Biseq: A package for analyzing targeted bisulfite sequencing data

Katja Hebestreit, Hans-Ulrich Klein

March 31, 2015

Contents

1 Introduction

The *BiSeq* package provides useful classes and functions to handle and analyze targeted bisulfite sequencing (BS) data such as reduced representation bisulfite sequencing (RRBS) data. In particular, it implements an algorithm to detect differentially methylated regions (DMRs), as described in detail in [**?**]. The package takes already aligned BS data from one or multiple samples. Until now, it was used for the analysis of CpG methylation of human and mouse samples only.

2 Data import and classes

2.1 Sample data

As sample data we use a small part of a recently published data set, see [**?**]. It comprises RRBS data from 10 samples of CpG sites from genomic regions on p arms of chromosome 1 and 2 covered in at least one sample. Data was obtained from 5 bone marrow probes of patients with acute promyelocytic leukemia (APL) and 5 control samples (APL in remission). RRBS data was preprocessed with the Bismark software version 0.5 [**?**].

2.2 Import of Bismark's methylation output files

Bismark [**?**] a bisulfite read mapper and methylation caller. *BiSeq* allows the import of Bismark output files.

```
> library(BiSeq)
```

`readBismark` imports the CpG context output files created by the methylation extractor of Bismark:

```
> readBismark(files, colData)
```

The argument `files` point to files created by Bismark's `methylation_extractor` and `bismark2bedGraph` (see the man page of `readBismark` for details on how to retrieve the right input files). `colData` specifies the sample names and additional phenotype information. This method returns a *BSraw* object.

2.3 The *BSraw* and *BSrel* classes

The *BiSeq* package contains the classes *BSraw* and *BSrel*, both derived from *RangedSummarizedExperiment*.

2.3.1 The BSraw class

The *BSraw* class is a container for 'raw' RRBS data. It comprises sample information together with CpG positions and numbers of reads spanning the CpG positions as well as the number of methylated reads.

A *BSraw* object consists of four slots:

1. A `list` of arbitrary content describing the overall experiment, accessible with `metadata`.

2. A *GRanges* of the positions of CpG-sites covered by BS in at least one sample, accessible with `rowRanges`.

3. A *DataFrame* of samples and the values of variables measured on those samples, accessible with `colData`.

4. An `assays` slot containing a *SimpleList* of two matrices, one containing the numbers of reads (accessible with `totalReads`) and the other the numbers of methylated reads (accessible with `methReads`).

A new *BSraw* object can also be created by hand:

```
> metadata <- list(Sequencer = "Instrument", Year = "2013")
> rowRanges <- GRanges(seqnames = "chr1",
                       ranges = IRanges(start = c(1,2,3), end = c(1,2,3)))
> colData <- DataFrame(group = c("cancer", "control"),
                         row.names = c("sample_1", "sample_2"))
> totalReads <- matrix(c(rep(10L, 3), rep(5L, 3)), ncol = 2)
> methReads <- matrix(c(rep(5L, 3), rep(5L, 3)), ncol = 2)
> BSraw(metadata = metadata,
        rowRanges = rowRanges,
        colData = colData,
        totalReads = totalReads,
        methReads = methReads)
```

Nevertheless, users will most likely create *BSraw* objects when use `readBismark` to load data.

We load and show the APL data:

```
> data(rrbs)
> rrbs

class: BSraw
dim: 10502 10
metadata(0):
assays(2): totalReads methReads
rownames(10502): 1456 1457 ... 4970981 4970982
```

```
rowData names(0):
colnames(10): APL1 APL2 ... APL11624 APL5894
colData names(1): group
```

We show the sample characteristics slot:

```
> colData(rrbs)
```

```
DataFrame with 10 rows and 1 column
            group
         <factor>
APL1           APL
APL2           APL
APL3           APL
APL7           APL
APL8           APL
APL10961   control
APL11436   control
APL11523   control
APL11624   control
APL5894    control
```

The first CpG sites on chromosome 1 which were covered:

```
> head(rowRanges(rrbs))
```

```
GRanges object with 6 ranges and 0 metadata columns:
       seqnames           ranges strand
          <Rle>        <IRanges>  <Rle>
  1456     chr1 [870425, 870425]      +
  1457     chr1 [870443, 870443]      +
  1458     chr1 [870459, 870459]      +
  1459     chr1 [870573, 870573]      +
  1460     chr1 [870584, 870584]      +
  1461     chr1 [870599, 870599]      +
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

The coverage of the first CpG sites per sample:

```
> head(totalReads(rrbs))
```

```
     APL1 APL2 APL3 APL7 APL8 APL10961 APL11436 APL11523
1456   39    6   10    0    0       48       31       65
1457   39    6   10    0    0       48       31       65
1458   39    6   10    0    0       48       27       65
1459   20   26   49   48   39       27       23       34
1460   20   26   49   48   39       27       23       34
1461   20   26   49   48   39       27       22       34
     APL11624 APL5894
1456       39      29
1457       39      29
1458       39      29
1459       29      15
1460       28      15
1461       29      15
```

The number of methylated reads of the first CpG sites per sample:

```
> head(methReads(rrbs))
```

```
     APL1 APL2 APL3 APL7 APL8 APL10961 APL11436 APL11523
1456   32    6    7    0    0       15       23       16
1457   33    6    7    0    0       18       10       19
1458   33    6   10    0    0       20       10       19
1459   13   20   34   41   32        3        8        8
1460   14   18   35   37   33        2        4        4
1461   14   16   35   40   31        5        5        5
     APL11624 APL5894
1456        7       7
1457        2       7
1458        2       3
1459        6       4
1460        3       0
1461        1       2
```

2.3.2 The BSrel class

The *BSrel* is a container for 'relative' methylation levels of BS data. It comprises sample information together with CpG positions and the relative methylation values (between 0 and 1).

A *BSrel* object consists of four slots:

1. A `list` of arbitrary content describing the overall experiment, accessible with `metadata`.
2. A *GRanges* of the positions of CpG-sites covered by BS in at least one sample, accessible with `rowRanges`.
3. A *DataFrame* of samples and the values of variables measured on those samples, accessible with `colData`.
4. An `assays` slot containing a *SimpleList* of a matrix with the relative methylation levels (between 0 and 1), accessible with `methLevel`.

A new *BSraw* object can be created by:

```
> methLevel <- matrix(c(rep(0.5, 3), rep(1, 3)), ncol = 2)
> BSrel(metadata = metadata,
        rowRanges = rowRanges,
        colData = colData,
        methLevel = methLevel)
```

We can convert a *BSraw* object to a *BSrel* object easily:

```
> rrbs.rel <- rawToRel(rrbs)
> rrbs.rel
```

```
class: BSrel
dim: 10502 10
metadata(0):
assays(1): methLevel
rownames(10502): 1456 1457 ... 4970981 4970982
rowData names(0):
colnames(10): APL1 APL2 ... APL11624 APL5894
colData names(1): group
```

The relative methylation values of the first CpG sites:

```
> head(methLevel(rrbs.rel))
```

```
          APL1      APL2      APL3      APL7      APL8
1456 0.8205128 1.0000000 0.7000000       NaN       NaN
1457 0.8461538 1.0000000 0.7000000       NaN       NaN
1458 0.8461538 1.0000000 1.0000000       NaN       NaN
1459 0.6500000 0.7692308 0.6938776 0.8541667 0.8205128
1460 0.7000000 0.6923077 0.7142857 0.7708333 0.8461538
1461 0.7000000 0.6153846 0.7142857 0.8333333 0.7948718
       APL10961  APL11436  APL11523   APL11624   APL5894
1456 0.31250000 0.7419355 0.2461538 0.17948718 0.2413793
1457 0.37500000 0.3225806 0.2923077 0.05128205 0.2413793
1458 0.41666667 0.3703704 0.2923077 0.05128205 0.1034483
1459 0.11111111 0.3478261 0.2352941 0.20689655 0.2666667
1460 0.07407407 0.1739130 0.1176471 0.10714286 0.0000000
1461 0.18518519 0.2272727 0.1470588 0.03448276 0.1333333
```

2.4 Data handling

All methods for *RangedSummarizedExperiment* objects are applicable for *BSraw* and *BSrel* objects:

```
> dim(rrbs)

[1] 10502    10

> colnames(rrbs)

 [1] "APL1"     "APL2"     "APL3"     "APL7"     "APL8"
 [6] "APL10961" "APL11436" "APL11523" "APL11624" "APL5894"
```

We can return subsets of samples or CpG sites:

```
> rrbs[,"APL2"]
> ind.chr1 <- which(seqnames(rrbs) == "chr1")
> rrbs[ind.chr1,]
```

We can also subset by overlaps with a *GRanges* object:

```
> region <- GRanges(seqnames="chr1",
                    ranges=IRanges(start = 875200,
                                   end = 875500))
```

```
> findOverlaps(rrbs, region)
> subsetByOverlaps(rrbs, region)
```

We can sort *BSraw* and *BSrel* objects into ascending order of CpG sites positions on chromosomes:

```
> sort(rrbs)
```

BSraw and *BSrel* objects can be combined and splitted:

```
> combine(rrbs[1:10,1:2], rrbs[1:1000, 3:10])
> split(rowRanges(rrbs),
      f = as.factor(as.character(seqnames(rrbs))))
```

3 Quality control

Via two very simple methods it is possible to compare the sample's coverages. `covStatistics` lists the number of CpG sites that were covered per sample together with the median of the coverage of these CpG sites. `covBoxplots` represent the coverage distributions per sample.

```
> covStatistics(rrbs)

$Covered_CpG_sites
    APL1     APL2     APL3     APL7     APL8 APL10961
    5217     4240     4276     3972     3821     5089
APL11436 APL11523 APL11624  APL5894
    5169     6922     6483     7199

$Median_coverage
    APL1     APL2     APL3     APL7     APL8 APL10961
      12        5       12       15       11       10
APL11436 APL11523 APL11624  APL5894
       6        8        4        5
```

```
> covBoxplots(rrbs, col = "cornflowerblue", las = 2)
```

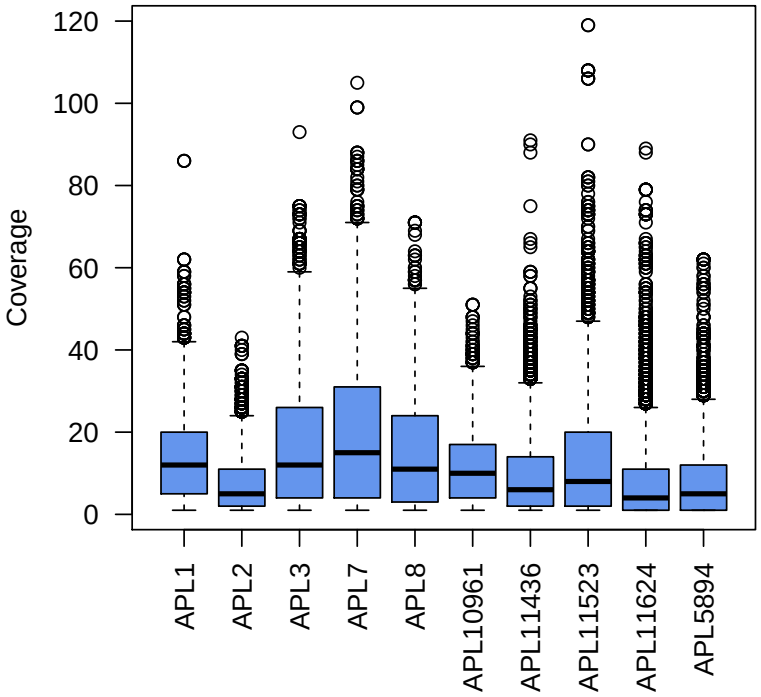

Figure 1: Sample wise coverage distributions

4 Detection of DMRs within groups of samples

The algorithm to detect differentially methylated regions (DMRs) within two groups of samples (e.g. cancer and control) is described in detail in [**?**]. To better understand this User's guide it is helpful to know the rough procedure. The DMR detection is a five-steps approach:

1. Definition of CpG clusters
2. Smooth methylation data within CpG clusters
3. Model and test group effect for each CpG site within CpG clusters
4. Apply hierarchical testing procedure:
   (a) Test CpG clusters for differential methylation and control weighted FDR on cluster
   (b) Trim rejected CpG clusters and control FDR on single CpGs
5. Define DMR boundaries

Please see [**?**] for more details.

4.1 Definition of CpG clusters

In order to smooth the methylation data we first have to detect CpG clusters (regions with a high spatial density of covered CpG sites). Within a *BSraw* object `clusterSites` searches for agglomerations of CpG sites across all samples. In a first step the data is reduced to CpG sites covered in `round(perc.samples*ncol(object))` samples (here: 4 samples), these are called 'frequently covered CpG sites'. In a second step regions are detected where not less than `min.sites` frequently covered CpG sites are sufficiantly close to each other (`max.dist`. Note, that the frequently covered CpG sites are considered to define the boundaries of the CpG clusters only. For the subsequent analysis the methylation data of all CpG sites within these clusters are used.

We perform the analysis on a subset of our data to save time:

```
> rrbs.small <- rrbs[1:1000,]
> rrbs.clust.unlim <- clusterSites(object = rrbs.small,
                                   groups = colData(rrbs)$group,
                                   perc.samples = 4/5,
                                   min.sites = 20,
                                   max.dist = 100)
```

`rrbs.clust.unlim` is a again *BSraw* object but restricted to CpG sites within CpG clusters. Each CpG site is assigned to a cluster:

```
> head(rowRanges(rrbs.clust.unlim))
```

```
GRanges object with 6 ranges and 1 metadata column:
         seqnames           ranges strand |  cluster.id
            <Rle>        <IRanges>  <Rle> | <character>
    1513     chr1 [872335, 872335]      * |      chr1_1
    1514     chr1 [872369, 872369]      * |      chr1_1
  401911     chr1 [872370, 872370]      * |      chr1_1
    1515     chr1 [872385, 872385]      * |      chr1_1
  401912     chr1 [872386, 872386]      * |      chr1_1
    1516     chr1 [872412, 872412]      * |      chr1_1
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

The underlying CpG clusters can also be converted to a *GRanges* object with the start and end positions:

```
> clusterSitesToGR(rrbs.clust.unlim)
```

```
GRanges object with 6 ranges and 1 metadata column:
      seqnames           ranges strand | cluster.id
         <Rle>        <IRanges>  <Rle> |   <factor>
  [1]     chr1 [872335, 872616]      * |     chr1_1
  [2]     chr1 [875227, 875470]      * |     chr1_2
  [3]     chr1 [875650, 876028]      * |     chr1_3
  [4]     chr1 [876807, 877458]      * |     chr1_4
  [5]     chr1 [877684, 877932]      * |     chr1_5
  [6]     chr2 [ 45843,  46937]      * |     chr2_1
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

4.2 Smooth methylation data

In the smoothing step CpG sites with high coverages get high weights. To reduce bias due to unusually high coverages we limit the coverage, e.g. to the 90% quantile:

```
> ind.cov <- totalReads(rrbs.clust.unlim) > 0
> quant <- quantile(totalReads(rrbs.clust.unlim)[ind.cov], 0.9)
> quant

90%
 32

> rrbs.clust.lim <- limitCov(rrbs.clust.unlim, maxCov = quant)
```

We then smooth the methylation values of CpG sites within the clusters with the default bandwidth `h = 80` base pairs. It is possible - and recommended - to parallelize this step by setting `mc.cores`, to 6 cores for instance, if there are 6 available.

```
> predictedMeth <- predictMeth(object = rrbs.clust.lim)
```

`predictedMeth` is a *BSrel* object with smoothed relative methylation levels for each CpG site within CpG clusters:

```
> predictedMeth

class: BSrel
dim: 344 10
metadata(0):
assays(1): methLevel
rownames(344): 1 2 ... 343 344
rowData names(1): cluster.id
colnames(10): APL1 APL2 ... APL11624 APL5894
colData names(1): group
```

The effect of the smoothing step can be shown with the `plotMeth` function:

```
> covBoxplots(rrbs.clust.lim, col = "cornflowerblue", las = 2)
```

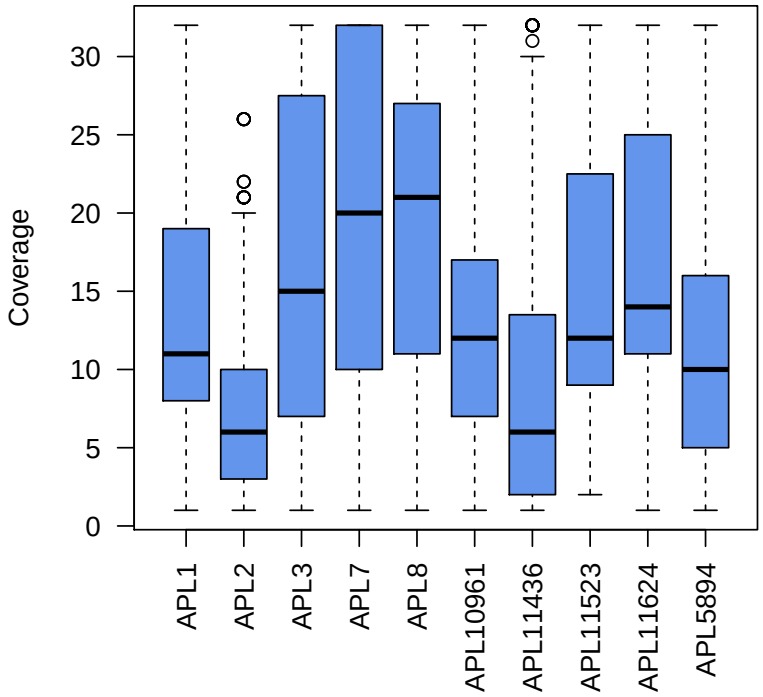

Figure 2: Sample wise coverage distributions after coverage limitation

```
> plotMeth(object.raw = rrbs[,6],
           object.rel = predictedMeth[,6],
           region = region,
           lwd.lines = 2,
           col.points = "blue",
           cex = 1.5)
```

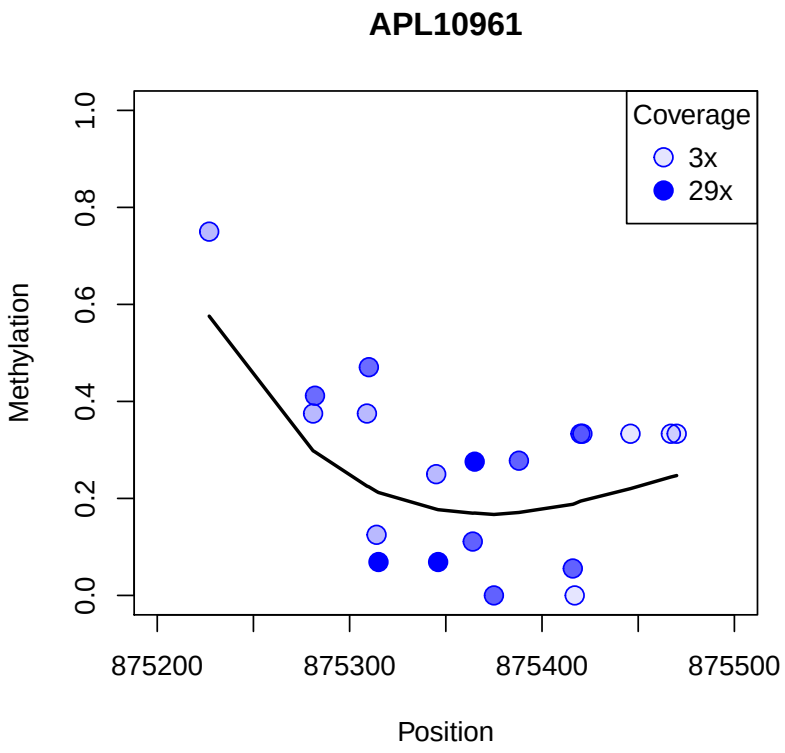

Figure 3: Raw data together with smoothed methylation levels

4.3 Model and test group effect

We observe a differential methylation between cancer and control for some CpG sites:

```
> cancer <- predictedMeth[, colData(predictedMeth)$group == "APL"]
> control <- predictedMeth[, colData(predictedMeth)$group == "control"]
> mean.cancer <- rowMeans(methLevel(cancer))
> mean.control <- rowMeans(methLevel(control))
> plot(mean.control,
      mean.cancer,
      col = "blue",
      xlab = "Methylation in controls",
      ylab = "Methylation in APLs")
```

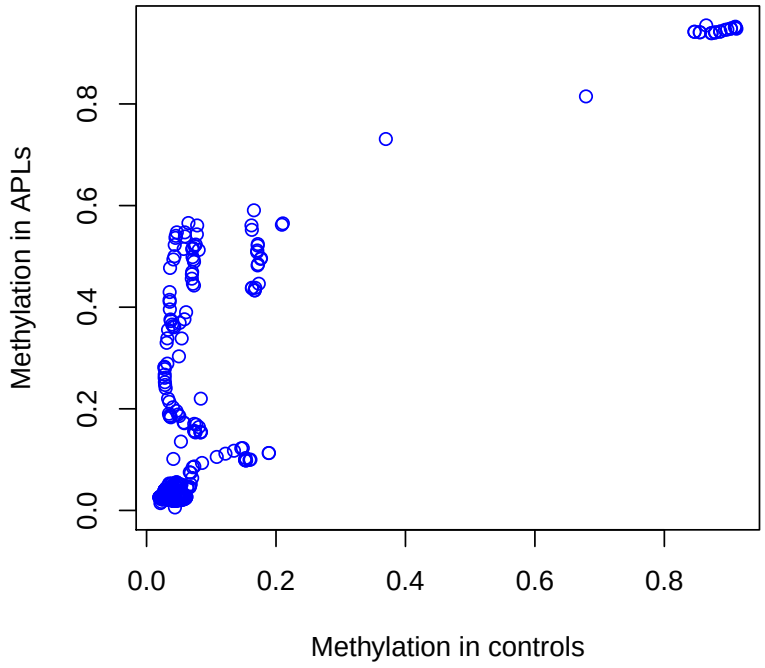

Figure 4: Smoothed methylation levels in APL and control samples

To detect the CpG sites where the DNA methylation differs between APL and control samples we model the methylation within a beta regression with the group as explanatory variable and use the Wald test to test if there is a

group effect:

```
> ## To shorten the run time set mc.cores, if possible!
> betaResults <- betaRegression(formula = ~group,
                                link = "probit",
                                object = predictedMeth,
                                type = "BR")

> ## OR:
> data(betaResults)
```

`betaResults` is a *data.frame* containing model and test information for each CpG site:

```
> head(betaResults)
```

```
        chr    pos        p.val meth.group1 meth.group2
chr1.1 chr1 872335 0.0011317652   0.9525098   0.8635983
chr1.2 chr1 872369 0.0007678027   0.9414368   0.8444060
chr1.3 chr1 872370 0.0008347451   0.9414314   0.8448725
chr1.4 chr1 872385 0.0010337477   0.9412217   0.8521862
chr1.5 chr1 872386 0.0010975571   0.9410544   0.8526863
chr1.6 chr1 872412 0.0035114839   0.9378250   0.8718024
        meth.diff   estimate std.error pseudo.R.sqrt
chr1.1 0.08891149 -0.5730618 0.1760266     0.6304051
chr1.2 0.09703074 -0.5542171 0.1647422     0.6291904
chr1.3 0.09655890 -0.5522164 0.1652843     0.6246474
chr1.4 0.08903549 -0.5192564 0.1582531     0.6108452
chr1.5 0.08836810 -0.5156622 0.1579728     0.6090794
chr1.6 0.06602261 -0.4018161 0.1376551     0.5851744
       cluster.id
chr1.1     chr1_1
chr1.2     chr1_1
chr1.3     chr1_1
chr1.4     chr1_1
chr1.5     chr1_1
chr1.6     chr1_1
```

By setting `type = "BR"` the maximum likelihood with bias reduction is called. This is especially useful, when the sample size is small, see [**?**]. The

mean of the response (methylation) is linked to a linear predictor described by `~ x1 + x2` using a link function while the precision parameter is assumed to be constant. Sometimes the variance of DNA methylation is dependent on the group factor, e.g. the methylation variance in cancer samples is often higher than in normal samples. These additional regressors can be linked to the precision parameter within the formula of type `~ x1 + x2 | y1 + y2` where the regressors in the two parts can be overlapping, see the documentation in the *betareg* package.

4.4 Test CpG clusters for differential methylation

The aim is to detect CpG clusters containing at least one differentially methylated location. To do so the P values p from the Wald tests are transformed to Z scores: $z = \Phi^{-1}(1-p)$, which are normally distributed under Null hypothesis (no group effect). As cluster test statistic a standardized Z score average is used. To estimate the standard deviation of the Z scores we have to estimate the correlation and hence the variogram of methylation between two CpG sites within a cluster. The estimation of the standard deviation requires that the distribution of the Z scores follows a standard normal distribution. However, if methylation in both groups differs for many CpG sites the density distribution of P values shows a peak near 0. To ensure that the P values are roughly uniformly distributed to get a variance of the Z scores that is Gaussian with variance 1 we recommend to estimate the variogram (and hence the correlation of Z scores) under the null hypothesis. To do so we model the beta regression again for resampled data:

```
> ## Both resampled groups should have the same number of
> ## cancer and control samples:
> predictedMethNull <- predictedMeth[,c(1:4, 6:9)]
> colData(predictedMethNull)$group.null <- rep(c(1,2), 4)
> ## To shorten the run time, please set mc.cores, if possible!
> betaResultsNull <- betaRegression(formula = ~group.null,
                                    link = "probit",
                                    object = predictedMethNull,
                                    type="BR")

> ## OR:
> data(betaResultsNull)
```

We estimate the variogram for the Z scores obtained for the resampled data:

```
> vario <- makeVariogram(betaResultsNull)
> ## OR:
> data(vario)
```

Based on the variogram plot we evaluate the sill (usually near 1) of the variogram and smooth the curve:

```
> plot(vario$variogram$v)
> vario.sm <- smoothVariogram(vario, sill = 0.9)
> lines(vario.sm$variogram[,c("h", "v.sm")],
      col = "red", lwd = 1.5)
> grid()
```

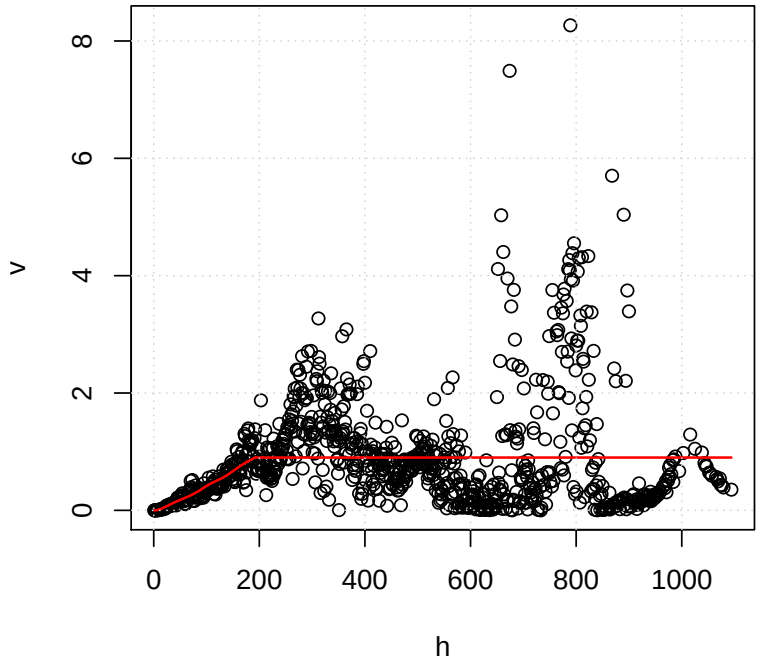

Figure 5: Estimated variogram together with the smoothed curve

The `vario.sm` object is a list of two:

```
> names(vario.sm)

[1] "variogram" "pValsList"

> head(vario.sm$variogram)

  h         v.sm
1 1 0.0000000000
2 2 0.0000000000
3 3 0.0000000000
4 4 0.0004532616
5 5 0.0019789519
6 6 0.0038612942

> head(vario.sm$pValsList[[1]])

        chr    pos     p.val meth.group1 meth.group2
chr1.1 chr1 872335 0.8567258   0.9267336   0.9191601
chr1.2 chr1 872369 0.7910594   0.9180561   0.9056264
chr1.3 chr1 872370 0.7855283   0.9187319   0.9060236
chr1.4 chr1 872385 0.7462536   0.9225722   0.9080750
chr1.5 chr1 872386 0.7462667   0.9225955   0.9081423
chr1.6 chr1 872412 0.8575813   0.9165966   0.9093737
         meth.diff    estimate std.error pseudo.R.sqrt
chr1.1 0.007573505 -0.05244358 0.2904759   0.006696118
chr1.2 0.012429670 -0.07781982 0.2937315   0.013223677
chr1.3 0.012708263 -0.07993265 0.2937384   0.013863179
chr1.4 0.014497213 -0.09359431 0.2892434   0.020416968
chr1.5 0.014453228 -0.09334700 0.2884945   0.020484807
chr1.6 0.007222875 -0.04562887 0.2542651   0.006457349
       cluster.id    z.score pos.new
chr1.1     chr1_1 -1.0657239       1
chr1.2     chr1_1 -0.8101026      35
chr1.3     chr1_1 -0.7910010      36
chr1.4     chr1_1 -0.6627467      51
chr1.5     chr1_1 -0.6627876      52
chr1.6     chr1_1 -1.0695155      78
```

We replace the `pValsList` object (which consists of the test results of the resampled data) by the test results of interest (for group effect):

```
> ## auxiliary object to get the pValsList for the test
> ## results of interest:
> vario.aux <- makeVariogram(betaResults, make.variogram=FALSE)
> vario.sm$pValsList <- vario.aux$pValsList
> head(vario.sm$pValsList[[1]])
```

```
        chr    pos        p.val meth.group1 meth.group2
chr1.1 chr1 872335 0.0011317652   0.9525098   0.8635983
chr1.2 chr1 872369 0.0007678027   0.9414368   0.8444060
chr1.3 chr1 872370 0.0008347451   0.9414314   0.8448725
chr1.4 chr1 872385 0.0010337477   0.9412217   0.8521862
chr1.5 chr1 872386 0.0010975571   0.9410544   0.8526863
chr1.6 chr1 872412 0.0035114839   0.9378250   0.8718024
        meth.diff   estimate std.error pseudo.R.sqrt
chr1.1 0.08891149 -0.5730618 0.1760266     0.6304051
chr1.2 0.09703074 -0.5542171 0.1647422     0.6291904
chr1.3 0.09655890 -0.5522164 0.1652843     0.6246474
chr1.4 0.08903549 -0.5192564 0.1582531     0.6108452
chr1.5 0.08836810 -0.5156622 0.1579728     0.6090794
chr1.6 0.06602261 -0.4018161 0.1376551     0.5851744
       cluster.id  z.score pos.new
chr1.1     chr1_1 3.053282       1
chr1.2     chr1_1 3.167869      35
chr1.3     chr1_1 3.143485      36
chr1.4     chr1_1 3.080361      51
chr1.5     chr1_1 3.062480      52
chr1.6     chr1_1 2.695753      78
```

`vario.sm` now contains the smoothed variogram under the Null hypothesis together with the P values (and Z scores) from the Wald test, that the group has no effect on methylation. The correlation of the Z scores between two locations in a cluster can now be estimated:

```
> locCor <- estLocCor(vario.sm)
```

We test each CpG cluster for the presence of at least one differentially methylated location at q what can be interpreted as the size-weighted FDR on clusters:

```
> clusters.rej <- testClusters(locCor,
                               FDR.cluster = 0.1)

3 CpG clusters rejected.

> clusters.rej$clusters.reject

GRanges object with 3 ranges and 2 metadata columns:
      seqnames           ranges strand |           X
         <Rle>        <IRanges>  <Rle> | <character>
  [1]     chr1 [872335, 872616]      * |      chr1_1
  [2]     chr1 [875227, 875470]      * |      chr1_2
  [3]     chr2 [ 45843,  46937]      * |      chr2_1
          p.value
         <numeric>
  [1] 6.867163e-03
  [2] 4.295184e-07
  [3] 1.881082e-09
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

4.5 Trim significant CpG clusters

We then trim the rejected CpG clusters that is to remove the not differentially methylated CpG sites at q_1 what can be interpreted as the location-wise FDR:

```
> clusters.trimmed <- trimClusters(clusters.rej,
                                   FDR.loc = 0.05)
> head(clusters.trimmed)

                chr    pos        p.val meth.group1
chr1_1.chr1.1  chr1 872335 0.0011317652   0.9525098
chr1_1.chr1.2  chr1 872369 0.0007678027   0.9414368
chr1_1.chr1.3  chr1 872370 0.0008347451   0.9414314
chr1_1.chr1.4  chr1 872385 0.0010337477   0.9412217
chr1_1.chr1.5  chr1 872386 0.0010975571   0.9410544
chr1_2.chr1.21 chr1 875227 0.0003916052   0.7175829
               meth.group2  meth.diff   estimate std.error
chr1_1.chr1.1    0.8635983 0.08891149 -0.5730618 0.1760266
```

```
chr1_1.chr1.2    0.8444060 0.09703074 -0.5542171 0.1647422
chr1_1.chr1.3    0.8448725 0.09655890 -0.5522164 0.1652843
chr1_1.chr1.4    0.8521862 0.08903549 -0.5192564 0.1582531
chr1_1.chr1.5    0.8526863 0.08836810 -0.5156622 0.1579728
chr1_2.chr1.21   0.3648251 0.35275781 -0.9212669 0.2598282
                pseudo.R.sqrt cluster.id  z.score pos.new
chr1_1.chr1.1       0.6304051     chr1_1 3.053282       1
chr1_1.chr1.2       0.6291904     chr1_1 3.167869      35
chr1_1.chr1.3       0.6246474     chr1_1 3.143485      36
chr1_1.chr1.4       0.6108452     chr1_1 3.080361      51
chr1_1.chr1.5       0.6090794     chr1_1 3.062480      52
chr1_2.chr1.21      0.6818046     chr1_2 3.358661       1
                       p.li
chr1_1.chr1.1  0.019767638
chr1_1.chr1.2  0.019953150
chr1_1.chr1.3  0.021679300
chr1_1.chr1.4  0.028817819
chr1_1.chr1.5  0.030594942
chr1_2.chr1.21 0.004852623
```

`clusters.trimmed` is a *data.frame* object containing all differentially methylated CpG sites. The `p.li` column contains the P values estimated in the cluster trimming step, see [**?**].

4.6 Definition of DMR boundaries

We can now define the boundaries of DMRs as rejected CpG sites within which rejected CpG sites solely are located. Within the DMRs the distance between neighbored rejected CpG sites should not exceed `max.dist` base pairs (usually the same as for `max.dist` in `clusterSites`), otherwise, the DMR is splitted. DMRs are also splitted if the methylation difference switches from positive to negative, or vice versa, if `diff.dir = TRUE`. That way we ensure that within a DMR all CpG sites are hypermethylated, and hypomethylated respectively.

```
> DMRs <- findDMRs(clusters.trimmed,
                   max.dist = 100,
                   diff.dir = TRUE)
> DMRs
```

```
GRanges object with 4 ranges and 4 metadata columns:
      seqnames           ranges strand |
         <Rle>        <IRanges>  <Rle> |
  [1]     chr1 [872335, 872386]      * |
  [2]     chr1 [875227, 875470]      * |
  [3]     chr2 [ 46126,  46718]      * |
  [4]     chr2 [ 46915,  46937]      * |
                 median.p median.meth.group1
                <numeric>          <numeric>
  [1]   0.0010337476937105   0.94143135870377
  [2] 6.67719258580069e-06  0.502443544206678
  [3]  3.9604046820675e-05  0.438629367812053
  [4]   0.0148459621327497   0.13636374336524
      median.meth.group2   median.meth.diff
                <numeric>          <numeric>
  [1]  0.852186207117906 0.0890354930388882
  [2]  0.182082007625853  0.319474180010203
  [3] 0.0776818250851119  0.355363178661131
  [4] 0.0369197526986237  0.099443990666616
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

5 Detection of DMRs between two samples

If there are two samples only to be compared we can use the `compareTwoSamples` function which determines the differences per CpG site and aggregates the sites surpassing the minimum difference `minDiff`:

```
> DMRs.2 <- compareTwoSamples(object = predictedMeth,
                              sample1 = "APL1",
                              sample2 = "APL10961",
                              minDiff = 0.3,
                              max.dist = 100)
```

Some of the DMRs detected within these two samples overlap with the group-wise DMRs:

```
> sum(overlapsAny(DMRs.2,DMRs))

[1] 1
```

6 Testing of predefined genomic regions

There are sometimes biological and/or technical reasons for testing predefined genomic regions for differential methylation. For example, one might want to test for methylation differences in known CpG islands or promototer regions. In this scenario it is not of interest which exact CpG sites are differentially methylated. The methods of choice for testing genomic regions are BiSeq and the global test (which we also implemented in this package) [**?**]. Here, we want to give a short guidance on how to use the *BiSeq* package in order to test predefined genomic regions using BiSeq and the global test.

6.1 Testing predefined genomic regions using the BiSeq method

Testing predefined genomic regions using the BiSeq method is mostly equivalent to the procedure shown above for DMR detection. Instead of the definition of CpG clusters (which we described in section 4.1) we will just add the region information to the CpG sites in the *BSraw* object. In this example, we assume that we want to test gene promoters for differential methylation.

We filter out all CpG sites of the *BSraw* object that do not fall into any promoter region. The remaining CpG sites get the promoter identifier in a column named `cluster.id`:

```
> data(promoters)
> data(rrbs)
> rrbs.red <- subsetByOverlaps(rrbs, promoters)
> ov <- findOverlaps(rrbs.red, promoters)
> rowRanges(rrbs.red)$cluster.id[queryHits(ov)] <- promoters$acc_no[subjectHits(
> head(rowRanges(rrbs.red))

GRanges object with 6 ranges and 1 metadata column:
          seqnames         ranges strand |   cluster.id
             <Rle>      <IRanges>  <Rle> |  <character>
  4636436     chr2 [46089, 46089]      + | NM_001077710
  4636437     chr2 [46095, 46095]      + | NM_001077710
  4636438     chr2 [46104, 46104]      + | NM_001077710
  4636439     chr2 [46111, 46111]      + | NM_001077710
  4636440     chr2 [46113, 46113]      + | NM_001077710
```

```
  4636441     chr2 [46126, 46126]      + | NM_001077710
  -------
  seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

From here on we can use the same pipeline as usual: Smoothing and modeling (sections 4.2 and 4.3) of the DNA methylation data and subsequent testing of the regions (section 4.4).

6.2 Testing predefined genomic regions using the Global Test

Goeman et al. [**?**] proposed a score test for testing high dimensional alternatives. The approach is implemented in the R/Bioconductor package *globaltest*. The *BiSeq* package offers a wrapper function to apply Goeman's Global Test directly to RRBS data given as a *BSrel* object:

```
> data(promoters)
> data(rrbs)
> rrbs <- rawToRel(rrbs)
> promoters <- promoters[overlapsAny(promoters, rrbs)]
> gt <- globalTest(group~1,
                   rrbs,
                   subsets = promoters)
> head(gt)

  p-value Statistic Expected Std.dev #Cov
1 0.00975  2.50e+01     11.1    4.89  206
2 0.05381  2.05e+01     11.1    5.61   67
3 0.05381  2.05e+01     11.1    5.61   67
4 1.00000  1.85e-31     11.1   14.81    8
5 1.00000  1.85e-31     11.1   14.81    8
6 0.06744  1.84e+01     11.1    4.88  149
```

If the parameter `subsets` is not given, the null hypothesis is that no CpG site (rather than region) is associated with the given response variable.

7 Further data processing

The `plotMethMap` function is helpful to evaluate DMRs graphically. Via `zlim = c(0,1)` that is passed to the `heatmap` function we ensure that green stands for a relative methalytion of 0 and red stands for a relative methylation of 1:

```
> rowCols <- c("magenta", "blue")[as.numeric(colData(predictedMeth)$group)]
> plotMethMap(predictedMeth,
              region = DMRs[3],
              groups = colData(predictedMeth)$group,
              intervals = FALSE,
              zlim = c(0,1),
              RowSideColors = rowCols,
              labCol = "", margins = c(0, 6))
```

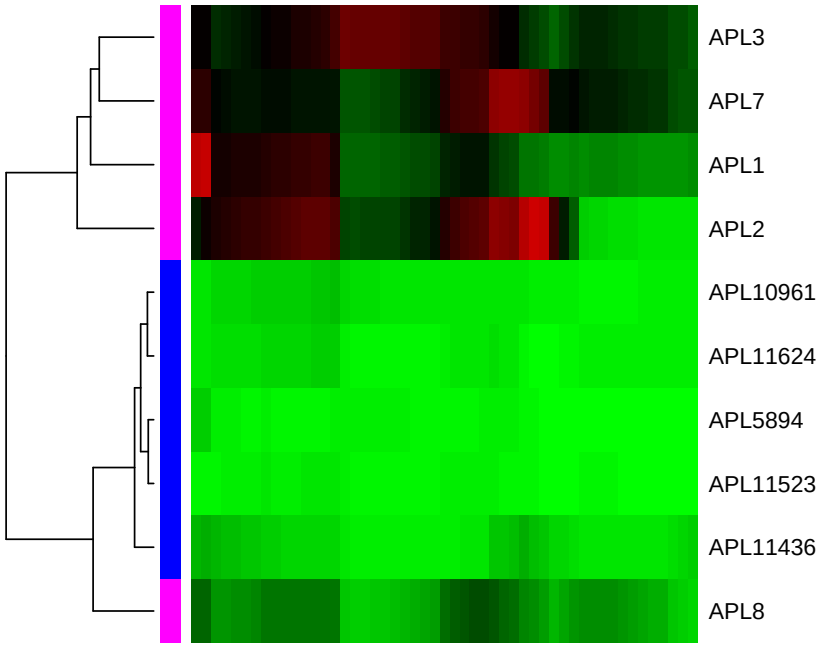

Figure 6: Methylation map of smoothed methylation data within a detected DMR together with hierarchical clustering of the samples

To represent the smoothed methylation curves we can use the `plotSmooth-`

`Meth` function:

```
> plotSmoothMeth(object.rel = predictedMeth,
                 region = DMRs[3],
                 groups = colData(predictedMeth)$group,
                 group.average = FALSE,
                 col = c("magenta", "blue"),
                 lwd = 1.5)
> legend("topright",
        legend=levels(colData(predictedMeth)$group),
        col=c("magenta", "blue"),
        lty=1, lwd = 1.5)
```

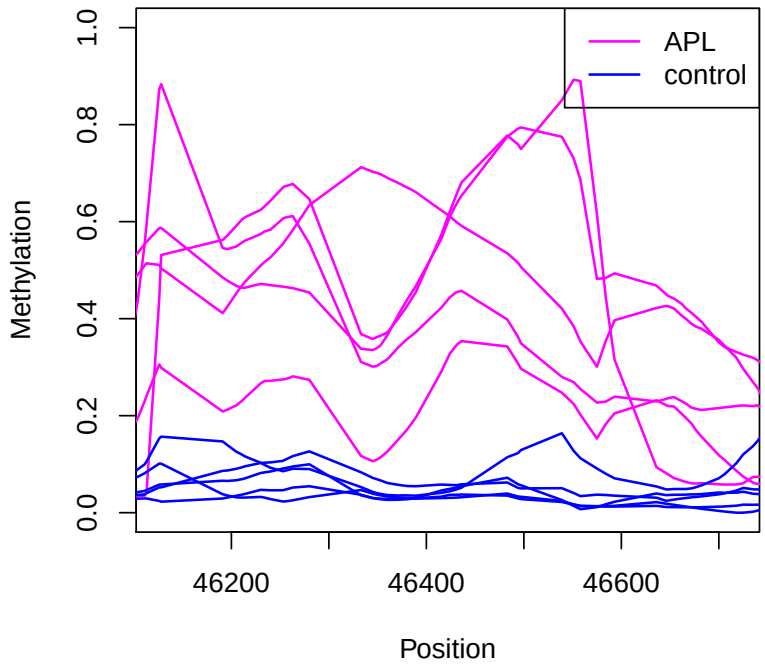

Figure 7: The smoothed methylation curves for all samples within a detected DMR

We can annotate the detected DMRs by means of a *GRanges* object, e.g. a list of promoter regions. In case of an overlapping of both *GRanges* objects the DMR is marked as `TRUE`, or with the respective identifier in the promoter list:

```
> data(promoters)
> head(promoters)

GRanges object with 6 ranges and 1 metadata column:
      seqnames               ranges strand |       acc_no
         <Rle>            <IRanges>  <Rle> |  <character>
  [1]     chr1 [66998824, 67000324]      * |    NM_032291
  [2]     chr1 [ 8383389,  8384889]      * | NM_001080397
  [3]     chr1 [16766166, 16767666]      * | NM_001145277
  [4]     chr1 [16766166, 16767666]      * | NM_001145278
  [5]     chr1 [16766166, 16767666]      * |    NM_018090
  [6]     chr1 [50489126, 50490626]      * |    NM_032785
  -------
  seqinfo: 24 sequences from an unspecified genome; no seqlengths

> DMRs.anno <- annotateGRanges(object = DMRs,
                               regions = promoters,
                               name = 'Promoter',
                               regionInfo = 'acc_no')
> DMRs.anno

GRanges object with 4 ranges and 5 metadata columns:
      seqnames           ranges strand |     median.p
         <Rle>        <IRanges>  <Rle> |    <numeric>
  [1]     chr1 [872335, 872386]      * | 1.033748e-03
  [2]     chr1 [875227, 875470]      * | 6.677193e-06
  [3]     chr2 [ 46126,  46718]      * | 3.960405e-05
  [4]     chr2 [ 46915,  46937]      * | 1.484596e-02
      median.meth.group1 median.meth.group2
                <numeric>          <numeric>
  [1]          0.9414314         0.85218621
  [2]          0.5024435         0.18208201
  [3]          0.4386294         0.07768183
  [4]          0.1363637         0.03691975
```

```
    median.meth.diff     Promoter
           <numeric>  <character>
[1]       0.08903549         <NA>
[2]       0.31947418         <NA>
[3]       0.35536318 NM_001077710
[4]       0.09944399 NM_001077710
-------
seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

`plotBindingSites` plots the average methylation around given genomic regions, e.g. protein binding sites. Here, we compare the methyation in and around promoter regions between APL and controls:

```
> plotBindingSites(object = rrbs,
                   regions = promoters,
                   width = 4000,
                   group = colData(rrbs)$group,
                   col = c("magenta", "blue"),
                   lwd = 1.5)
> legend("top",
        legend=levels(colData(rrbs)$group),
        col=c("magenta", "blue"),
        lty=1, lwd = 1.5)
```

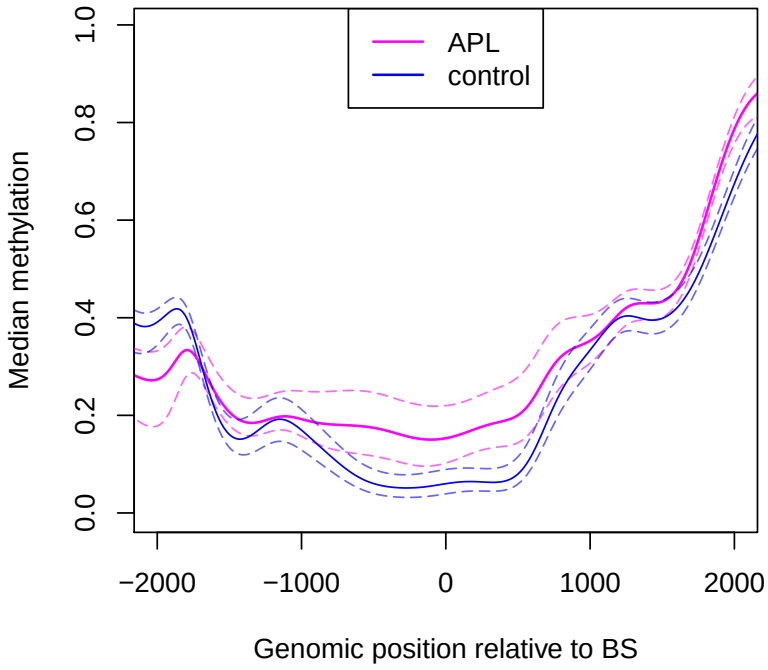

Figure 8: Methylation around 1,000 promoters; Position 0 refers to the centers of the promoters

The raw and relative methylation data can also be viewed in the Integra-

tive Genomics Viewer (IGV; freely available for download from www.broadinstitute.org/igv) [**?**]. To do so we first write the methylation information of each sample within the *BSraw* or *BSrel* object to a bed file:

```
> track.names <- paste(colData(rrbs)$group,
                       "_",
                       gsub("APL", "", colnames(rrbs)),
                       sep="")
> writeBED(object = rrbs,
           name = track.names,
           file = paste(colnames(rrbs), ".bed", sep = ""))
> writeBED(object = predictedMeth,
           name = track.names,
           file = paste(colnames(predictedMeth), ".bed", sep = ""))
```

We can load the bed files of the raw data in the IGV. The integers beneath the CpG marks represent the numbers of sequencing reads covering the CpG sites:

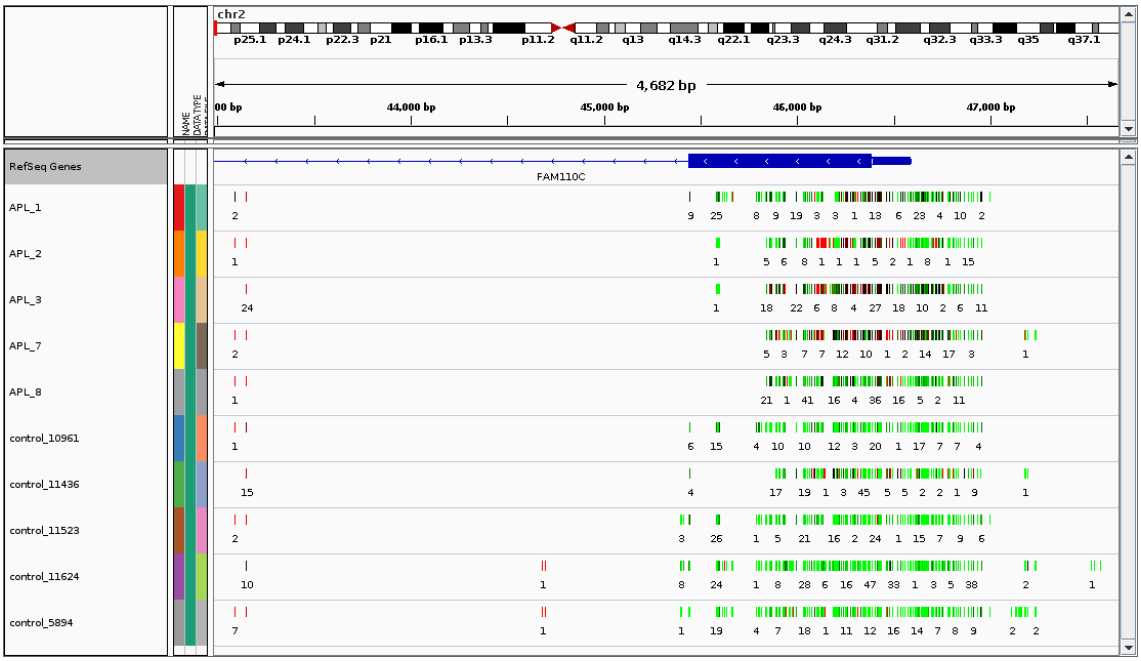

Figure 9: IGV snapshot of the raw data in and around a detected DMR

We can also load the smoothed methylation levels:

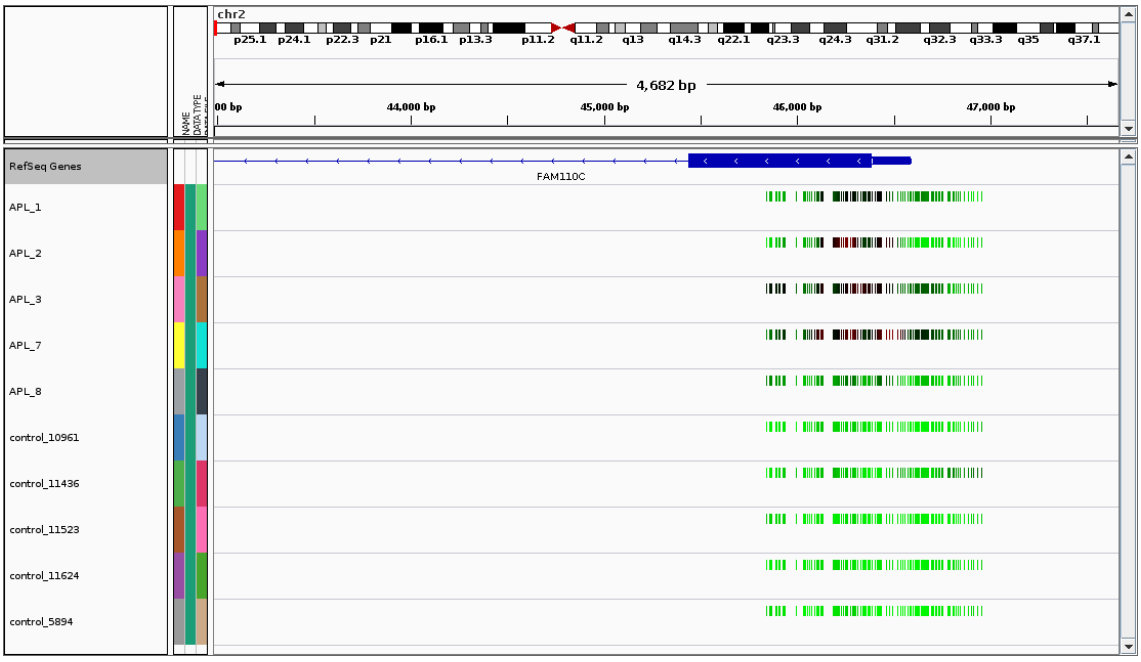

Figure 10: IGV snapshot of the smoothed data in and around a detected DMR