The *bigmelon* Package

Tyler Gorrie-Stone, Ayden Saffari, Karim Malki and Leonard C Schalkwyk

October 17, 2016

1 About

The *bigmelon* package for Illumina methylation data provides a fast and convenient way to apply a variety of different normalisation methods to your data, such as those previously described by Pidsley et al. [1] and implemented in the package *wateRmelon*. Bigmelon extends the capability of *wateRmelon* to higher dimensional data, allowing much larger data sets containing many more arrays to be processed, while also providing a convenient way to store this data for future access. This has been achieved by adapting methods from the *gdsfmt* package, originally designed for handling SNP data, which through efficient memory use and management is able to overcome some of the memory overheads associated with handling big data in *R*.

2 Installation

bigmelon works with existing *Bioconductor* packages and therefore has a number of dependencies. The install.packages() should install the required packages automatically, but should this not succeed for any reason, the following commands can be used to install these manually:

```
> source('http://bioconductor.org/biocLite.R')
> biocLite('wateRmelon', 'gdsfmt')
```

Install the latest package from a local copy (located in the current working directory of your R session):

```
> install.packages('bigmelon_0.99.11.tar.gz', repos = NULL, type = 'source')
```

3 Using Bigmelon

```
> library(bigmelon)
```

3.1 Loading Data

There are multiple methods that can be used to load in data into a gds object. These can either be from GenomeStudio final report text files or from raw binary (.IDAT) files.

3.1.1 IDAT Files

IDAT files are the raw intensities obtained from DNA methylation microarrays and are split into two files per sample (one Red Channel and one Green Channel). These are typically read into R using *minfi* or *methylumi*. In *bigmelon* IDAT files can be read in using the `iadd` or `iadd2` functions. These functions pass to `methylumIDATepic` (*wateRmelon*) to read in the data - although it should be noted that the full annotation of features are not included when reading from idats.

`iadd` and `iadd2` function differently. `iadd` will take a vector of barcodes, `iadd2` will accept a directory pathway and extract all IDAT files within the specified path.

`iadd2` also has the functionality to read IDAT files in chunks. This is useful if you are attempting to read in a lot of data at once and do not have sufficient memory on your workstation to support this. **This method is recommended if you are using a workstation bounded by memory limitations.**

```
> # read in an IDAT file with barcode 'sentrixid_rnncnn'
> gfile <- iadd('sentrixid_rnncnn', gds = 'melon.gds')
> gfile <- iadd2('Data/IDATLocations/dataset', gds = 'melon.gds', chunksize = 100)
```

3.1.2 ExpressionSet Objects

You may have been given a *MethylumiSet*, *RGChannelSet* or *MethylSet* instead of idats. These can be passed to `es2gds` to convert the data into a *gds.class*

Henceforth we will convert the ExpressionSet data object 'melon' packaged within *wateRmelon* to demonstrate further down stream analysis.

```
> data(melon)
> gfile <- es2gds(melon, 'melon.gds')
```

3.1.3 Text Files

To read in text files, the `methyLumiR` function from *methylumi* can be used. If using this method, we suggest saving the unnormalised, uncorrected version of the data. We also recommend keeping the barcode names (SentrixID_RnnCnn) as the column headers or in a separate dataframe. Alternatively you can use the function `finalreport2gds` that will output a .gds object.

```
> library(methylumi)
> # read Illumina methylation data into a MethyLumiSet object
> melon <- methyLumiR('finalreport.txt')
> # read Illumina methylation final report into a gds.class object.
> gfile <- finalreport2gds('finalreport.txt', gds='melon.gds')
```

Assuming you have used `methyLumiR` you would then need to convert the resultant object to a Genomic Data Structure (GDS) data file. This can also be achieved using the function `es2gds` which can convert *MethyLumiSet* objects (from *methylumi*, *RGChannelSet* and *MethylSet* objects (from *minfi*) as described above.

```
> # convert a MethyLumiSet object to a gds.class object
> gfile <- es2gds(melon, 'melon.gds')
```

3.2 Opening and Closing gds files

Now that you have created a .gds file you can continue working on it within the same R session. Or close the file for later use, or to share with others.

The functions `openfn.gds` and `closefn.gds` are used.

```
> # Closing File
> closefn.gds(gfile)
> # Opening File
> gfile <- openfn.gds('melon.gds')
```

Recommended: See `?openfn.gds`

3.3 Exploring the gds.class

The resulting *gds.class* may be different to any other data-structure you have previously used. Simply, it resembles an S4 object but instead of slots there are nodes with a *gdsn.class* class. To access these, specialized functions need to be used as common R functionality (@ and $) are not yet existent for these objects.

When printing the *gds.object* we are given an almost directory-like output.

```
> print(gfile)
```

```
File: /tmp/RtmpyZSwVy/Rbuild32965b3c56c/bigmelon/vignettes/melon.gds (1.4M)
+    [  ]
|--+ description        *
|--+ betas   { Float64 3363x12, 315.3K }
|--+ pvals   { Float64 3363x12, 315.3K }
|--+ methylated   { Int32 3363x12, 157.6K }
|--+ unmethylated   { Int32 3363x12, 157.6K }
|--+ fData   [ data.frame ] *
|  |--+ TargetID   { Str8 3363, 39.8K }
|  |--+ ProbeID_A   { Str8 3363, 29.6K }
|  |--+ ProbeID_B   { Str8 3363, 29.6K }
|  |--+ ILMNID   { Str8 3363, 39.8K }
|  |--+ NAME   { Str8 3363, 39.8K }
|  |--+ ADDRESSA_ID   { Str8 3363, 29.6K }
|  |--+ ALLELEA_PROBESEQ   { Str8 3363, 167.5K }
|  |--+ ADDRESSB_ID   { Str8 3363, 8.1K }
|  |--+ ALLELEB_PROBESEQ   { Str8 3363, 33.3K }
|  |--+ INFINIUM_DESIGN_TYPE   { Str8 3363, 9.3K }
|  |--+ NEXT_BASE   { Str8 3363, 3.9K }
|  |--+ COLOR_CHANNEL   { Str8 3363, 5.1K }
|  |--+ FORWARD_SEQUENCE   { Str8 3363, 402.7K }
|  |--+ GENOME_BUILD   { Str8 3363, 9.7K }
```

```
|  |--+ CHR   { Str8 3363, 7.5K }
|  |--+ MAPINFO   { Str8 3363, 29.7K }
|  |--+ SOURCESEQ   { Str8 3363, 164.3K }
|  |--+ CHROMOSOME_36   { Str8 3363, 7.5K }
|  |--+ COORDINATE_36   { Str8 3363, 29.7K }
|  |--+ STRAND   { Str8 3363, 6.5K }
|  |--+ PROBE_SNPS   { Str8 3363, 8.3K }
|  |--+ PROBE_SNPS_10   { Str8 3363, 4.8K }
|  |--+ RANDOM_LOCI   { Str8 3363, 3.3K }
|  |--+ METHYL27_LOCI   { Str8 3363, 3.8K }
|  |--+ UCSC_REFGENE_NAME   { Str8 3363, 27.8K }
|  |--+ UCSC_REFGENE_ACCESSION   { Str8 3363, 46.5K }
|  |--+ UCSC_REFGENE_GROUP   { Str8 3363, 27.0K }
|  |--+ UCSC_CPG_ISLANDS_NAME   { Str8 3363, 38.1K }
|  |--+ RELATION_TO_UCSC_CPG_ISLAND   { Str8 3363, 13.3K }
|  |--+ PHANTOM   { Str8 3363, 6.9K }
|  |--+ DMR   { Str8 3363, 3.8K }
|  |--+ ENHANCER   { Str8 3363, 5.0K }
|  |--+ HMM_ISLAND   { Str8 3363, 25.8K }
|  |--+ REGULATORY_FEATURE_NAME   { Str8 3363, 18.6K }
|  |--+ REGULATORY_FEATURE_GROUP   { Str8 3363, 20.1K }
|  |--+ DHS   { Str8 3363, 4.4K }
|  |--+ Index   { Str8 3363, 18.0K }
|  \--+ X38   { Str8 3363, 3.3K }
|--+ pData   [ data.frame ] *
|  |--+ sampleID   { Str8 12, 216B }
|  |--+ label   { Str8 12, 216B }
|  \--+ sex   { Str8 12, 24B }
|--+ QCmethylated   { Int32 835x12, 39.1K }
|--+ QCunmethylated   { Int32 835x12, 39.1K }
|--+ QCrownames   { Str8 835, 10.1K }
|--+ history   [ data.frame ] *
|  |--+ submitted   { Str8 4, 80B }
|  |--+ finished   { Str8 4, 80B }
|  \--+ command   { Str8 4, 118B }
\--+ paths   { Str8 2, 30B }
```

From this output we can see some useful information about our object such as the file name, total object size and the name, size and type of each node within the gds object.

Typically a *bigmelon* gds file is comprised of some common nodes these being: betas, methylated, unmethylated, pvals, fData, pData, and History. If you are familiar with the `MethyLumiSet` this will be immediately familiar to you. If not a brief description is as follows

- betas: The ratio between Methylated and Unmethylated intensities - most commonly used for analysis
- methylated: The methylated intensities
- unmethylated: The unmethylated intensities
- pvals: The detection P values of the array
- NBeads(not shown): The total beadcount (per feature) on the array.

- fData: The feature data, which contains all relavent biological information to CpG probes within the micro-array (rows).
- pData: The pheno data, which contains information relevant to biological samples (columns).
- history: Brief description of operations applied to the file.

To access the data represented in the object we need to use the function `index.gdsn`

```
> index.gdsn(gfile, 'betas')
```

```
+ betas   { Float64 3363x12, 315.3K }
```

```
> class(index.gdsn(gfile, 'betas'))
```

```
[1] "gdsn.class"
```

```
> # Access nodes with additional nodes inside
> index.gdsn(gfile, 'fData/TargetID')
```

```
+ fData/TargetID   { Str8 3363, 39.8K }
```

Alternatively, there are some accessors written for the common object names see `?'bigmelon-accessors'`. Majority of these accessors will pass to `index.gdsn` but if the object is small enough the accessor may read the object into memory without further indexing.

```
> betas(gfile)
```

```
+ betas   { Float64 3363x12, 315.3K }
```

```
> class(betas(gfile))
```

```
[1] "gdsn.class"
```

If the directory-tree output is hard to interpret or you wish to list all available nodes the function `ls.gdsn` allows you to view the contents of a gds file in a vector.

```
> ls.gdsn(gfile)
```

```
 [1] "description"    "betas"          "pvals"          "methylated"     "unmethylated"
 [6] "fData"          "pData"          "QCmethylated"   "QCunmethylated" "QCrownames"
[11] "history"        "paths"
```

```
> # Look into nodes with additional nodes
> ls.gdsn(index.gdsn(gfile, 'fData'))
```

```
 [1] "TargetID"                  "ProbeID_A"
 [3] "ProbeID_B"                 "ILMNID"
 [5] "NAME"                      "ADDRESSA_ID"
 [7] "ALLELEA_PROBESEQ"          "ADDRESSB_ID"
 [9] "ALLELEB_PROBESEQ"          "INFINIUM_DESIGN_TYPE"
[11] "NEXT_BASE"                 "COLOR_CHANNEL"
[13] "FORWARD_SEQUENCE"          "GENOME_BUILD"
[15] "CHR"                       "MAPINFO"
[17] "SOURCESEQ"                 "CHROMOSOME_36"
[19] "COORDINATE_36"             "STRAND"
[21] "PROBE_SNPS"                "PROBE_SNPS_10"
[23] "RANDOM_LOCI"               "METHYL27_LOCI"
```

```
[25] "UCSC_REFGENE_NAME"            "UCSC_REFGENE_ACCESSION"
[27] "UCSC_REFGENE_GROUP"           "UCSC_CPG_ISLANDS_NAME"
[29] "RELATION_TO_UCSC_CPG_ISLAND" "PHANTOM"
[31] "DMR"                          "ENHANCER"
[33] "HMM_ISLAND"                   "REGULATORY_FEATURE_NAME"
[35] "REGULATORY_FEATURE_GROUP"     "DHS"
[37] "Index"                        "X38"
```

3.4 Exploring the gdsn.class

You may ask the question - 'How do I access **that** juicy data?'. To do this, the functions `read.gdsn` and `readex.gdsn` are used. `read.gdsn` will load the entire object represented in a gdsn.class object into memory. While `readex.gdsn` allows you to specify a subset to load into memory.

```
> # Call a gdsn.class node
> anode <- betas(gfile)
> anode
```

```
+ betas   { Float64 3363x12, 315.3K }
```

```
> class(anode)
```

```
[1] "gdsn.class"
```

```
> # All data
> dat <- read.gdsn(anode)
> dim(dat)
```

```
[1] 3363   12
```

```
> head(dat)
```

```
        [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]    [,9]   [,10]
[1,] 0.67233 0.71083 0.67504 0.69099 0.75096 0.70100 0.69015 0.70555 0.73616 0.69577
[2,] 0.57508 0.37251 0.65755 0.48510 0.60844 0.57751 0.55881 0.58700 0.56206 0.51776
[3,] 0.75909 0.77251 0.78121 0.78238 0.83259 0.77071 0.67976 0.81480 0.79522 0.77952
[4,] 0.39772 0.52589 0.30660 0.43188 0.36626 0.51246 0.43329 0.29387 0.50085 0.49857
[5,] 0.70793 0.75873 0.75429 0.76494 0.77341 0.76085 0.75011 0.70859 0.76726 0.71562
[6,] 0.43090 0.44909 0.51544 0.50341 0.48614 0.48561 0.46985 0.40216 0.44669 0.51309
       [,11]   [,12]
[1,] 0.71457 0.71944
[2,] 0.50770 0.54396
[3,] 0.70430 0.71377
[4,] 0.48607 0.36159
[5,] 0.70947 0.75575
[6,] 0.55050 0.54047
```

```
> # Subset!
> datsub <- readex.gdsn(anode, sel = list(1:5, 1:3))
> dim(datsub)
```

```
[1] 5 3
```

```
> datsub
```

```
        [,1]    [,2]    [,3]
[1,] 0.67233 0.71083 0.67504
[2,] 0.57508 0.37251 0.65755
[3,] 0.75909 0.77251 0.78121
[4,] 0.39772 0.52589 0.30660
[5,] 0.70793 0.75873 0.75429
```

You may immediately notice that the rownames and column names of the matrix are missing. This is an unfortunate side-effect of using `read.gdsn` because such information is not stored within the specified gds node. However within *bigmelon* we have written a wrapper-function for `read.gdsn` (and `readex.gdsn`) to load data into R. This is achieved using `'['`. The purpose of this is to enable similar indexing operations that most will be familiar with.

```
> # Re-using node from previous example
> anode
```

```
+ betas   { Float64 3363x12, 315.3K }
```

```
> datsub <- anode[1:5,1:3]
> dim(datsub)
```

```
[1] 5 3
```

```
> datsub
```

```
           6057825008_R01C01 6057825008_R01C02 6057825008_R02C01
cg00000029           0.67233           0.71083           0.67504
cg00000108           0.57508           0.37251           0.65755
cg00000109           0.75909           0.77251           0.78121
cg00000165           0.39772           0.52589           0.30660
cg00000236           0.70793           0.75873           0.75429
```

```
> # Additionally, the row and col names can be turned off
> anode[1:5, 1:3, name = FALSE]
```

```
        [,1]    [,2]    [,3]
[1,] 0.67233 0.71083 0.67504
[2,] 0.57508 0.37251 0.65755
[3,] 0.75909 0.77251 0.78121
[4,] 0.39772 0.52589 0.30660
[5,] 0.70793 0.75873 0.75429
```

There are a few more tricks that are possible in *bigmelon* that we will briefly explore here.

```
> # Logical Indexing
> anode[1:5,c(TRUE,FALSE,FALSE)]
```

```
           6057825008_R01C01 6057825008_R02C02 6057825008_R04C01 6057825008_R05C02
cg00000029           0.67233           0.69099           0.69015           0.69577
cg00000108           0.57508           0.48510           0.55881           0.51776
cg00000109           0.75909           0.78238           0.67976           0.77952
```

```
cg00000165           0.39772           0.43188           0.43329           0.49857
cg00000236           0.70793           0.76494           0.75011           0.71562
```

```
> # Ordering calls
> anode[c(5,9,1,500,345), c(8,4,1,3)]
```

```
           6057825008_R04C02 6057825008_R02C02 6057825008_R01C01 6057825008_R02C01
cg00000236           0.70859           0.76494           0.70793           0.75429
cg00000363           0.10832           0.12452           0.11314           0.12508
cg00000029           0.70555           0.69099           0.67233           0.67504
cg00020649           0.03407           0.03478           0.02913           0.03470
cg00014272           0.07078           0.07480           0.06852           0.06607
```

```
> # Indexing by characters (and drop functionality)
> anode[c('cg00000029', 'cg00000236'), '6057825008_R02C01', drop = FALSE]
```

```
           6057825008_R02C01
cg00000029           0.67504
cg00000236           0.75429
```

```
> # Loading entire data (no indexing)
> dat <- anode[ , ]
> dim(dat)
```

```
[1] 3363   12
```

Additionally it is possible to call a gds node from a gds.class within the '[' indexing.

```
> gfile[1:5, 1:3, node = 'betas', name = TRUE]
```

```
           6057825008_R01C01 6057825008_R01C02 6057825008_R02C01
cg00000029           0.67233           0.71083           0.67504
cg00000108           0.57508           0.37251           0.65755
cg00000109           0.75909           0.77251           0.78121
cg00000165           0.39772           0.52589           0.30660
cg00000236           0.70793           0.75873           0.75429
```

```
> gfile[1:5, 1:3, node = 'methylated', name = TRUE]
```

```
           6057825008_R01C01 6057825008_R01C02 6057825008_R02C01
cg00000029              2926              2940              2715
cg00000108              4301              2060              5075
cg00000109              1503              1630              1821
cg00000165               943              1381               910
cg00000236              2322              2607              2898
```

As a brief side note, the row and column names are still stored within the gds data file. Located at the bottom of each gds data file will be a node labelled as "paths". This contains a string to where the row and column names are stored. These are determined by default upon the creation of the gds data file but in events where they are incorrect they can be corrected with the `redirect.gds`.

```
> read.gdsn(index.gdsn(gfile, "paths"))
```

```
[1] "fData/TargetID" "pData/sampleID"
```

```
> head(read.gdsn(index.gdsn(gfile, "fData/TargetID")))
[1] "cg00000029" "cg00000108" "cg00000109" "cg00000165" "cg00000236" "cg00000289"
> head(read.gdsn(index.gdsn(gfile, "pData/sampleID")))
[1] "6057825008_R01C01" "6057825008_R01C02" "6057825008_R02C01" "6057825008_R02C02"
[5] "6057825008_R03C01" "6057825008_R03C02"
```

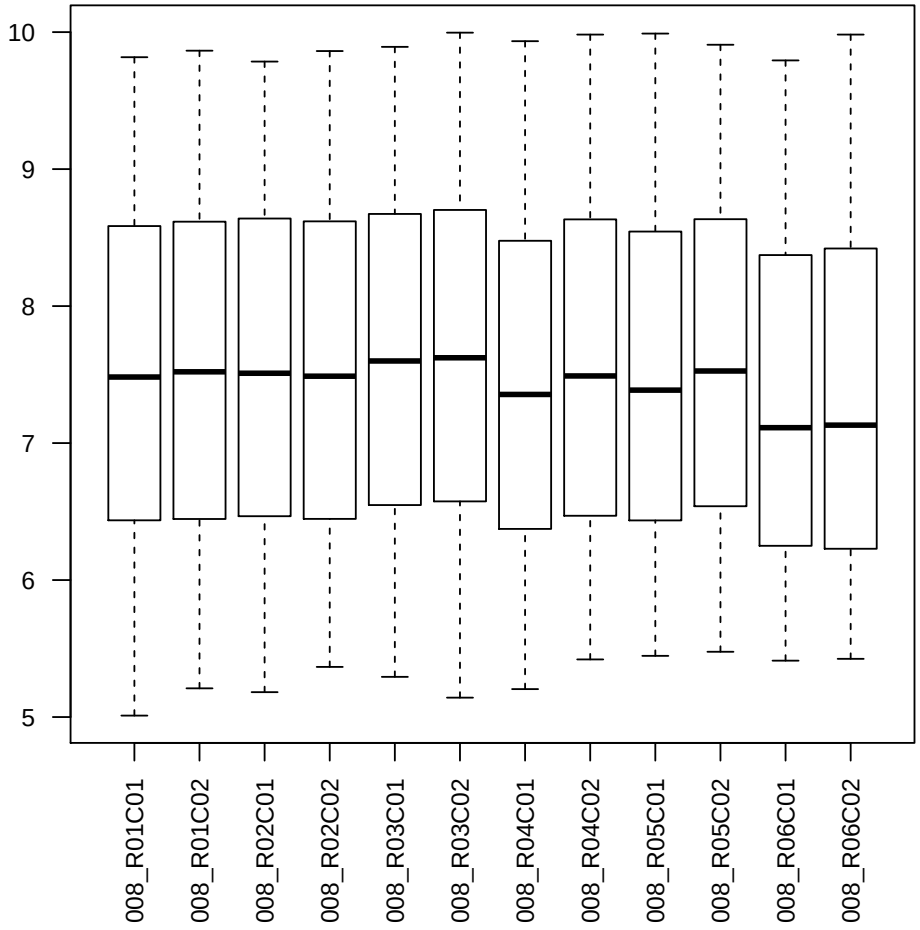
4 Preprocessing

4.1 Quality Control

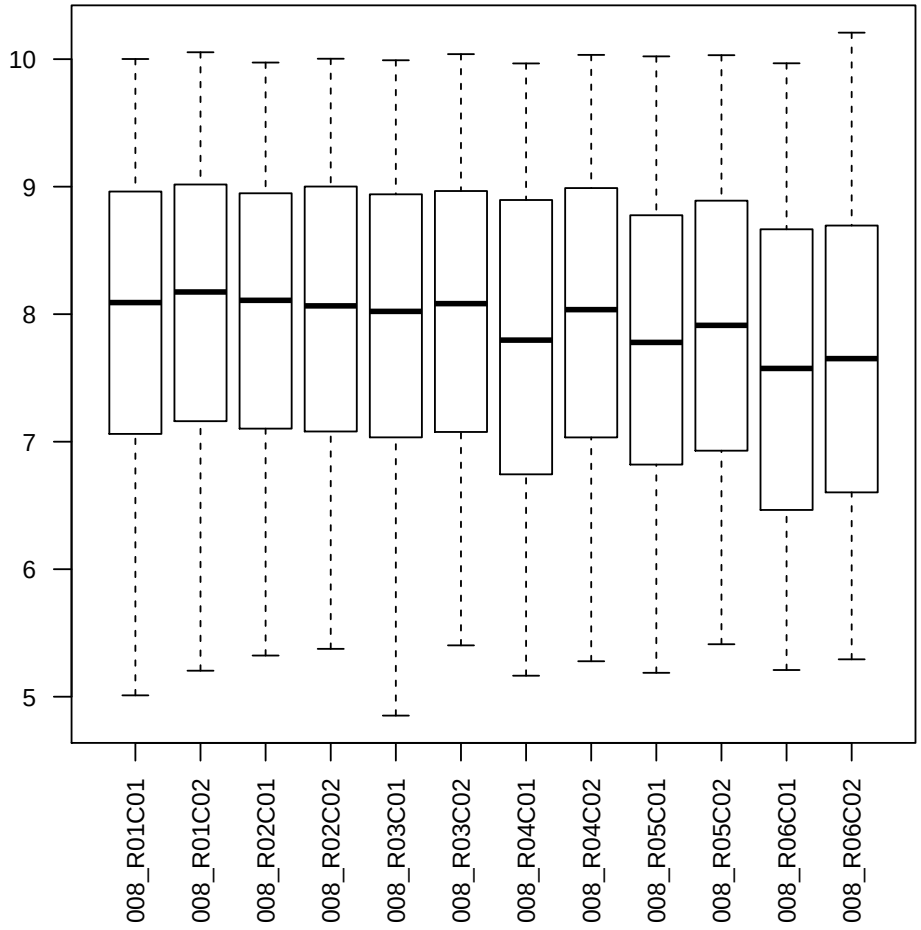
Prior to data analysis, you may find it is necessary to perform some quality control and normalization. Within bigmelon, we have some functions can assist with the QC but you can use whatever functions you like.

Typical workflows involve visualizing raw intensities:

```
> rawmet <- methylated(gfile)[,]
> rawume <- unmethylated(gfile)[,]
> boxplot(log(rawmet), las=2, cex.axis=0.8)
```




```
> boxplot(log(rawume), las=2, cex.axis=0.8)
```

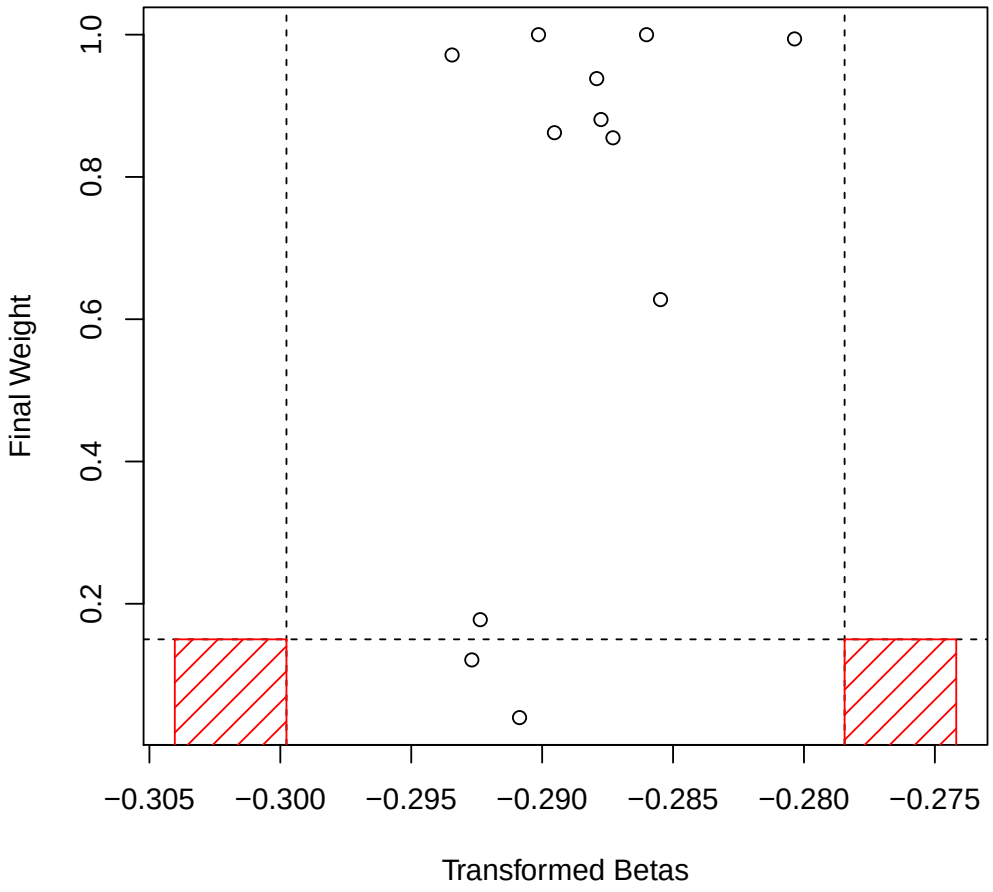


Alternatively it is possible to used some highly specialized functions available within *bigmelon*.

```
> rawbet <- betas(gfile)[,]
```

```
> outlyx(rawbet, plot = TRUE)
```

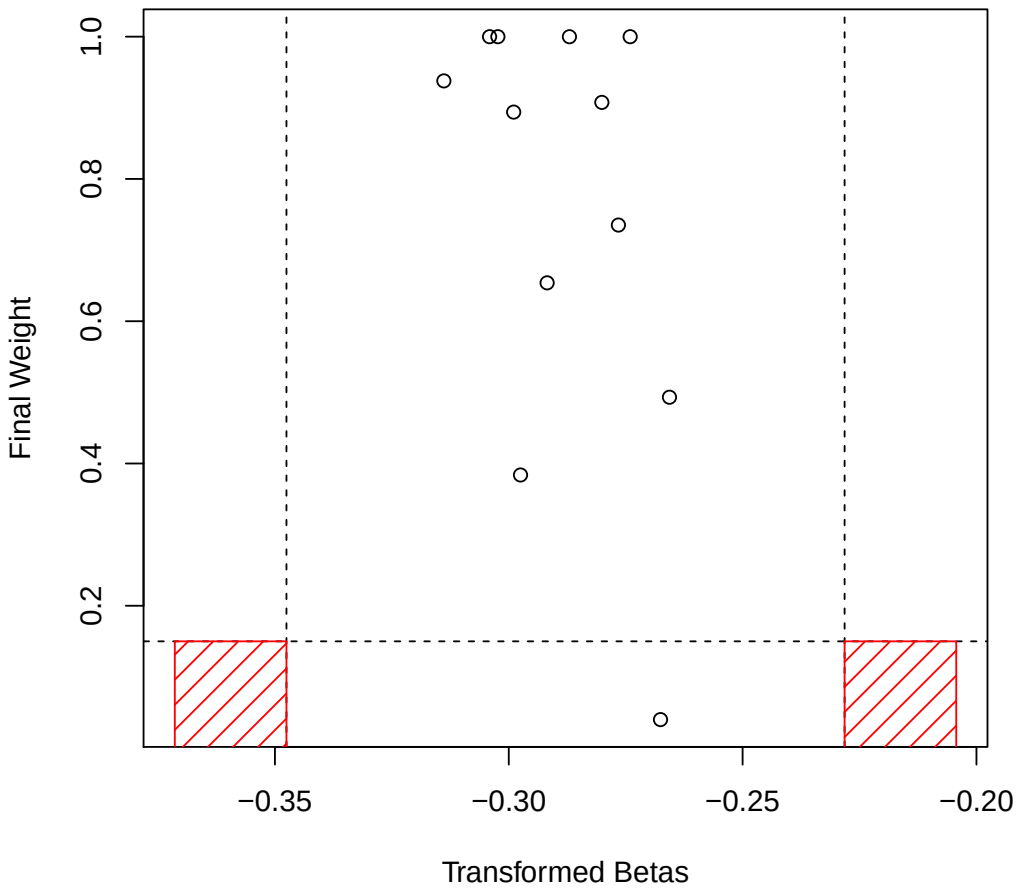
```
                    iqr    mv outliers
6057825008_R01C01 FALSE FALSE    FALSE
6057825008_R01C02 FALSE FALSE    FALSE
6057825008_R02C01 FALSE  TRUE    FALSE
6057825008_R02C02 FALSE FALSE    FALSE
6057825008_R03C01 FALSE FALSE    FALSE
6057825008_R03C02 FALSE FALSE    FALSE
6057825008_R04C01 FALSE FALSE    FALSE
6057825008_R04C02 FALSE  TRUE    FALSE
6057825008_R05C01 FALSE FALSE    FALSE
6057825008_R05C02 FALSE FALSE    FALSE
6057825008_R06C01 FALSE FALSE    FALSE
6057825008_R06C02 FALSE FALSE    FALSE
```



If the data is too large to load into memory, one can use the *bigmelon* method which determines outliers with a small subset of data.

```
> outlyx(gfile, plot = TRUE, perc = 0.01)
```

```
                    iqr    mv outliers
6057825008_R01C01 FALSE FALSE    FALSE
6057825008_R01C02 FALSE FALSE    FALSE
6057825008_R02C01 FALSE FALSE    FALSE
6057825008_R02C02 FALSE FALSE    FALSE
6057825008_R03C01 FALSE FALSE    FALSE
6057825008_R03C02 FALSE  TRUE    FALSE
6057825008_R04C01 FALSE FALSE    FALSE
6057825008_R04C02 FALSE FALSE    FALSE
6057825008_R05C01 FALSE FALSE    FALSE
6057825008_R05C02 FALSE FALSE    FALSE
6057825008_R06C01 FALSE FALSE    FALSE
6057825008_R06C02 FALSE FALSE    FALSE
```



Filtering probes/features by detection p-values also provides another straightforward approach for removing both failed samples and probes. The `pfilter` function discards samples with more than 1 % of probes above .05 detection p-value threshold, and probes with any samples with beadcount under 3 or more than 1% above the p-value threshold.

n.b. This will perform irreversible subsetting procedures onto the gds file and will **not** work if the gds gfile is in read mode.

```
> pfilter(gfile)
```

```
NBeads missing, using betas instead...
0 samples having 1 % of sites with a detection p-value greater than 0.05 were removed
Samples removed:
72 sites were remove as beadcount <3 in 5 % of samples
40 sites having 1 % of samples with a detection p-value greater than 0.05 were removed
```

It is at this point worth mentioning that we do not need to assign our results from these functions since the data is not stored with memory.

4.2 Backing Up/Storing Raw data

Before performing any function that will noticeably change the data, you may want to create a physical back-up of the gds file so you do not have to retrace your steps incase you lose progress. The `backup.gdsn` function serves as an easy way to copy a node you may be interested in to your gds file incase you need it for later.

```
> backup.gdsn(gds = NULL, node = index.gdsn(gfile, 'betas'))
> ls.gdsn(index.gdsn(gfile, 'backup'))
```

```
[1] "betas"
```

Alternatively you can create a new gdsfile to store the copy or use the `copyto.gdsn`.

```
> f <- createfn.gds('melon2.gds')
> backup.gdsn(gds = f, node = index.gdsn(gfile, 'betas'))
> f
```

```
File: /tmp/RtmpyZSwVy/Rbuild32965b3c56c/bigmelon/vignettes/melon2.gds (305.0K)
+    [  ]
\--+ backup   [  ]
   \--+ betas   { Float64 3252x12, 304.9K }
```

```
> copyto.gdsn(node = f, source = index.gdsn(gfile, 'betas'), name = 'betacopy')
> f
```

```
File: /tmp/RtmpyZSwVy/Rbuild32965b3c56c/bigmelon/vignettes/melon2.gds (610.0K)
+    [  ]
|--+ backup   [  ]
|  \--+ betas   { Float64 3252x12, 304.9K }
\--+ betacopy   { Float64 3252x12, 304.9K }
```

```
> copyto.gdsn(node = gfile, source = index.gdsn(gfile, 'betas'), name='betacopy')
> # Close File
> closefn.gds(f)
```

4.3 Normalization

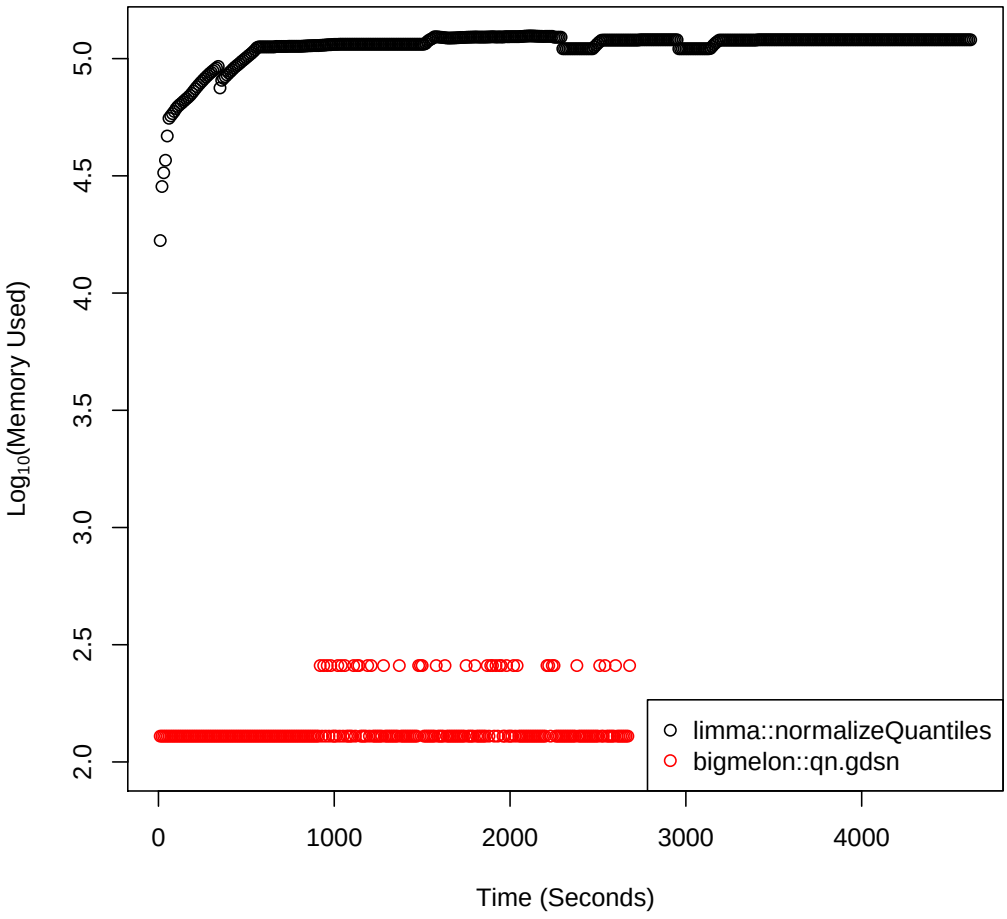
Within *bigmelon* there are numerous normalisation methods that can be used. The method `dasen` will work well for most data sets. **n.b.** This will perform irreversible procedures on the data. And will replace raw intensities with the normalised ones. This will not work if gds file is in read mode.

```
> dasen(gfile)
> # Alternatively it is possible to store normalized betas to a separate node
> # If you want to keep the raw data
> dasen(gfile, node="normbeta")
> index.gdsn(gfile, "normbeta")
```

```
+ normbeta   { Float64 3252x12, 304.9K }
```

Due to how the normalisation process is broken down within *bigmelon* there is only ever a small amount of memory required throughout data analysis. For example when attempting to process 4000 EPIC array samples (>850,000 features), totalling around 28Gb of data. Simple quantile normalisation procedures

quickly use up all available memory to attempt such feat. Whereas within bigmelon, the same analyses uses considerably less memory and (in this circumstance) provide a 1000 fold decrease in memory use.



5 Analysis

While we cannot recommend any advice about how to perform your statistical analysis we will demonstrate how to make the most out of the *bigmelon* package. Within *gdsfmt* there are many functions written that are specialized for gds files. Notably the `apply.gdsn` function is particularly useful as it will perform functions upon specified margins efficiently instead of loading the entire object into R to perform analysis.

```
> # Example of apply.gdsn
> apply.gdsn(betas(gfile), margin = 2, as.is='double', FUN = function(x,y){
+ mean(x, na.rm=y)
+ }, y = TRUE)
```

```
 [1] 0.4291725 0.4293673 0.4280593 0.4287714 0.4295415 0.4297671 0.4273242 0.4281040
 [9] 0.4284095 0.4298338 0.4285670 0.4297393
```

You can define your own functions to supply as an argument to FUN. Please do explore `apply.gdsn` as it is extremely versatile and it can additionally store outputs straight the a gds node if needed.

There will ofcourse be some analyses that may not be amenable to high dimensional data but if analysis can be broken down into column/row wise operations then it is possible.

Currently, all available methods within *wateRmelon* with the exception of `seabi`, `swan`, `tost`, `fuks` and `BMIQ` have been optimised for memory usage.

6 Back-Porting

Should you find it necessary to convert your gds object back into memory (perhaps for some specialised analyses) you can use the functions `gds2mlumi` and `gds2mset` which will build a `MethyLumiSet` object and `MethylSet` object in your enviroment.

```
> gds2mlumi(gfile)
```

```
Object Information:
MethyLumiSet (storageMode: lockedEnvironment)
assayData: 3252 features, 12 samples
  element names: betas, methylated, pvals, unmethylated
protocolData: none
phenoData
  sampleNames: 6057825008_R01C01 6057825008_R01C02 ... 6057825008_R06C02 (12
    total)
  varLabels: sampleID label sex
  varMetadata: labelDescription
featureData
  featureNames: 1 2 ... 3252 (3252 total)
  fvarLabels: TargetID ProbeID_A ... X38 (38 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
Major Operation History:
            submitted            finished
1 2012-10-17 14:23:16 2012-10-17 14:23:20
2 2012-10-17 17:11:19 2012-10-17 17:11:20
3 2012-10-17 17:11:48 2012-10-17 17:11:48
4 2016-10-17 20:28:31 2016-10-17 20:28:31
5 2016-10-17 20:28:32 2016-10-17 20:28:32
6 2016-10-17 20:28:32 2016-10-17 20:28:32
7 2016-10-17 20:28:32 2016-10-17 20:28:34
8 2016-10-17 20:28:34 2016-10-17 20:28:37
9 2016-10-17 20:28:37 2016-10-17 20:28:37
                                        command
1                 methylumiR(filename = "fr2.txt")
2                           Subset of 46 samples.
3                           Subset of 12 samples.
```

```
4         MethylumiSet converted to gds (bigmelon)
5                       pfilter applied (bigmelon)
6               Subset of 3252 rows and 12 samples
7        Normalized with dasen method (wateRmelon)
8        Normalized with dasen method (wateRmelon)
9 Converted to methylumi with gds2mlumi (bigmelon)
```

```
> gds2mset(gfile, anno="450k")
```

```
MethylSet (storageMode: lockedEnvironment)
assayData: 3252 features, 12 samples
  element names: Meth, Unmeth
An object of class 'AnnotatedDataFrame': none
Annotation
  array: IlluminaHumanMethylation450k
  annotation: ilmn12.hg19
Preprocessing
  Method: Converted from gdsfmt to MethylSet (bigmelon)
  minfi version: 1.20.0
  Manifest version: NA
```

7 Finishing an R session

As this workflow is in its infancy there are some issues that have yet to be ironed out. Notably there have been observed instances of data-loss when connection to a gds file has been interrupted without proper closure using `closefn.gds`. As such it is **imperative** that once you are ready to exit R, you must close the connection to the gds file and then exit R.

```
> # Closing the connection
> closefn.gds(gfile)
```

8 Session Info

```
> sessionInfo()
```

```
R version 3.3.1 (2016-06-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C                LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                   LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets  methods
```

```
[9] base

other attached packages:
 [1] bigmelon_1.0.0
 [2] gdsfmt_1.10.0
 [3] wateRmelon_1.18.0
 [4] illuminaio_0.16.0
 [5] IlluminaHumanMethylation450kanno.ilmn12.hg19_0.6.0
 [6] ROC_1.50.0
 [7] lumi_2.26.0
 [8] methylumi_2.20.0
 [9] minfi_1.20.0
[10] bumphunter_1.14.0
[11] locfit_1.5-9.1
[12] iterators_1.0.8
[13] foreach_1.4.3
[14] Biostrings_2.42.0
[15] XVector_0.14.0
[16] SummarizedExperiment_1.4.0
[17] FDb.InfiniumMethylation.hg19_2.2.0
[18] org.Hs.eg.db_3.4.0
[19] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
[20] GenomicFeatures_1.26.0
[21] AnnotationDbi_1.36.0
[22] GenomicRanges_1.26.0
[23] GenomeInfoDb_1.10.0
[24] IRanges_2.8.0
[25] S4Vectors_0.12.0
[26] ggplot2_2.1.0
[27] reshape2_1.4.1
[28] scales_0.4.0
[29] matrixStats_0.51.0
[30] limma_3.30.0
[31] Biobase_2.34.0
[32] BiocGenerics_0.20.0

loaded via a namespace (and not attached):
 [1] httr_1.2.1               nor1mix_1.2-2            splines_3.3.1
 [4] affy_1.52.0              doRNG_1.6                Rsamtools_1.26.0
 [7] RSQLite_1.0.0            lattice_0.20-34          quadprog_1.5-5
[10] chron_2.3-47             digest_0.6.10            RColorBrewer_1.1-2
[13] colorspace_1.2-7         preprocessCore_1.36.0    Matrix_1.2-7.1
[16] plyr_1.8.4               GEOquery_2.40.0          siggenes_1.48.0
[19] XML_3.98-1.4             biomaRt_2.30.0           genefilter_1.56.0
[22] zlibbioc_1.20.0          xtable_1.8-2             affyio_1.44.0
[25] BiocParallel_1.8.0       nleqslv_3.0.3            openssl_0.9.4
[28] annotate_1.52.0          mgcv_1.8-15              beanplot_1.2
[31] pkgmaker_0.22            crayon_1.3.2             survival_2.39-5
```

```
[34] magrittr_1.5             mclust_5.2               nlme_3.1-128
[37] MASS_7.3-45              BiocInstaller_1.24.0     tools_3.3.1
[40] registry_0.3             data.table_1.9.6         BiocStyle_2.2.0
[43] stringr_1.1.0            munsell_0.4.3            rngtools_1.2.4
[46] base64_2.0               grid_3.3.1               RCurl_1.95-4.8
[49] bitops_1.0-6             gtable_0.2.0             codetools_0.2-15
[52] multtest_2.30.0          DBI_0.5-1                reshape_0.8.5
[55] R6_2.2.0                 GenomicAlignments_1.10.0 rtracklayer_1.34.0
[58] KernSmooth_2.23-15       stringi_1.1.2            Rcpp_0.12.7
```

9 References

[1] Pidsley R, Wong CCY, Volta M, Lunnon K, Mill J, Schalkwyk LC (2013) A data-driven approach to preprocessing Illumina 450K methylation array data. BMC genomics, 14(1), 293.

[2] Zheng X, Levine D, Shen J, Gogarten SM, Laurie C, Weir BS (2012) A high-performance computing toolset for relatedness and principal component analysis of SNP data. Bioinformatics, 28, 3326-3328.