Using funtooNorm

Celia Greenwood, Stepan Grinek, Kathleen Oros Klein

January 4, 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 [1, 2]. 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

The user needs to provide the following information:

- `sigA, sigB:` Two matrices containing the signal A and signal B information from the IDAT files. These matrices should have columns representing each sample, and rows for each probe. The probe names should index the rows, and the sample names should index the columns. These can be obtained from the Illumina IDAT files using Genome Studio (http://www.illumina.com/applications/microarrays/microarray-software/genomestudio.html) by extracting Signal_A and Signal_B from the Sample Methylation Profile, or these can be obtained directly from the IDAT files with Bioconductor packages such as `illuminaio` and `IlluminaDataTestFiles`, [3]. `illuminaio` provides the function readIDAT to extract signals from the IDAT files. Control probe signals can be extracted using probe summaries provided in file 4343238080_A_ProbeSummary.txt.gz of `IlluminaDataTestFiles`.

- `controlred, controlgrn:` Two matrices containing the control probe signals, separately for the red and green channels. The column labels (sample names) for each matrix should match the column labels of the signal A and signal B matrices. This information can also be extracted from the IDAT files using Genome Studio, by extracting Signal_Grn and Signal_Red from the Control Probe Profile or extracted directly from the IDAT files as described above.

- `cell_type:` A list of the cell types or tissues, in the order corresponding to the columns of the signal A and signal B matrices. There must be at least two different cell or tissue types in the data or the program will terminate with a warning.

The program also requires the following information, but default matrices are provided and used if the parameters are set to NULL in the function call.

- `Annot:` An annotation table, containing information on probe names, probe type, and color (for probes of type I). This can be extracted from the Illumina annotation information for the Infinium BeadChip. A default annotation table is provided if not supplied including all probes on the 450K array.

- `cp.types:` A list of the types of control probes to be used in the normalization. This may be obtainable from the row names of the control probe matrices (controlred, controlgrn).

Finally, a number of parameters control whether intermediate calculations should be stored, simply so that the analysis can be performed in stages if desired.

We have provided a small data set containing $N = 93$ individuals and 20,000 probes to demonstrate the usage of the package. The samples are either from cord blood or foetal placenta tissue.

Here is a basic call to normalize this sample data set:

```
>     library(funtooNorm)
>     ncmp <-4
>     funtoonormout <- funtoonorm(sigA=sigAsample, sigB=sigBsample, Annot=NULL,
+                         controlred=matred, controlgrn=matgrn,
+                         cp.types=NULL, cell_type = cell_type,
+                         ncmp=4, ncv.fold=10, logged.data=FALSE, save.quant=TRUE,
+                         type.fits="PCR", apply.fits=TRUE, validate=FALSE)

Since Annot is NULL using default annotation.
Since cp.types is NULL and rownames of control probe matrices do not contain,
 this information, analysis will use default cp.types.
Checking sanity of the data...
```

```
Assuming data have not been previously log transformed, and applying a log transformation,
Data is ok.

>
```

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.

Choice of the number of components can be facilitated by examining a series of fits with different numbers of components. When `validate=TRUE`, funtooNorm produces two files showing the root mean squared errors from cross-validated fits, for different numbers of components. Results are displayed across the quantiles of the signal distributions, separately for A and B signals, and for type I red, type I green, and type II probes. By default, funtooNorm will perform 10-fold cross-validation, but this can be changed with the parameter `ncv.fold`.

Other parameters include:

`save.quant:` When `TRUE`, the quantiles should be saved. When `FALSE`, saved quantiles from a previous run will be loaded and used.

`apply.fits:` When `TRUE`, the results of the model fitting process should be used - i.e. the original data should be normalized. This parameter can be set fo `FALSE` when exploring the desired number of components.

`logged.data:` If `TRUE`, the `sigA` and `sigB` matrices, as well as the control probe data matrices (controlred, controlgrn) are assumed to have been previously log transformed prior to sending the data into the algorithm. If `logged.data=FALSE`, then these data will be log transformed (log(1+x)) inside the algorithm.

The following call performs cross-validation to assess the performance of the model fitting. Note that here the `ncmp` parameter does not need to be specified.

```
>     #This call  will perform cross validation to find optimal value
>     #of parameter ncmp for PLS regression:
>     funtoonormmout <- funtoonorm(sigA=sigAsample, sigB=sigBsample,
+                       controlred=matred, controlgrn=matgrn, cp.type = cp.types, cell_type = cell_type,
+                       ncv.fold=10, logged.data=FALSE, save.quant=TRUE,
+                       type.fits="PCR", apply.fits=FALSE, validate = TRUE)

Since Annot is NULL using default annotation.
Checking sanity of the data...
Assuming data have not been previously log transformed, and applying a log transformation,
Data is ok.
```

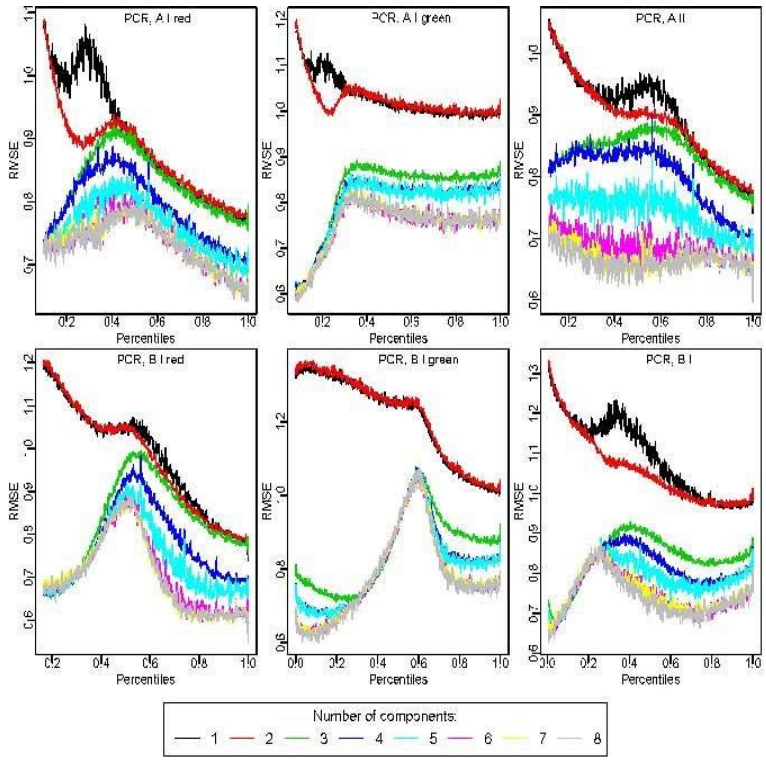

Figure 1: Cross-validated root mean squared errors across percentiles of the signal distributions for different numbers of PCR components. Top: signal A; Bottom: signal B; Left: probe type I red; Middle: probe type I green; Right: probe type II.

```
 Starting validation with  8  components...
Done. Check your working directory for the file "validationcurves.PCR.pdf" and validationcurves.PLS.pdf"
>
>
```

Calling the funtoonorm function with `validate = TRUE` produces two files, corresponding to PCR and PLS regressions, where each file contains a set of plots, one for each type of probe and colour (probe type I red, signal A, ... probe type II, signal B). By looking at figures 1 and 2 the goal is to choose the smallest value of `ncmp` where the cross-validated root mean squared error is fairly small across the quantiles. We have set 4 as the default value for `ncmp`.

After choosing a desired number of components, in order to run the program to normalize the data,

```
>     #This call  will normalize the data, using parameter ncmp for the number of components
>     # and PLR regresion
>     funtoonormmout <- funtoonorm(sigA=sigAsample, sigB=sigBsample,
+                     controlred=matred, controlgrn=matgrn, cp.type = cp.types, cell_type = cell_type,
```

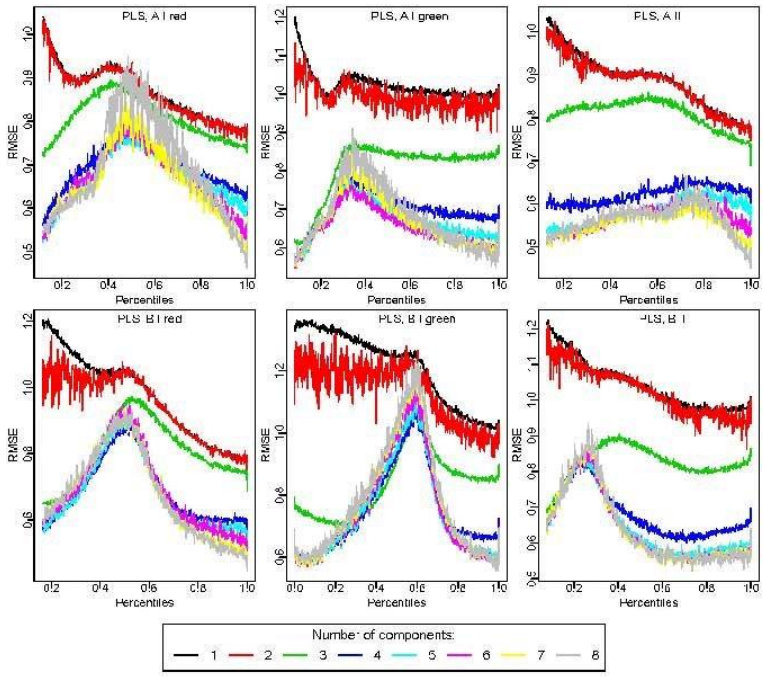

Figure 2: Cross-validated root mean squared errors across percentiles of the signal distributions for different numbers of PLS components. Top: signal A; Bottom: signal B; Left: probe type I red; Middle: probe type I green; Right: probe type II.

```
+                       logged.data=FALSE, save.quant=FALSE, ncmp=4,
+                       type.fits="PLS", apply.fits=TRUE, validate = FALSE)
Since Annot is NULL using default annotation.
Checking sanity of the data...
Assuming data have not been previously log transformed, and applying a log transformation,
Data is ok.
>
```

To access the performance of normalization function one can use a measure of intra-replicate differences `M`, described in [4]. 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(funtoonormout$origBeta, individualID) # M for data before the normalization
[1] 3.335332
> agreement(funtoonormout$newBeta, individualID)  # M for data after normalization
[1] 3.33375
```

3 FuntooNorm and the minfi package

The `minfi` package [2] 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 2) 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

class: GenomicRatioSet
dim: 20000 93
exptData(0):
```

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

```
assays(1): Beta
rownames: NULL
rowRanges metadata column names(0):
colnames(93): 5975819019_R01C01 5975819019_R01C02 ... 6229050136_R06C01
  6229050136_R06C02
colData names(2): Sample_Name sex
Annotation
  array:
Preprocessing
  Method: NA
  minfi version: NA
  Manifest version: NA
```

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.

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

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

```
> ctrlInfo <- getProbeInfo(manifest, type="Control")
> unique(ctrlInfo$Type)
```

```
 [1] "STAINING"                "EXTENSION"
 [3] "HYBRIDIZATION"           "TARGET REMOVAL"
 [5] "BISULFITE CONVERSION I"  "BISULFITE CONVERSION II"
 [7] "SPECIFICITY I"           "SPECIFICITY II"
 [9] "NON-POLYMORPHIC"         "NEGATIVE"
[11] "RESTORATION"             "NORM_A"
[13] "NORM_G"                  "NORM_C"
[15] "NORM_T"
```

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

```
> data(RGsetEx)
> RGsetEx

RGChannelSet (storageMode: lockedEnvironment)
assayData: 622399 features, 6 samples
  element names: Green, Red
An object of class 'AnnotatedDataFrame'
  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
```

The matrices with red and green signals need to be extracted first.

```
> matRed <- getRed(RGsetEx)
> matGrn <- getGreen(RGsetEx)
```

The following code extracts the control probe data from the red and green matrices.

```
> ctrlRed <- matRed[rownames(matRed) %in% ctrlInfo$Address,]
> ctrlGrn <- matGrn[rownames(matGrn) %in% ctrlInfo$Address,]
```

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 1two different colors provide the methylated and the unmethylated measurements.

Therefore, the procedure to extract the A (methylated) and B (unmethylated) signals from the `matRed` and `matGrn` matrices is slightly different for the Type I and Type II probes. The following code first extracts the information on the two probe Types from the manifest and then creates the signal A matrix for Type I probes. Note that `funtooNorm` requires the row names of the signal matrices to be the probe names whereas the `matRed` and `matGrn` matrices have the position as rownames.

```
> typeI_info <- getProbeInfo(manifest, type="I")
> typeII_info <- getProbeInfo(manifest, type="II")
```

```
> typeIARed <- typeI_info[typeI_info$Color=="Red",]
> typeIAGrn <- typeI_info[typeI_info$Color=="Grn",]
> sigAIRed <- matRed[rownames(matRed) %in% typeIARed$AddressA,]
> rownames(sigAIRed) <- typeIARed$Name
> sigAIGrn <- matGrn[rownames(matGrn) %in% typeIAGrn$AddressA,]
> rownames(sigAIGrn) <- typeIAGrn$Name
> sigAI <- rbind(sigAIRed, sigAIGrn)
```

The extraction of the signal A matrix for Type II probes is much simpler:

```
> sigAII <- matRed[rownames(matRed) %in% typeII_info$AddressA,]
> rownames(sigAII) <- typeII_info$Name
```

Now these the Type I and Type II signal A matrices can be combined:

```
> sigA <- rbind(sigAI, sigAII)
```

The following code block repeats the same steps for the unmethylated B signal matrix:

```
> typeIBRed <- typeI_info[typeI_info$Color=="Red",]
> typeIBGrn <- typeI_info[typeI_info$Color=="Grn",]
> sigBIRed <- matRed[rownames(matRed) %in% typeIBRed$AddressB,]
> rownames(sigBIRed) <- typeIBRed$Name
> sigBIGrn <- matGrn[rownames(matGrn) %in% typeIBGrn$AddressB,]
> rownames(sigBIGrn) <- typeIBGrn$Name
> sigBI <- rbind(sigBIRed, sigBIGrn)
> sigBII <- matGrn[rownames(matGrn) %in% typeII_info$AddressA,]
> rownames(sigBII) <- typeII_info$Name
> sigB <- rbind(sigBI, sigBII)
```

`funtooNorm` requires a vector with the cell type of each of the 6 amples in the RGsetEx data set.

```
> cellType <- as.factor(c(rep("cellTypeA", 2), rep("cellTypeB", 4)))
```

For the `Annot` option of `funtooNorm` we set up the annotations selecting only the probes that are in the example data set and then order those according to the order in the signal matrices.

```
> data(Annot)
> annot <- Annot[rownames(Annot) %in% rownames(sigA),]
> annot <- annot[match(rownames(sigA), rownames(annot)),]
```

After these conversion steps, we can run `funtooNorm()` on the `RgsetEx` data:

```
> ftRGsetExOut <- funtoonorm(sigA=sigA, sigB=sigB, controlred=ctrlRed,
+                            controlgrn=ctrlGrn, cell_type=cellType,
+                            Annot=annot)

Since cp.types is NULL and rownames of control probe matrices do not contain,
 this information, analysis will use default cp.types.
Checking sanity of the data...
Assuming data have not been previously log transformed, and applying a log transformation,
Data is ok.
```

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.