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

Tom Mayo

Modified: 12 Aug, 2014. Compiled: May 15, 2016

Contents

1 Introduction

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) [**?**] to perform the test, adjusting this measure to account for changes in the coverage profile between the testing groups.

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 [**?**].

2 Analysis Pipeline

2.1 Reading in Data

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

For illustrative purposes, we have included data from two RRBS datasets via the ENCODE consortium [**?**], 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)
```

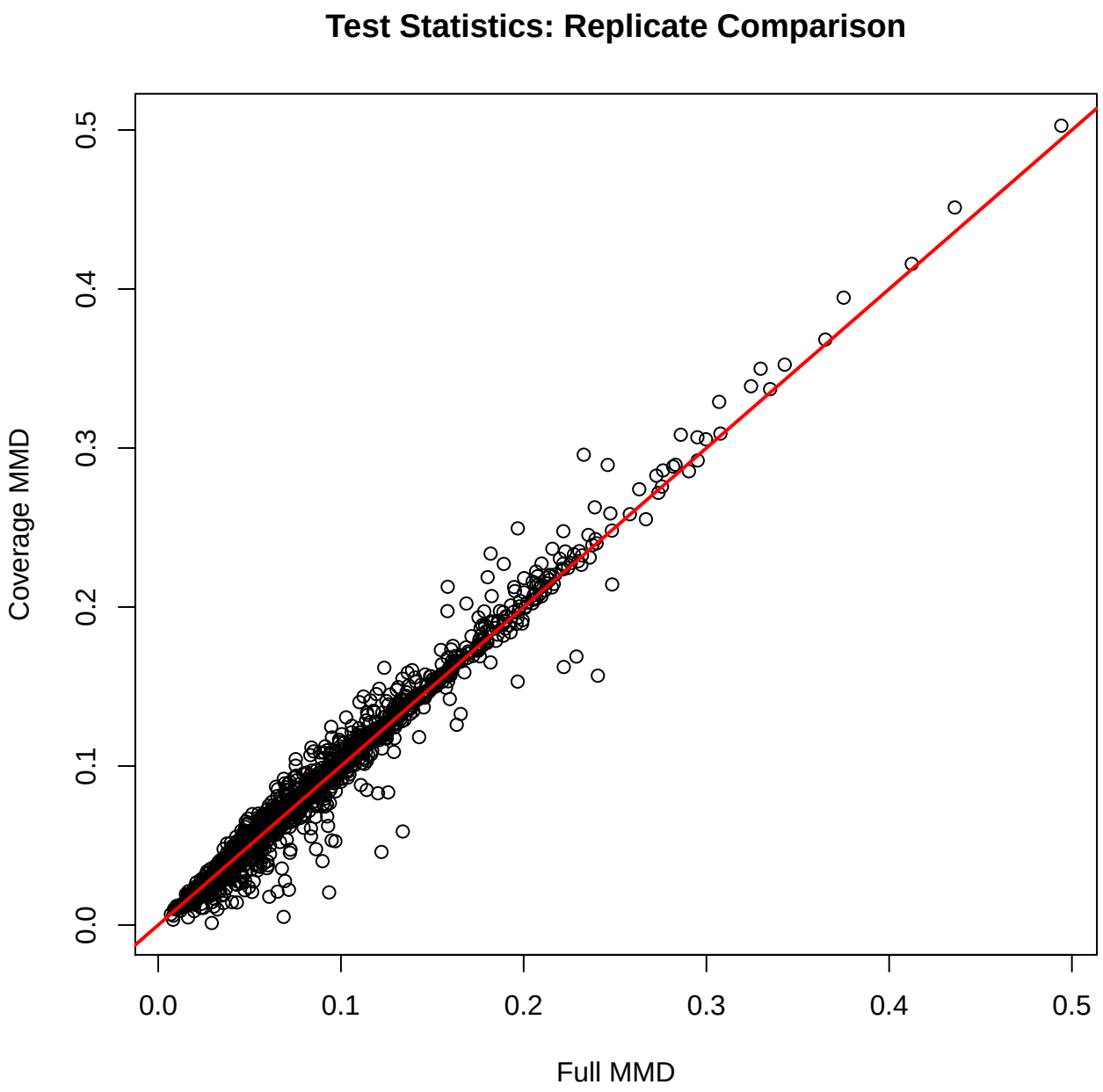
```
##      H1-hESC1  vs  H1-hESC2 H1-hESC1  vs  K562-1 H1-hESC1  vs  K562-2
## [1,]             0.13245172            0.3979882            0.3998015
## [2,]             0.21273851            0.1709892            0.1664227
## [3,]             0.11562054            0.1408935            0.1493031
## [4,]             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
```

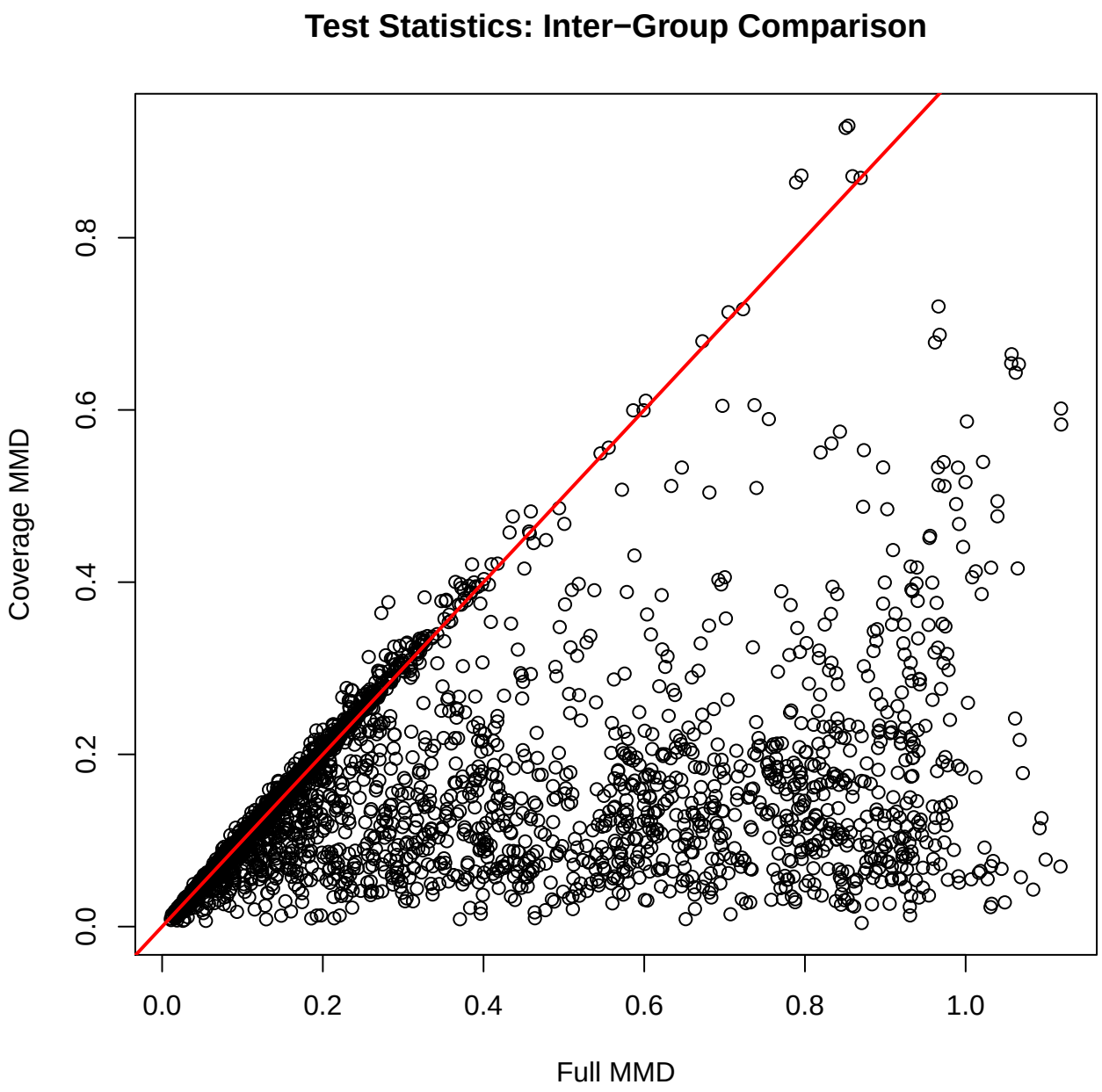
In the matrices we have stored, we can see from the column names that the columns pertaining to inter-replicate values are 1 and 6, while 2 to 5 detail inter-group comparisons, since there are . We can plot the values for replicates as follows:

```
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.

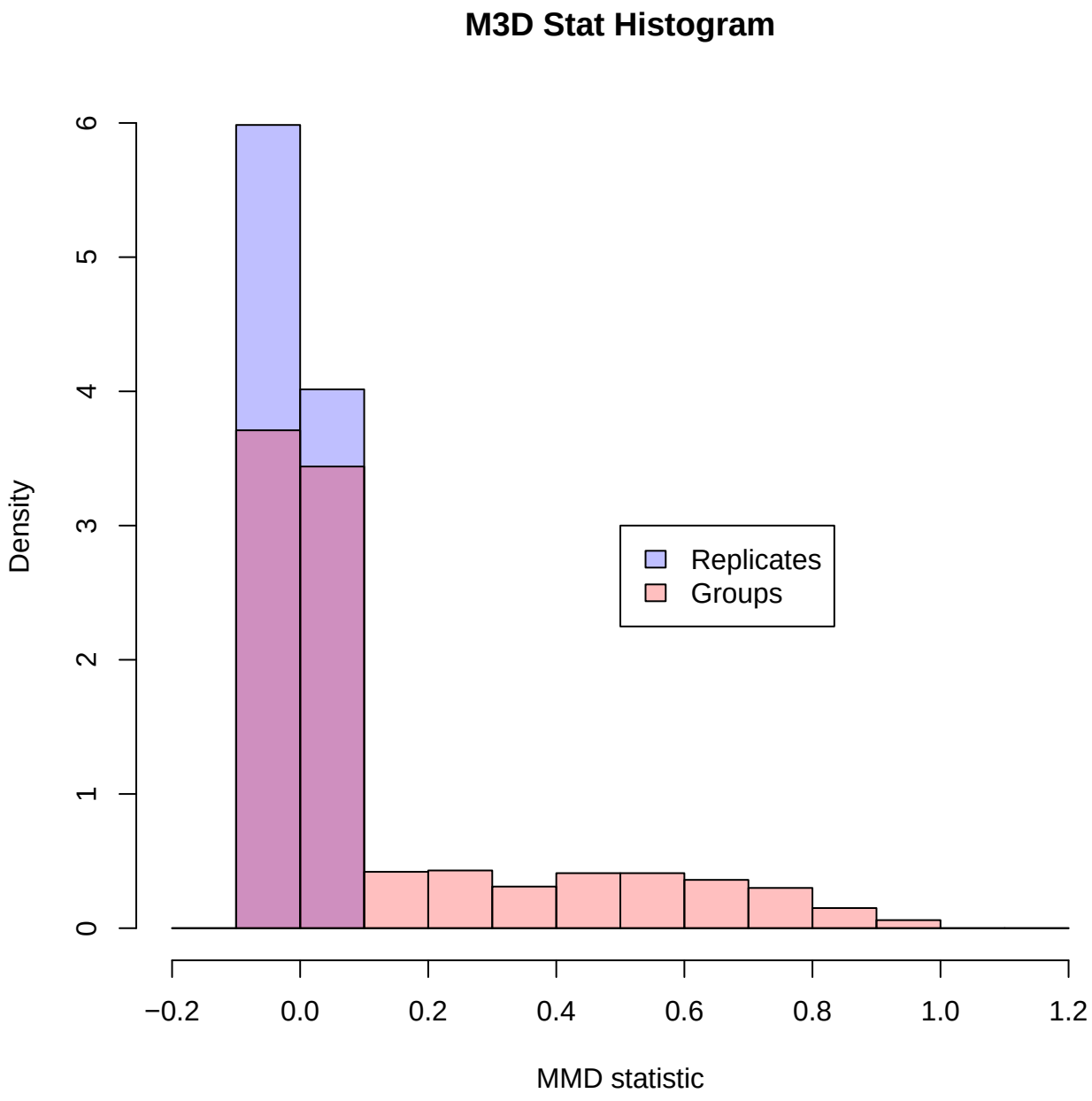
```
groupCols <- 2:5
plot(as.vector(MMDlistDemo$Full[,groupCols]),
    as.vector(MMDlistDemo$Coverage[,groupCols]),
    xlab='Full MMD',ylab='Coverage MMD')
title('Test Statistics: Inter-Group Comparison')
abline(a=0,b=1,col='red',lwd=2)
```



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)
```

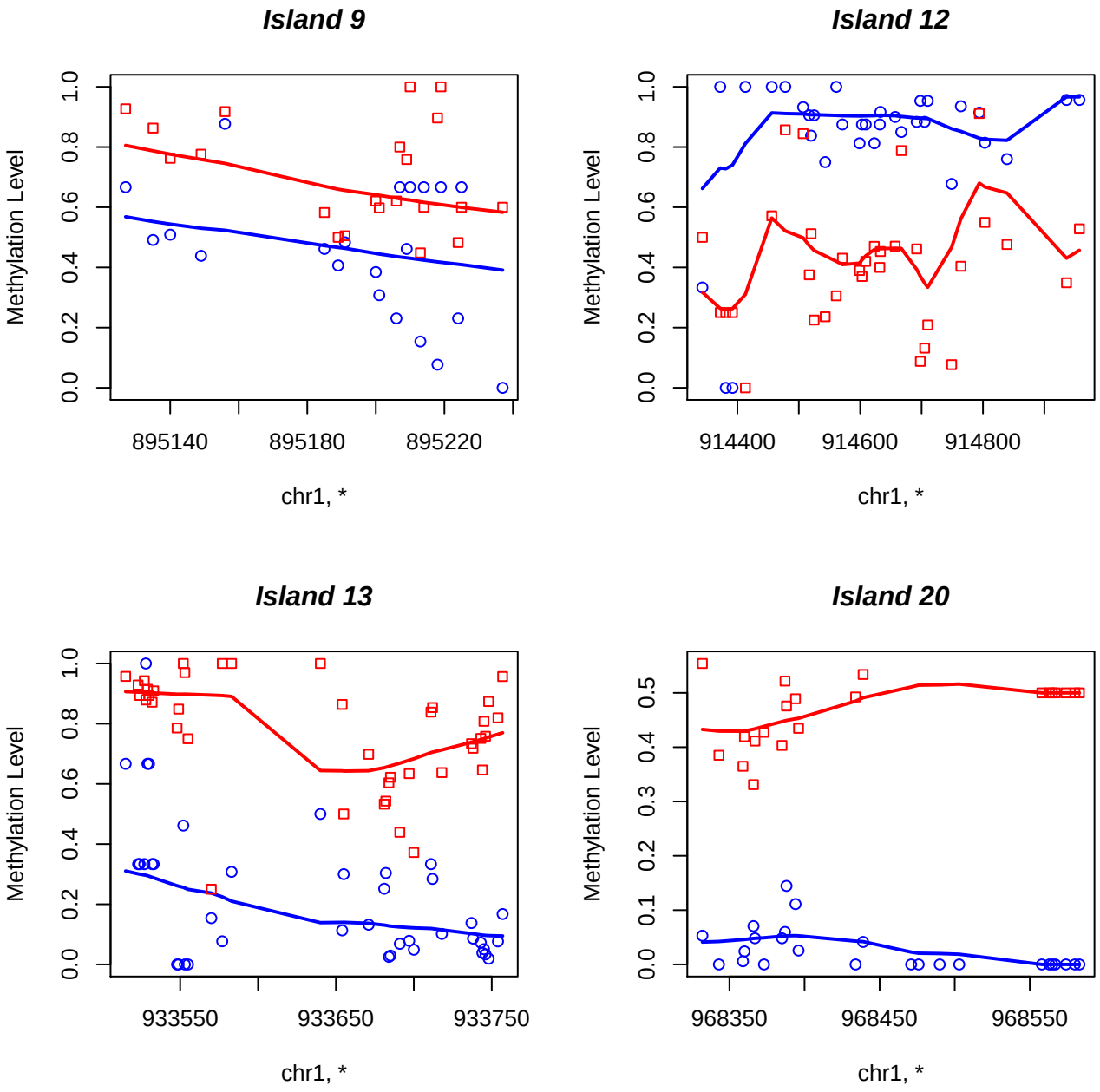
```
## [1]  9 12 13 20 21 22
```

```
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)
}
```



3 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 readENCODEdata function

provided in the package. This function is a adaption of BiSeq's readBismark file to handle this format. Unfortunately, RRBS data files frequently have differing layouts, but this function is readily adaptable to new layouts.

```
# 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 <- readENCODEdata(files,colData)
```

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.

4 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.

5 sessionInfo()

This vignette was built using:

```
sessionInfo()
```

```
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] M3D_1.6.2                  BiSeq_1.12.0               Formula_1.2-1
##  [4] SummarizedExperiment_1.2.2 Biobase_2.32.0             GenomicRanges_1.24.0
##  [7] GenomeInfoDb_1.8.2         IRanges_2.6.0              S4Vectors_0.10.0
## [10] BiocGenerics_0.18.0
##
## loaded via a namespace (and not attached):
##  [1] highr_0.6               formatR_1.4             XVector_0.12.0
##  [4] bitops_1.0-6            tools_3.3.0             zlibbioc_1.18.0
##  [7] RSQLite_1.0.0           annotate_1.50.0         evaluate_0.9
## [10] lattice_0.20-33         Matrix_1.2-6            DBI_0.4-1
## [13] rtracklayer_1.32.0      stringr_1.0.0           knitr_1.13
## [16] Biostrings_2.40.0       lmtest_0.9-34           grid_3.3.0
## [19] nnet_7.3-12             globaltest_5.26.0       flexmix_2.3-13
## [22] AnnotationDbi_1.34.2    survival_2.39-4         XML_3.98-1.4
## [25] BiocParallel_1.6.2      lokern_1.1-6            magrittr_1.5
## [28] splines_3.3.0           Rsamtools_1.24.0        modeltools_0.2-21
## [31] sfsmisc_1.1-0           GenomicAlignments_1.8.0 BiocStyle_2.0.2
## [34] xtable_1.8-2            betareg_3.0-5           sandwich_2.3-4
## [37] stringi_1.0-1           RCurl_1.95-4.8          zoo_1.7-13
```