DNA methylation analysis in R

Olivier Elemento
ole2001@med.cornell.edu

Matt Teater
mrt2001@med.cornell.edu

Institute for Computational Biomedicine
Weill Cornell Medical College
New York, NY, USA

June, 2015

1 Introduction

In this lab, we will cover some functionality provided by the *methylkit* R package for the analysis of DNA methylation data obtained from Reduced Representation Bisulfite Sequencing (RRBS) experiments. We will use this package to characterize the methylome, detect differentially methylated CpGs, call differentially methylated regions, and correlate DNA methylation alterations with changes in gene expression.

2 Load package

We will primarily be using the *methylKit* package (https://code.google.com/p/methylkit/) for this lab. In addition, we will also be using the *biomaRt* package (http://www.bioconductor.org/packages/release/bioc/html/biomaRt.html) to map gene identifiers to gene symbols.

```
> library(methylKit)
> library(biomaRt)
```

3 Import data

In this lab, we will use a dataset that contains two paired test and control samples, respectively. There are a number of ways to align bisulfite converted reads and call percent methylation per base, see *methylKit* pacakge vignette fror methods directly supported by *methylKit*. In our case, we will use methylation call files that has previously been generated using the AMP pipeline (for more info, visit https://code.google.com/p/amp-errbs/).

There are two ways we can import our methylation call files into R. The first one is to read a set of methylation call files that have test/control conditions

specified to a *methylRawList* object.

```
> file.list <- list("methylcall_test1.txt", "methylcall_test2.txt",
+ "methylcall_ctrl1.txt", "methylcall_ctrl2.txt")
> myobj <- read(file.list, sample.id=list("test1", "test2", "ctrl1",
+ "ctrl2"), assembly="hg19", treatment=c(1, 1, 0, 0), context="CpG")
```

Another way to read the methylation calls is to start directly from SAM files. The SAM file must be sorted and only SAM files from Bismark aligner are supported. Check *read.bismark()* function to learn more about this.
methylRawList objects can be coerced to a *GRanges* object from the *GenomicRanges* package. This may give you additional flexibility when customizing your analyses.

```
> as(myobj[[2]], "GRanges")
```

```
GRanges with 531146 ranges and 3 metadata columns:
           seqnames                ranges strand   |  coverage     numCs
              <Rle>             <IRanges>  <Rle>   | <integer> <numeric>
       [1]     chr1 [100001074, 100001074]      -   |         1         1
       [2]     chr1 [100001361, 100001361]      -   |         1         0
       [3]     chr1 [100001948, 100001948]      +   |         3         2
       [4]     chr1 [100003154, 100003154]      -   |         3         3
       [5]     chr1 [100004680, 100004680]      +   |         1         1
       [6]     chr1 [100007717, 100007717]      -   |         1         1
       [7]     chr1 [  1000172,   1000172]      -   |         3         1
       [8]     chr1 [  1000157,   1000157]      -   |         3         0
       [9]     chr1 [  1000165,   1000165]      -   |         3         1
       ...      ...                   ...    ... ...       ...       ...
  [531138]     chr1   [  999902,   999902]      +   |         6         0
  [531139]     chr1   [  999863,   999863]      +   |         6         0
  [531140]     chr1   [  999910,   999910]      +   |         6         0
  [531141]     chr1   [  999888,   999888]      +   |         5         0
  [531142]     chr1   [  999950,   999950]      +   |         3         2
  [531143]     chr1   [99992465, 99992465]      -   |         1         0
  [531144]     chr1   [ 9999280,  9999280]      +   |         1         1
  [531145]     chr1   [  999999,   999999]      -   |         1         0
  [531146]     chr1   [  999998,   999998]      +   |         1         0
              numTs
           <numeric>
       [1]         0
       [2]         1
       [3]         1
       [4]         0
       [5]         0
       [6]         0
       [7]         2
       [8]         3
       [9]         2
       ...       ...
```

```
  [531138]         6
  [531139]         6
  [531140]         6
  [531141]         5
  [531142]         1
  [531143]         1
  [531144]         0
  [531145]         1
  [531146]         1
  ---
  seqlengths:
   chr1
     NA
```

4 Characterize basic features of the data

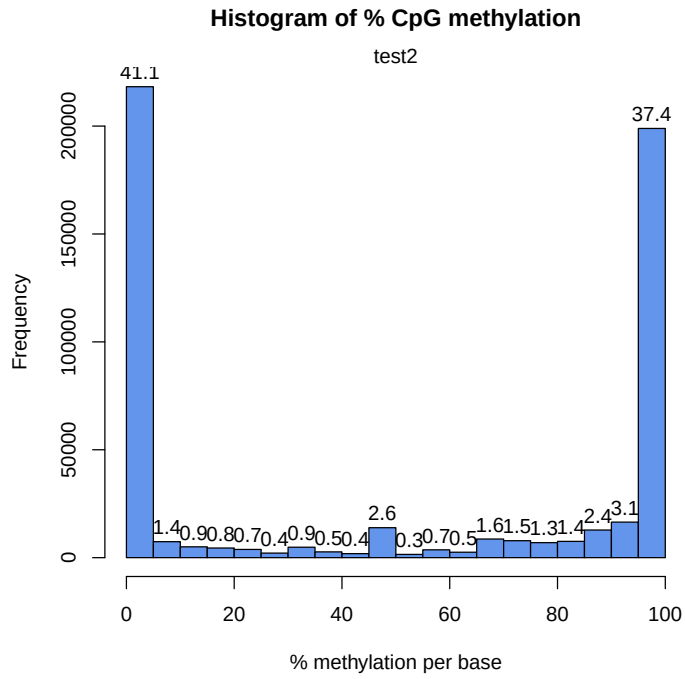
4.1 Assess methylation levels and depth of coverage

Now that we have read the methylation data into a *methylRawList* object, we can evaluate basic stats and QC of the methylation data, such as percent methylation. The following command prints out percent methylation statistics for the second sample, "test2:"

```
> getMethylationStats(myobj[[2]], plot=FALSE, both.strands=FALSE)

methylation statistics per base
summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00   54.55   50.27  100.00  100.00
percentiles:
        0%        10%        20%        30%        40%        50%        60%
  0.000000   0.000000   0.000000   0.000000   2.531646  54.545455  91.275168
       70%        80%        90%        95%        99%      99.5%      99.9%
100.000000 100.000000 100.000000 100.000000 100.000000 100.000000 100.000000
      100%
100.000000
```

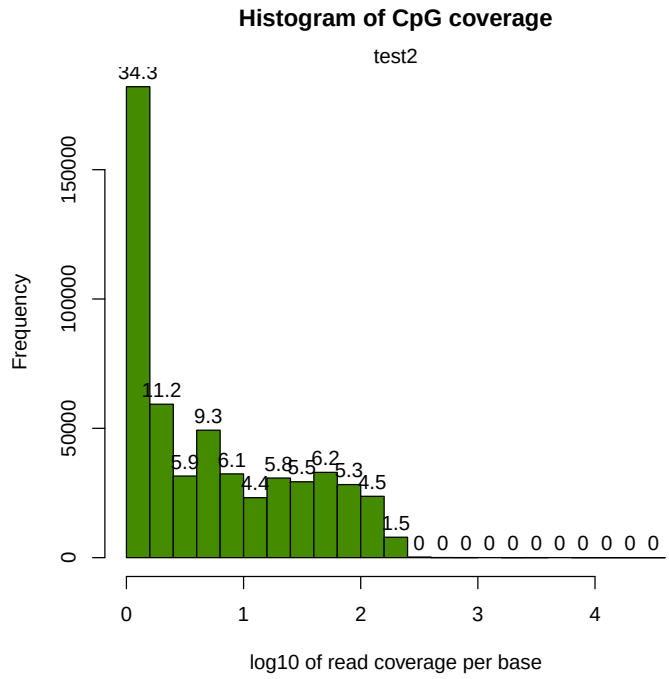
If we want to plot the histogram for percent methylation, we can do so with the same command, changing *plot=TRUE*. This will produce a histogram with numbers on the bars to denote what percentage of CpGs are contained in that bin. Typically, percent methylation should be bimodal and have peaks at both ends of the histogram, representing "unmethylated" and "methylated" CpGs.

```
> getMethylationStats(myobj[[2]], plot=TRUE, both.strands=FALSE)
```



We can also plot the read coverage per base in a similar way. Experiments with high PCR duplication bias will have a secondary peak toward the right hand side of the histogram.

```
> getCoverageStats(myobj[[2]], plot=T, both.strands=F)
```



4.2 Filtering samples based on read coverage

It is often useful to filter samples based on coverage. For example, if our samples are suffering from PCR bias, it would be advantageous to discard bases with very high read coverage. Furthermore, we should discard bases that have low read coverage. We can filter our *methylRawList* object and discard bases that have coverage below 10X and bases with greater than 99.9th percentile of coverage in each sample using the *filterByCoverage()* function.

```
> filtered.myobj <- filterByCoverage(myobj, lo.count=10, lo.perc=NULL,
+ hi.count=NULL, hi.perc=99.9)
```

4.3 Sample correlation

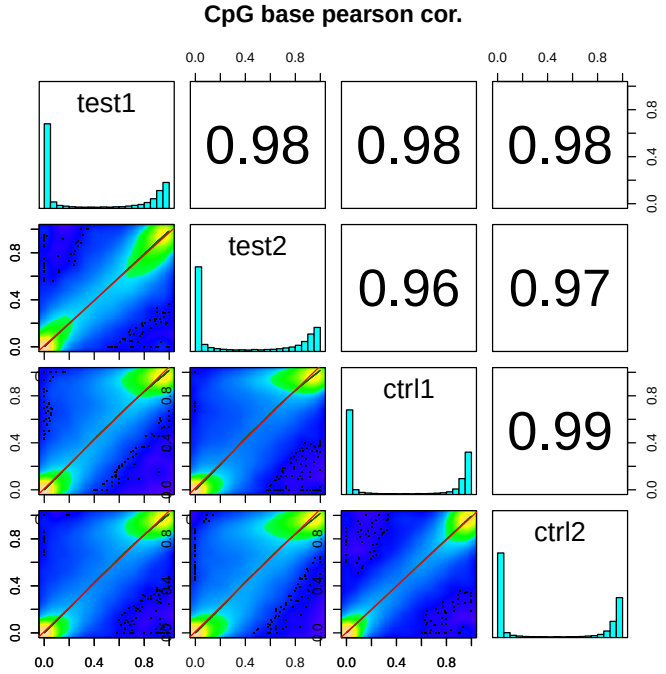
In order to perform further analysis, we need to identify bases that are represented in all samples. We can identify such bases within a *methylBase* object using the *unite()* function.

```
> meth <- unite(filtered.myobj, destrand=FALSE)
> head(meth)

methylBase object with 6 rows
--------------
   chr  start    end strand coverage1 numCs1 numTs1 coverage2 numCs2 numTs2
1 chr1  10497  10497      +       107    101      6       173    154     19
```

```
2 chr1  10525  10525      +       106    105      1       172    162     10
3 chr1  10542  10542      +       107    100      7       172    167      5
4 chr1 544718 544718      +       198     20    178        84      6     78
5 chr1 544739 544739      +       192     19    173        82      8     74
6 chr1 544743 544743      +       198      0    198        84      0     84
  coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
1       133    118     15        97     79     18
2       134    126      8        96     91      5
3       132    121     11        95     91      4
4        50     24     26       156     60     96
5        50     25     25       155     66     89
6        48      0     48       154      0    154
--------------
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg19
context: CpG
treament: 1 1 0 0
resolution: base

> nrow(meth)

[1] 122373
```

We can then generate scatterplots and correlation coefficients for our *methylBase* object using the *getCorrelation()* function.

```
> getCorrelation(meth, plot=T)

          test1     test2     ctrl1     ctrl2
test1 1.0000000 0.9822216 0.9778766 0.9758692
test2 0.9822216 1.0000000 0.9630499 0.9700467
ctrl1 0.9778766 0.9630499 1.0000000 0.9891585
ctrl2 0.9758692 0.9700467 0.9891585 1.0000000
```

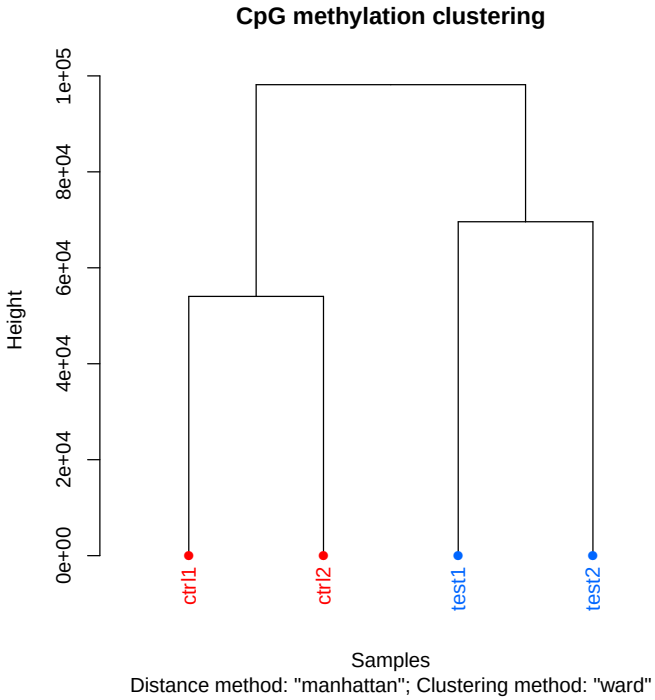


Question 1
How similar are replicate profiles to each other? How similar are profiles from different conditions to each other?

4.4 Cluster methylation profiles

We can cluster samples hierarchically using various distance metrics (e.g. "1-correlation", "euclidean", "manhattan", etc.), as well as the agglomeration methods, to be used in the clustering algorithm (e.g. Ward's method or single/complete linkage).

```
> clusterSamples(meth, dist='manhattan', method='ward', plot=TRUE)

Call:
hclust(d = d, method = HCLUST.METHODS[hclust.method])

Cluster method   : ward
Distance         : manhattan
Number of objects: 4
```



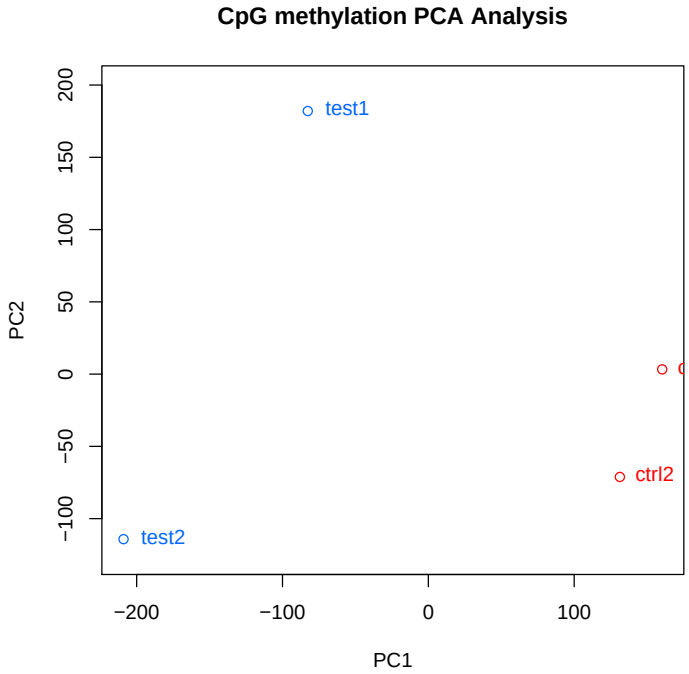
Setting the *plot=FALSE* will return a dendrogram object, which can be manipulated or used in other functions that work with dendrograms.

```
> hc <- clusterSamples(meth, dist='manhattan', method='ward', plot=FALSE)
> summary(hc)
```

```
            Length Class  Mode
merge       6      -none- numeric
height      3      -none- numeric
order       4      -none- numeric
labels      4      -none- character
method      1      -none- character
call        3      -none- call
dist.method 1      -none- character
```

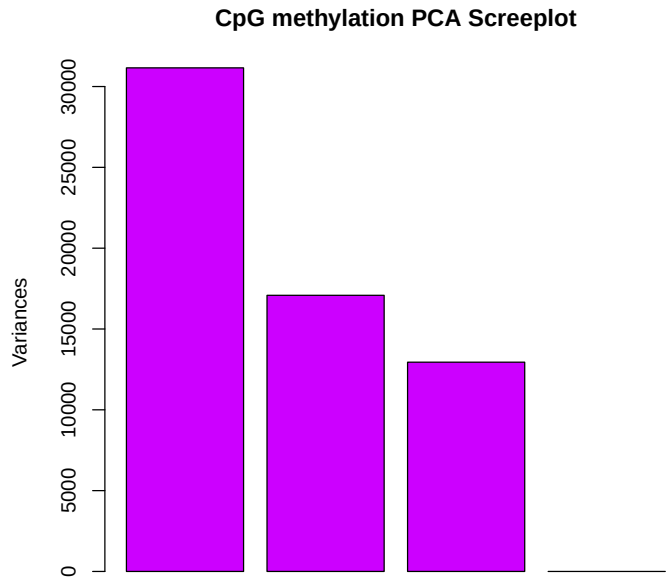
We can also visualize the similarity/difference of methylation profiles by performing a Principal Component Analysis (PCA) and plotting our samples' PC1 (principle component 1) and PC2 (principal component 2) on the x- and y-axes.

```
> PCASamples(meth)
```



Additionally, we can represent the proportion of the total variation in a dataset that is explained by each of the components in our PCA by generating a scree plot. This can help us to assess the importance of the components and identify how many of components are needed to summarise the data.

```
> PCASamples(meth, screeplot=TRUE)
```



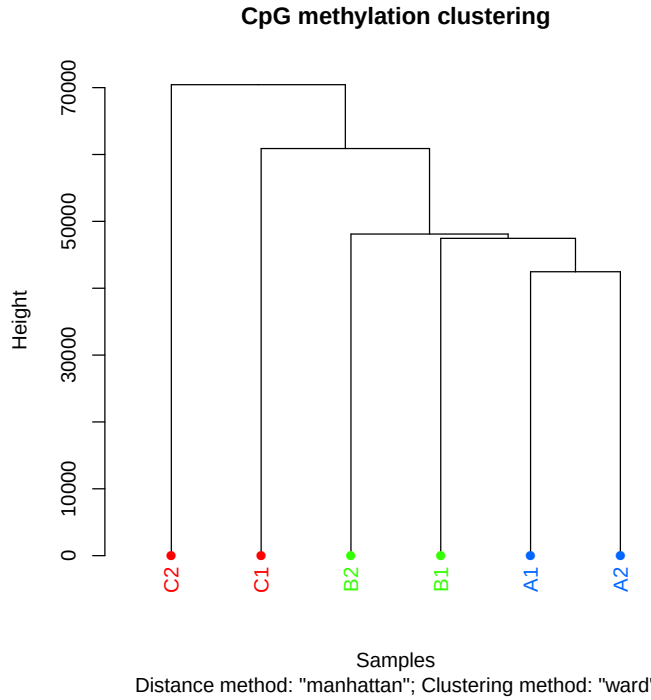
Question 2
Is it possible to cluster samples with more than two conditions?

Let us begin by loading an example dataset that utilizes three conditions.

```
> file.list <- list("condA_sample1.txt", "condA_sample2.txt", "condB_sample1.txt",
+ "condB_sample2.txt", "condC_sample1.txt", "condC_sample2.txt")
> clist <- read(file.list, sample.id=list("A1", "A2", "B1", "B2", "C1", "C2"),
+ assembly="hg19", treatment=c(2, 2, 1, 1, 0, 0), context="CpG")
> newMeth <- unite(clist)
```

We can then cluster our new *methylBase* object using the earlier *clusterSamples()* function.

```
> clusterSamples(newMeth, dist='manhattan', method='ward')

Call:
hclust(d = d, method = HCLUST.METHODS[hclust.method])

Cluster method   : ward
Distance         : manhattan
Number of objects: 6
```



5 Detect differentially methylated CpGs (DMCs)

5.1 Option 1: logistic regression

In logistic regression, information from each sample is specified (the number of methylated Cs and number unmethylated Cs at a given loci) and a logistic regression test will be applied to compare the fraction of methylated Cs across the test and control groups. [1]

```
> myDiff <- calculateDiffMeth(meth, slim=TRUE, weigthed.mean=TRUE, num.cores=1)
> myDiff_20p <- get.methylDiff(myDiff, difference=20, qvalue=0.01)
> head(myDiff_20p)

methylDiff object with 6 rows
--------------
     chr  start    end strand       pvalue       qvalue meth.diff
4   chr1 544718 544718      + 1.110223e-16 1.959537e-14 -31.55684
5   chr1 544739 544739      + 0.000000e+00 0.000000e+00 -34.53623
35  chr1 793769 793769      + 3.606654e-04 4.651105e-03 -23.74517
45  chr1 838050 838050      - 5.241575e-06 1.268631e-04 -22.86320
134 chr1 848739 848739      - 1.471445e-11 1.220422e-09 -29.07735
142 chr1 850976 850976      - 5.471909e-04 6.507645e-03 -24.05019
```

[1] NOTE: Depending on the methyKit version loaded, calculateDiffMeth() may require typo spelling 'weigthed.mean=TRUE' .

```
--------------
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg19
context: CpG
treament: 1 1 0 0
resolution: base
```

```
> nrow(myDiff_20p)
```

```
[1] 5355
```

This will give us all differentially methylated bases. If we want to identify only DMCs that are hyper- or hypomethylated, we can specify 'type.'

```
> myDiff_20p.hyper <- get.methylDiff(myDiff, difference=20, qvalue=0.01, type="hyper")
> nrow(myDiff_20p.hyper)
```

```
[1] 389
```

```
> myDiff_20p.hypo <- get.methylDiff(myDiff, difference=20, qvalue=0.01, type="hypo")
> nrow(myDiff_20p.hypo)
```

```
[1] 4966
```

5.2 Option 2: Fisher's exact test

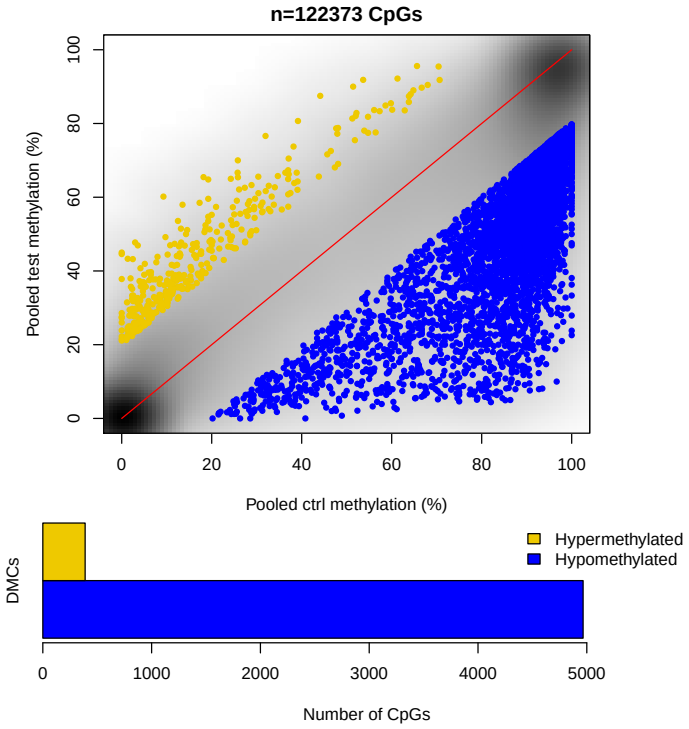
The Fisher's exact test compares the fraction of methylated Cs in test and control samples in the absence of replicates.

```
> pooled.obj <- pool(meth, sample.ids=c("test","control"))
> head(pooled.obj)
```

```
methylBase object with 6 rows
--------------
   chr  start    end strand coverage1 numCs1 numTs1 coverage2 numCs2 numTs2
1 chr1  10497  10497      +       280    255     25       230    197     33
2 chr1  10525  10525      +       278    267     11       230    217     13
3 chr1  10542  10542      +       279    267     12       227    212     15
4 chr1 544718 544718      +       282     26    256       206     84    122
5 chr1 544739 544739      +       274     27    247       205     91    114
6 chr1 544743 544743      +       282      0    282       202      0    202
--------------
sample.ids: test control
destranded FALSE
assembly: hg19
context: CpG
treament: 1 0
resolution: base
```

We can plot the pooled DMC data using a custom *DMCplot()* function:

```
> source("DMCplot.R")
> DMCplot(pooled.obj)
```



The differential methylation calculation speed can be increased substantially by utilizing multiple cores in a machine, if available. Both Fisher's exact test and logistic regression options can use a multiple-core option. The following command will run differential methylation calculation using two cores.

```
> myDiff <- calculateDiffMeth(meth, num.cores=2)
```

6 Assess genomic distribution of DMCs

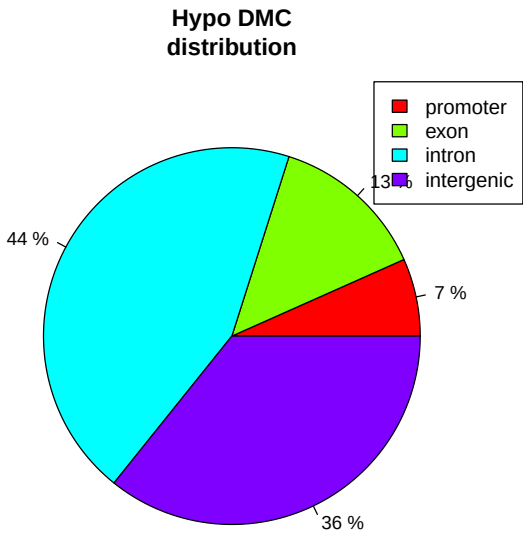
Using gene annotation, we can determine the genomic distribution of our DMCs. To do this, we will need to read the gene annotation from a bed file and annotate our DMCs using this information. This will tell us what percentage of our DMCs occurs within each feature (e.g. promoters, exons, introns). Gene annotations can be fetched using the *GenomicFeatures* package available from Bioconductor.org. In our case, we will load the hg19 annotation from a BED file.

```
> gene.obj <- read.transcript.features("refseq.hg19.bed.txt")
> diffAnnotate.hyper <- annotate.WithGenicParts(myDiff_20p.hyper, gene.obj)
```

```
> diffAnnotate.hypo <- annotate.WithGenicParts(myDiff_20p.hypo, gene.obj)
> getTargetAnnotationStats(diffAnnotate.hypo, percentage=TRUE, precedence=TRUE)

  promoter       exon     intron intergenic
  6.665324  13.431333  44.140153  35.763190
```

We can then plot this distribution using the *plotTargetAnnotation()* function, producing a pie chart.

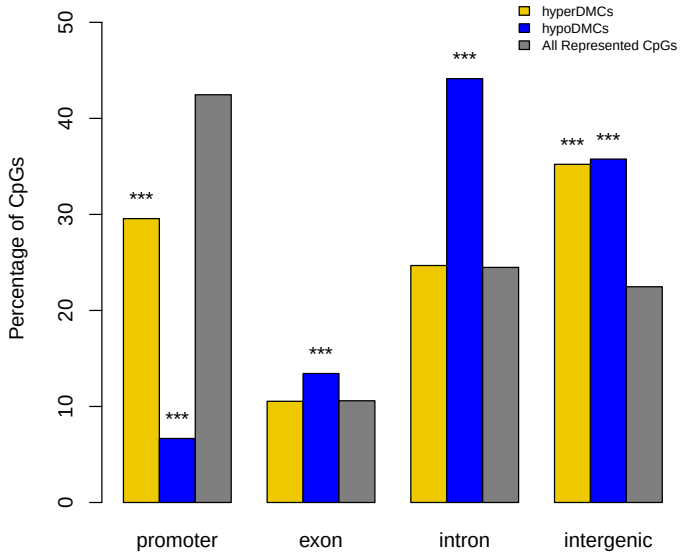
```
> plotTargetAnnotation(diffAnnotate.hypo, precedence=TRUE, main="Hypo DMC
+ distribution")
```



Question 3
How can be assess the significance of this genomic distribution?

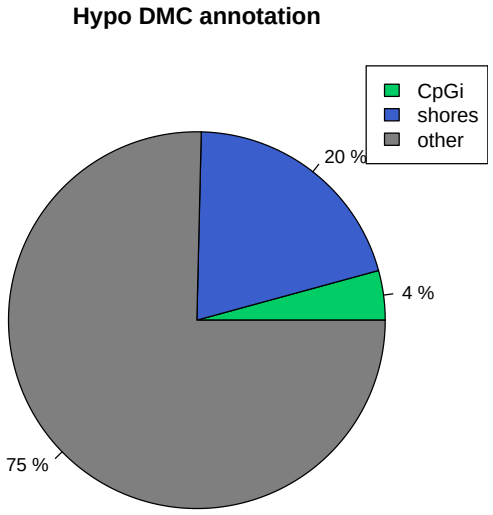
Significance of this distribution versus expectation (the set of all CpGs represented in our RRBS experiment) can be assessed using the binomial test. We can generate a barplot identifying significance using a custom *genomicDistBarplot()* function.

```
> diffAnnotate.allRepr <- annotate.WithGenicParts(myDiff, gene.obj)
> source("genomicDistBarplot.R")
> genomicDistBarplot(diffAnnotate.hyper,diffAnnotate.hypo,diffAnnotate.allRepr)
```



As earlier, the CpG island annotation can be plotted using the *plotTargetAnnotation()* function along with a CpG island annotation BED file. This will produce a pie chart showing what percentage of CpGs are on CpG islands, CpG island shores, and other regions.

```
> cpg.obj <- read.feature.flank("cpgi.hg19.bed.txt", feature.flank.name=c("CpGi",
+ "shores"))
> diffCpGann <- annotate.WithFeature.Flank(myDiff_20p.hypo, cpg.obj$CpGi, cpg.obj$shores,
+ feature.name = "CpGi", flank.name = "shores")
> plotTargetAnnotation(diffCpGann, col=c("springgreen3", "royalblue3", "grey50"), main=
+ "Hypo DMC annotation")
```



7 Calling differentially methylated regions (DMRs)

For some situations, it is desirable to summarize methylation information over tiling windows or over a set of predefined regions (e.g CpG islands), rather than investigating base-pair resolution. The following command tiles the genome with windows 1kb in length and 1kb step-size, summarizing the methylation information on those tiles. Similar to our DMC analysis, this will return a *methylRawList* object, which can subsequently be used with *unite()* and *calculateDiffMeth()* functions to call DMRs.

```
> tiles <- tileMethylCounts(myobj, win.size=1000, step.size=1000)
> head(tiles[[2]])

methylRaw object with 6 rows
--------------
   chr  start    end strand coverage numCs numTs
1 chr1  10001  11000      *      974   885    89
2 chr1  16001  17000      *        1     0     1
3 chr1  17001  18000      *        3     3     0
4 chr1  19001  20000      *        2     2     0
5 chr1  88001  89000      *        3     3     0
6 chr1 131001 132000      *        1     1     0
--------------
sample.id: test2
assembly: hg19
```

```
context: CpG
resolution: region
```

```
> meth.DMR <- unite(tiles, destrand=TRUE)
> myDiff.DMR <- calculateDiffMeth(meth.DMR, slim=TRUE, weigthed.mean=TRUE, num.cores=1)
> myDiff.DMR_20p.hyper <- get.methylDiff(myDiff.DMR, difference=20, qvalue=0.01,
+ type="hyper")
> myDiff.DMR_20p.hypo <- get.methylDiff(myDiff.DMR, difference=20, qvalue=0.01,
+ type="hypo")
```

Question 4

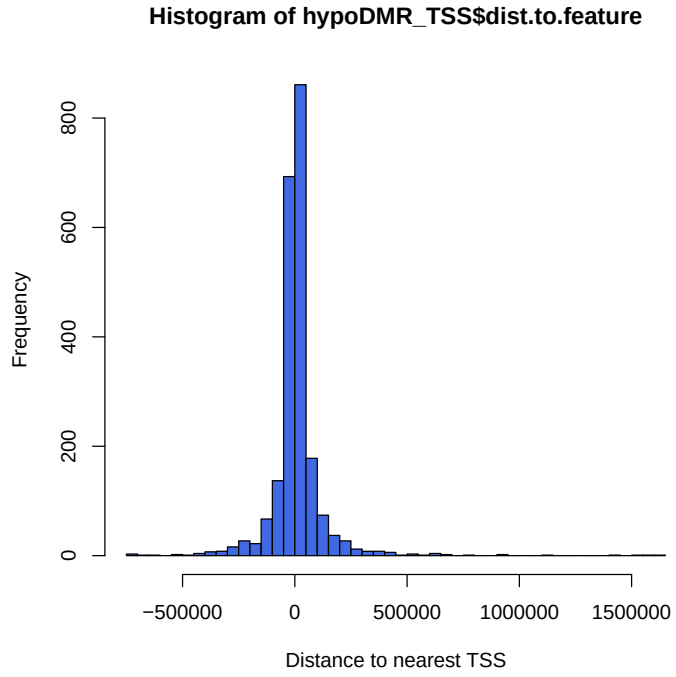
How can we identify which genes are affected by differential methylation?

To discern the biological impact of differential methylation events, it is important to identify genomic context. *methylKit* can annotate DMRs with regard to the nearest TSS using the *getAssociationWithTSS()* function.

```
> diffAnnotate.hypo <- annotate.WithGenicParts(myDiff.DMR_20p.hypo, gene.obj)
> hypoDMR_TSS <- getAssociationWithTSS(diffAnnotate.hypo)
> head(hypoDMR_TSS)
```

```
       target.row dist.to.feature feature.name feature.strand
1419            1           40390    NR_125957              -
1433            2           18183    NR_027055              -
1433.1          3            3183    NR_027055              -
1431            4           13680    NM_015658              -
1432            5            6498 NM_001291367              -
1438            6               0 NM_001291366              -
```

```
> hist(hypoDMR_TSS$dist.to.feature, col="royalblue", xlab="Distance to nearest TSS",
+ breaks=40)
```



To identify genes associated with DMRs, we can also use the *getAssociationWithTSS()* function and subset the results to identify genes that are near DMRs (e.g. TSS within 2kb).

```
> diffAnnotate.hyper <- annotate.WithGenicParts(myDiff.DMR_20p.hyper, gene.obj)
> hyperDMR_TSS <- getAssociationWithTSS(diffAnnotate.hyper)
> hyperDMR_genes <- unique(hyperDMR_TSS[hyperDMR_TSS$dist.to.feature<2000,]$feature.name)
> hypoDMR_genes <- unique(hypoDMR_TSS[hypoDMR_TSS$dist.to.feature<2000,]$feature.name)
```

If expression data is available, we could then assess the effect of differential methylation upon gene expression. Let's load our gene expression values:

```
> fpkm <- read.table("test_ctrl_fpkm.txt", header=T, row.names=1, sep="\t")
> head(fpkm)
```

```
             ctrl      test
AADACL3  0.000000  0.000000
AADACL4  0.000000  0.000000
ABCA4    0.000000  0.000000
ABCB10  10.433847  8.797743
ABCD3   11.736541 16.156906
ABL2     4.236589  2.641502
```

Question 5

Unfortunately, our DMR-associated genes are annotated using RefSeq transcript

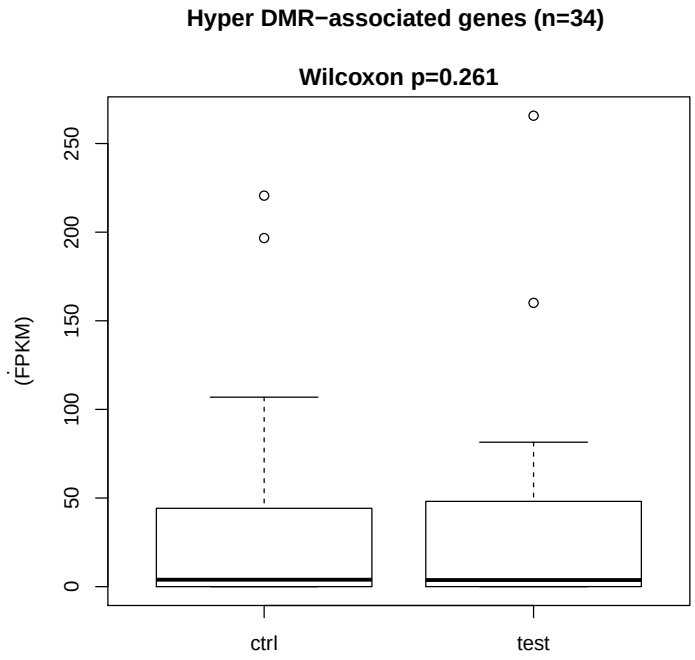
identifiers, but our gene expression data uses common gene symbols. How can we map the RefSeq identifiers to gene symbols?

Mapping reference identifiers between different databases (e.g. Refseq, Ensembl) or to gene symbols can easily be done using the *getBM()* function from the *biomaRt* package.

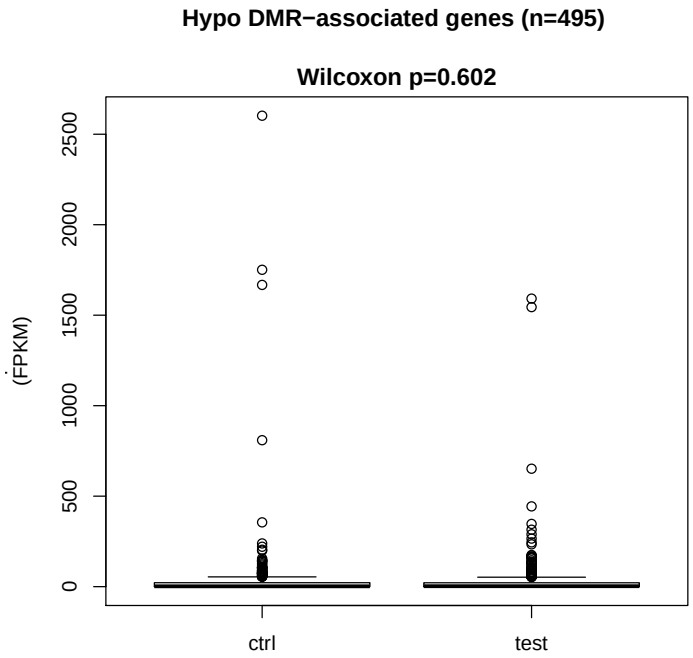
```
> hyperDMR_key <- getBM(attributes=c('hgnc_symbol', 'refseq_mrna'), filters=
+ 'refseq_mrna', values=hyperDMR_genes, mart=useMart("ensembl", dataset=
+ "hsapiens_gene_ensembl"))
> head(hyperDMR_key)

  hgnc_symbol  refseq_mrna
1       GSTM5    NM_000851
2       OPRD1    NM_000911
3      CLSTN1 NM_001009566
4      ZNF648 NM_001009992
5       C4BPB NM_001017365
6        ECE1 NM_001113348

> hypoDMR_key <- getBM(attributes=c('hgnc_symbol', 'refseq_mrna'), filters=
+ 'refseq_mrna', values=hypoDMR_genes, mart=useMart("ensembl", dataset=
+ "hsapiens_gene_ensembl"))
```

We can then use the gene expression data to generate boxplots, allowing us to assess expression differences.

```
> hyperDMR_symbol <- hyperDMR_key$hgnc_symbol[hyperDMR_key$hgnc_symbol%in%rownames(fpkm)]
> boxplot(fpkm[hyperDMR_symbol,], names=c("ctrl", "test"), ylab="Gene expression level
+ (FPKM)", main=paste("Hyper DMR-associated genes (n=", length(hyperDMR_symbol),") \n
+ Wilcoxon p=",signif(wilcox.test(fpkm[hyperDMR_symbol,1], fpkm[hyperDMR_symbol,2],
+ paired=T)$p.value,3),sep=""))
```




```
> hypoDMR_symbol <- hypoDMR_key$hgnc_symbol[hypoDMR_key$hgnc_symbol%in%rownames(fpkm)]
> boxplot(fpkm[hypoDMR_symbol,], names=c("ctrl", "test"), ylab="Gene expression level
+ (FPKM)", main=paste("Hypo DMR-associated genes (n=", length(hypoDMR_symbol),") \n
+ Wilcoxon p=",signif(wilcox.test(fpkm[hypoDMR_symbol,1], fpkm[hypoDMR_symbol,2],
+ paired=T)$p.value,3),sep=""))
```

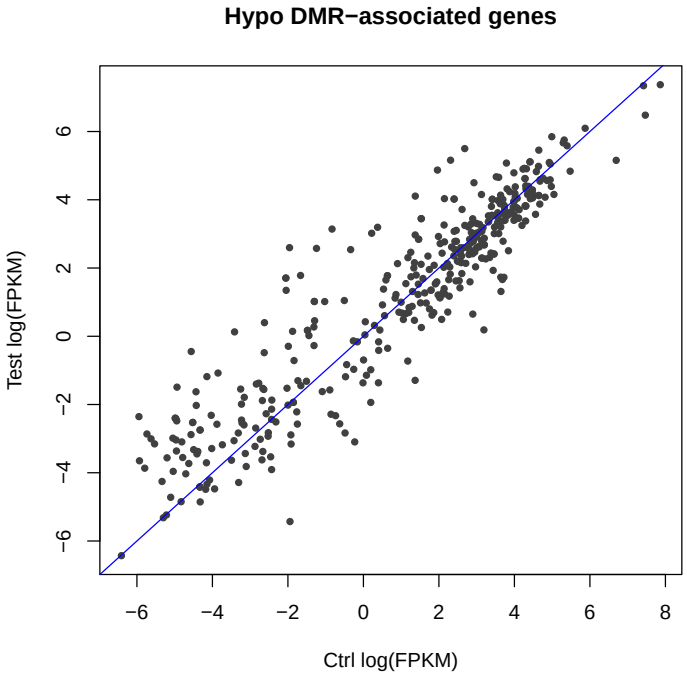


Question 6

Can we identify any potential candidate genes whose expression correlates with Hypo DMR-association?

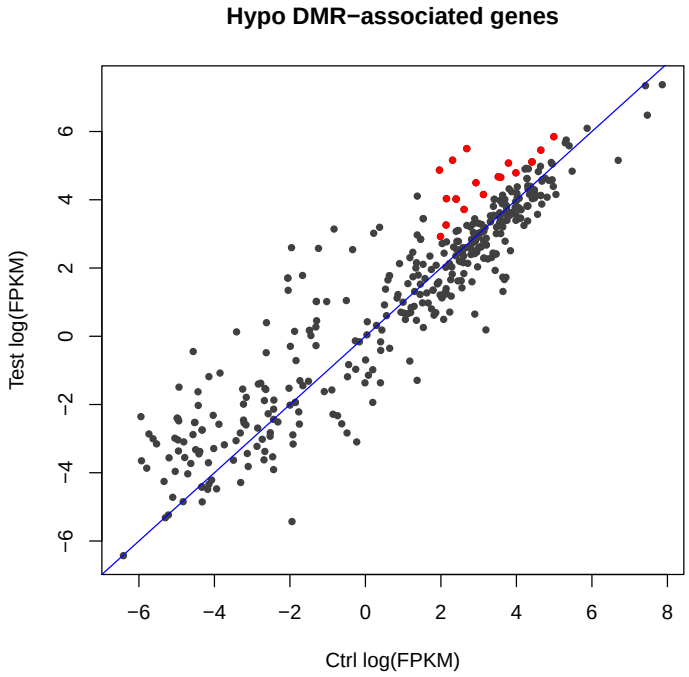
We can see from our boxplots that many DMR-associated genes are lowly expressed and that association with DMRs does not result in significant differences in gene expression. To evalulate the effect on individual genes, we can generate a ctrl vs test scatterplot of log-transformed FPKM values.

```
> plot(log(fpkm[hypoDMR_symbol,]), xlab="Ctrl log(FPKM)", ylab="Test log(FPKM)", pch=20,
+ col="grey25", main="Hypo DMR-associated genes"); abline(a=0, b=1, col="blue")
```



We can then highlight genes in this scatterplot that are reasonably expressed (FPKM>5) and are 2-fold up-regulated in test using the *points()* function.

```
> plot(log(fpkm[hypoDMR_symbol,]), xlab="Ctrl log(FPKM)", ylab="Test log(FPKM)", pch=20,
+ col="grey25", main="Hypo DMR-associated genes"); abline(a=0, b=1, col="blue")
> points(log(fpkm[hypoDMR_symbol,][fpkm[hypoDMR_symbol,2]/fpkm[hypoDMR_symbol,1]>2 &
+ fpkm[hypoDMR_symbol,1]>5,]), pch=20, col="red")
```



Question 7
How can we find the names of these candidate genes?

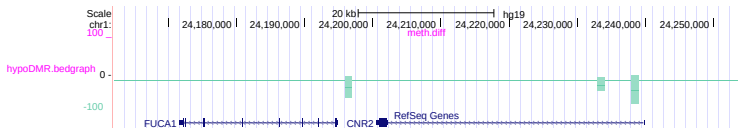
```
> rownames(fpkm[hypoDMR_symbol,][fpkm[hypoDMR_symbol,2]/fpkm[hypoDMR_symbol,1]>2 &
+ fpkm[hypoDMR_symbol,1]>5,])

 [1] "FUCA1"    "SEMA4A"   "PTP4A2"   "BCAR3"    "ACADM"    "RNF19B"
 [7] "PRDX1"    "S100A10"  "SLC2A5"   "BCAR3.1"  "AIM2"     "PMVK"
[13] "ICMT"     "CLIC4"    "IKBKE"    "NUCKS1"   "PTP4A2.1" "LBR"
[19] "STMN1"
```

7.1 Visualize DMRs using UCSC Genome Browser

We can create a bedgraph file using the *bedgraph()* function in the *methylKit* package.

```
> bedgraph(myDiff.DMR_20p.hypo, file="hypoDMR.bedgraph", 'meth.diff')
```

This can then be uploaded to the UCSC genome browser. The bar height will correspond to the methylation differnce between test and ctrl samples.



Alternatively we could use the *rtracklayer* package, we can upload a data track to the UCSC Genome Browser and manipulate the genomic views (for more info, visit http://www.bioconductor.org/packages/release/bioc/html/rtracklayer.html)