M^{3}D: Statistical Testing for RRBS Data Sets

Tom Mayo

Modified: 5 September, 2016. Compiled: April 24, 2017

Contents

1 Introduction

**** Please note, that as of 17th June 2016, we have introduced new 'lite' functions, detailed in section 3. These are faster and more memory efficient than their predecessors and we recommend their use where possible ****

RRBS experiments offer a cost-effective method for measuring the methylation levels of cytosines in CpG dense regions. Statistical testing for differences in CpG methylation between sample groups typically involves beta-binomial modelling of CpG loci individually and testing across samples, such as with the BiSeq and MethylSig packages. Individual CpGs are then chained together to output a list of putative differentially methylated regions (DMRs).

We take a different approach, instead testing pre-defined regions of interest for changes in the distribution of the methylation profiles. Changes are compared to inter-replicate differences to establish which regions vary in a manner that cannot be explained by replicate variation alone. We utilise the maximum mean discrepancy (MMD) [1] to perform the test, adjusting this measure to account for changes in the coverage profile between the testing groups. For a similar application of the MMD to ChIP-Seq data, please see the MMDiff Bioconductor package, and associated paper [2].

In this vignette, we run through an analysis of a toy data set using the M^{3}D method. We use the same data structures as the 'BiSeq' package, and assume that we have data stored in an rrbs structure with an accompanying GRanges object describing the regions of interest we want to test. For a fuller explanation and exploration of the method, please see the open access journal article [3].

2 Analysis Pipeline

2.1 Reading in Data

We use the same data structures as the 'BiSeq' package. Please see their manual for manipulating the data structures.

We have supplied a function for reading in different types of bed files. This can be done with the readBedFiles function. Which we use as follows:

```
rrbs <- readBedFiles(files, colData, bed_type = 'Encode', eData=NaN)
```

Where 'files' is a vector of character strings, with each string pointing to a bed file for a sample. 'colData' is a data frame that specifies the names of the samples and the groups (or conditions) to which they belong. For example, if we are comparing two ENCODE samples from H1-hESC cells to two samples from K562 leukemia cells, we would do the following, assuming the current working directory contains the files:

```
files <- c('wgEncodeHaibMethylRrbsH1hescHaibSitesRep1.bed.gz',
'wgEncodeHaibMethylRrbsH1hescHaibSitesRep2.bed.gz',
'wgEncodeHaibMethylRrbsK562HaibSitesRep1.bed.gz',
'wgEncodeHaibMethylRrbsK562HaibSitesRep2.bed.gz')
group <- factor(c('H1-hESC','H1-hESC','K562','K562'))
samples <- c('H1-hESC1','H1-hESC2','K562-1','K562-2')
colData <- DataFrame(group,row.names= samples)
```

Here DataFrame has been defined in the BiSeq package, so be sure to load that first.

As mentioned previously, bed files come in different styles or types. You can check yours on linus by using 'less' followed by the file name. This will print to the console a table of the first few rows, from which you can deduce which one you have.

The following description was taken from the RnBeads vignette. The names in inverted commas are all options that we pass to readBedFiles in the bed_type argument.

'BisSNP' The bed files are assumed to have been generated by the methylation calling tool BisSNP. A tab-separated file contains the chromosome name, start coordinate, end coordinate, methylation value in percent, the coverage, the strand, as well as additional information not used in the package. The file should contain a header line. Coordinates are 0-based, spanning the first and the last coordinate in a site (i.e. end-start = 1 for a CpG). Sites on the negative strand are shifted by +1. Here are some example lines (genome assembly hg19):

```
track name=file_sorted.realign.recal.cpg.filtered.sort.CG.bed type=bedDetail description="CG methylation
chr1    10496   10497   79.69   64      +       10496   10497   180,60,0        0       0
chr1    10524   10525   90.62   64      +       10524   10525   210,0,0 0       0
chr1    864802  864803  58.70   46      +       864802  864803  120,120,0       0       5
chr1    864803  864804  50.00   4       -       864803  864804  90,150,0        1       45
...
```

'EPP' The bed files are assumed to have the format as output files from the Epigenome Processing Pipeline developed by Fabian Müller and Christoph Bock. A tab-separated file contains: the chromosome name, start coordinate, end coordinate, methylation value and coverage as a string ('#M/#T'), methylation level scaled to the interval 0 to 1000, strand, and additional information not used in this package. The file should not contain a header line. Coordinates are 0-based, spanning the first coordinate in a site and the first coordinate outside the site (i.e. end-start = 2 for a CpG). Here are some example lines (genome assembly mm9):

```
chr1    3010957 3010959 '27/27' 1000    +
chr1    3010971 3010973 '10/20' 500     +
chr1    3011025 3011027 '57/70' 814     -
...
```

'bismarkCov' The cov files are assumed to have the format as defined by Bismark's coverage file output converted from its bedGraph output (Bismark's bismark2bedGraph module; see the section "Optional bedGraph output" in the Bismark User Guide). A tab-separated file contains: the chromosome name, cytosine coordinate, cytosine coordinate (again), methylation value in percent, number of methylated reads (#M) and the number of unmethylated reads (#U). The file should not contain a header line. Coordinates are 1-based. Strand information does not need to be provided, but is inferred from the coordinates: Coordinates on the negative strand specify the cytosine base position (G on the positive strand). Coordinates referring to cytosines not in CpG content are automatically discarded. Here are some example lines (genome assembly hg19):

```
...
chr9    73252   73252   100     1       0
chr9    73253   73253   0       0       1
chr9    73256   73256   100     1       0
```

```
chr9    73260   73260   0       0       1
chr9    73262   73262   100     1       0
chr9    73269   73269   100     1       0
...
```

'bismarkCytosine' The bed files are assumed to have the format as defined by Bismark's cytosine report output (Bismark's coverage2cytosine module; see the section "Optional genome-wide cytosine report output" in the Bismark User Guide). A tab-separated file contains: the chromosome name, cytosine coordinate, strand, number of methylated reads (#M), number of unmethylated reads (#U), and additional information not used in this package. The file should not contain a header line. Coordinates are 1-based. Coordinates on the negative strand specify the cytosine position (G on the positive strand). CpG without coverage are allowed, but not required. Here are some example lines (genome assembly hg19):

```
...
chr22   16050097        +       0       0       CG      CGG
chr22   16050098        −       0       0       CG      CGA
chr22   16050114        +       0       0       CG      CGG
chr22   16050115        −       0       0       CG      CGT
...
chr22   16115591        +       1       1       CG      CGC
chr22   16117938        −       0       2       CG      CGT
chr22   16122790        +       0       1       CG      CGC
...
```

'Encode' The bed files are assumed to have the format as the ones that can be downloaded from UCSC's ENCODE data portal. A tab-separated file contains: the chromosome name, start coordinate, end coordinate, some identifier, read coverage (#T), strand, start and end coordinates again (we discard this duplicated information), some color value, read coverage (#T) and the methylation percentage. The file should contain a header line. Coordinates are 0-based. Note that this file format is very similar but not identical to the 'BisSNP' one. Here are some example lines (genome assembly hg19):

```
track name="SL1815 MspIRRBS" description="HepG2_B1__GC_" visibility=2 itemRgb="On"
chr1    1000170 1000171 HepG2_B1__GC_   62      +       1000170 1000171 55,255,0    62      6
chr1    1000190 1000191 HepG2_B1__GC_   62      +       1000190 1000191 0,255,0     62      3
chr1    1000191 1000192 HepG2_B1__GC_   31      −       1000191 1000192 0,255,0     31      0
chr1    1000198 1000199 HepG2_B1__GC_   62      +       1000198 1000199 55,255,0    62      10
chr1    1000199 1000200 HepG2_B1__GC_   31      −       1000199 1000200 0,255,0     31      0
chr1    1000206 1000207 HepG2_B1__GC_   31      −       1000206 1000207 55,255,0    31      10
...
```

For illustrative purposes, we have included data from two RRBS datasets via the ENCODE consortium [4], namely the H1-hESC human embryonic stem cells and the K562 leukemia cell line. We clustered the data using the 'clusterSites' and 'clusterSitesToGR' functions in the BiSeq package, removed any islands with a total coverage of less than 100 in any island over any sample and included only the first 1000. Full details of where to download the data and how to read it in are included in the section 'Using ENCODE Data', so that this toy set and the corresponding full data set can be used.

We load and show the data with the following.

```
library(BiSeq)
library(M3D)
```

```
data(rrbsDemo)
rrbsDemo

## class: BSraw
## dim: 39907 4
## metadata(0):
## assays(2): totalReads methReads
## rownames(39907): 643464 643465 ... 687062 687063
## rowData names(1): cluster.id
## colnames(4): H1-hESC1 H1-hESC2 K562-1 K562-2
## colData names(1): group
```

The regions to be tested are stored in a Granges object, with each entry representing the start and end of the region. This can be loaded as follows:

```
data(CpGsDemo)
CpGsDemo

## GRanges object with 1000 ranges and 1 metadata column:
##          seqnames                 ranges strand | cluster.id
##            <Rle>              <IRanges>  <Rle> |   <factor>
##      [1]     chr1       [840131, 840357]      * |     chr1_1
##      [2]     chr1       [845246, 845561]      * |     chr1_2
##      [3]     chr1       [858978, 859375]      * |     chr1_3
##      [4]     chr1       [859529, 859727]      * |     chr1_4
##      [5]     chr1       [875730, 875953]      * |     chr1_5
##      ...      ...                    ...    ... .        ...
##    [996]     chr1 [168105869, 168106338]      * |  chr1_1009
##    [997]     chr1 [168148116, 168148585]      * |  chr1_1010
##    [998]     chr1 [168194959, 168195245]      * |  chr1_1011
##    [999]     chr1 [169075431, 169075641]      * |  chr1_1012
##   [1000]     chr1 [170633423, 170633656]      * |  chr1_1013
##   -------
##   seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

2.2 Computing the MMD, and coverage-MMD

The M^{3}D test statistic is calculated by finding the maximum mean discrepancy, over each island, of the full methylation data and the coverage data. Both of these are achieved using the M3D_Wrapper function. This outputs a list, with the first entry being a matrix of the pairwise full, methylation aware MMD of each possible sample pair, and the second being the coverage only equivalent.

The function requires a list of overlap locations, which we create as follows:

```
data(CpGsDemo)
data(rrbsDemo)
overlaps <- findOverlaps(CpGsDemo,rowRanges(rrbsDemo))
```

The components of the M^{3}D statistic are then generated with the code:

```
MMDlistDemo <- M3D_Wrapper(rrbsDemo, overlaps)
```

Alternatively, you can load this data directly:

```
data(MMDlistDemo)
```

We show the structure of each entry, where column names correspond to the sample pairs being tested.

```
# the full MMD
head(MMDlistDemo$Full)

##      H1-hESC1  vs  H1-hESC2 H1-hESC1  vs  K562-1 H1-hESC1  vs  K562-2
## [1,]             0.13328796           0.3711302           0.3882027
## [2,]             0.15843597           0.1991325           0.1901730
## [3,]             0.11149443           0.1376077           0.1476828
## [4,]             0.05180868           0.1450703           0.1196117
## [5,]             0.17830486           0.2438087           0.1983181
## [6,]             0.02382480           0.0972950           0.1378252
##      H1-hESC2  vs  K562-1 H1-hESC2  vs  K562-2 K562-1  vs  K562-2
## [1,]           0.23581746          0.22551184        0.06755053
## [2,]           0.14194300          0.14213741        0.04260434
## [3,]           0.03963481          0.05186073        0.01802223
## [4,]           0.11307940          0.07325263        0.05798640
## [5,]           0.07915218          0.10918123        0.09577036
## [6,]           0.09148931          0.13047384        0.05080778

# the coverage only MMD
```

```
head(MMDlistDemo$Coverage)
```

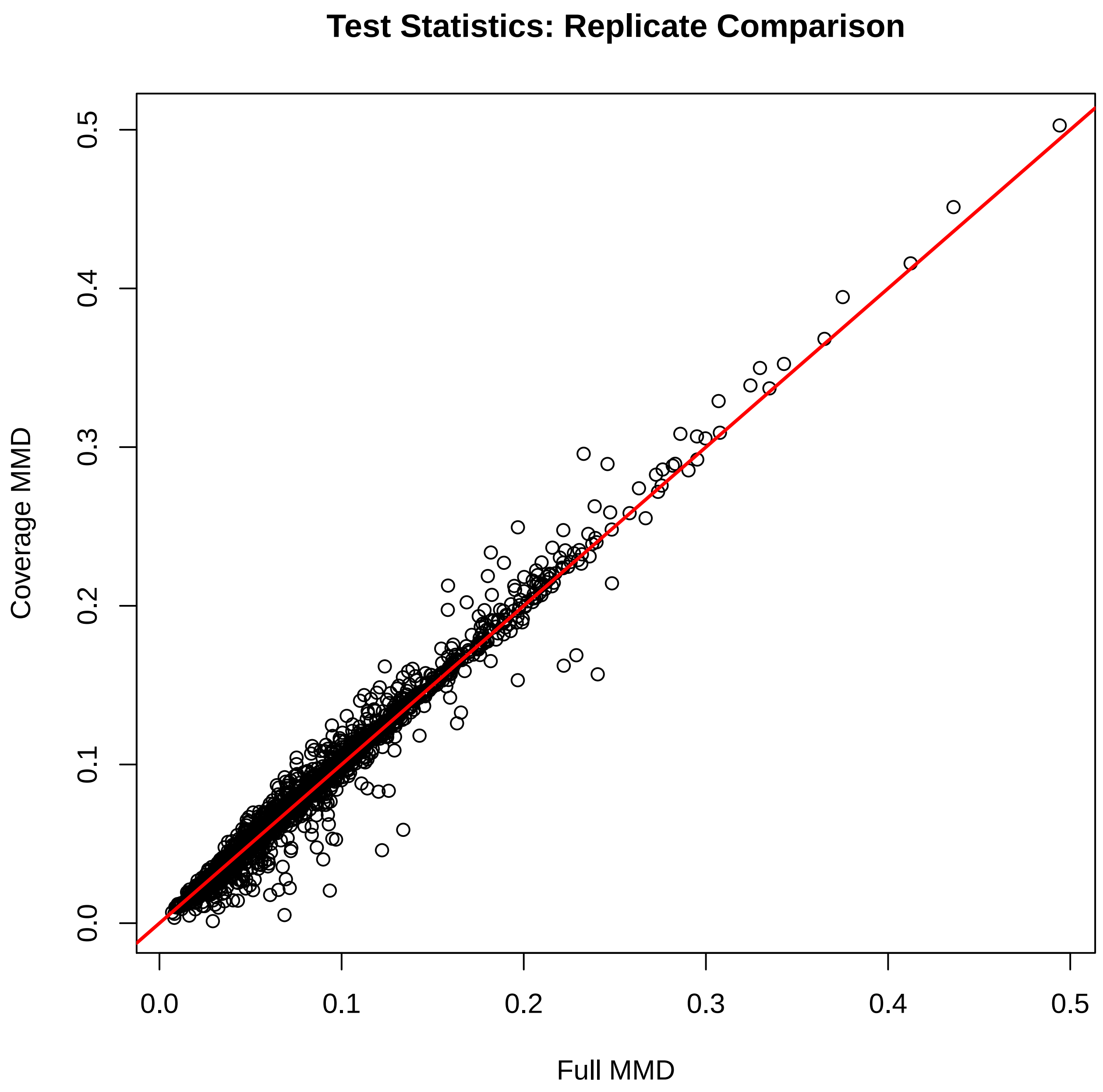
```
##      H1-hESC1  vs  H1-hESC2 H1-hESC1  vs  K562-1 H1-hESC1  vs  K562-2
## [1,]            0.13245172           0.3979882           0.3998015
## [2,]            0.21273851           0.1709892           0.1664227
## [3,]            0.11562054           0.1408935           0.1493031
## [4,]            0.05353733           0.1461974           0.1205286
## [5,]            0.17977249           0.2456640           0.2034446
## [6,]            0.02360063           0.0988395           0.1365794
##      H1-hESC2  vs  K562-1 H1-hESC2  vs  K562-2 K562-1  vs  K562-2
## [1,]          0.26189520          0.23546355         0.06338745
## [2,]          0.05580718          0.07902125         0.04269711
## [3,]          0.04014058          0.04976709         0.01636765
## [4,]          0.11230110          0.07220178         0.05731698
## [5,]          0.07999092          0.11754403         0.09997088
## [6,]          0.09302259          0.12916722         0.04774588
```

The M^{3}D test statistic uses the difference between these values.

```
M3Dstat <- MMDlistDemo$Full-MMDlistDemo$Coverage
```

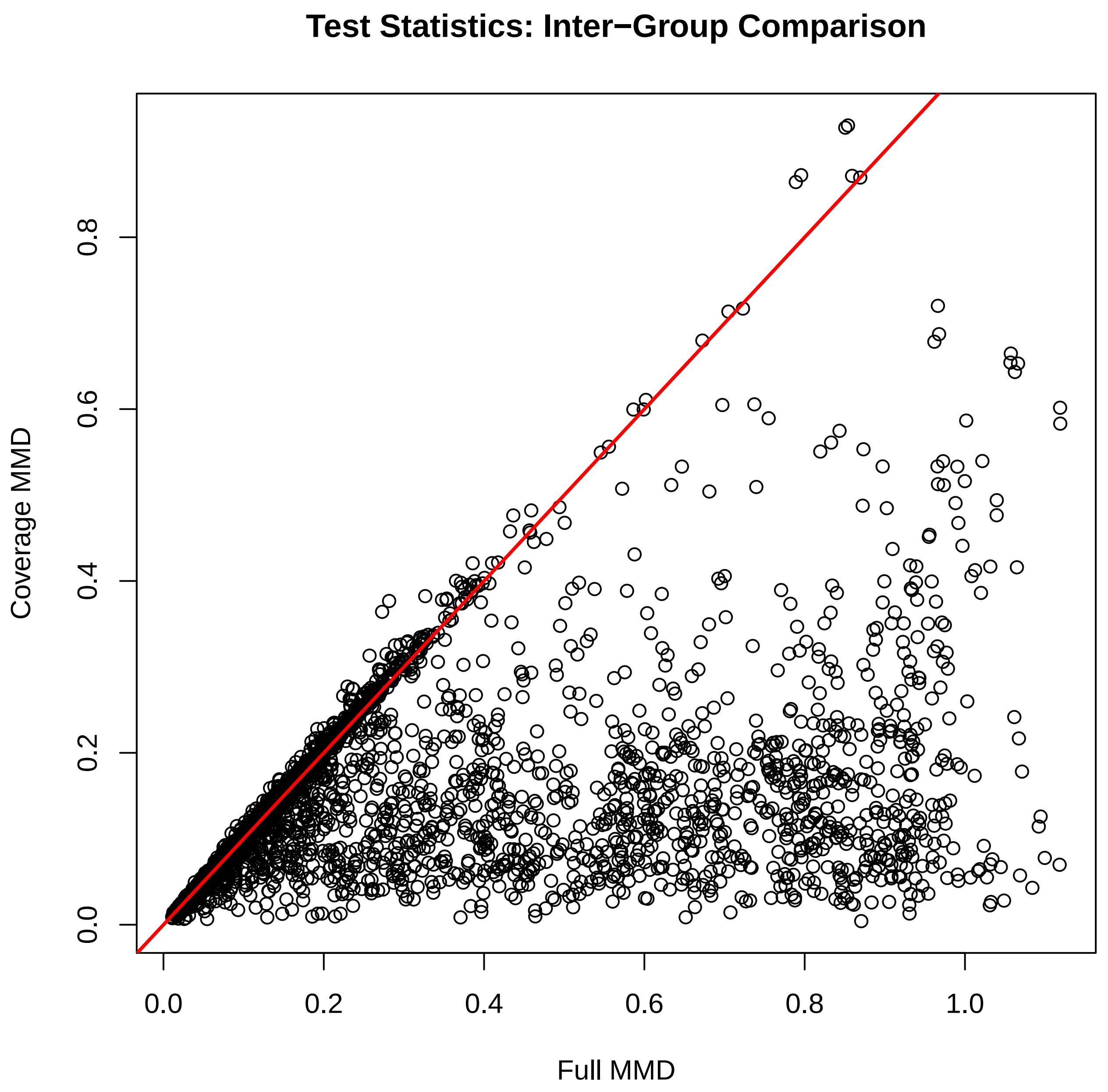
In the matrices we have stored, we can see from the column names that the columns pertaining to inter-replicate values are 1 and 6, while 2 to 5 detail inter-group comparisons, since there are . We can plot the values for replicates as follows:

```
repCols <- c(1,6)
plot(as.vector(MMDlistDemo$Full[,repCols]),
    as.vector(MMDlistDemo$Coverage[,repCols]),
    xlab='Full MMD',ylab='Coverage MMD')
    title('Test Statistics: Replicate Comparison')
abline(a=0,b=1,col='red',lwd=2)
```



And analogously for the inter-group values. Note that with the replicates, the values are close to the red line, representing equality. With the inter-group metrics, we see that, with some of the regions, the full MMD is greater than the coverage only version. This forms the basis of the M^{3}D test statistic, which is used in the following section.

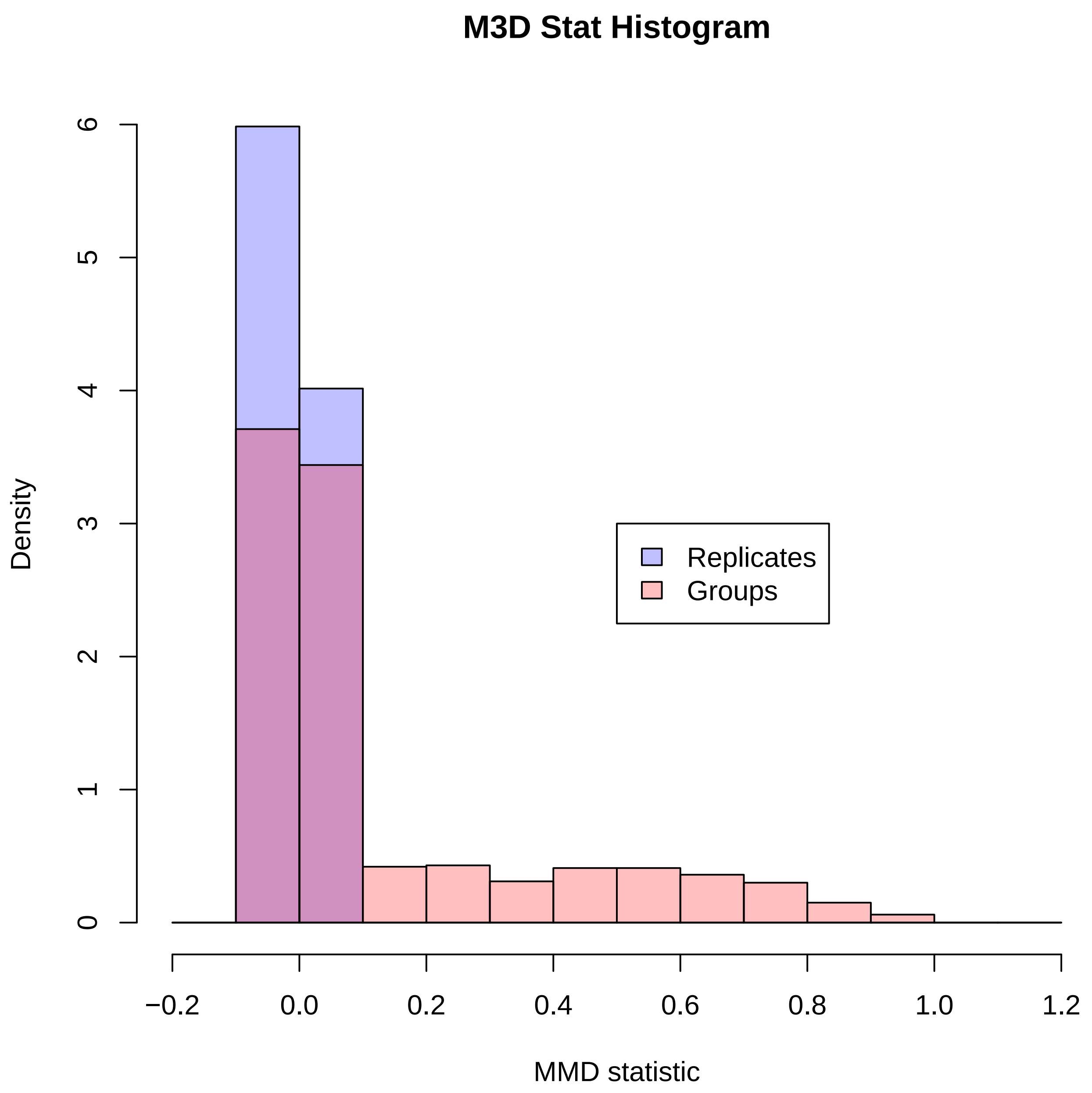
```
groupCols <- 2:5
plot(as.vector(MMDlistDemo$Full[,groupCols]),
    as.vector(MMDlistDemo$Coverage[,groupCols]),
    xlab='Full MMD',ylab='Coverage MMD')
title('Test Statistics: Inter-Group Comparison')
abline(a=0,b=1,col='red',lwd=2)
```



This can be summarised in a histogram of the M^{3}D test statistics.

```
repCols <- c(1,6)
groupCols <- 2:5
M3Dstat <- MMDlistDemo$Full - MMDlistDemo$Coverage
breaks <- seq(-0.2,1.2,0.1)
# WT reps
hReps <- hist(M3Dstat[,repCols], breaks=breaks,plot=FALSE)
# mean between groups
hGroups <- hist(rowMeans(M3Dstat[,groupCols]),breaks=breaks,plot=FALSE)
col1 <- "#0000FF40"
col2 <- "#FF000040"
```

```
plot(hReps,col=col1, freq=FALSE, ylab='Density',
    xlab='MMD statistic', main= 'M3D Stat Histogram')
plot(hGroups, col=col2, freq=FALSE, add=TRUE)
legend(x=0.5, y =3, legend=c('Replicates', 'Groups'), fill=c(col1, col2))
```



2.3 Generating p-values

P-values are calculated by using the M^{3}D test statistic observed between all of the replicates - our 'null' distribution. For each island, we take the mean of the inter-group test-statistics, and calculate the likelihood of observing that value or higher among the inter-replicate values.

We provide two methods for this calculation. 'empirical' gives the empirical probabilities, and is the default method. In the event that we see M^{3}D test statistics between groups above the range of the inter-replicate values, we get p-values of 0. This is not a concern for large samples, but in case of small samples, we recommend using the 'model' option, whereby an exponential is fitted to the tails of the distribution and p-values are calculated from the exponential.

The function 'pvals' computes this, taking in the raw data, the regions, the test statistic and the names of the two groups being compared as stored in the rrbs object.

This outputs a list with 2 entries. The one we are concerned with is FDRmean, the adjusted p-values for each region. The unadjusted values are stored in 'Pmean'.

The structure can be explored with:

```
data(PDemo)
```

Or calculated via:

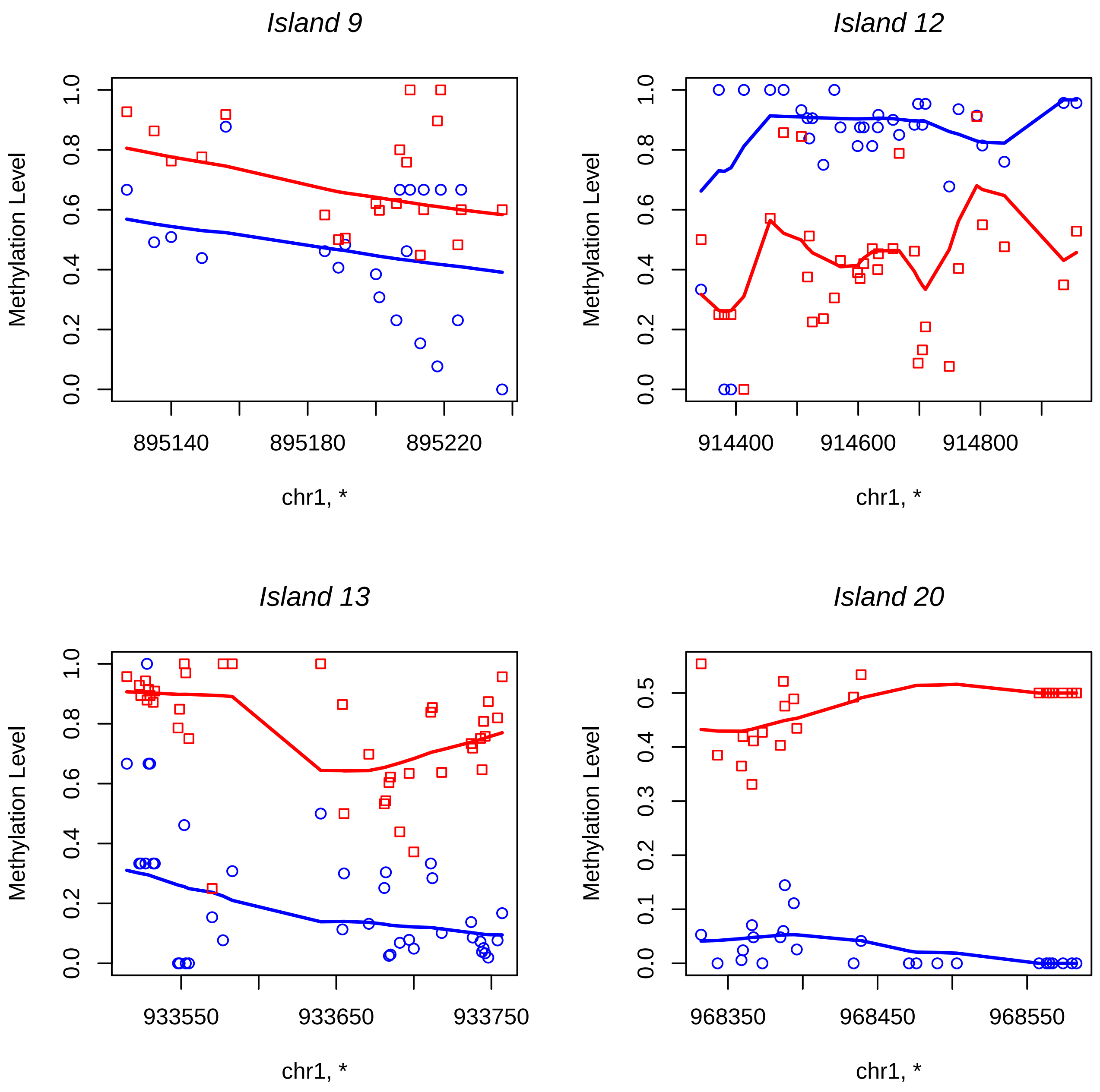
```
group1 <- unique(colData(rrbsDemo)$group)[1]
group2 <-unique(colData(rrbsDemo)$group)[2]
PDemo <- pvals(rrbsDemo, CpGsDemo, M3Dstat,
    group1, group2, smaller=FALSE, comparison='allReps', method='empirical', closePara=0.005)
head(PDemo$FDRmean)

## [1] 0.98489426 0.01125402 0.72746781 0.63871473 0.87210584 0.65576324
```

PDemo$FDRlist is then a vector with an adjusted p-value for each region being tested. To find which regions fall below a certain threshold, we can test very simply. For example, using a cut off FDR of 1%:

```
called <- which(PDemo$FDRmean<=0.01)
head(called)

## [1]  9 12 13 20 21 22

head(CpGsDemo[called])

## GRanges object with 6 ranges and 1 metadata column:
##       seqnames             ranges strand | cluster.id
##          <Rle>          <IRanges>  <Rle> |   <factor>
##   [1]     chr1 [895127, 895237]      * |     chr1_9
##   [2]     chr1 [914343, 914957]      * |    chr1_12
##   [3]     chr1 [933515, 933757]      * |    chr1_13
##   [4]     chr1 [968332, 968583]      * |    chr1_20
##   [5]     chr1 [968837, 969320]      * |    chr1_21
##   [6]     chr1 [969425, 969767]      * |    chr1_22
##   -------
##   seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

We can use the function 'plotMethProfile' to view a smoothed methylation profile for the called regions, taking the mean of the individual methylation levels within each group.

```
par(mfrow=c(2,2))
for (i in 1:4){
    CpGindex <- called[i]
    plotMethProfile(rrbsDemo, CpGsDemo, 'H1-hESC', 'K562', CpGindex)
}
```



2.4 Identifying highly variable regions: outlier detection

As of October 2016, M3D now supports identifying highly variable regions in the null distribution, and excluding them from the analysis. We have noticed in some experiments, some regions of the genome have very high variability, which is not representative of the usual, highly regulated patterns. We propose to identify these regions first, then perfrom the statistical test in the usual manner. If you are struggling to find DMRs using M3D, this may be a reason.

To do so, we have an option 'outlier_test' in the pvals functions. By default it is set to FALSE, so to activate the test set this to 'TRUE'. Our method is to identify any regions, among those presented to the function, which have a pair of regions in the null distribution that is greater than 8 times the standard deviation of the null, above the mean of the null. Additionally, we insist that this region must be in the top 2.5 percent of the variation within the null group. These values

(8 and 2.5 percent) are set by the parameters 'cut_off' and 'sds' respectively. If you are performing many tests across different groups, one can set the threshold directly using 'thresh'. This will overwrite the calculation of the threshold.

In this setting, the p-values and FDR-valeus returned are as before, but with 'highly variable' regions returing NA. Additionally, there is a third entry in the returned list, 'HighVariation' which gives the indices of the identified highly variable regions.

One can confirm any selections are sensible by plotting all samples in the region using the plotMethProfileSamples function (which gives the first group in blue and the second in red).

3 Using the 'lite' functions

When dealing with larger data sets such as those consisting of many replicates, or if we simply want faster processing times, we may want to reduce the amount of data stored by the M3D method. The lite functions provide such a feature.

M3D_Wrapper_lite and M3D_Para_lite operate exactly as M3D_Wrapper and M3D_Para, so that instead of entering the following:

```
MMDlistDemo <- M3D_Wrapper(rrbsDemo, overlaps)
```

We can enter

```
MMDlistDemo <- M3D_Wrapper_lite(rrbsDemo, overlaps)
```

The difference is in how the results are stored and returned. In the lite functions, instead of retruning the full MMD and the coverage only MMD, for the user to subtract, the function automatically subtracts the coverage only MMD from the full MMD and returns the difference.

Another memory and time saving step is that, since the mean of the inter group M3D statistics is used to calculate the p-values, the lite functions calculate this mean and store it in the first column. Thus in an m by n replicate comparison, we save $m * n - 1$ columns of data.

Clearly the new structure will not work with the original pvals function. A pvals_lite function has been created to cope with the new output. The workflow has therefore changed from:

```
MMDlistDemo <- M3D_Wrapper(rrbsDemo, overlaps)
M3Dstat <- MMDlistDemo$Full-MMDlistDemo$Coverage
PDemo <- pvals(rrbsDemo, CpGsDemo, M3Dstat,
    group1, group2, smaller=FALSE, comparison='allReps', method='empirical', closePara=0.005)
```

Where group1 and group2 have already been defined, to:

```
MMDlistDemo <- M3D_Wrapper_lite(rrbsDemo, overlaps)
PDemo <- pvals_lite(rrbsDemo, CpGsDemo, M3Dstat,
    group1, group2, smaller=FALSE, comparison='allReps', method='empirical', closePara=0.005)
```

The two differences are that we no longer subtract the two composite MMD values, and we use '_lite' at the end of each function.

4 Using ENCODE data

This section will demonstrate how to load ENCODE RRBS data for use with the package. Data is available for download from: http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeHaibMethylRrbs/

For this package we have used the H1-ESC and K562 cell lines, each of which have two replicates. We therefore download the files named: 'wgEncodeHaibMethylRrbsH1hescHaibSitesRep1.bed.gz', 'wgEncodeHaibMethylRrbsH1hescHaibSitesRep2.bed.gz', 'wgEncodeHaibMethylRrbsK562HaibSitesRep1.bed.gz' and 'wgEncodeHaibMethylRrbsK562HaibSitesRep2.bed.gz'.

To load in the data, change directory to the location of the 4 files named above and use the readBedFiles function provided in the package, with bed_type set to 'Encode'. This function is a adaption of BiSeq's readBismark file to handle a variety of bed file types. Please get in touch if you have a bed file that doesn't fit with any of the options of the package.

```
# change working directory to the location of the files
group <- factor(c('H1-hESC','H1-hESC','K562','K562'))
samples <- c('H1-hESC1','H1-hESC2','K562-1','K562-2')
colData <- DataFrame(group,row.names= samples)
files <- c('wgEncodeHaibMethylRrbsH1hescHaibSitesRep1.bed.gz',
           'wgEncodeHaibMethylRrbsH1hescHaibSitesRep2.bed.gz',
           'wgEncodeHaibMethylRrbsK562HaibSitesRep1.bed.gz',
           'wgEncodeHaibMethylRrbsK562HaibSitesRep2.bed.gz')
rrbs <- readBedFiles(files,colData, bed_type = ENCODE)
```

We then generate the GRanges file of regions as in the BiSeq package (if we are not using a list of regions of interest).

```
# Create the CpGs
rrbs.CpGs <- clusterSites(object=rrbs,groups=unique(colData(rrbs)$group),
                          perc.samples = 3/4, min.sites = 20, max.dist=100)
CpGs <- clusterSitesToGR(rrbs.CpGs)
```

In this example, we cut out regions with a total coverage of less than 100 in each sample, which is performed as follows:

```
inds <- vector()
overlaps <- findOverlaps(CpGs,rowRanges(rrbs.CpGs))
for (i in 1:length(CpGs)){
  covs <- colSums(totalReads(rrbs.CpGs)[subjectHits(
        overlaps[queryHits(overlaps)==i]),])
  if (!any(covs<=100)){
    inds <- c(inds,i)
  }
}
CpGs <- CpGs[inds]
rm(inds)
```

Next, to create the toy dataset here, we took only the first 1000 regions. It is important to update the 'overlaps' object if you do this, so that it details the overlaps between the right objects.

```
# reduce the rrbs to only the cytosine sites within regions
rrbs <- rrbs.CpGs
CpGs <- CpGs[1:1000] # first 1000 regions
overlaps <- findOverlaps(CpGs,rowRanges(rrbs))
rrbs <- rrbs[subjectHits(overlaps)]
overlaps <- findOverlaps(CpGs,rowRanges(rrbs))
```

Following these steps produces the toy data in this package.

5 Acknowledgements

This package was developed at the University of Edinburgh in the School of Informatics, with support from Guido Sanguinetti and Gabriele Schweikert.

The work was supported by grants EP/F500385/1 and BB/F529254/1 from the UK Engineering and Physical Sciences Research Council, UK Biotechnology and Biological Sciences Research Council, and the UK Medical Research Council.

6 sessionInfo()

This vignette was built using:

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.2 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so
##
## 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] parallel  stats4    stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] M3D_1.10.0                 BiSeq_1.16.0               Formula_1.2-1
##  [4] SummarizedExperiment_1.6.0 DelayedArray_0.2.0         matrixStats_0.52.2
##  [7] Biobase_2.36.0             GenomicRanges_1.28.0       GenomeInfoDb_1.12.0
## [10] IRanges_2.10.0             S4Vectors_0.14.0           BiocGenerics_0.22.0
##
## loaded via a namespace (and not attached):
##  [1] zoo_1.8-0                modeltools_0.2-21        splines_3.4.0
##  [4] lattice_0.20-35          htmltools_0.3.5          rtracklayer_1.36.0
##  [7] yaml_2.1.14              XML_3.98-1.6             survival_2.41-3
## [10] DBI_0.6-1                BiocParallel_1.10.0      GenomeInfoDbData_0.99.0
## [13] stringr_1.2.0            zlibbioc_1.22.0          Biostrings_2.44.0
## [16] globaltest_5.30.0        evaluate_0.10            memoise_1.1.0
## [19] knitr_1.15.1             lmtest_0.9-35            flexmix_2.3-13
## [22] AnnotationDbi_1.38.0     highr_0.6                Rcpp_0.12.10
## [25] xtable_1.8-2             backports_1.0.5          annotate_1.54.0
## [28] XVector_0.16.0           Rsamtools_1.28.0         BiocStyle_2.4.0
## [31] digest_0.6.12            stringi_1.1.5            grid_3.4.0
## [34] rprojroot_1.2            tools_3.4.0              bitops_1.0-6
## [37] sandwich_2.3-4           magrittr_1.5             RCurl_1.95-4.8
## [40] RSQLite_1.1-2            lokern_1.1-8             Matrix_1.2-9
## [43] betareg_3.1-0            rmarkdown_1.4            sfsmisc_1.1-0
## [46] GenomicAlignments_1.12.0 nnet_7.3-12              compiler_3.4.0
```

References

[1] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alexander J Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19:513, 2007.

[2] Gabriele Schweikert, Botond Cseke, Thomas Clouaire, Adrian Bird, and Guido Sanguinetti. MMDiff: quantitative testing for shape changes in ChIP-Seq data sets. *BMC genomics*, 14(1):826, 2013.

[3] Tom R Mayo, Gabriele Schweikert, and Guido Sanguinetti. M3d: a kernel-based test for spatially correlated changes in methylation profiles. *Bioinformatics*, page btu749, 2014.

[4] ENCODE Project Consortium and others. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, 2012.