probe ought to have values clustered around 3 distinct values corresponding to homo-, and hetero-zygotes.

Out-of-band (or ghost) probes

The function `getOOB` applied to an `RGChannelSet` retrieves the so-called "out-of-band" (OOB) probes. These are the measurements of Type I probes in the "wrong" color channel. The function returns a list with two matrices, named `Red` and `Grn`.

```
> oob <- getOOB(rgset)
```

11 Exercises

1) Before processing a RGChannelSet further, could you remove the probes which failed more than 50% of the samples in the example dataset?

2) For the top loci that we found differentially methylated for the predicted sex, could you tell if those loci are mostly mapped to the X and Y chromosomes?

3) It is known that the Beta-value distribution of the Type I probes is different from the Beta value distribution of the Type II probes. Can you verify this with by plotting the Beta-value distribution density for each type separately?

References

[1] Martin J. Aryee, Andrew E. Jaffe, Hector Corrada-Bravo, Christine Ladd-Acosta, Andrew P. Feinberg, Kasper D. Hansen, and Rafael A. Irizarry. Minfi: A flexible and

comprehensive Bioconductor package for the analysis of Infinium DNA Methylation microarrays. *Bioinformatics*, 2014. doi: 10.1093/bioinformatics/btu049.

[2] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y H Yang, and Jianhua Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004. doi: 10.1186/gb-2004-5-10-r80.

[3] Jean-Philippe Fortin, Elana J Fertig, and Kasper D Hansen. shinyMethyl: interactive quality control of Illumina 450k DNA methylation arrays in R. *F1000Research*, 3(175), 2014. doi: 10.12688/f1000research.4680.1.

[4] Jean-Philippe Fortin and Kasper D. Hansen. *shinyMethyl repository*, 2014. URL https://github.com/Jfortin1/shinyMethyl. Online.

[5] Yi-an Chen, Mathieu Lemire, Sanaa Choufani, Darci T Butcher, Daria Grafodatskaya, Brent W Zanke, Steven Gallinger, Thomas J Hudson, and Rosanna Weksberg. Discovery of cross-reactive probes and polymorphic cpgs in the illumina infinium humanmethylation450 microarray. *Epigenetics*, 8(2):203–9, 2013. doi: 10.4161/epi.23470.

[6] Jovana Maksimovic, Lavinia Gordon, and Alicia Oshlack. SWAN: Subset quantile Within-Array Normalization for Illumina Infinium HumanMethylation450 BeadChips. *Genome Biology*, 13(6):R44, 2012. doi: 10.1186/gb-2012-13-6-r44.

[7] Nizar Touleimat and Jörg Tost. Complete pipeline for Infinium Human Methylation 450K BeadChip data processing using subset quantile normalization for accurate DNA methylation estimation. *Epigenomics*, 4(3):325–341, 2012. doi: 10.2217/epi.12.21.

[8] Jean-Philippe Fortin, Aurelie Labbe, Mathieu Lemire, Brent W. Zanke, Thomas J. Hudson, Elana J. Fertig, Celia M.T. Greenwood, and Kasper D. Hansen. Functional normalization of 450k methylation array data improves replication in large cancer studies. *bioRxiv*, 2014. doi: 10.1101/002956. URL http://biorxiv.org/content/early/2014/02/23/002956.

[9] Andrew E Jaffe, Peter Murakami, Hwajin Lee, Jeffrey T Leek, M Daniele Fallin, Andrew P Feinberg, and Rafael A Irizarry. Bump hunting to identify differentially methylated regions in epigenetic epidemiology studies. *International Journal of Epidemiology*, 41(1):200–209, 2012. doi: 10.1093/ije/dyr238.

[10] Rafael A. Irizarry, Martin Aryee, Hector Corrada Bravo, Kasper D. Hansen, and Harris A. Jaffee. *bumphunter: Bump Hunter*. R package version 1.2.0.

[11] Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genetics*, 3(9):1724–1735, 2007. doi: 10.1371/journal.pgen.0030161.

[12] Jeffrey T Leek and John D Storey. A general framework for multiple testing dependence. *Proceedings of the National Academy of Sciences*, 105(48):18718–18723, 2008. doi: 10.1073/pnas.0808709105.

[13] Andrew E Jaffe and Rafael A Irizarry. Accounting for cellular heterogeneity is critical in epigenome-wide association studies. *Genome Biol*, 15(2):R31, Feb 2014. doi: 10.1186/gb-2014-15-2-r31.

[14] Andrew E Jaffe. *FlowSorted.Blood.450k: Illumina HumanMethylation data on sorted blood cell populations*. R package version 1.0.2.

[15] Eugene Andres Houseman, William P Accomando, Devin C Koestler, Brock C Christensen, Carmen J Marsit, Heather H Nelson, John K Wiencke, and Karl T Kelsey. Dna methylation arrays as surrogate measures of cell mixture distribution. *BMC Bioinformatics*, 13:86, 2012. doi: 10.1186/1471-2105-13-86.