

# The bigmelon Package

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

April 27, 2025

## 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 capabilities of the *wateRmelon* to higher dimensional data, allowing larger data sets containing many more arrays to be processed simultaneously, while also providing convenient storage for data for future access and sharing with peers. 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 the memory overheads associated with handling big data in R.

## 2 Quick-Start

---

This section will briefly describe how to import, access and 'analyse' data using bigmelon and the gds file format. All the functions described in this section are described in full in the later portions of this document.

```
> library(bigmelon)
> data(melon)
> # Convert methylumiset or methylset objects to .gds
> gfile <- es2gds(melon, 'melon.gds')
> # 'melon.gds' file created in current working directory
> dir()

[1] "bigmelon.Rnw" "bigmelon.tex" "logmemuse.pdf" "melon.gds"

> # Access data with:
> betas(gfile)

+ betas    { Float64 3363x12, 315.3K }

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

## The bigmelon Package

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

> # Get betas with '[' notation
> betas(gfile)[1:5,1:5]

      6057825008_R01C01 6057825008_R01C02 6057825008_R02C01
cg000000029          0.67233          0.71083          0.67504
cg000000108          0.57508          0.37251          0.65755
cg000000109          0.75909          0.77251          0.78121
cg000000165          0.39772          0.52589          0.30660
cg000000236          0.70793          0.75873          0.75429
      6057825008_R02C02 6057825008_R03C01
cg000000029          0.69099          0.75096
cg000000108          0.48510          0.60844
cg000000109          0.78238          0.83259
cg000000165          0.43188          0.36626
cg000000236          0.76494          0.77341

> # Or call from gfile
> gfile[1:5, 1:5, node = 'methylated']

      6057825008_R01C01 6057825008_R01C02 6057825008_R02C01
cg000000029          2926          2940          2715
cg000000108          4301          2060          5075
cg000000109          1503          1630          1821
cg000000165          943          1381          910
cg000000236          2322          2607          2898
      6057825008_R02C02 6057825008_R03C01
cg000000029          3153          3534
cg000000108          2816          4354
cg000000109          1758          2059
cg000000165          1138          1068
cg000000236          2727          2891

> # Preprocess data with pfilter and dasen
> pfilter(gfile)

NBeads missing, using betas instead...

> dasen(gfile)
> # Note you do not have to store the output
> # because the functions make changes to disk
>
> # Use apply.gdsn (or clusterApply.gdsn) to perform apply-like operations
> meth <- methylated(gfile)
> apply.gdsn(meth, 2, median, as.is='double', na.rm = TRUE)

 [1] 1817.0 1883.0 1862.5 1820.5 2035.5 2077.5 1603.0 1826.5 1664.0 1881.0
[11] 1248.0 1284.5
```

## The bigmelon Package

```
> # Calculating Horvath's epigenetic ages with agep
> data(coef)
> agep(gfile, coeff=coef)

      custom_age n_missing
6057825008_R01C01 0.7937227      351
6057825008_R01C02 0.7954131      351
6057825008_R02C01 0.7959686      351
6057825008_R02C02 0.7955614      351
6057825008_R03C01 0.7812136      351
6057825008_R03C02 0.7959020      351
6057825008_R04C01 0.7894942      351
6057825008_R04C02 0.7910460      351
6057825008_R05C01 0.7941418      351
6057825008_R05C02 0.7870633      351
6057825008_R06C01 0.7893790      351
6057825008_R06C02 0.7897976      351

> # Close .gds file
> closefn.gds(gfile)
> # Open a .gds file
> gfile <- openfn.gds('melon.gds')
```

## 3 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:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+   install.packages("BiocManager")
> BiocManager::install('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')
```

## 4 Using Bigmelon

---

### 4.1 Loading Data to gds format

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 (preferred method).

#### 4.1.1 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 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` (from *wateRmelon*) to read in the data - although it should be noted that the full annotation of the features are not included when reading from idats. And can be added to the

`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. You will need to be in the same directory as the idat files for this to work.

`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)
```

#### 4.1.2 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*. Such circumstance can arise when one wants to use a particular normalisation methodology only available to specific packages.

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')
```

## The bigmelon Package

### 4.1.3 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 (SentrrixID\_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')
```

## 5 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` Generally only one instance of a `gds` can be opened per R session, this can be disabled by setting `allow.fork` and `allow.duplicate` arguments in `openfn.gds` to `TRUE`.

## 6 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.

## The bigmelon Package

```
> print(gfile)
```

```
File: /private/var/folders/r0/l4fjk6cj5xj0j3brt4bplpl40000gt/T/Rtmp5UXPmJ/Rbuild12ae285ce008/bigmelon
```

```
+ [ ]
```

```
|--> 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 }
```

## The bigmelon Package

```
| |--+ 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, 87B }
| |--+ finished  { Str8 4, 87B }
| \--+ 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 relevant 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'))
```

## The bigmelon Package

```
[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"
[5] "unmethylated"    "fData"           "pData"           "QCmethylated"
[9] "QCunmethylated"  "QCrownames"      "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"
```



## The bigmelon Package

```
[35] "REGULATORY_FEATURE_GROUP"    "DHS"  
[37] "Index"                       "X38"
```

### 6.1 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]  
[1,] 0.67233 0.71083 0.67504 0.69099 0.75096 0.70100 0.69015 0.70555 0.73616  
[2,] 0.57508 0.37251 0.65755 0.48510 0.60844 0.57751 0.55881 0.58700 0.56206  
[3,] 0.75909 0.77251 0.78121 0.78238 0.83259 0.77071 0.67976 0.81480 0.79522  
[4,] 0.39772 0.52589 0.30660 0.43188 0.36626 0.51246 0.43329 0.29387 0.50085  
[5,] 0.70793 0.75873 0.75429 0.76494 0.77341 0.76085 0.75011 0.70859 0.76726  
[6,] 0.43090 0.44909 0.51544 0.50341 0.48614 0.48561 0.46985 0.40216 0.44669  
      [,10] [,11] [,12]  
[1,] 0.69577 0.71457 0.71944  
[2,] 0.51776 0.50770 0.54396  
[3,] 0.77952 0.70430 0.71377  
[4,] 0.49857 0.48607 0.36159  
[5,] 0.71562 0.70947 0.75575  
[6,] 0.51309 0.55050 0.54047  
  
> # Subset!  
> datsub <- readex.gdsn(anode, sel = list(1:5, 1:3))  
> dim(datsub)  
[1] 5 3
```

## The bigmelon Package

```
> 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
cg000000029          0.67233          0.71083          0.67504
cg000000108          0.57508          0.37251          0.65755
cg000000109          0.75909          0.77251          0.78121
cg000000165          0.39772          0.52589          0.30660
cg000000236          0.70793          0.75873          0.75429

> # Additionally, the row and col names can be disabled
> 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. Most of the indexing tricks that can be implemented on matrices can be performed on gdsn.nodes.

## The bigmelon Package

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

      6057825008_R01C01 6057825008_R02C02 6057825008_R04C01
cg00000029          0.67233          0.69099          0.69015
cg00000108          0.57508          0.48510          0.55881
cg00000109          0.75909          0.78238          0.67976
cg00000165          0.39772          0.43188          0.43329
cg00000236          0.70793          0.76494          0.75011

      6057825008_R05C02
cg00000029          0.69577
cg00000108          0.51776
cg00000109          0.77952
cg00000165          0.49857
cg00000236          0.71562

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

      6057825008_R04C02 6057825008_R02C02 6057825008_R01C01
cg00000236          0.70859          0.76494          0.70793
cg00000363          0.10832          0.12452          0.11314
cg00000029          0.70555          0.69099          0.67233
cg00020649          0.03407          0.03478          0.02913
cg00014272          0.07078          0.07480          0.06852

      6057825008_R02C01
cg00000236          0.75429
cg00000363          0.12508
cg00000029          0.67504
cg00020649          0.03470
cg00014272          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[ , ] # Not recommended for large data.
> dim(dat)

[1] 3363 12
```

## The bigmelon Package

Additionally it is possible to call a gds node from a gds.class within the '[' indexing. This is particularly useful if you have a foreign matrix in your gds object with the name "foobar", it will be possible to implement the below structures to retrieve specific data.

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

	6057825008_R01C01	6057825008_R01C02	6057825008_R02C01
cg000000029	0.67233	0.71083	0.67504
cg000000108	0.57508	0.37251	0.65755
cg000000109	0.75909	0.77251	0.78121
cg000000165	0.39772	0.52589	0.30660
cg000000236	0.70793	0.75873	0.75429

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

	6057825008_R01C01	6057825008_R01C02	6057825008_R02C01
cg000000029	2926	2940	2715
cg000000108	4301	2060	5075
cg000000109	1503	1630	1821
cg000000165	943	1381	910
cg000000236	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] "cg000000029" "cg000000108" "cg000000109" "cg000000165" "cg000000236"
```

```
[6] "cg000000289"
```

```
> head(read.gdsn(index.gdsn(gfile, "pData/sampleID")))
```

```
[1] "6057825008_R01C01" "6057825008_R01C02" "6057825008_R02C01"
```

```
[4] "6057825008_R02C02" "6057825008_R03C01" "6057825008_R03C02"
```

## 7 Preprocessing

### 7.1 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)
```

## The bigmelon Package



Alternatively it is possible to use 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

The bigmelon Package

6057825008_R05C02	FALSE	FALSE	FALSE
6057825008_R06C01	FALSE	FALSE	FALSE
6057825008_R06C02	FALSE	FALSE	FALSE



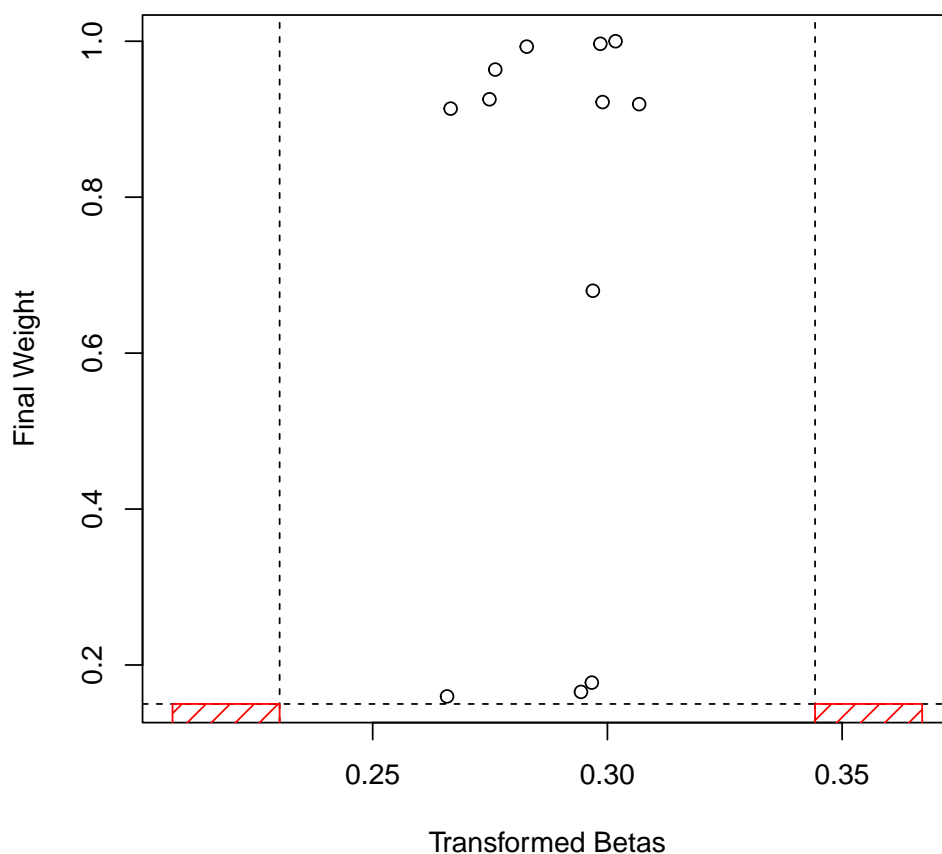
If the data is too large to load into memory, one can use the *bigmelon* method which determines outliers with a small subset of data.

```
> 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 FALSE   FALSE
6057825008_R04C01 FALSE FALSE   FALSE
```

## The bigmelon Package

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 is in read mode.

```
> pfilter(gfile)
```



## The bigmelon Package

```
NBeads missing, using betas instead...
```

Alternatively if you do not wish to perform subsetting at this time one can use `pfilter.gds` to get the list of failed probes and use those for subsetting at a later time.

### 7.2 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 `gds` file 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: /private/var/folders/r0/l4fjk6cj5xj0j3brt4bplpl40000gt/T/Rtmp5UXPmJ/Rbuild12ae285ce008/bigmel
+   [ ]
\--+ backup   [ ]
    \--+ betas { Float64 3252x12, 304.9K }

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

File: /private/var/folders/r0/l4fjk6cj5xj0j3brt4bplpl40000gt/T/Rtmp5UXPmJ/Rbuild12ae285ce008/bigmel
+   [ ]
|--+ 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)
```

### 7.3 Normalization

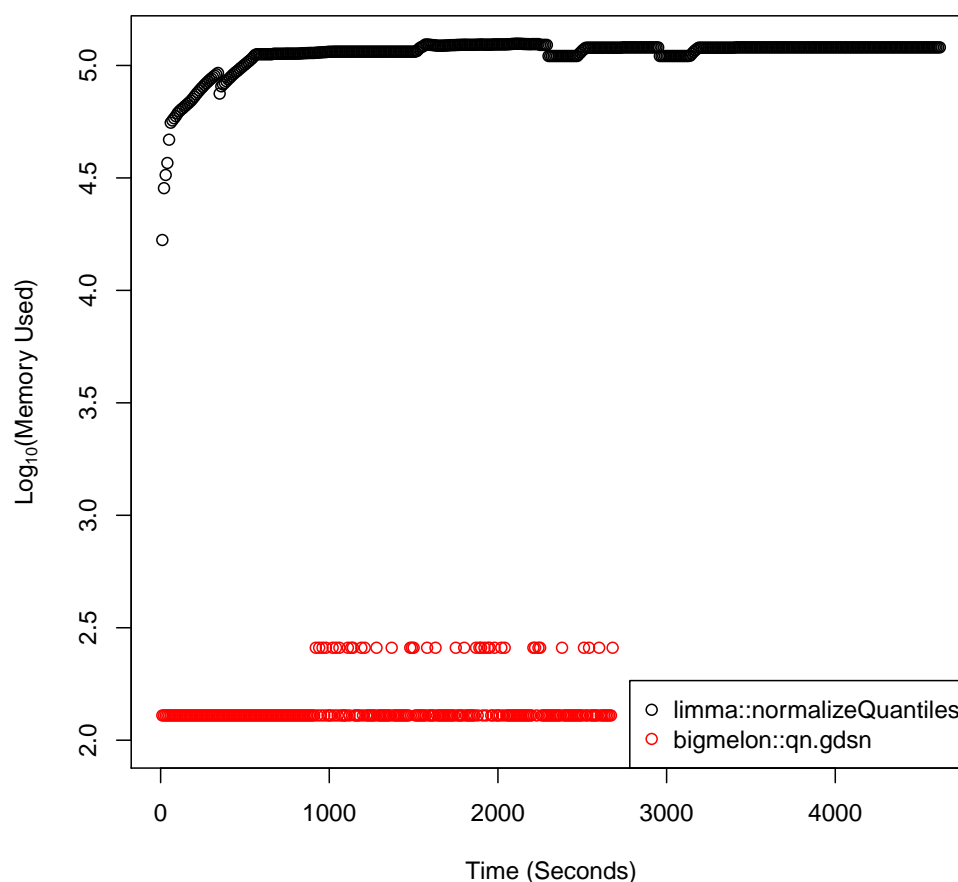
Within *bigmelon* there are numerous normalisation methods that can be used. The method `dasen` will work well for most data sets. **n.b.** This will perform irreversible procedures on the data. And will replace raw intensities with the normalised ones. This will not work if `gds` file is in read mode.

## The bigmelon Package

```
> 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 mem-



ory use.

## 8 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] 0.4291726 0.4293674 0.4280594 0.4287715 0.4295416 0.4297672 0.4273244
[8] 0.4281042 0.4284096 0.4298339 0.4285671 0.4297395
```

Additional analyses that could be useful to an EWAS in particular are age prediction and estimating cell types, these can be done with `agep` and `estimateCellCounts.gds`

```
> # Age Prediction
> data(coef) # Load up a set of coefficient (horvaths)
> agep(gfile, coef)

      custom_age n_missing
6057825008_R01C01 0.7937227      351
6057825008_R01C02 0.7954131      351
6057825008_R02C01 0.7959686      351
6057825008_R02C02 0.7955614      351
6057825008_R03C01 0.7812136      351
6057825008_R03C02 0.7959020      351
6057825008_R04C01 0.7894942      351
6057825008_R04C02 0.7910460      351
6057825008_R05C01 0.7941418      351
6057825008_R05C02 0.7870633      351
6057825008_R06C01 0.7893790      351
6057825008_R06C02 0.7897976      351

> # Alternatively
> agep(betas(gfile), coef) # or index.gdsn(gfile, 'foobar') for a different matrix

      custom_age n_missing
6057825008_R01C01 0.7937227      351
6057825008_R01C02 0.7954131      351
6057825008_R02C01 0.7959686      351
6057825008_R02C02 0.7955614      351
6057825008_R03C01 0.7812136      351
6057825008_R03C02 0.7959020      351
6057825008_R04C01 0.7894942      351
```

## The bigmelon Package

6057825008_R04C02	0.7910460	351
6057825008_R05C01	0.7941418	351
6057825008_R05C02	0.7870633	351
6057825008_R06C01	0.7893790	351
6057825008_R06C02	0.7897976	351

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.

## 9 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 environment.

```
> 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: cg000000029 cg000000108 ... rs9839873 (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
```

## The bigmelon Package

```
4 2025-04-27 15:43:58.771464 2025-04-27 15:43:58.792405
5 2025-04-27 15:43:59.031111 2025-04-27 15:43:59.045399
6 2025-04-27 15:43:59.045681 2025-04-27 15:43:59.069517
7 2025-04-27 15:43:59.085818 2025-04-27 15:43:59.724838
8 2025-04-27 15:43:59.725557 2025-04-27 15:44:00.379507
9 2025-04-27 15:44:00.402429 2025-04-27 15:44:00.460831
                                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")

class: MethylSet
dim: 3252 12
metadata(0):
assays(2): Meth Unmeth
rownames(3252): cg000000029 cg000000108 ... rs966367 rs9839873
rowData names(0):
colnames(12): 6057825008_R01C01 6057825008_R01C02 ... 6057825008_R06C01
               6057825008_R06C02
colData names(3): sampleID label sex
Annotation
  array: IlluminaHumanMethylation450k
  annotation: ilmn12.hg19
Preprocessing
  Method: Converted from gdsfmt to MethylSet (bigmelon)
  minfi version: 1.55.0
  Manifest version: NA
```

## 10 Adding additional objects to gds objects

---

Often, it will be necessary to make use of a transformation while additionally preserving the original betas. This can be done using the `add.gdsn` which is described in the `gdsfmt` vignette in great detail.

## 11 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)
```

## 12 Session Info

---

```
> sessionInfo()

R version 4.5.0 Patched (2025-04-21 r88169)
Platform: aarch64-apple-darwin20
Running under: macOS Ventura 13.7.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib; LAPACK

locale:
 [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/New_York
tzcode source: internal

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

other attached packages:
 [1] bigmelon_1.35.0
 [2] gdsfmt_1.45.0
 [3] wateRmelon_2.15.0
 [4] illuminaio_0.51.0
 [5] IlluminaHumanMethylation450kanno.ilmn12.hg19_0.6.1
 [6] ROC_1.85.0
 [7] lumi_2.61.0
 [8] methyllumi_2.55.0
 [9] minfi_1.55.0
[10] bumphunter_1.51.0
```

## The bigmelon Package

```
[11] locfit_1.5-9.12
[12] iterators_1.0.14
[13] foreach_1.5.2
[14] Biostrings_2.77.0
[15] XVector_0.49.0
[16] SummarizedExperiment_1.39.0
[17] MatrixGenerics_1.21.0
[18] FDb.InfiniumMethylation.hg19_2.2.0
[19] org.Hs.eg.db_3.21.0
[20] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
[21] GenomicFeatures_1.61.0
[22] AnnotationDbi_1.71.0
[23] GenomicRanges_1.61.0
[24] GenomeInfoDb_1.45.0
[25] IRanges_2.43.0
[26] S4Vectors_0.47.0
[27] ggplot2_3.5.2
[28] reshape2_1.4.4
[29] scales_1.4.0
[30] matrixStats_1.5.0
[31] limma_3.65.0
[32] Biobase_2.69.0
[33] BiocGenerics_0.55.0
[34] generics_0.1.3
```

loaded via a namespace (and not attached):

```
[1] RColorBrewer_1.1-3      jsonlite_2.0.0
[3] magrittr_2.0.3          farver_2.1.2
[5] rmarkdown_2.29          BiocIO_1.19.0
[7] vctrs_0.6.5             multtest_2.65.0
[9] memoise_2.0.1           Rsamtools_2.25.0
[11] DelayedMatrixStats_1.31.0 RCurl_1.98-1.17
[13] askpass_1.2.1           htmltools_0.5.8.1
[15] S4Arrays_1.9.0          curl_6.2.2
[17] Rhdf5lib_1.31.0         SparseArray_1.9.0
[19] rhdf5_2.53.0            KernSmooth_2.23-26
[21] nor1mix_1.3-3           plyr_1.8.9
[23] cachem_1.1.0            GenomicAlignments_1.45.0
[25] lifecycle_1.0.4        pkgconfig_2.0.3
[27] Matrix_1.7-3            R6_2.6.1
[29] fastmap_1.2.0           GenomeInfoDbData_1.2.14
[31] digest_0.6.37           siggenes_1.83.0
[33] reshape_0.8.9           RSQLite_2.3.9
[35] base64_2.0.2            mgcv_1.9-3
[37] httr_1.4.7              abind_1.4-8
```

## The bigmelon Package

[39] compiler_4.5.0	beanplot_1.3.1
[41] rngtools_1.5.2	bit64_4.6.0-1
[43] withr_3.0.2	BiocParallel_1.43.0
[45] DBI_1.2.3	HDF5Array_1.37.0
[47] MASS_7.3-65	openssl_2.3.2
[49] DelayedArray_0.35.1	rjson_0.2.23
[51] tools_4.5.0	rentrez_1.2.3
[53] glue_1.8.0	quadprog_1.5-8
[55] h5mread_1.1.0	restfulr_0.0.15
[57] nlme_3.1-168	rhdf5filters_1.21.0
[59] grid_4.5.0	gtable_0.3.6
[61] tzdb_0.5.0	preprocessCore_1.71.0
[63] tidyr_1.3.1	data.table_1.17.0
[65] hms_1.1.3	xml2_1.3.8
[67] pillar_1.10.2	stringr_1.5.1
[69] genefilter_1.91.0	splines_4.5.0
[71] dplyr_1.1.4	lattice_0.22-7
[73] survival_3.8-3	rtracklayer_1.69.0
[75] bit_4.6.0	GEQuery_2.77.0
[77] annotate_1.87.0	tidyselect_1.2.1
[79] knitr_1.50	xfun_0.52
[81] scrime_1.3.5	statmod_1.5.0
[83] stringi_1.8.7	UCSC.utils_1.5.0
[85] yaml_2.3.10	evaluate_1.0.3
[87] codetools_0.2-20	tibble_3.2.1
[89] BiocManager_1.30.25	affyio_1.79.0
[91] cli_3.6.5	xtable_1.8-4
[93] dichromat_2.0-0.1	Rcpp_1.0.14
[95] png_0.1-8	XML_3.99-0.18
[97] readr_2.1.5	blob_1.2.4
[99] mclust_6.1.1	doRNG_1.8.6.2
[101] sparseMatrixStats_1.21.0	bitops_1.0-9
[103] affy_1.87.0	nleqslv_3.3.5
[105] purrr_1.0.4	crayon_1.5.3
[107] BiocStyle_2.37.0	rlang_1.1.6
[109] KEGGREST_1.49.0	

## 13 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.



## The bigmelon Package

[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.