methylKit: User Guide

Altuna Akalin*
altuna.akalin@mdc-berlin.de

October 20, 2014

Contents

*Author of the vignette. See Acknowledgements for a list of contributors.

1 Introduction

In this manual, we will show how to use the methylKit package. methylKit is an R package for analysis and annotation of DNA methylation information obtained by high-throughput bisulfite sequencing. The package is designed to deal with sequencing data from RRBS and its variants. But it can potentially handle whole-genome bisulfite sequencing data if proper input format is provided.

1.1 DNA methylation

DNA methylation in vertebrates typically occurs at CpG dinucleotides, however non-CpG Cs are also methylated in certain tissues such as embryonic stem cells. DNA methylation can act as an epigenetic control mechanism for gene regulation. Methylation can hinder binding of transcription factors and/or methylated bases can be bound by methyl-binding-domain proteins which can recruit chromatin remodeling factors. In both cases, the transcription of the regulated gene will be effected. In addition, aberrant DNA methylation patterns have been associated with many human malignancies and can be used in a predictive manner. In malignant tissues, DNA is either hypo-methylated or hyper-methylated compared to the normal tissue. The location of hyper- and hypo-methylated sites gives a distinct signature to many diseases. Traditionally, hypo-methylation is associated with gene transcription (if it is on a regulatory region such as promoters) and hyper-methylation is associated with gene repression.

1.2 High-throughput bisulfite sequencing

Bisulfite sequencing is a technique that can determine DNA methylation patterns. The major difference from regular sequencing experiments is that, in bisulfite sequencing DNA is treated with bisulfite which converts cytosine residues to uracil, but leaves 5-methylcytosine residues unaffected. By sequencing and aligning those converted DNA fragments it is possible to call methylation status of a base. Usually, the methylation status of a base determined by a high-throughput bisulfite sequencing will not be a binary score, but it will be a percentage. The percentage simply determines how many of the bases that are aligning to a given cytosine location in the genome have actual C bases in the reads. Since bisulfite treatment leaves methylated Cs intact, that percentage will give us percent methylation score on that base. The reasons why we will not get a binary response are 1) the probable sequencing errors in high-throughput sequencing experiments 2) incomplete bisulfite conversion 3) (and a more likely scenario) is heterogeneity of samples and heterogeneity of paired chromosomes from the same sample

2 Basics

2.1 Reading the methylation call files

We start by reading in the methylation call data from bisulfite sequencing with `read` function. Reading in the data this way will return a methylRawList object which stores methylation information per sample for each covered base. The methylation call files are basically text files that contain percent methylation score per base. A typical methylation call file looks like this:

```
        chrBase   chr    base strand coverage freqC  freqT
1 chr21.9764539 chr21 9764539      R       12 25.00  75.00
2 chr21.9764513 chr21 9764513      R       12  0.00 100.00
3 chr21.9820622 chr21 9820622      F       13  0.00 100.00
4 chr21.9837545 chr21 9837545      F       11  0.00 100.00
5 chr21.9849022 chr21 9849022      F      124 72.58  27.42
```

Most of the time bisulfite sequencing experiments have test and control samples. The test samples can be from a disease tissue while the control samples can be from a healthy tissue. You can read a set of methylation call files that have test/control conditions giving `treatment` vector option. For sake of subsequent analysis, file.list, sample.id and treatment option should have the same order. In the following example, first two files are have the sample ids "test1" and "test2" and as determined by treatment vector they belong to the same group. The third and fourth files have sample ids "ctrl1" and "ctrl2" and they belong to the same group as indicated by the treatment vector.

```
> library(methylKit)
> file.list=list( system.file("extdata", "test1.myCpG.txt", package = "methylKit"),
```

```
+                 system.file("extdata", "test2.myCpG.txt", package = "methylKit"),
+                 system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
+                 system.file("extdata", "control2.myCpG.txt", package = "methylKit") )
> # read the files to a methylRawList object: myobj
> myobj=read(file.list,
+            sample.id=list("test1","test2","ctrl1","ctrl2"),
+            assembly="hg18",
+            treatment=c(1,1,0,0),
+            context="CpG"
+            )
>
>
```

2.2 Reading the methylation calls from sorted Bismark alignments

Alternatively, methylation percentage calls can be calculated from sorted SAM file(s) from Bismark aligner and read-in to the memory. Bismark is a popular aligner for bisulfite sequencing reads [1]. `read.bismark` function is designed to read-in Bismark SAM files as `methylRaw` or `methylRawList` objects which store per base methylation calls. SAM files must be sorted by chromosome and read position columns, using 'sort' command in unix-like machines will accomplish such a sort easily.

The following command reads a sorted SAM file and creates a `methylRaw` object for CpG methylation.The user has the option to save the methylation call files to a folder given by `save.folder` option. The saved files can be read-in using the `read` function when needed.

```
> my.methRaw=read.bismark(
+            location=system.file("extdata", "test.fastq_bismark.sorted.min.sam",
+                                 package = "methylKit"),
+              sample.id="test1",assembly="hg18",read.context="CpG",save.folder=getwd())
```

It is also possible to read multiple SAM files at the same time, check `read.bismark` documentation.

2.3 Descriptive statistics on samples

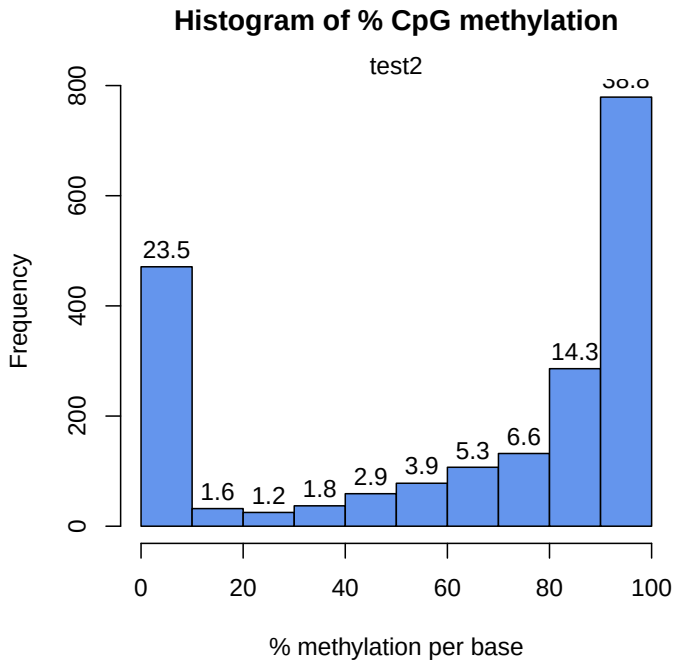
Since we read the methylation data now, we can check the basic stats about the methylation data such as coverage and percent methylation. We now have a `methylRawList` object which contains methylation information per sample. The following command prints out percent methylation statistics for second sample: "test2"

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

```
methylation statistics per base
summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   20.00   82.79   63.17   94.74  100.00
percentiles:
       0%       10%       20%       30%       40%       50%       60%       70%
  0.00000   0.00000   0.00000  48.38710  70.00000  82.78556  90.00000  93.33333
      80%       90%       95%       99%     99.5%     99.9%      100%
 96.42857 100.00000 100.00000 100.00000 100.00000 100.00000 100.00000
```

The following command plots the histogram for percent methylation distribution.The figure below is the histogram and numbers on bars denote what percentage of locations are contained in that bin. Typically, percent methylation histogram should have two peaks on both ends. In any given cell, any given base are either methylated or not. Therefore, looking at many cells should yield a similar pattern where we see lots of locations with high methylation and lots of locations with low methylation.

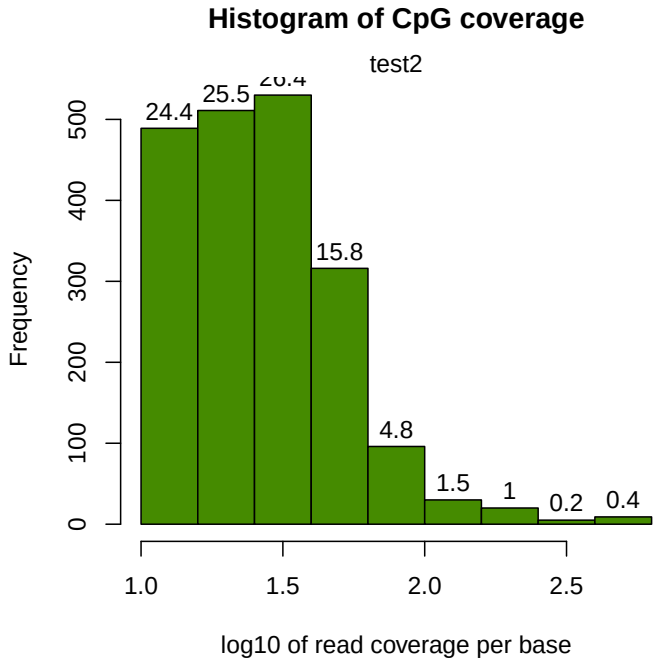
```
> getMethylationStats(myobj[[2]],plot=T,both.strands=F)
```



We can also plot the read coverage per base information in a similar way, again numbers on bars denote what percentage of locations are contained in

that bin. Experiments that are highly suffering from PCR duplication bias will have a secondary peak towards the right hand side of the histogram.

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



2.4 Filtering samples based on read coverage

It might be useful to filter samples based on coverage. Particularly, if our samples are suffering from PCR bias it would be useful to discard bases with very high read coverage. Furthermore, we would also like to discard bases that have low read coverage, a high enough read coverage will increase the power of the statistical tests. The code below filters a `methylRawList` and discards bases that have coverage below 10X and also discards the bases that have more than 99.9th percentile of coverage in each sample.

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

3 Comparative analysis

3.1 Merging samples

In order to do further analysis, we will need to get the bases covered in all samples. The following function will merge all samples to one object for base-pair locations that are covered in all samples. Setting `destrand`=TRUE (the default is FALSE) will merge reads on both strands of a CpG dinucleotide. This provides better coverage, but only advised when looking at CpG methylation (for CpH methylation this will cause wrong results in subsequent analyses). In addition, setting `destrand`=TRUE will only work when operating on base-pair resolution, otherwise setting this option TRUE will have no effect. The `unite()` function will return a `methylBase` object which will be our main object for all comparative analysis. The `methylBase` object contains methylation information for regions/bases that are covered in all samples.

```
> meth=unite(myobj, destrand=FALSE)
```

Let us take a look at the data content of methylBase object:

```
> head(meth)
```

```
methylBase object with 6 rows
--------------
    chr   start     end strand coverage1 numCs1 numTs1 coverage2 numCs2 numTs2
1 chr21 9853296 9853296      +        17     10      7        333    268     65
2 chr21 9853326 9853326      +        17     12      5        329    249     79
3 chr21 9860126 9860126      +        39     38      1         83     78      5
4 chr21 9906604 9906604      +        68     42     26        111     97     14
5 chr21 9906616 9906616      +        68     52     16        111    104      7
6 chr21 9906619 9906619      +        68     59      9        111    109      2
  coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
1        18     16      2       395    341     54
2        16     14      2       379    284     95
3        83     83      0        41     40      1
4        23     18      5        37     33      4
5        23     14      9        37     27     10
6        22     18      4        37     29      8
--------------
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg18
context: CpG
treament: 1 1 0 0
resolution: base
```

By default, `unite` function produces bases/regions covered in all samples. That requirement can be relaxed using "min.per.group" option in `unite` function.

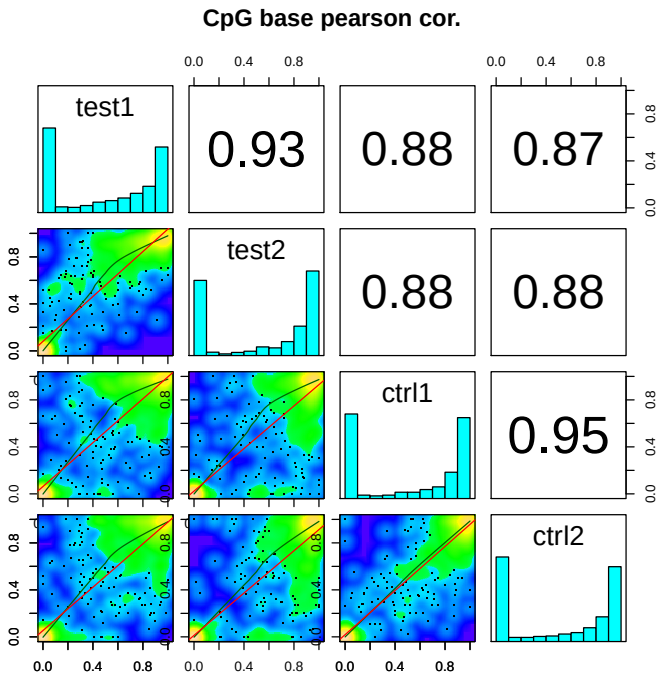
```
> # creates a methylBase object. Only CpGs covered at least in 1 sample per group will be re
> # there were two groups defined by the treatment vector given during the creation of myobj
> meth.min=unite(myobj,min.per.group=1L)
```

3.2 Sample Correlation

We can check the correlation between samples using `getCorrelation`. This function will either plot scatter plot and correlation coefficients or just print a correlation matrix

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

```
          test1     test2     ctrl1     ctrl2
test1 1.0000000 0.9252530 0.8767865 0.8737509
test2 0.9252530 1.0000000 0.8791864 0.8801669
ctrl1 0.8767865 0.8791864 1.0000000 0.9465369
ctrl2 0.8737509 0.8801669 0.9465369 1.0000000
```



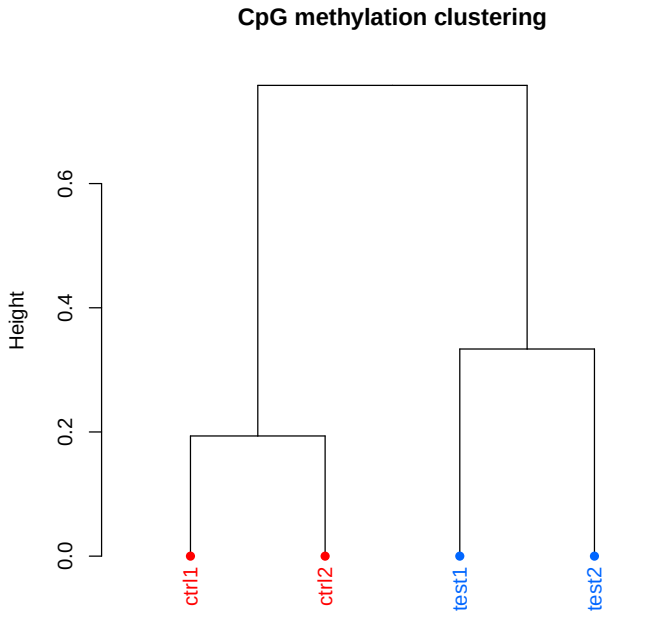
3.3 Clustering samples

We can cluster the samples based on the similarity of their methylation profiles. The following function will cluster the samples and draw a dendrogram.

```
> clusterSamples(meth, dist="correlation", method="ward", plot=TRUE)

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

Cluster method   : ward.D
Distance         : pearson
Number of objects: 4
```

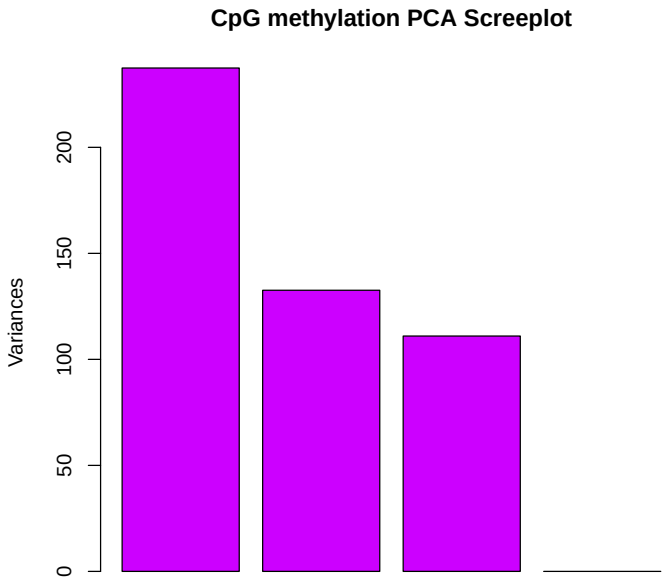


Setting the plot=FALSE will return a dendrogram object which can be manipulated by users or fed in to other user functions that can work with dendrograms.

```
> hc = clusterSamples(meth, dist="correlation", method="ward", plot=FALSE)
```

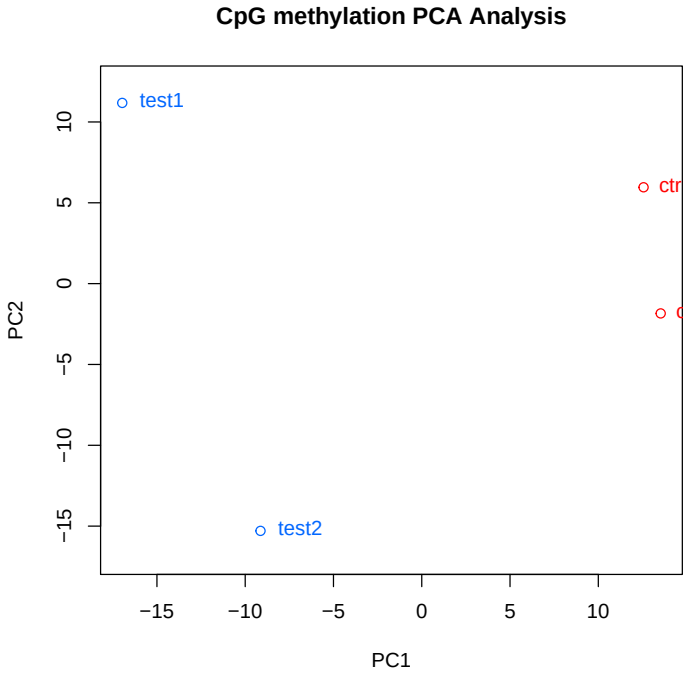
We can also do a PCA analysis on our samples. The following function will plot a scree plot for importance of components.

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



We can also plot PC1 and PC2 axis and a scatter plot of our samples on those axis which will reveal how they cluster.

```
> PCASamples(meth)
```



3.4 Batch effects

We have implemented some rudimentary functionality for batch effect control. You can check which one of the principal components are statistically associated with the potential batch effects such as batch processing dates, age of subjects, sex of subjects using `assocComp`. The function gets principal components from the percent methylation matrix derived from the input `methylBase` object, and checks for association. The tests for association are either via Kruskal-Wallis test or Wilcoxon test for categorical attributes and correlation test for numerical attributes for samples such as age. If you are convinced that some principal components are accounting for batch effects, you can remove those principal components from your data using `removeComp`.

```
> # make some batch data frame
> # this is a bogus data frame
> # we don't have batch information
> # for the example data
> sampleAnnotation=data.frame(batch_id=c("a","a","b","b"),
+                             age=c(19,34,23,40))
> as=assocComp(mBase=meth,sampleAnnotation)
> as
```

```
$pcs
             PC1        PC2         PC3         PC4
test1 -0.4978699 -0.5220504  0.68923849 -0.06737363
test2 -0.4990924 -0.4805506 -0.71827964  0.06365693
ctrl1 -0.5016543  0.4938800  0.08068700  0.70563101
ctrl2 -0.5013734  0.5026102 -0.05014261 -0.70249091

$vars
[1] 92.271885  4.525328  1.870950  1.331837

$association
               PC1       PC2       PC3       PC4
batch_id 0.3333333 0.3333333 1.0000000 1.0000000
age      0.5864358 0.6794346 0.3140251 0.3467957
```

```
> # construct a new object by removing the first pricipal component
> # from percent methylation value matrix
> newObj=removeComp(meth,comp=1)
```

In addition to the methods described above, if you have used other ways to correct for batch effects and obtained a corrected percent methylation matrix, you can use `reconstruct` function to reconstruct a corrected `methylBase` object. Users have to supply a corrected percent methylation matrix and `methylBase` object (where the uncorrected percent methylation matrix obtained from) to the `reconstruct` function. Corrected percent methylation matrix should have the same row and column order as the original percent methylation matrix. All of these functions described in this section work on a `methylBase` object that does not have missing values (that means all bases in methylBase object should have coverage in all samples).

```
> mat=percMethylation(meth)
> # do some changes in the matrix
> # this is just a toy example
> # ideally you want to correct the matrix
> # for batch effects
> mat[mat==100]=80
> # reconstruct the methylBase from the corrected matrix
> newobj=reconstruct(mat,meth)
>
```

3.5 Tiling windows analysis

For some situations, it might be desirable to summarize methylation information over tiling windows rather than doing base-pair resolution analysis. `methylKit` provides functionality to do such analysis. The function below tiles the genome with windows 1000bp length and 1000bp step-size and summarizes the methylation information on those tiles. In this case, it returns a `methylRawList` object

which can be fed into `unite` and `calculateDiffMeth` functions consecutively to get differentially methylated regions. The tilling function adds up C and T counts from each covered cytosine and returns a total C and T count for each tile.

```
> tiles=tileMethylCounts(myobj,win.size=1000,step.size=1000)
> head(tiles[[1]],3)
```

```
methylRaw object with 3 rows
--------------
    chr   start     end strand coverage numCs numTs
1 chr21 9764001 9765000      *       24     3    21
2 chr21 9820001 9821000      *       13     0    13
3 chr21 9837001 9838000      *       11     0    11
--------------
sample.id: test1
assembly: hg18
context: CpG
resolution: region
```

3.6 Finding differentially methylated bases or regions

`calculateDiffMeth()` function is the main function to calculate differential methylation. Depending on the sample size per each set it will either use Fisher's exact or logistic regression to calculate P-values. P-values will be adjusted to Q-values using SLIM method [2].

```
> myDiff=calculateDiffMeth(meth)
```

After q-value calculation, we can select the differentially methylated regions/bases based on q-value and percent methylation difference cutoffs. Following bit selects the bases that have q-value<0.01 and percent methylation difference larger than 25%. If you specify `type="hyper"` or `type="hypo"` options, you will get hyper-methylated or hypo-methylated regions/bases.

```
> # get hyper methylated bases
> myDiff25p.hyper=get.methylDiff(myDiff,difference=25,qvalue=0.01,type="hyper")
> #
> # get hypo methylated bases
> myDiff25p.hypo=get.methylDiff(myDiff,difference=25,qvalue=0.01,type="hypo")
> #
> #
> # get all differentially methylated bases
> myDiff25p=get.methylDiff(myDiff,difference=25,qvalue=0.01)
```

We can also visualize the distribution of hypo/hyper-methylated bases/regions per chromosome using the following function. In this case, the example set includes only one chromosome. The `list` shows percentages of hypo/hyper methylated bases over all the covered bases in a given chromosome.

```
> diffMethPerChr(myDiff,plot=FALSE,qvalue.cutoff=0.01, meth.cutoff=25)

$diffMeth.per.chr
    chr number.of.hypomethylated percentage.of.hypomethylated
1 chr21                       59                     6.126687
  number.of.hypermethylated percentage.of.hypermethylated
1                        75                     7.788162

$diffMeth.all
  percentage.of.hypermethylated number.of.hypermethylated
1                      7.788162                        75
  percentage.of.hypomethylated number.of.hypomethylated
1                    6.126687                        59
```

3.6.1 Finding differentially methylated bases using multiple-cores

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 based test are able to use multiple-core option.
The following piece of code will run differential methylation calculation using 2 cores.

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

4 Annotating differentially methylated bases or regions

We can annotate our differentially methylated regions/bases based on gene annotation. In this example, we read the gene annotation from a bed file and annotate our differentially methylated regions with that information. This will tell us what percentage of our differentially methylated regions are on promoters/introns/exons/intergenic region. Similar gene annotation can be fetched using `GenomicFeatures` package available from Bioconductor.org.

```
> gene.obj=read.transcript.features(system.file("extdata", "refseq.hg18.bed.txt",
+                                               package = "methylKit"))
> #
> # annotate differentially methylated Cs with promoter/exon/intron using annotation data
> #
> annotate.WithGenicParts(myDiff25p,gene.obj)

summary of target set annotation with genic parts
133 rows in target set
--------------
--------------
percentage of target features overlapping with annotation :
```

```
  promoter       exon     intron intergenic
  27.81955   15.03759   34.58647   57.14286


percentage of target features overlapping with annotation (with promoter>exon>intron precede
  promoter       exon     intron intergenic
  27.81955    0.00000   15.03759   57.14286


percentage of annotation boundaries with feature overlap :
  promoter       exon     intron
0.28604119 0.02683483 0.17068273


summary of distances to the nearest TSS :
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      5     828   45160   52030   94640  313500
```

Similarly, we can read the CpG island annotation and annotate our differentially methylated bases/regions with them.

```
> # read the shores and flanking regions and name the flanks as shores
> # and CpG islands as CpGi
> cpg.obj=read.feature.flank(system.file("extdata", "cpgi.hg18.bed.txt",
+                                        package = "methylKit"),
+                            feature.flank.name=c("CpGi","shores"))
> #
> #
> diffCpGann=annotate.WithFeature.Flank(myDiff25p,cpg.obj$CpGi,cpg.obj$shores,
+                                       feature.name="CpGi",flank.name="shores")
```

4.1 Regional analysis

We can also summarize methylation information over a set of defined regions such as promoters or CpG islands. The function below summarizes the methylation information over a given set of promoter regions and outputs a `methylRaw` or `methylRawList` object depending on the input.

```
> promoters=regionCounts(myobj,gene.obj$promoters)
> head(promoters[[1]])

methylRaw object with 6 rows
--------------
    chr    start      end strand coverage numCs numTs
1 chr21 17806094 17808094      +     3722    19  3703
2 chr21 10119796 10121796      -     1725  1171   554
3 chr21 10011791 10013791      -     7953  6662  1290
```

```
4 chr21 10119808 10121808      -     1725  1171   554
5 chr21 15357997 15359997      -    15573    26 15544
6 chr21 16023366 16025366      +    11694    16 11676
--------------
sample.id: test1
assembly: hg18
context: CpG
resolution: region
```

4.2 Convenience functions for annotation objects

After getting the annotation of differentially methylated regions, we can get the distance to TSS and nearest gene name using the `getAssociationWithTSS` function.

```
> diffAnn=annotate.WithGenicParts(myDiff25p,gene.obj)
> # target.row is the row number in myDiff25p
> head(getAssociationWithTSS(diffAnn))
```

```
     target.row dist.to.feature feature.name feature.strand
60            1          106111    NM_199260              -
60.1          2          106098    NM_199260              -
60.2          3          106092    NM_199260              -
60.3          4          105919    NM_199260              -
60.4          5           85265    NM_199260              -
60.5          6           68287    NM_199260              -
```

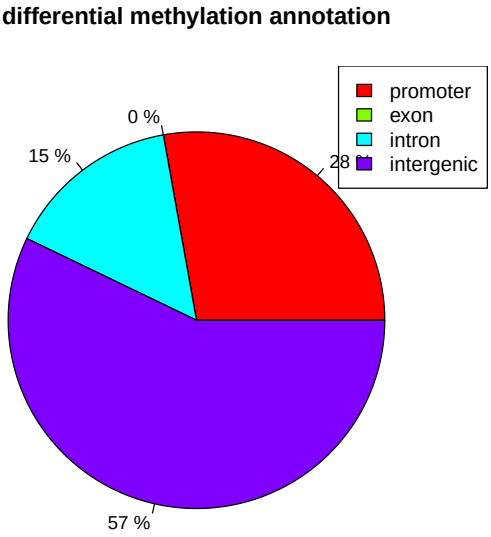
It is also desirable to get percentage/number of differentially methylated regions that overlap with intron/exon/promoters

```
> getTargetAnnotationStats(diffAnn,percentage=TRUE,precedence=TRUE)
```

```
  promoter       exon     intron intergenic
  27.81955    0.00000   15.03759   57.14286
```

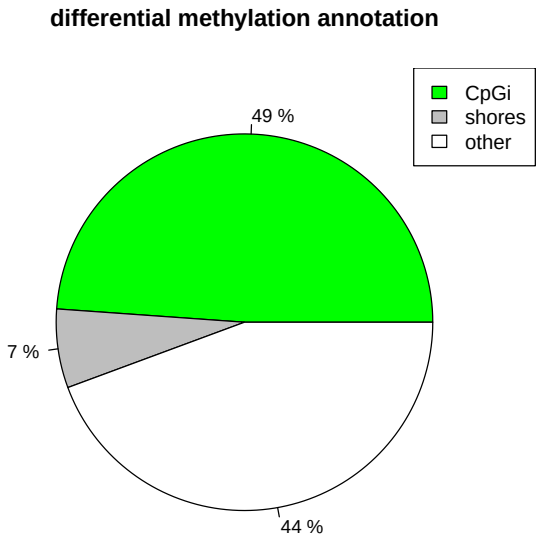
We can also plot the percentage of differentially methylated bases overlapping with exon/intron/promoters

```
> plotTargetAnnotation(diffAnn,precedence=TRUE,
+     main="differential methylation annotation")
```



We can also plot the CpG island annotation the same way. The plot below shows what percentage of differentially methylated bases are on CpG islands, CpG island shores and other regions.

```
> plotTargetAnnotation(diffCpGann,col=c("green","gray","white"),
+         main="differential methylation annotation")
```



It might be also useful to get percentage of intron/exon/promoters that overlap with differentially methylated bases.

```
> getFeatsWithTargetsStats(diffAnn,percentage=TRUE)

  promoter       exon     intron
0.28604119 0.02683483 0.17068273
```

5 methylKit convenience functions

5.1 coercion

Most `methylKit` objects (methylRaw,methylBase and methylDiff) can be coerced to `GRanges` objects from `GenomicRanges` package. Coercing methylKit objects to `GRanges` will give users additional flexibility when customizing their analyses.

```
> class(meth)

[1] "methylBase"
attr(,"package")
[1] "methylKit"
```

```
> as(meth,"GRanges")

GRanges object with 963 ranges and 12 metadata columns:
        seqnames              ranges strand   | coverage1    numCs1    numTs1
           <Rle>           <IRanges>  <Rle>   | <integer> <numeric> <numeric>
    [1]    chr21   [9853296, 9853296]      +   |        17        10         7
    [2]    chr21   [9853326, 9853326]      +   |        17        12         5
    [3]    chr21   [9860126, 9860126]      +   |        39        38         1
    [4]    chr21   [9906604, 9906604]      +   |        68        42        26
    [5]    chr21   [9906616, 9906616]      +   |        68        52        16
    ...      ...                 ...    ... ...       ...       ...       ...
  [959]    chr21 [19855690, 19855690]      +   |        27        26         1
  [960]    chr21 [19855706, 19855706]      +   |        27        27         0
  [961]    chr21 [19855711, 19855711]      +   |        18        18         0
  [962]    chr21 [19943653, 19943653]      +   |        12        12         0
  [963]    chr21 [19943695, 19943695]      +   |        12        11         1
        coverage2    numCs2    numTs2 coverage3    numCs3    numTs3 coverage4
        <integer> <numeric> <numeric> <integer> <numeric> <numeric> <integer>
    [1]       333       268        65        18        16         2       395
    [2]       329       249        79        16        14         2       379
    [3]        83        78         5        83        83         0        41
    [4]       111        97        14        23        18         5        37
    [5]       111       104         7        23        14         9        37
    ...       ...       ...       ...       ...       ...       ...       ...
  [959]        19        17         2        34        34         0        12
  [960]        19        19         0        34        34         0        12
  [961]        18        15         3        34        34         0        12
  [962]        32        30         2        26        25         1        24
  [963]        32        32         0        26        26         0        27
           numCs4    numTs4
        <numeric> <numeric>
    [1]       341        54
    [2]       284        95
    [3]        40         1
    [4]        33         4
    [5]        27        10
    ...       ...       ...
  [959]        12         0
  [960]        11         1
  [961]        12         0
  [962]        22         2
  [963]        24         3
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> class(myDiff)
```

```
[1] "methylDiff"
attr(,"package")
[1] "methylKit"
```

```
> as(myDiff,"GRanges")
```

```
GRanges object with 963 ranges and 2 metadata columns:
        seqnames               ranges strand   |              qvalue
           <Rle>            <IRanges>  <Rle>   |           <numeric>
    [1]    chr21   [9853296, 9853296]      +   |  0.0215658126063664
    [2]    chr21   [9853326, 9853326]      +   |   0.592173028310101
    [3]    chr21   [9860126, 9860126]      +   |  0.0697808391745449
    [4]    chr21   [9906604, 9906604]      +   |   0.259453089661393
    [5]    chr21   [9906616, 9906616]      +   | 0.00432220069940899
    ...      ...                  ...     ... ...                 ...
  [959]    chr21 [19855690, 19855690]      +   |  0.0660274910679262
  [960]    chr21 [19855706, 19855706]      +   |   0.282958728725395
  [961]    chr21 [19855711, 19855711]      +   |  0.0446545923565476
  [962]    chr21 [19943653, 19943653]      +   |   0.592173028310101
  [963]    chr21 [19943695, 19943695]      +   |   0.396045532748492
                 meth.diff
                 <numeric>
    [1]    -7.01210653753026
    [2] -0.00951196312286129
    [3]    -4.11158117398202
    [4]     -7.3463687150838
    [5]     18.8175046554935
    ...                  ...
  [959]    -6.52173913043478
  [960]     2.17391304347827
  [961]    -8.33333333333333
  [962]     1.45454545454545
  [963]      3.3876500857633
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

5.2 select

We can also select rows from `methylRaw`, `methylBase` and `methylDiff` objects with "select" function. An appropriate methylKit object will be returned as a result of "select" function. Or you can use the "[" notation to subset the methylKit objects.

```
> select(meth,1:5) # get first 10 rows of a methylBase object
```

```
methylBase object with 5 rows
--------------
```

```
    chr   start     end strand coverage1 numCs1 numTs1 coverage2 numCs2 numTs2
1 chr21 9853296 9853296      +        17     10      7        333    268     65
2 chr21 9853326 9853326      +        17     12      5        329    249     79
3 chr21 9860126 9860126      +        39     38      1         83     78      5
4 chr21 9906604 9906604      +        68     42     26        111     97     14
5 chr21 9906616 9906616      +        68     52     16        111    104      7
  coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
1        18     16      2       395    341     54
2        16     14      2       379    284     95
3        83     83      0        41     40      1
4        23     18      5        37     33      4
5        23     14      9        37     27     10
--------------
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg18
context: CpG
treament: 1 1 0 0
resolution: base

> myDiff[21:25,] # get 5 rows of a methylDiff object

methylDiff object with 5 rows
--------------
     chr   start     end strand       pvalue       qvalue  meth.diff
21 chr21 9913543 9913543      + 1.254379e-02 2.632641e-02 -13.343109
22 chr21 9913575 9913575      + 2.755448e-01 3.161628e-01  -5.442623
23 chr21 9927527 9927527      + 1.120126e-07 9.257475e-07 -46.109840
24 chr21 9944505 9944505      + 0.000000e+00 0.000000e+00 -51.017943
25 chr21 9944663 9944663      - 1.790779e-05 7.678302e-05 -28.099911
--------------
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg18
context: CpG
treament: 1 1 0 0
resolution: base
```

5.3 reorganize

`methylBase` and `methylRawList` can be reorganized by `reorganize` function. The function can subset the objects based on provided sample ids, it also creates a new treatment vector determining which samples belong to which group. Order of sample ids should match the treatment vector order.

```
> # creates a new methylRawList object
> myobj2=reorganize(myobj,sample.ids=c("test1","ctrl2"),treatment=c(1,0) )
```

```
> # creates a new methylBase object
> meth2 =reorganize(meth,sample.ids=c("test1","ctrl2"),treatment=c(1,0) )
```

5.4 percMethylation

Percent methylation values can be extracted from `methylBase` object by using `percMethylation` function.

```
> # creates a matrix containing percent methylation values
> perc.meth=percMethylation(meth)
```

6 Frequently Asked Questions

Detailed answers to some of the frequently asked questions and various how-tos can be found at `http://zvfak.blogspot.com/search/label/methylKit`. In addition, `http://code.google.com/p/methylkit/` has online documentation and links to tutorials and other related material. You can also check methylKit Q&A forum for answers `https://groups.google.com/forum/#!forum/methylkit_discussion`.

Apart from those here are some of the frequently asked questions.

6.1 How can I select certain regions/bases from `methylRaw` or `methylBase` objects ?

see `?select` or `help("[", package = "methylKit")`

6.2 How can I find if my regions of interest overlap with exon/intron/promoter/CpG island etc.?

Currently, we will be able to tell you if your regions/bases overlap with the genomic features or not. see ?getMembers.

6.3 How can I find the nearest TSS associated with my CpGs

see ?getAssociationWithTSS

6.4 How do you define promoters and CpG island shores

Promoters are defined by options at `read.transcript.features` function. The default option is to take -1000,+1000bp around the TSS and you can change that. Same goes for CpG islands when reading them in via `read.feature.flank` function. Default is to take 2000bp flanking regions on each side of the CpG island as shores. But you can change that as well.

6.5 What does Bismark SAM output look like, where can I get more info?

Check the Bismark [1] website and there are also example files that ship with the package. Look at their formats and try to run different variations of `read.bismark()` command on the example files.

6.6 How can I reorder or remove samples at/from `methylRawList` or `methylBase` objects ?

see ?reorganize

6.7 Should I normalize my data?

`methylKit` comes with a simple `normalizeCoverage()` function to normalize read coverage distributions between samples. Ideally, you should first filter bases with extreme coverage to account for PCR bias using `filterByCoverage()` function, then run `normalizeCoverage()` function to normalize coverage between samples. These two functions will help reduce the bias in the statistical tests that might occur due to systematic over-sampling of reads in certain samples.

6.8 How can I force methylKit to use Fisher's exact test?

`methylKit` decides which test to use based on number of samples per group. In order to use Fisher's exact there must be one sample in each of the test and control groups. So if you have multiple samples for group, the package will employ Logistic Regression based test. However, you can use `pool()` function to pool samples in each group so that you have one representative sample per group. `pool()` function will sum up number of Cs and Ts in each group. We recommend using `filterByCoverage()` and `normalizeCoverage()` functions prior to using `pool()`. see ?pool

6.9 Can use data from other aligners than Bismark in methylKit ?

Yes, you can. methylKit can read any generic methylation percentage/ratio file as long as that text file contains columns for chromosome, start, end, strand, coverage and number of methylated cytosines. However, methylKit can only process SAM files from Bismark. For other aligners, you need to get a text file containing the minimal information described above. Some aligners will come with scripts or built-in tools to provide such files. See `http://zvfak.blogspot.com/2012/10/how-to-read-bsmap-methylation-ratio.html` for how to read methylation ratio files from BSMAP [3] aligner.

7 Acknowledgements

This package is developed at Weill Cornell Medical College by Altuna Akalin with important code contributions from Matthias Kormaksson mk375@cornell.edu and Sheng Li shl2018@med.cornell.edu. We wish to thank especially Maria E. Figueroa, Francine Garret-Bakelman, Christopher Mason and Ari Melnick for their contribution of ideas, data and support. Their support and discussions lead to development of methylKit.

8 R session info

```
> sessionInfo()

R version 3.1.1 (2014-07-10)
Platform: x86_64-apple-darwin13.1.0 (64-bit)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] methylKit_0.9.3

loaded via a namespace (and not attached):
 [1] BiocGenerics_0.12.0  chron_2.3-45         data.table_1.9.4
 [4] GenomeInfoDb_1.2.0   GenomicRanges_1.18.1 IRanges_2.0.0
 [7] KernSmooth_2.23-13   parallel_3.1.1       plyr_1.8.1
[10] Rcpp_0.11.3          reshape2_1.4         S4Vectors_0.4.0
[13] stats4_3.1.1         stringr_0.6.2        tools_3.1.1
[16] XVector_0.6.0
```

References

[1] Felix Krueger and Simon R Andrews. Bismark: a flexible aligner and methylation caller for Bisulfite-Seq applications. *Bioinformatics (Oxford, England)*, 27(11):1571–2, June 2011.

[2] Hong-Qiang Wang, Lindsey K Tuominen, and Chung-Jui Tsai. SLIM: a sliding linear model for estimating the proportion of true null hypotheses in datasets with dependence structures. *Bioinformatics (Oxford, England)*, 27(2):225–31, January 2011.

[3] Yuanxin Xi and Wei Li. BSMAP: whole genome bisulfite sequence MAPping program. *BMC bioinformatics*, 10(1):232, January 2009.