Using funtooNorm

Celia Greenwood, Stepan Grinek, Kathleen Oros Klein

March 17, 2016

1 Introduction

The `funtooNorm` package provides a function for normalization of Illumina Infinium Human Methylation 450 BeadChip (Illumina 450K) data when there are samples from multiple tissues or cell types. The algorithm in this package represents an extension of a normalization method introduced recently by [**?**, **?**]. In addition to the percentile-specific adjustments implemented in funNorm, funtooNorm is specifically designed to allow flexibility in the adjustments for different cell types or tissue types. Percentiles of methylation levels may vary across cell types and hence the original funNorm may not be ideal if applied to all samples simultaneously. Normalization separately for each cell type may introduce unwanted variability if sample sizes are small. Therefore, this algorithm provides flexibility while optimizing the sample size used to estimate the corrections.

Note that the current version of the package does not do a good job of normalizing the Y chromosome probes; the funNorm method performs better. In a subsequent version of the package we will address this issue.

2 Package use

2.1 Terminology

As the `minfi` vignette describes nicely, the 450k array contains two types of probes:

> 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.

Therefore, we dissociate the 6 types of signals in our method : `AIGrn`, `BIGrn`, `AIRed`, `BIRed`, `AII` and `BII`, where the **A** (methylated) and **B** (unmethylated) are on **Green** or **Red** chanel depending **Type I** (Red or Green) and **Type II**. We will carefully talk about position when refering to a CpG and not about probe since the number of probe per position depend on the type of these position.

The **beta** value is computed with a default offset of `100` like ILLUMINA standard but can be easily change.

2.2 Reading Data

The package use a `SampleSet` on which you can apply functions. It will contain your chip data and a matching cell type for each sample. The first step will be to provide your data to a new `SampleSet`. There is two way to load your data to the package. Using the output of `GenomeStudio` or using raw `IDAT` files and use `minfi` package:

- `GenomeStudio:` The function `fromGenStudFiles` take three arguments, the control probes file, the signal intensity file (we should add some specific information about gGenomeStudio if necessary here), and the cell_type vector. There is two way to pass your data to the package.
- `From IDAT:` Using of `minfi` package, you should create a `RGChannelSet` object containing all your samples and use the function `fromRGChannelSet` to create your `SampleSet`. Please reffer to `minfi` vignette on how to create a `RGChannelSet`. The phenotype data of your object should contain a column name `cell_type`, you can access it via `pData()`. There must be at least two different cell or tissue types in the data or the program will terminate with a warning.

2.3 DataSet

We have provided a small data set containing $N = 129$ samples from chip 450K to demonstrate the usage of the package. The samples are from different types plasma cells.

```
>     require(funtooNorm)
>     require(minfiData)
>     pData(RGsetEx)$cell_type <- rep(c("type1","type2"),3)
>     mySampleSet=fromRGChannelSet(RGsetEx)
>     mySampleSet

SampleSet object built from  minfi
Data:  485577 positionsand  6 samples
   cell type:
   528 quantiles
funtooNorm Normalization was not applied

>
```

Now you get the sampleSet ready for normalisation, you can already get the Beta value before normalization

```
>     origBeta <- getRawBeta(mySampleSet)
```

Before normalizing you need to choose the ideal number of component for your data, we have set 4 as the default value for `ncmp`.

Choice of the number of components can be facilitated by examining a series of fits with different numbers of components : Calling the `plotValidationGraph` function with `type.fits = "PCR"` produces a set of plots, showing the root mean squared errors from cross-validated fits, for different numbers of components, SEPARATELY for each type of signal ("AIGrn" "BIGrn" "AIRed" "BIRed" "AII" and "BII"). By looking at figures **??** the goal is to choose the smallest value of `ncmp` where the cross-validated root mean squared error is fairly small across the quantiles. By default, funtooNorm will perform 10-fold cross-validation, but this can be changed with the parameter `ncv.fold`. You can set `type.fits = "validationcurve.pdf"` or change the type of fit to `"PLS"`

```
>     plotValidationGraph(mySampleSet, type.fits="PCR")
```

Here is a basic call to normalize this sample data set: funtooNorm will fit either principal component regression (PCR) or partial least squares regression (PLS) by specifying `type.fits="PCR"` or`type.fits="PLS"`. The default is set to "PCR" to match funNorm. An important user-chosen parameter is `ncmp`, the number of components to be included in either of these two models; these components are calculated from the control probe data and cell type data.

```
>     #This call  will perform cross validation to find optimal value
>     #of parameter ncmp for PCR regression:
>     mySampleSet=funtooNorm(mySampleSet,type.fits="PCR",ncmp=3)
>     mySampleSet

SampleSet object built from  minfi
Data:  485577 positionsand  6 samples
   cell type:
   528 quantiles
funtooNorm Normalization was applied

>     normBeta <- getNormBeta(mySampleSet)
>
```

To assess the performance of normalization function one can use a measure of intra-replicate differences `M`, described in [**?**]. We provide a function `agreement` implementing this measure. It takes as arguments a matrix of beta values and a vector of individual ID's. For the function to work some elements of individual's vector, obviously, should be identical. The returned value of `M` is expected to be similar for the data before and after normalization:

```
> #agreement(origBeta, c(1:5,5)) # M for data before the normalization
> #agreement(normBeta, c(1:5,5)) # M for data after normalization
```

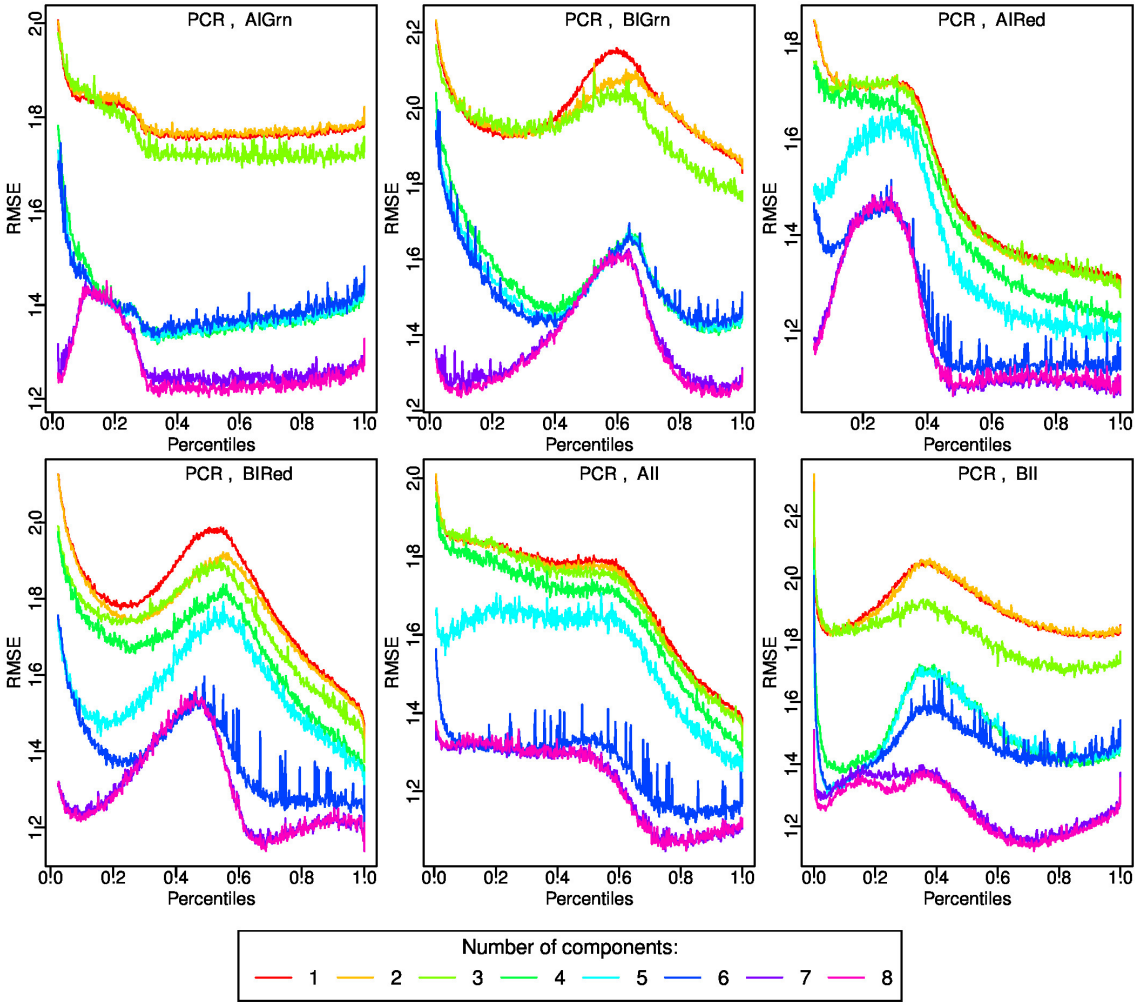

Figure 1: Cross-validated root mean squared errors across percentiles of the signal distributions for different numbers of PCR components. Top: signal A; Bottom: signal B; Left: probe type I red; Middle: probe type I green; Right: probe type II.

3 FuntooNorm and the minfi package

The `minfi` package [**?**] contains several tools for analyzing and visualizing Illumina's 450k array data. This section shows the interoperability of this package with the `funtooNorm` package.

```
> library(minfi)
```

3.1 Downstream use of the funtooNorm output

Since normalization is rarely the final goal, this section shows how to convert the output of `funtooNorm()` (the `funtoonromout` object created in section **??**) to a `GenomicRatioSet` object, so that it can be used by other tools in `minfi` like `bumphunter()` or `blockFinder()`.

A `GenomicRatioSet` object requires some phenotype information, so the following creates a `DataFrame`[1] with (random) gender information.

```
> #phenoData <- DataFrame(Sample_Name=colnames(funtoonormout$newBeta),
> #                       sex=sample(c("M", "F"), 93, replace=TRUE))
> #rownames(phenoData) <- phenoData$Sample_Name
>

> #includedProbes <- Annot[which(Annot$probe %in%
> #                              rownames(funtoonormout$newBeta)),]
> #genomerange <- GRanges(seqnames=includedProbes$probe,
> #                       ranges=includedProbes$Mapinfo, strand=NA)
>
> #grs <- GenomicRatioSet(gr=genomerange,
> #                       Beta=funtoonormout$newBeta,
> #                       preprocessMethod="funtooNorm",
> #                       pData=phenoData)
>
```

The default print method of a `GenomicRatioSet` object shows basic information of that object. In this example things were kept simple in order to show the bare necessities.

```
> #grs
```

library(minfiData)

[1] The `DataFrame` class is part of the S4Vectors package on Bioconductor, which is loaded by `minfi`

3.2 Using the example data from `minfi`

Here we will show how to use the FuntooNorm functions on the `RGsetEx` example data of the `minfi` package.

First, load the `minfiData` package that contains the example data set.

```
> library(minfiData)
```

The `IlluminaHumanMethylation450kmanifest` object contains information on the array, which we need to extract various types of information from.

```
> pData(RGsetEx)$cell_type <- rep(c("type1","type2"),3)
```

Information on the type of control probes on the array can be found like this:

```
> mySampleSet <- fromRGChannelSet(RGsetEx)
```

Next, let's load the `RGsetEx` data set and have a look at the summary of its contents:

```
> mySampleSet
```

```
SampleSet object built from  minfi
Data:  485577 positionsand  6 samples
   cell type:
   528 quantiles
funtooNorm Normalization was not applied
```

References

[1] Fortin, J.-P., et al. (2014). Functional normalization of 450K methylation array data improves replication in large cancer studies. *Genome Biology*, 15: p. 503.

[2] Aryee, M.J., et al. (2014). Minfi: a flexible and comprehensive Bioconductor package for the analysis of Infinium DNA methylation microarrays. *Bioinformatics*, 30(10): p. 1363-9.

[3] Smith M., et al. (2013). illuminaio: An open source IDAT parsing tool for Illumina microarrays. *F1000Research*, 2:264, 2013.

[4] Kathleen Oros Klein et al. (2015). *funtooNorm:* An improvement of the funNorm normalization method for methylation data from multiple cell or tissue types. Manuscript submitted.