biomvRCNS: Copy Number study and Segmentation for multivariate biological data.

Yang Du*

July 16, 2014

Abstract

With high throughput experiments like tiling array and NGS, researchers are looking for continuous homogeneous segments or signal peaks, which would represent chromatin states, methylation ratio, transcripts or genome regions of deletion and amplification. While in a normal experimental set-up, these profiles would be generated for multiple samples or conditions with replicates. In the package *biomvRCNS*, a Hidden Semi Markov Model and one homogeneous segmentation model are implemented and tailored to handle multiple genomic profiles, with the aim of assisting in transcripts detection using high throughput technology like RNA-seq or tiling array, and copy number analysis using aCGH or targeted sequencing.

1 Introduction

To begin with *biomvRCNS*, load the package and read the manual page.

```
> library(biomvRCNS)
```

In the package, 3 main functions are provided for the batch processing of multiple chromosome regions across samples: `biomvRhsmm`, a hidden semi Markov model (HSMM) (**?**); `biomvRseg`, a maximum likelihood based homogeneous segmentation model; and a third `biomvRmgmr`, custom batch function using max-gap-min-run algorithm. In the following sections we will illustrate their functionalities using example data. Currently the package does not deal with data correction, so input should be normalized by reference or paired sample and corrected for factor of interest before passing down.

2 Example of array CGH data set of Coriell cell lines

Extracted from packge *DNACopy* (**?**), the `coriell` data contains two aCGH studies (GM05296 and GM13330) of Corriel cell lines taken from **?**. In particular, with 2271 mapped features in total across 22 autosomes and chromosome X.

All three main functions accept common data matrix plus positional information as input or a *GRanges* object with data matrix stored in the meta columns. To get started, we first build a *GRanges* object from *data.frame*.

```
> data('coriell', package='biomvRCNS')
> head(coriell, n=3)
```

*email: yang.du@uni-rostock.de

```
       Clone Chromosome Position Coriell.05296 Coriell.13330
1 GS1-232B23          1        1      0.000359      0.207470
2 RP11-82d16          1      469      0.008824      0.063076
3 RP11-62m23          1     2242     -0.000890      0.123881
```

```
> xgr<-GRanges(seqnames=coriell[,2],
+         IRanges(start=coriell[,3], width=1, names=coriell[,1]))
> values(xgr)<-DataFrame(coriell[,4:5], row.names=NULL)
> xgr<-sort(xgr)
> head(xgr, n=3)
```

```
GRanges object with 3 ranges and 2 metadata columns:
             seqnames       ranges strand | Coriell.05296 Coriell.13330
                <Rle>    <IRanges>  <Rle> |     <numeric>     <numeric>
  GS1-232B23        1 [   1,    1]      * |      0.000359       0.20747
  RP11-82d16        1 [ 469,  469]      * |      0.008824      0.063076
  RP11-62m23        1 [2242, 2242]      * |      -0.00089      0.123881
  -------
  seqinfo: 23 sequences from an unspecified genome; no seqlengths
```

Please be sure that the data is sorted with respect to their positions before feeding to the models.

2.1 Genomic segmentation with Hidden-semi Markov model

First we use the hidden-semi Markov model with the batch function `biomvRhsmm`, which will sequentially process each chromosome identified by the *seqnames* (using dummy name when no *GRanges* supplied in `x` or `xRange`), thus for non-continuous regions on the same chromosome user should give different *seqnames* to each part of the data. Within this package, there is one argument *grp*, for all main batch functions, which is used to assign data columns to groups according to the experimental design, say technical replicates or biological replicates. Sample columns within the same group could be treated simultaneously in the modelling process as well as iteratively. [1] In this example, the two profiles are considered independent and not similar, thus been given different values in the *grp* vector. Additionally there is a built-in automatic grouping method, given a valid clustering method *cluster.m* and *grp* set to `NULL`. By default, all data columns are assumed to be from the same group.

```
> rhsmm<-biomvRhsmm(x=xgr, maxbp=1E5, J=3, soj.type='gamma',
+         com.emis=T, emis.type='norm', prior.m='quantile', grp=c(1,2))
```

```
> show(rhsmm)
```

```
Object is of class: 'biomvRCNS'
List of parameters used in the model:
J, maxk, maxbp, maxgap, soj.type, emis.type, q.alpha, r.var, iterative, cMethod, maxit, tol,
grp, cluster.m, avg.m, prior.m, trim, na.rm, soj.par, emis.par

The segmented ranges:
GRanges object with 102 ranges and 3 metadata columns:
        seqnames           ranges strand |        SAMPLE STATE        AVG
           <Rle>        <IRanges>  <Rle> |         <Rle> <Rle>      <Rle>
    [1]        1 [     1, 108746]      * | Coriell.05296     2  0.0091220
```

[1] Simultaneous treatment within group is currently available for *emis.type* equals 'mvnorm' or 'mvt' in `biomvRhsmm`, *poolGrp=TRUE* in `biomvRmgmr` and *twoStep=FALSE* in `biomvRseg`.

```
  [2]        1 [112204, 218166]      * | Coriell.05296     2  0.0138270
  [3]        1 [110293, 110293]      * | Coriell.05296     1 -0.0791300
  [4]        1 [220439, 240001]      * | Coriell.05296     1 -0.0083905
  [5]        1 [     1,  36207]      * | Coriell.13330     3  0.0874010
  ...      ...              ...    ... .           ...   ...        ...
 [98]       22 [ 20553,  33001]      * | Coriell.13330     3   0.130433
 [99]       23 [     1, 155001]      * | Coriell.05296     3   0.676184
[100]       23 [     1,  98906]      * | Coriell.13330     2  -0.053510
[101]       23 [125572, 155001]      * | Coriell.13330     2  -0.012260
[102]       23 [103194, 122966]      * | Coriell.13330     1  -0.101480
-------
seqinfo: 23 sequences from an unspecified genome; no seqlengths
```

In the above run, we limit the model complexity by setting the *maxbp* to $1E5$, which will restrict the maximum sojourn time to *maxbp*. J is the number of states in the HSMM model, this argument can be given explicitly or estimated from prior information provided in *xAnno*. Argument *soj.type* defines the type of sojourn distribution; with Gamma distributed sojourn, the neighbouring position will tend to have the same state, and transit to other states if far apart. In this way the sojourn distribution fully incorporate the positional information into the probabilistic framework.

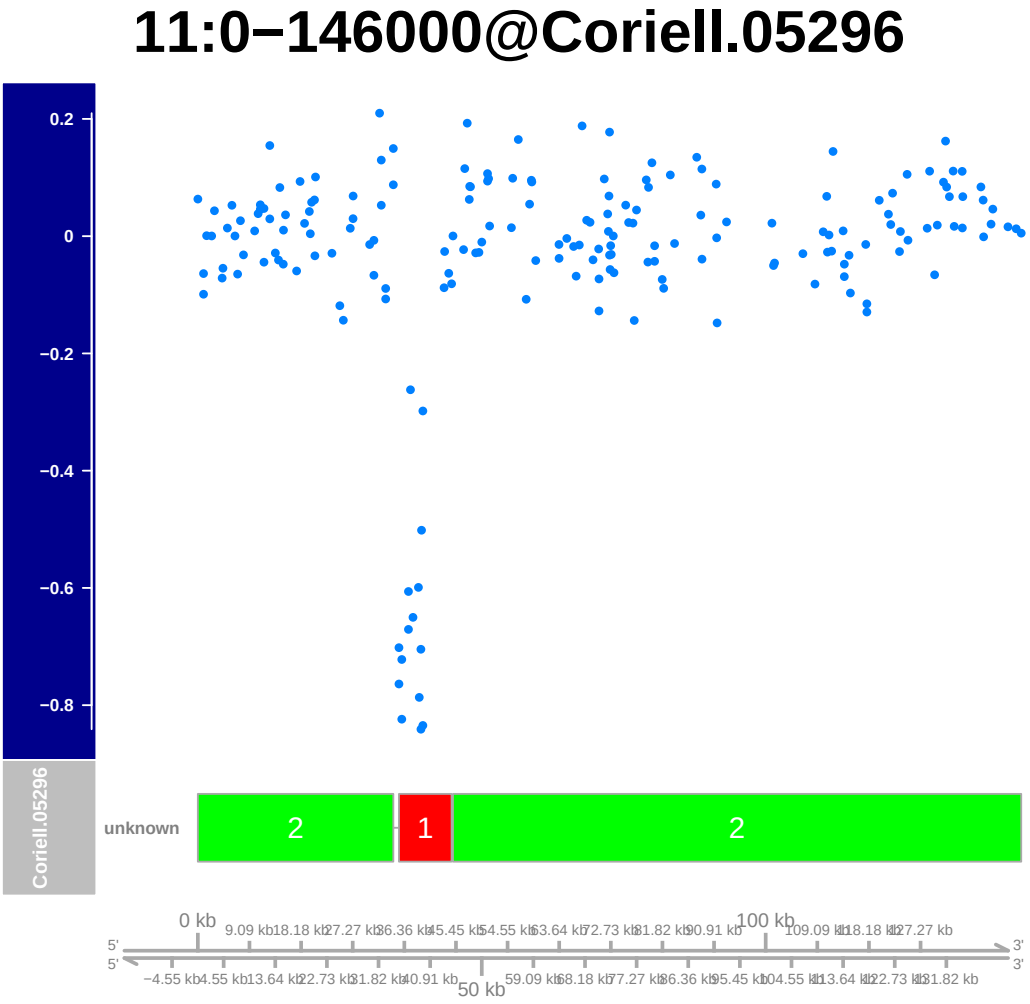
Argument *emis.type* controls the distribution of emission probability, in this case the log2 ratio of aCGH data is considered to follow Normal distribution. The emission density could be estimated using all data or only data on the respective region or chromosome (identified by unique `seqnames`), controlling via *com.emis*. In this case, the ratios cross chromosomes are directly comparable, thus *com.emis* was set to true. The prior of the emission parameters could be controlled by supplying *q.alpha* and *r.var* with *prior.m='quantile'*, or automatically determined through a clustering process with *prior.m='cluster'*.

The function will then call C codes and estimate the most likely state sequence, with either *cMethod='F-B'* or *cMethod='Viterbi'*. The *F-B* method (default) uses a forward-backward algorithm described in **?**, which gives a smooth state sequence, whereas the Viterbi algorithm with *cMethod='Viterbi'* will use the state profile estimated by the forward-backward algorithm and rebuild the most likely state sequence. The parameter *maxit* controls the maximum iteration of the EM algorithm. When assessing aCGH data, the quantile method should be able to give a good estimation of the emission desity priors, one can also adjust *q.alpha* and *r.var* for better control over the mean-variance relationships in extreme states. SInce we are not training a prediction model, but trying to derive the most likely state sequence, one iteration of the EM procedure is sufficient.

The function returns an object of class *biomvRCNS*, in which the `res` slot is a *GRanges* object containing the summary of each estimated segments. There are three meta columns: column `SAMPLE` gives the column name of which sample this segment belongs to; column `STATE`, the estimated state for each segment, the lower state number represents state with lower mean value, thus in this example, a state of 1 could represent region of deletion and 3 for region of duplication, whereas state 2 could be considered copy neutral; column `AVG`, gives the segment average value, which could take the form of (trimmed) mean or median controlled by *avg.m*. The original input is also kept and returned in slot `x` with the estimated most likely state assignment and associated probability.

A `plot` method has been implemented for *biomvRCNS* object using package *Gviz*, by default the `plot` method tries to output graphics to multiple EPS/PDF files for each chromosome region and sample. Multiple samples could also be overlaid on the same image, by passing *sampleInOne=TRUE* in the `plot` method. Here we set *tofile=FALSEALSE* to output graphics to the current device, and only show resulting graphics for chromosome 11 from sample Coriell.05296.

```
> obj<-biomvRGviz(exprgr=xgr[seqnames(xgr)=='11', 'Coriell.05296'],
+         seggr=rhsmm@res[mcols(rhsmm@res)[,'SAMPLE']=='Coriell.05296'], tofile=FALSE)
```



2.2 Using other methods provided in the package

In this section, we use the other two batch functions to process the `coriell` data. First we use `biomvRseg`, in which a similar segmentation method like in the package *tillingArray* (**?**) is implemented and extended to handle Poisson and Negative binomial distributed data. The function shares several argument with `biomvRhsmm`, like *maxbp* and *grp*. The *maxseg* gives the maximum number of segment per chromosome region, while the optimal number of segment per chromosome region is determined internally by assessing the likelihood with optional penalty terms, by default *penalty='BIC'* is used. Another option is to use modified Bayes information criterion *penalty='mBIC'* (**?**), as in the CBS algorithm used in *DNAcopy*. The function proceed in the following manner: assuming within each group sample columns exhibit similar patterns, and thus be processed simultaneously in the first step. By maximizing the likelihood the optimal number of segments is selected for each group. And in a second step if *twoStep=TRUE* or merging is necessary, the candidate segments produced in the first step are merged with respect to each sample, thus forcing sample columns in the same group to have a more unified segmentation result yet keeping it possible to have sample specific pattern.

```
> rseg<-biomvRseg(x=xgr, maxbp=4E4, maxseg=10, family='norm', grp=c(1,2))
```

```
> head(rseg@res)
```

```
GRanges object with 6 ranges and 3 metadata columns:
      seqnames           ranges strand |        SAMPLE        AVG       STATE
         <Rle>        <IRanges>  <Rle> |   <character>  <numeric> <character>
  [1]        1 [     1,  35001]      * | Coriell.05296  0.0045915         LOW
  [2]        1 [ 35106,  36207]      * | Coriell.05296  0.1335200        HIGH
  [3]        1 [ 37117,  85833]      * | Coriell.05296  0.0129660        HIGH
  [4]        1 [ 87215,  89493]      * | Coriell.05296 -0.1010400         LOW
  [5]        1 [ 91001, 132148]      * | Coriell.05296  0.0066565         LOW
  [6]        1 [132171, 136942]      * | Coriell.05296  0.1421530        HIGH
  -------
  seqinfo: 23 sequences from an unspecified genome; no seqlengths
```

After the example run, the function returns a *biomvRCNS* object, containing similar information as the previous `biomvRhsmm` run, except that the *STATE* column now only have a binary state value of either "HIGH" or "LOW", which is simply graded as 'HIGH' if the segment average is higher than the grand average of the whole region, and 'LOW' otherwise.

It is also possible to use the simple max-gap-min-run algorithm to segment aCGH profiles, by calling `biomvRmgmr`. But due to the binary nature of the algorithm, one have to run twice in order to get both extremely high and low segments, then combine the resulting *GRanges* manually.

```
> rmgmrh<-biomvRmgmr(xgr, q=0.9, high=T, maxgap=1000, minrun=2500, grp=c(1,2))
> rmgmrl<-biomvRmgmr(xgr, q=0.1, high=F, maxgap=1000, minrun=2500, grp=c(1,2))
> res<-c(rmgmrh@res, rmgmrl@res)
```

3 Example of transcript detection with RNA-seq data from ENCODE

The data contains gene expressions and transcript annotations in the region of the human TP53 gene (chr17:7,560,001-7,610,000 from the Human February 2009 (GRCh37/hg19) genome assembly), which is part of the long RNA-seq data generated by ENCODE (**?**) /Cold Spring Harbor Lab, containing 2 cell types (GM12878 and K562) with 2 replicates each. The libraries were sequenced on the Illumina GAIIx platform as paired-ends for 76 or 101 cycles for each read. The average depth of sequencing was 200 million reads (100 million paired-ends). The data were mapped against hg19 using Spliced Transcript Alignment and Reconstruction (STAR).

To generate local read counts, alignment files were pulled from UCSC (`http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeCshlLongRnaSeq/`) using package *Rsamtools*. And subsequently reads were counted in each non-overlapping unit sized window for the region (chr17:7,560,001-7,610,000). In the pre-compiled data `encodeTP53` , a window size of 25bp was used with the chunk of code below.

```
> winsize<-25
> cgr<-GRanges("chr17", strand='-',
+         IRanges(start=seq(7560001, 7610000, winsize), width =winsize))
> bf<-system.file("extdata", "encodeFiles.txt", package = "biomvRCNS")
> bamfiles<-read.table(bf, header=T, stringsAsFactors=F)
> library(Rsamtools)
> which<-GRanges("chr17", IRanges(7560001, 7610000))
> param<-ScanBamParam(which=which, what=scanBamWhat())
> for(i in seq_len(nrow(bamfiles))){
+         frd<-scanBam(bamfiles[i,1], param=param)
+         frdgr<-GRanges("chr17", strand=frd[[1]]$strand,
+                 IRanges(start=frd[[1]]$pos , end = frd[[1]]$pos+frd[[1]]$qwidth-1))
+         mcols(cgr)<-DataFrame(mcols(cgr), DOC=countOverlaps(cgr, frdgr))
+ }
```

Alternatively one can also operate on base pair resolution, in which case a *Rle* object should be preferred to store the count data for lower memory footprint and better efficiency. Also to speed things up, one could set *useMC=T* to enable parallel processing of multiple `seqnames`, the number of parallel process could be set by *options(mc.cores=n)*.

```
> cgr<-GRanges("chr17", strand='-',
+         IRanges(seq(7560001, 7610000), width=1))
> bf<-system.file("extdata", "encodeFiles.txt", package = "biomvRCNS")
> bamfiles<-read.table(bf, header=T, stringsAsFactors=F)
> library(Rsamtools)
> which<-GRanges("chr17", IRanges(7560001, 7610000))
> param<-ScanBamParam(which=which, flag=scanBamFlag(isMinusStrand=TRUE))
> for(i in seq_len(nrow(bamfiles))){
+         cod<-coverage(BamFile(bamfiles[i,1]), param=param)[['chr17']][7560001:7610000]
+         mcols(cgr)<-DataFrame(mcols(cgr), DOC=cod)
+ }
```

The pre-compiled data `encodeTP53` also includes the regional annotation of TP53 RNAs isoforms, `gmgr`, which were derived from the ENCODE Gene Annotations (GENCODE), `http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeGencodeV4/wgEncodeGencodeManualV4.gtf.gz)` , and subset to only isoforms of TP53 gene and neighboring genes in the region. The annotation object `gmgr` could be rebuilt with the following lines using the included file under `extdata`. The additional file 'gmodTP53.gff' can also be directly imported as *GRanges* using *rtracklayer*.

```
> af<-system.file("extdata", "gmodTP53.gff", package = "biomvRCNS")
> gtfsub<-read.table(af, fill=T, stringsAsFactors=F)
> gmgr<-GRanges("chr17", IRanges(start=gtfsub[, 4], end=gtfsub[, 5],
+         names=gtfsub[, 13]), strand=gtfsub[, 7], TYPE=gtfsub[, 3])
```

We first load the `encodeTP53` data, poll the read counts for each cell type and add 1 to the base count to increase stability.

```
> data(encodeTP53)
> cgr<-encodeTP53$cgr
> gmgr<-encodeTP53$gmgr
> mcols(cgr)<-DataFrame(
+         Gm12878=1+rowSums(as.matrix(mcols(cgr)[,1:2])),
+         K562=1+rowSums(as.matrix(mcols(cgr)[,3:4])) )
```

For count data from sequencing, the *emis.type* could be set to either `'pois'` or `'nbinom'`, though `'pois'` is preferred for sharp boundary detection. For the sojourn settings, instead of using the uninformative flat prior, we here use estimates from other data source as a prior. We load the *TxDb.Hsapiens.UCSC.hg19.knownGene* known gene database, and pass the *TxDb* object to *xAnno*. Then internally sojourn parameters and state number J will be estimated from *xAnno* by calling function `sojournAnno`. When given a *TxDb* object to *xAnno*, state number would be set to 3 and each represents 'intergenic', 'intron' and 'exon'. One can also supply a named *list* object with initial values for parameters of distribution specified by *soj.type*. For emission, since the highly dispersed nature of count data, we set the prior for emission mean to be more extreme, with *q.alpha=0.01*.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb<-TxDb.Hsapiens.UCSC.hg19.knownGene
> rhsmm<-biomvRhsmm(x=cgr, xAnno=txdb, maxbp=1E3, soj.type='gamma',
+         emis.type='pois', prior.m='quantile', q.alpha=0.01)
```

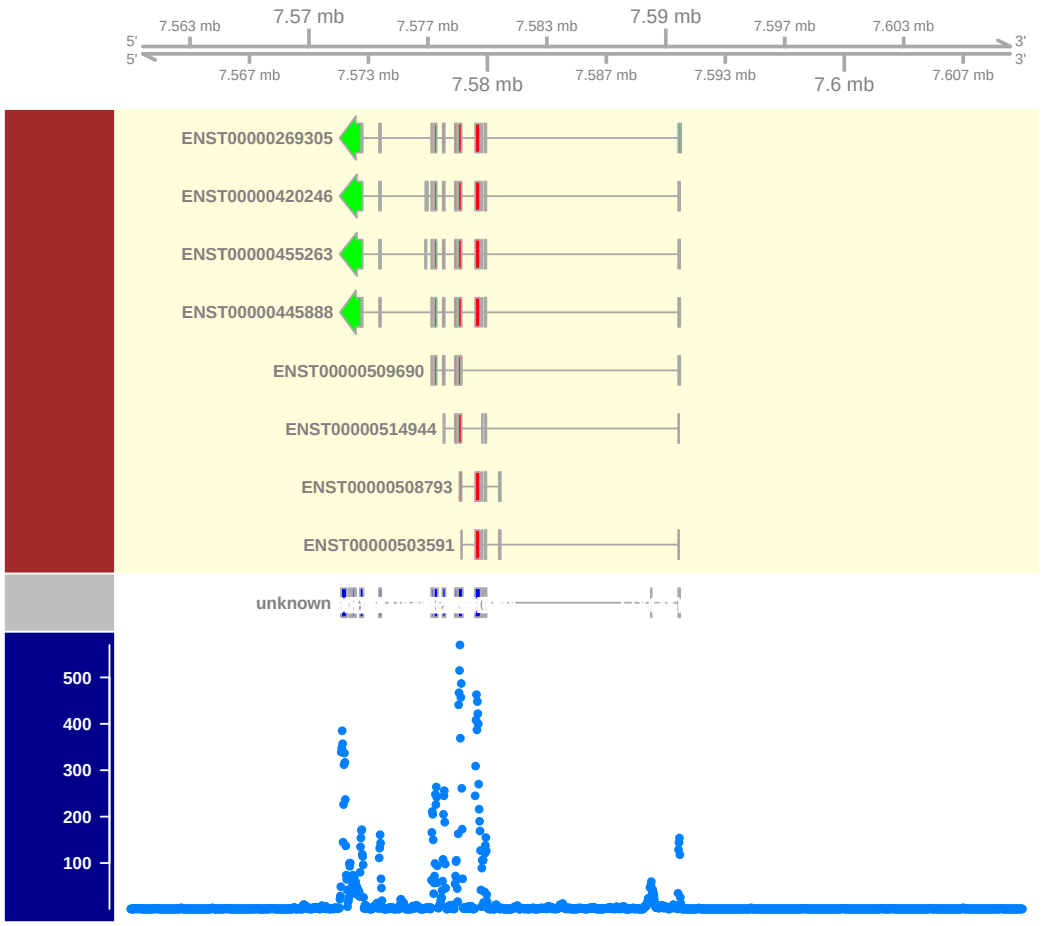
As in the ENCODE guide (**?**), the study identified the p53 isoform observed in K562 cells has a longer 3'UTR than the isoform seen in the GM12878 cell line. So here we plot our model estimates and consider the third state, namely 'exon', to represent detected transcripts. And the HSMM model clearly picked up the extra transcripts of the K562 cell line at the 3'UTR.

```
> rhsmm@res[mcols(rhsmm@res)[,'STATE']=='exon']
```

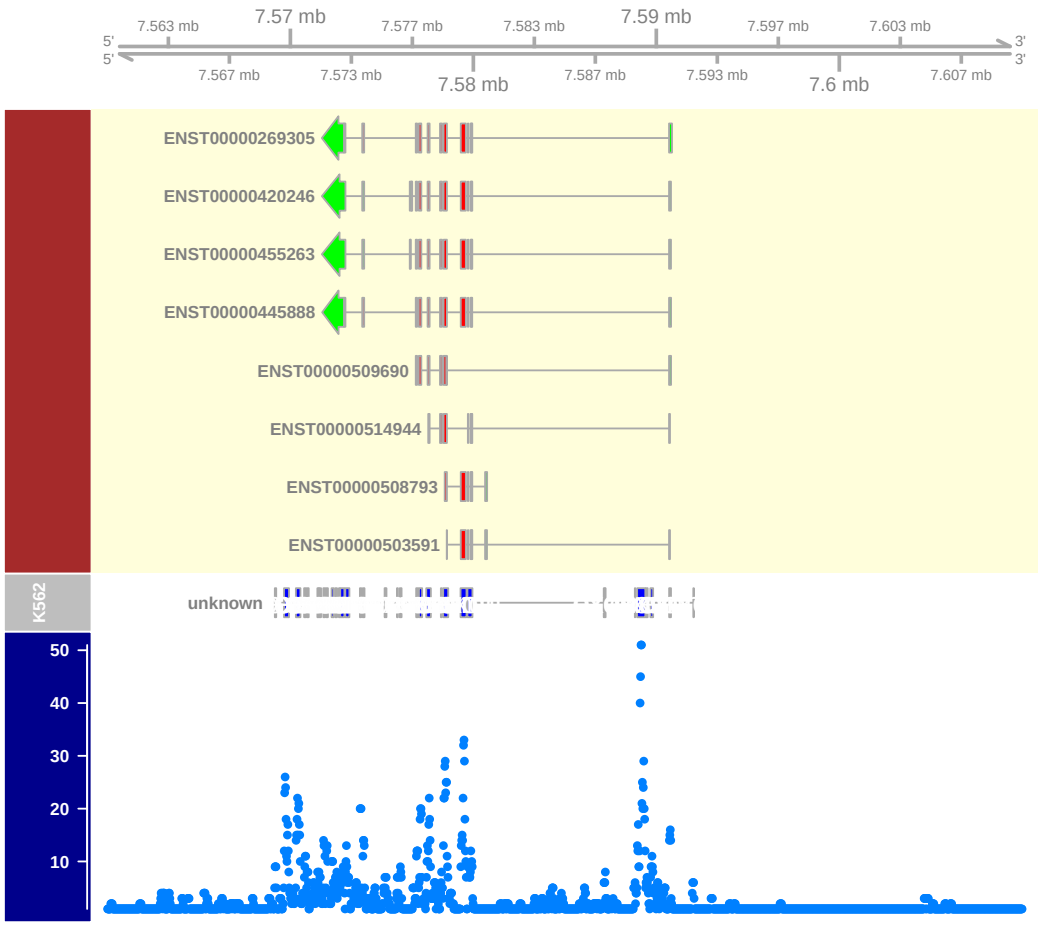
```
GRanges object with 52 ranges and 3 metadata columns:
       seqnames             ranges strand |  SAMPLE STATE   AVG
          <Rle>          <IRanges>  <Rle> |   <Rle> <Rle> <Rle>
   [1]    chr17 [7571801, 7572125]      - | Gm12878  exon   312
   [2]    chr17 [7572251, 7572350]      - | Gm12878  exon    96
   [3]    chr17 [7572426, 7572550]      - | Gm12878  exon    61
   [4]    chr17 [7572601, 7572625]      - | Gm12878  exon    60
   [5]    chr17 [7572851, 7573050]      - | Gm12878  exon   127
   ...      ...                ...    ... .     ...   ...   ...
  [48]    chr17 [7588951, 7589400]      - |    K562  exon  20.0
  [49]    chr17 [7589426, 7589525]      - |    K562  exon   6.0
  [50]    chr17 [7589676, 7589825]      - |    K562  exon   9.0
  [51]    chr17 [7590701, 7590800]      - |    K562  exon  14.5
  [52]    chr17 [7592026, 7592050]      - |    K562  exon   6.0
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
> g<-mcols(rhsmm@res)[,'STATE']=='exon' & mcols(rhsmm@res)[,'SAMPLE']=='Gm12878'
> obj<-biomvRGviz(exprgr=cgr[,'Gm12878'], gmgr=gmgr,
+         seggr=rhsmm@res[g], plotstrand='-', regionID='TP53', tofile=FALSE)
```

’53@chr17:7560000−7610000@Gm128




```
> k<-mcols(rhsmm@res)[,'STATE']=='exon' & mcols(rhsmm@res)[,'SAMPLE']=='K562'
> obj<-biomvRGviz(exprgr=cgr[,'K562'], gmgr=gmgr,
+   seggr=rhsmm@res[k], plotstrand='-', regionID='TP53', tofile=FALSE)
```



Now we can locate those novel detected fragments in K562 cell line comparing to the annotation and those detected in Gm12878 cell line. One can then follow up those findings either by gene structure prediction using local nucleotides composition or by experimental validation.

```
> nK2gm<-queryHits(findOverlaps(rhsmm@res[k], gmgr))
> nK2G<-queryHits(findOverlaps(rhsmm@res[k], rhsmm@res[g]))
> rhsmm@res[k][setdiff(seq_len(sum(k)), unique(c(nK2G, nK2gm)))]
```

```
GRanges object with 19 ranges and 3 metadata columns:
       seqnames              ranges strand | SAMPLE STATE   AVG
          <Rle>           <IRanges>  <Rle> |  <Rle> <Rle> <Rle>
   [1]    chr17 [7569151, 7569225]      - |   K562  exon     9
   [2]    chr17 [7569651, 7569925]      - |   K562  exon    15
   [3]    chr17 [7570301, 7570550]      - |   K562  exon    16
   [4]    chr17 [7570751, 7570850]      - |   K562  exon    10
   [5]    chr17 [7570901, 7571000]      - |   K562  exon     8
   ...      ...                 ...    ... .    ...   ...   ...
  [15]    chr17 [7587201, 7587225]      - |   K562  exon     8
  [16]    chr17 [7588826, 7588850]      - |   K562  exon     6
  [17]    chr17 [7589426, 7589525]      - |   K562  exon     6
  [18]    chr17 [7589676, 7589825]      - |   K562  exon     9
```

```
  [19]    chr17 [7592026, 7592050]      - |   K562  exon     6
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The other 2 batch functions could also be similarly applied here.

```
> rseg<-biomvRseg(x=cgr, maxbp=1E3, maxseg=20, family='pois')
> rmgmr<-biomvRmgmr(x=cgr, q=0.99, maxgap=50, minrun=100)
```

4 Example of differentially methylated region (DMR) detection

As an example, we include a toy dataset extracted from *BiSeq* (**?**), which is a small subset of a published study (**?**), comprising intermediate differential methylation results prior to DMR detection. We first load the `variosm` data, the data contains a *GRanges* object `variosm` with two meta columns: '`meth.diff`', methylation difference between the two sample groups; '`p.val`', significance level from the Wald test. What we will show here latter could be applied on other pipelines as well, using similar data input.

```
> data(variosm)
> head(variosm, n=3)
```

```
GRanges object with 3 ranges and 2 metadata columns:
      seqnames           ranges strand |          meth.diff                p.val
         <Rle>        <IRanges>  <Rle> |          <numeric>            <numeric>
  [1]     chr1 [872335, 872335]      * | 0.0889114878437597  0.00113176519852419
  [2]     chr1 [872369, 872369]      * | 0.0970307417273086 0.000767802697850362
  [3]     chr1 [872370, 872370]      * |  0.096558901255241 0.000834745131903375
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

In the *BiSeq* work-flow, they use an approach similar to the max-gap-min-run algorithm to define the DMR boundaries, by prior filtering and comparing the differential test statistics with a user specified significance level in the candidate regions. The positional information of methylation sites is taking into account by locating and testing highly correlated cluster regions in the filtering process.

We now use the `biomvRhsmm` model to detect DMR, since there are mainly two types of measurement associated with differential methylation studies like we have here, one is the difference in the methylation ratio and the other one is the significance level from differential test. The methylation difference gives information about the directionality of the change as well as the size, and the significance level gives the confidence in claiming differential events.

So here we utilize both information for the DMR detection. We implicitly ask the model to give 3 states, since J is default to 3, in which case the three states may each represent hypomethylated regions, undefined null regions, and hypermethylated regions respectively when modelling `meth.diff`; While modelling significance level these states would represent highly confident regions, lowly confident regions or / and null results. For both scenarios, we are more interested in extreme states where we have consistent differences and low P-values. However the distribution of `p.val` and `meth.diff` are both highly asymmetric, we thus enable the cluster mode for emission prior initialization by setting *prior.m*='cluster' . And due to the non-uniformly located CpG sites, one may split inter-spreading long segments with parameter *maxgap*=100.

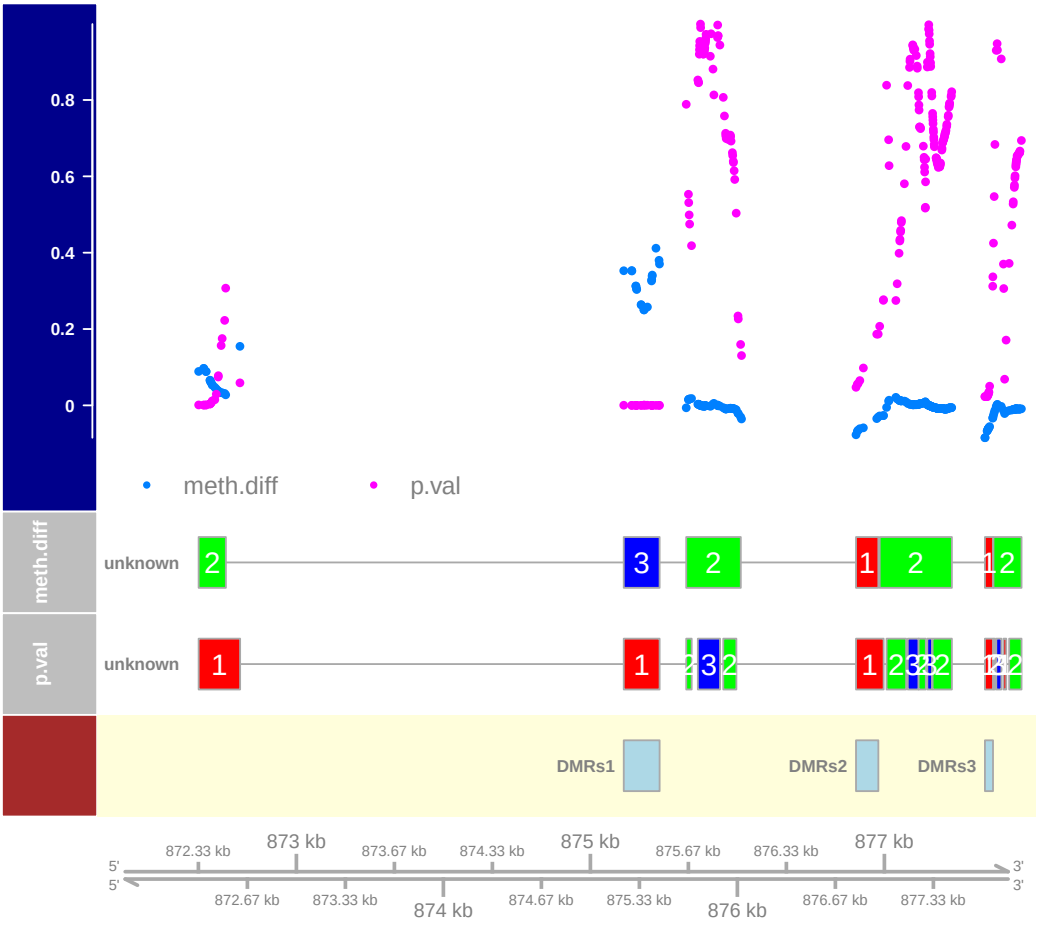
```
> rhsmm<-biomvRhsmm(x=variosm, maxbp=100, prior.m='cluster', maxgap=100)
```

```
> hiDiffgr<-rhsmm@res[mcols(rhsmm@res)[,'STATE']!=2
+         & mcols(rhsmm@res)[,'SAMPLE']=='meth.diff']
> dirNo<-mcols(hiDiffgr)[,'STATE']=='1' & mcols(hiDiffgr)[,'AVG']>0 |
+         mcols(hiDiffgr)[,'STATE']=='3' & mcols(hiDiffgr)[,'AVG']<0
> hiDiffgr<- hiDiffgr[!dirNo]
> loPgr<-rhsmm@res[mcols(rhsmm@res)[,'STATE']==1
+         & mcols(rhsmm@res)[,'SAMPLE']=='p.val']
> DMRs<-reduce(intersect(hiDiffgr, loPgr), min.gapwidth=100)
> idx<-findOverlaps(variosm, DMRs, type='within')
> mcols(DMRs)<-DataFrame(cbind(TYPE='DMR', aggregate(as.data.frame(mcols(variosm[queryHits(idx)])),
+         by=list(DMR=subjectHits(idx)), FUN=median)[,-1]))
> names(DMRs)<-paste0('DMRs', seq_along(DMRs))
> DMRs
```

```
GRanges object with 5 ranges and 3 metadata columns:
        seqnames           ranges strand |     TYPE   meth.diff        p.val
           <Rle>        <IRanges>  <Rle> | <factor>   <numeric>    <numeric>
  DMRs1     chr1 [875227, 875470]      * |      DMR  0.31947418 6.677193e-06
  DMRs2     chr1 [876807, 876958]      * |      DMR -0.06108219 6.500328e-02
  DMRs3     chr1 [877684, 877738]      * |      DMR -0.06123008 2.844639e-02
  DMRs4     chr2 [ 46126,  46280]      * |      DMR  0.41008524 1.818530e-07
  DMRs5     chr2 [ 46389,  46558]      * |      DMR  0.44823172 1.890819e-06
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

After the model fitting, by intersecting regions with extreme `meth.diff` and regions with low `p.val`, we can locate those detected DMRs, returned with their average `meth.diff` and `p.val`. Comparing to the regions detected in the *BiSeq* vignette, the two sets of regions are largely similar except for two regions: (chr1:872335,872386), which in our case the `meth.diff` has not been considered high enough due to the highly asymmetric distribution of 'meth.diff'; another region (chr2:46915,46937) resides in the tail of chromosome 2 with low density of methylation sites, which has been sorted to the intermediate states due to the lack of support from both the emission level and the sojourn time. However it is worth mentioning that due to the filtering applied in their work-flow, they built wider regions out of a smaller set of more significant sites; while in our case, the regions are more refined and especially we identified two hypomethylated regions.

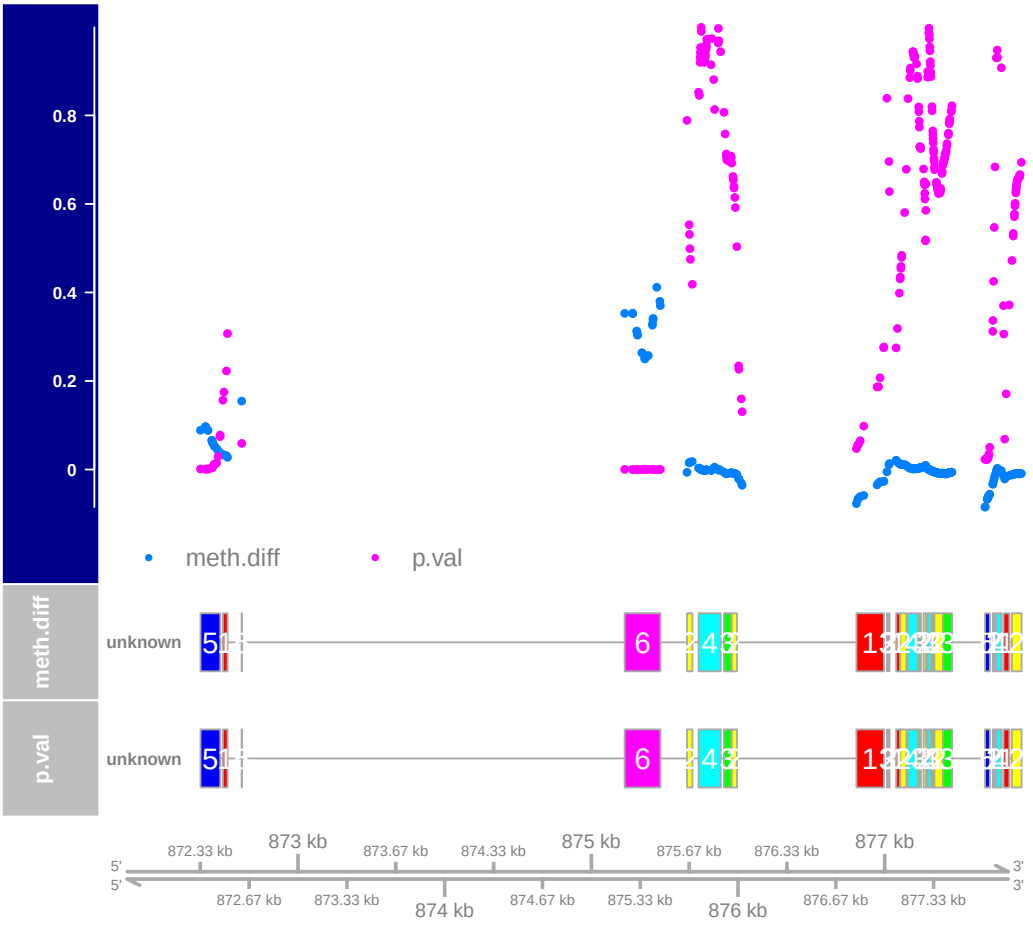
```
> plot(rhsmm, gmgr=DMRs, tofile=FALSE)
```



Other than the iterative fitting of individual models for the two profiles, it is also possible to take advantage of the multivariate nature of the data. Since in this case we are interested in locating regions show high difference in the methylation level, while also achieving high significance in the statistical test. To better capture the covariance structure of possible combination of the two profiles, we use the multivariate Normal for the emission and raise the number of state to a relatively high number, for example 6. Also we enable the '*com.emis*' mode to learn from the whole data rather than individual chromosome. After the model fitting, we inspect the estimated segment profile produced by the `plot` method. We could see that segments labelled with state '6' could be consider as the most significant DMRs, which show high `meth.diff` and with very low `p.val`, while state '5' could be considered as potential candidates for DMRs with relatively lower confidence. After the model fitting, one can also access the fitted emission and sojourn parameters. One can see from the emission parameter and also from the figure, the state 1 could be potentially linked to hypomethylated regions, where the methylation difference average is under 0 yet with relatively low `p.val`.

```
> rhsmm<-biomvRhsmm(x=variosm, J=6, maxbp=100, emis.type='mvnorm',
+  prior.m='cluster', maxgap=100, com.emis=T)
```

```
> plot(rhsmm, tofile=FALSE)
```




```
> DMRs<-reduce(rhsmm@res[mcols(rhsmm@res)['STATE']=='6'], min.gapwidth=100)
> idx<-findOverlaps(variosm, DMRs, type='within')
> mcols(DMRs)<-DataFrame(cbind(TYPE='DMR', aggregate(as.data.frame(mcols(variosm[queryHits(idx)])),
+         by=list(DMR=subjectHits(idx)), FUN=median)[,-1], stringsAsFactors=F))
> names(DMRs)<-paste0('DMRs', seq_along(DMRs))
> DMRs

GRanges object with 2 ranges and 3 metadata columns:
        seqnames            ranges strand |        TYPE meth.diff        p.val
           <Rle>         <IRanges>  <Rle> | <character> <numeric>    <numeric>
  DMRs1     chr1 [875227, 875470]      * |         DMR 0.3194742 6.677193e-06
  DMRs2     chr2 [ 46126,  46718]      * |         DMR 0.3553632 3.960405e-05
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths

> cDMRs<-reduce(rhsmm@res[mcols(rhsmm@res)['STATE']=='5'], min.gapwidth=100)
> idx<-findOverlaps(variosm, cDMRs, type='within')
> mcols(cDMRs)<-DataFrame(cbind(TYPE='cDMR', aggregate(as.data.frame(mcols(variosm[queryHits(idx)])),
+         by=list(cDMRs=subjectHits(idx)), FUN=median)[,-1], stringsAsFactors=F))
> names(cDMRs)<-paste0('cDMRs', seq_along(cDMRs))
> cDMRs
```

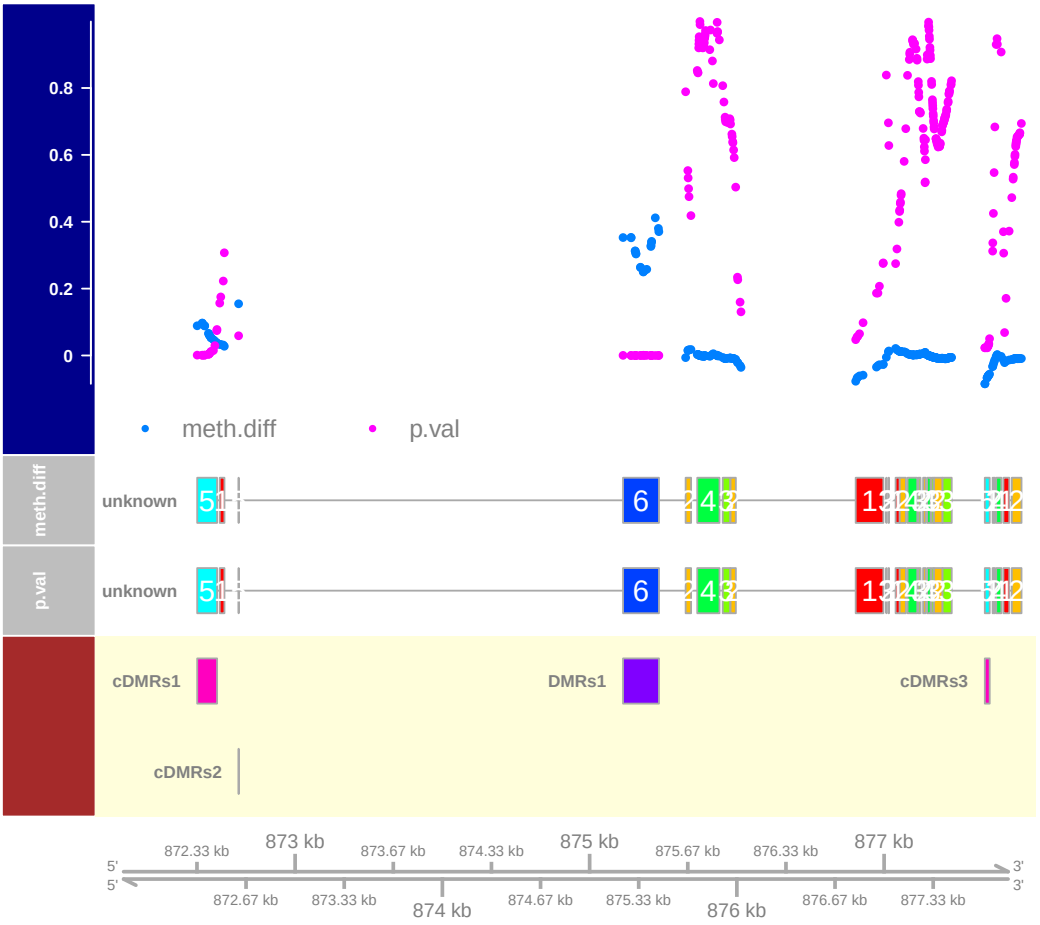
```
GRanges object with 6 ranges and 3 metadata columns:
         seqnames           ranges strand |        TYPE   meth.diff       p.val
            <Rle>        <IRanges>  <Rle> | <character>   <numeric>   <numeric>
  cDMRs1     chr1 [872335, 872469]      * |        cDMR  0.06064747 0.005942674
  cDMRs2     chr1 [872616, 872616]      * |        cDMR  0.15466604 0.059000939
  cDMRs3     chr1 [877684, 877716]      * |        cDMR -0.06185686 0.027555602
  cDMRs4     chr2 [ 45843,  46113]      * |        cDMR  0.13787294 0.055902671
  cDMRs5     chr2 [ 46725,  46752]      * |        cDMR  0.12334614 0.027668535
  cDMRs6     chr2 [ 46915,  46937]      * |        cDMR  0.09944399 0.014845962
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths

> rhsmm@param$emis.par['chr1',][[1]]


$var
$var[[1]]
              meth.diff         p.val
meth.diff  0.0014900896 -0.0001065013
p.val     -0.0001065013  0.0071132700

$var[[2]]
              meth.diff         p.val
meth.diff  8.660973e-05 -0.0003151823
p.val     -3.151823e-04  0.0044724795

$var[[3]]
             meth.diff        p.val
meth.diff 2.935401e-05 9.750245e-05
p.val     9.750245e-05 2.324940e-03

$var[[4]]
              meth.diff         p.val
meth.diff  3.064254e-06 -1.686886e-05
p.val     -1.686886e-05  1.055942e-03

$var[[5]]
             meth.diff        p.val
meth.diff 6.968485e-03 8.083678e-05
p.val     8.083678e-05 1.153695e-03

$var[[6]]
              meth.diff         p.val
meth.diff  0.0070819026 -2.259439e-04
p.val     -0.0002259439  6.130125e-05


$mu
$mu[[1]]
  meth.diff       p.val
-0.02066372  0.23425762

$mu[[2]]
   meth.diff        p.val
-0.007890973  0.631601677

$mu[[3]]
```

```
   meth.diff        p.val
-0.007865075  0.760032309

$mu[[4]]
  meth.diff       p.val
0.001136895 0.942443888

$mu[[5]]
 meth.diff      p.val
0.04309182 0.02937310

$mu[[6]]
  meth.diff       p.val
3.21372e-01 6.59215e-05
```

```
> rhsmm@param$soj.par['chr1',][[1]]
```

```
$scale
[1]   7.094867  12.880280  22.528988  19.636070 442.751509  82.985092

$shape
[1] 5.98484325 2.75899525 1.45863754 2.16831102 0.01084226 0.07593698
```

```
> plot(rhsmm, gmgr=c(DMRs, cDMRs), tofile=FALSE)
```



5 More

To be continued ...

6 Session information

```
> sessionInfo()
```

```
R version 3.3.1 (2016-06-21)
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] grid      stats4    parallel  stats     graphics  grDevices utils     datasets  methods
```

```
[10] base

other attached packages:
 [1] cluster_2.0.5                       TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
 [3] GenomicFeatures_1.26.0              AnnotationDbi_1.36.0
 [5] Biobase_2.34.0                      biomvRCNS_1.14.0
 [7] Gviz_1.18.0                         GenomicRanges_1.26.0
 [9] GenomeInfoDb_1.10.0                 IRanges_2.8.0
[11] S4Vectors_0.12.0                    BiocGenerics_0.20.0

loaded via a namespace (and not attached):
 [1] SummarizedExperiment_1.4.0    VariantAnnotation_1.20.0      splines_3.3.1
 [4] lattice_0.20-34               colorspace_1.2-7              htmltools_0.3.5
 [7] rtracklayer_1.34.0            chron_2.3-47                  interactiveDisplayBase_1.12.0
[10] survival_2.39-5               XML_3.98-1.4                  foreign_0.8-67
[13] DBI_0.5-1                     ensembldb_1.6.0               BiocParallel_1.8.0
[16] RColorBrewer_1.1-2            matrixStats_0.51.0            plyr_1.8.4
[19] zlibbioc_1.20.0               Biostrings_2.42.0             munsell_0.4.3
[22] gtable_0.2.0                  mvtnorm_1.0-5                 latticeExtra_0.6-28
[25] biomaRt_2.30.0                BiocInstaller_1.24.0          httpuv_1.3.3
[28] Rcpp_0.12.7                   xtable_1.8-2                  acepack_1.3-3.3
[31] BSgenome_1.42.0               scales_0.4.0                  Hmisc_3.17-4
[34] XVector_0.14.0                mime_0.5                      Rsamtools_1.26.0
[37] gridExtra_2.2.1               AnnotationHub_2.6.0           ggplot2_2.1.0
[40] digest_0.6.10                 biovizBase_1.22.0             shiny_0.14.1
[43] tools_3.3.1                   bitops_1.0-6                  RCurl_1.95-4.8
[46] RSQLite_1.0.0                 dichromat_2.0-0               Formula_1.2-1
[49] Matrix_1.2-7.1                data.table_1.9.6              httr_1.2.1
[52] R6_2.2.0                      rpart_4.1-10                  GenomicAlignments_1.10.0
[55] nnet_7.3-12
```