

methyKit: User Guide

Altuna Akalin Matthias Kormaksson
ala2027@med.cornell.edu mk375@cornell.edu

Sheng Li
shl2018@med.cornell.edu

August 16, 2012

Contents

1	Introduction	2
1.1	DNA methylation	2
1.2	High-throughput bisulfite sequencing	2
2	Basics	3
2.1	Reading the methylation call files	3
2.2	Reading the methylation calls from sorted Bismark alignments .	3
2.3	Descriptive statistics on samples	4
2.4	Filtering samples based on read coverage	6
3	Comparative analysis	6
3.1	Merging samples	6
3.2	Sample Correlation	7
3.3	Clustering samples	8
3.4	Tiling windows analysis	11
3.5	Finding differentially methylated bases or regions	11
3.5.1	Finding differentially methylated bases using multiple-cores	12
4	Annotating differentially methylated bases or regions	13
4.1	Regional analysis	14
4.2	Convenience functions for annotation objects	14
5	methyKit convenience functions	16
5.1	coercion	16
5.2	select	19
5.3	reorganize	21
5.4	percMethylation	21
6	Frequently Asked Questions	22

7	Acknowledgements	22
8	R session info	22

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 `methyRaw` or `methyRawList` 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 `methyRaw` 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 = "methyKit"),
+       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 `methyRawList` 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:

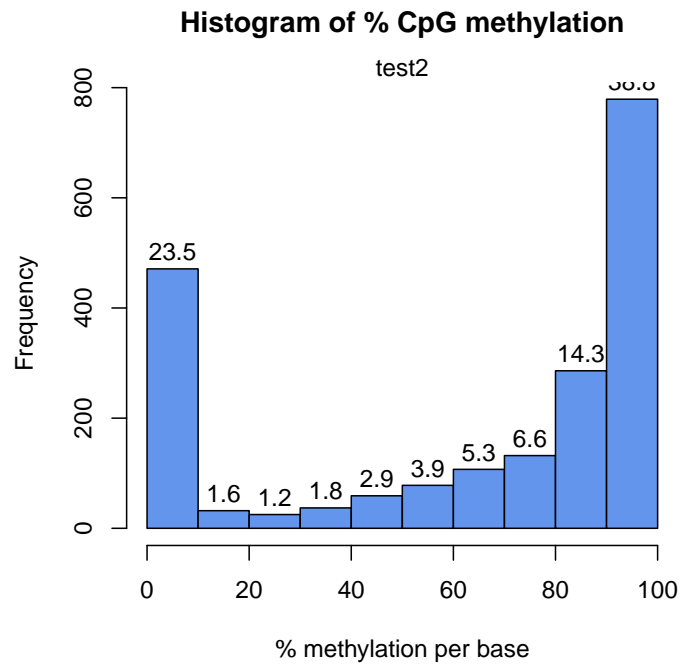
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.00	20.00	82.79	63.17	94.74	100.00

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	100.00000

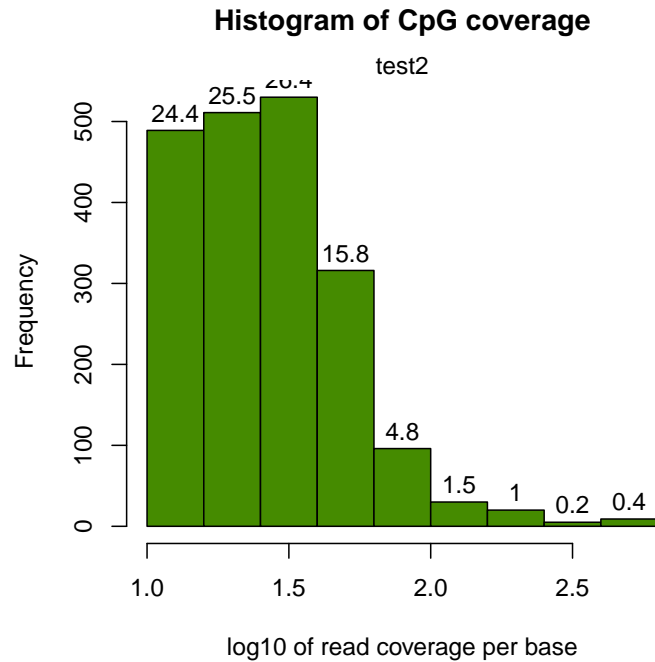
The following command plots the histogram for percent methylation distribution. The figure below is the histogram and numbers on bars denote what percentage of locations are contained in that bin. Typically, percent methylation histogram should have two peaks on both ends. In any given cell, any given 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 `methyRawList` 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 `methyBase` object which will be our main object for all comparative analysis. The `methyBase` 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 `methyBase` object:

```
> head(meth)
```

	id	chr	start	end	strand	coverage1	numCs1	numTs1
1	chr21.10011833	chr21	10011833	10011833	+	174	173	1
2	chr21.10011841	chr21	10011841	10011841	+	173	164	9
3	chr21.10011855	chr21	10011855	10011855	+	175	175	0
4	chr21.10011858	chr21	10011858	10011858	+	175	131	44
5	chr21.10011861	chr21	10011861	10011861	+	174	147	27
6	chr21.10011872	chr21	10011872	10011872	+	167	160	7

	coverage2	numCs2	numTs2	coverage3	numCs3	numTs3	coverage4	numCs4	numTs4
1	18	18	0	40	34	6	14	14	0
2	20	19	1	40	18	22	14	8	6
3	21	21	0	39	29	10	14	12	2
4	21	20	1	39	31	8	13	8	5
5	20	15	5	39	13	26	13	9	4
6	20	19	1	39	34	5	14	8	6

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

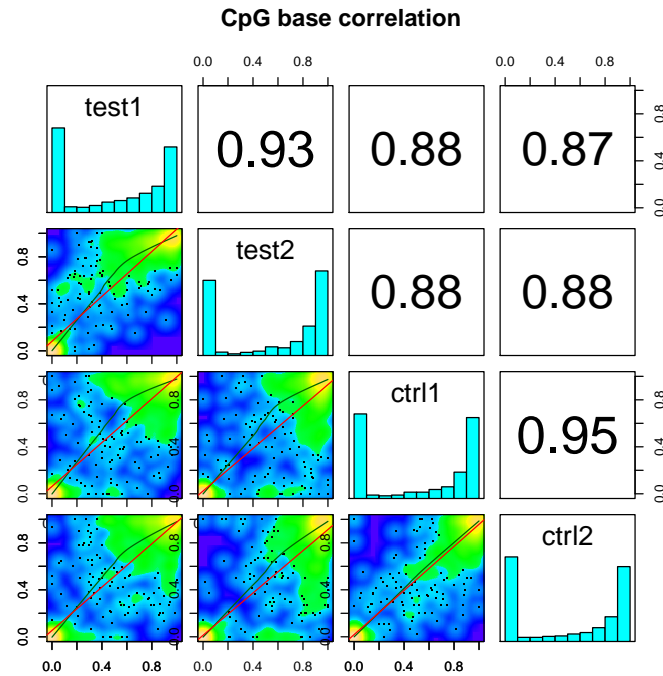
```
> # creates a methyBase object. Only CpGs covered at least in 1 sample per group will be retained
> # there were two groups defined by the treatment vector given during the creation of myobj
> meth.min=unite(myobj,min.per.group=1L)
```

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

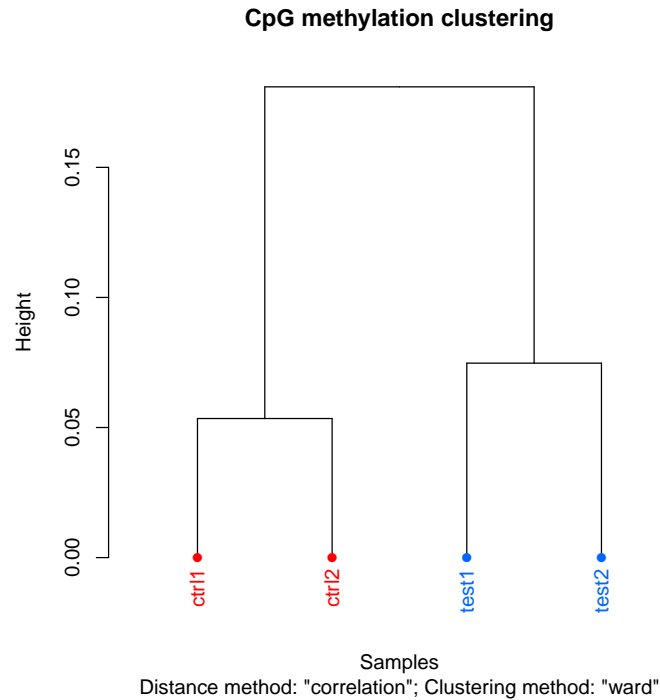
We can cluster the samples based on the similarity of their methylation profiles. The following function will cluster the samples and draw a dendrogram.

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

Call:

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

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

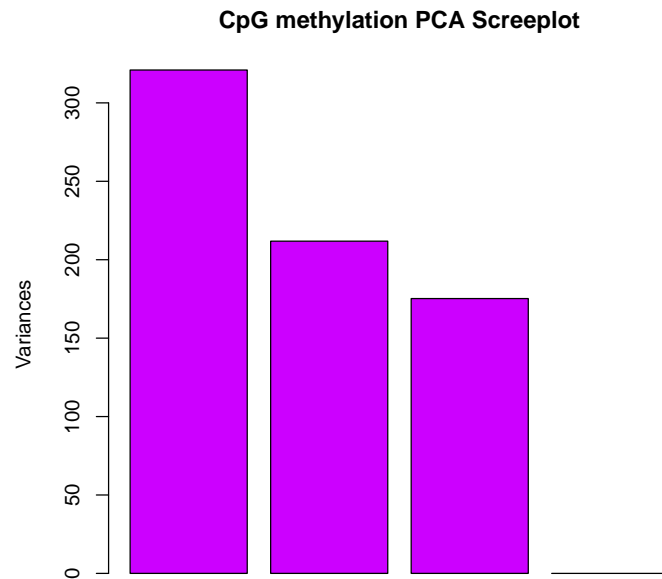



Setting the `plot=FALSE` will return a dendrogram object which can be manipulated by users or fed in to other user functions that can work with dendrograms.

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

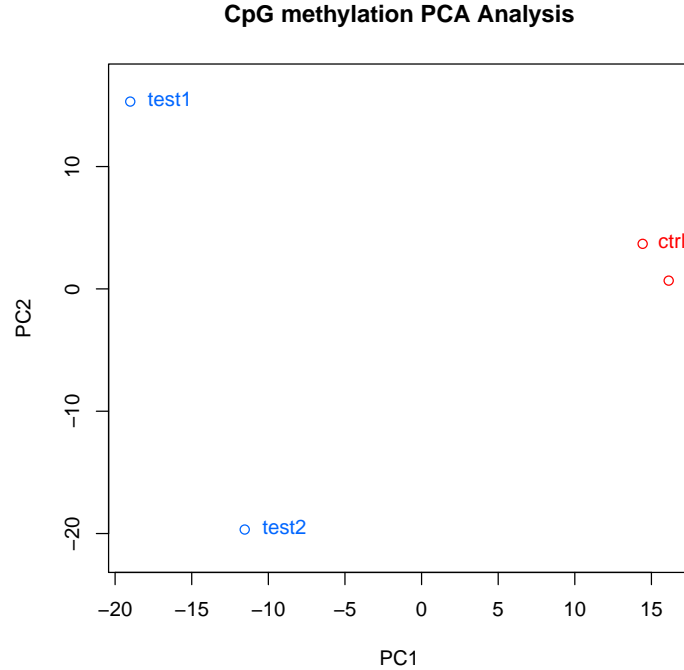
We can also do a PCA analysis on our samples. The following function will plot a scree plot for importance of components.

```
> 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 Tiling windows analysis

For some situations, it might be desirable to summarize methylation information over tiling windows rather than doing base-pair resolution analysis. `methyKit` 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 `methyRawList` object which can be fed into `unite` and `calculateDiffMeth` functions consecutively to get differentially methylated regions.

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

	id	chr	start	end	strand	coverage	numCs	numTs
1	chr21.9764001.9765000	chr21	9764001	9765000	*	24	3	21
2	chr21.9820001.9821000	chr21	9820001	9821000	*	13	0	13
3	chr21.9837001.9838000	chr21	9837001	9838000	*	11	0	11

3.5 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

3.5.1 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 = "methyKit"))
> #
> # 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 precedence)
```

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]])

```

	id	chr	start	end	strand	coverage	numCs
1	chr21.17806094.17808094.NA	chr21	17806094	17808094	+	1834	7
2	chr21.10119796.10121796.NA	chr21	10119796	10121796	-	79	44
3	chr21.10011791.10013791.NA	chr21	10011791	10013791	-	3697	2982
4	chr21.10119808.10121808.NA	chr21	10119808	10121808	-	79	44
5	chr21.15357997.15359997.NA	chr21	15357997	15359997	-	8613	16
6	chr21.16023366.16025366.NA	chr21	16023366	16025366	+	6296	5

```

  numTs
1 1827
2   35
3  715
4   35
5 8594
6 6291

```

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	951	NM_199260	-
60.1	2	931	NM_199260	-
60.2	3	838	NM_199260	-
60.3	4	828	NM_199260	-

60.4	5	802	NM_199260	-
60.5	6	723	NM_199260	-

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

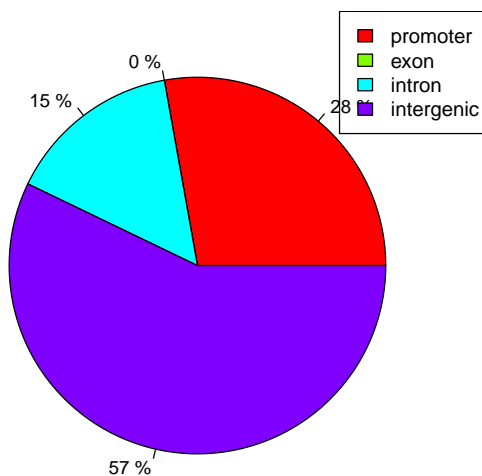
```
> getTargetAnnotationStats(diffAnn,percentage=TRUE,precedence=TRUE)
```

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

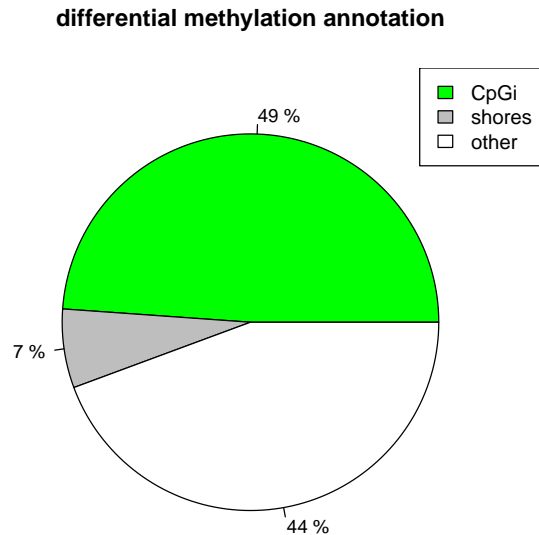
```
> plotTargetAnnotation(diffAnn,precedence=TRUE,
+   main="differential methylation annotation")
```

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 customising their analyses.

```

> class(meth)

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

```



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

GRanges with 963 ranges and 13 elementMetadata cols:

	seqnames		ranges	strand		id	coverage1
	<Rle>		<IRanges>	<Rle>		<factor>	<integer>
[1]	chr21	[10011833,	10011833]	+		chr21.10011833	174
[2]	chr21	[10011841,	10011841]	+		chr21.10011841	173
[3]	chr21	[10011855,	10011855]	+		chr21.10011855	175
[4]	chr21	[10011858,	10011858]	+		chr21.10011858	175
[5]	chr21	[10011861,	10011861]	+		chr21.10011861	174
[6]	chr21	[10011872,	10011872]	+		chr21.10011872	167
[7]	chr21	[10011876,	10011876]	+		chr21.10011876	160
[8]	chr21	[10011878,	10011878]	+		chr21.10011878	150
[9]	chr21	[10011925,	10011925]	-		chr21.10011925	120
...
[955]	chr21	[9944505,	9944505]	+		chr21.9944505	37
[956]	chr21	[9944663,	9944663]	-		chr21.9944663	61
[957]	chr21	[9959407,	9959407]	+		chr21.9959407	44
[958]	chr21	[9959541,	9959541]	-		chr21.9959541	26
[959]	chr21	[9959569,	9959569]	-		chr21.9959569	25
[960]	chr21	[9959577,	9959577]	-		chr21.9959577	25
[961]	chr21	[9959644,	9959644]	-		chr21.9959644	21
[962]	chr21	[9959650,	9959650]	-		chr21.9959650	21
[963]	chr21	[9967634,	9967634]	-		chr21.9967634	10
	numCs1	numTs1	coverage2	numCs2	numTs2	coverage3	numCs3
	<numeric>	<numeric>	<integer>	<numeric>	<numeric>	<integer>	<numeric>
[1]	173	1	18	18	0	40	34
[2]	164	9	20	19	1	40	18
[3]	175	0	21	21	0	39	29
[4]	131	44	21	20	1	39	31
[5]	147	27	20	15	5	39	13
[6]	160	7	20	19	1	39	34
[7]	148	12	21	18	3	38	24
[8]	134	16	20	19	1	37	20
[9]	65	55	37	21	16	68	21
...
[955]	2	35	147	56	91	86	79
[956]	19	42	116	71	45	45	35
[957]	17	27	118	58	60	52	49
[958]	12	14	76	44	32	39	37
[959]	17	8	77	69	8	40	40
[960]	25	0	77	71	6	40	40
[961]	0	21	97	50	47	59	52
[962]	6	15	103	57	46	59	51
[963]	0	10	61	25	36	93	62
	numTs3	coverage4	numCs4	numTs4			

```

      <numeric> <integer> <numeric> <numeric>
[1]          6          14          14          0
[2]         22          14           8          6
[3]         10          14          12          2
[4]          8          13           8          5
[5]         26          13           9          4
[6]          5          14           8          6
[7]         14          11           9          2
[8]         17          12          12          0
[9]         47          20           6          14
...
[955]         7          40          25          15
[956]        10          31          25           6
[957]         3          40          27          13
[958]         2          39          32           7
[959]         0          39          35           4
[960]         0          39          36           3
[961]         7          31          14          17
[962]         8          32          21          11
[963]        31          56          29          27
---
seqlengths:
chr21
NA

> class(myDiff)

[1] "methyldiff"
attr(,"package")
[1] "methyldiff"

> as(myDiff, "GRanges")

GRanges with 963 ranges and 3 elementMetadata cols:
      seqnames      ranges strand |      id      qvalue
      <Rle>      <IRanges> <Rle> |      <factor>      <numeric>
[1]   chr21 [10011833, 10011833]   + | chr21.10011833 8.543092e-04
[2]   chr21 [10011841, 10011841]   + | chr21.10011841 6.049801e-13
[3]   chr21 [10011855, 10011855]   + | chr21.10011855 4.579307e-09
[4]   chr21 [10011858, 10011858]   + | chr21.10011858 5.921730e-01
[5]   chr21 [10011861, 10011861]   + | chr21.10011861 8.162676e-08
[6]   chr21 [10011872, 10011872]   + | chr21.10011872 1.238123e-03
[7]   chr21 [10011876, 10011876]   + | chr21.10011876 1.933224e-04
[8]   chr21 [10011878, 10011878]   + | chr21.10011878 3.488683e-04
[9]   chr21 [10011925, 10011925]   - | chr21.10011925 8.543092e-04
...
[955]   chr21 [9944505, 9944505]   + | chr21.9944505 0.000000e+00

```

```

[956] chr21 [9944663, 9944663] - | chr21.9944663 7.678302e-05
[957] chr21 [9959407, 9959407] + | chr21.9959407 4.839266e-08
[958] chr21 [9959541, 9959541] - | chr21.9959541 3.145107e-06
[959] chr21 [9959569, 9959569] - | chr21.9959569 3.702161e-02
[960] chr21 [9959577, 9959577] - | chr21.9959577 4.922906e-01
[961] chr21 [9959644, 9959644] - | chr21.9959644 3.291132e-05
[962] chr21 [9959650, 9959650] - | chr21.9959650 6.575118e-05
[963] chr21 [9967634, 9967634] - | chr21.9967634 1.027764e-03

meth.diff
<numeric>
[1] 10.590278
[2] 46.670505
[3] 22.641509
[4] 2.040816
[5] 41.197462
[6] 16.476642
[7] 24.365768
[8] 24.693878
[9] 24.095252
...
[955] -51.017943
[956] -28.099911
[957] -36.312399
[958] -33.559578
[959] -10.622983
[960] -2.084885
[961] -30.960452
[962] -28.314428
[963] -25.862558
---
seqlengths:
chr21
NA

```

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.

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

Object of class "methylBase"

	id	chr	start	end	strand	coverage1	numCs1	numTs1
1	chr21.10011833	chr21	10011833	10011833	+	174	173	1
2	chr21.10011841	chr21	10011841	10011841	+	173	164	9
3	chr21.10011855	chr21	10011855	10011855	+	175	175	0

4	chr21.10011858	chr21	10011858	10011858	+	175	131	44
5	chr21.10011861	chr21	10011861	10011861	+	174	147	27
6	chr21.10011872	chr21	10011872	10011872	+	167	160	7
7	chr21.10011876	chr21	10011876	10011876	+	160	148	12
8	chr21.10011878	chr21	10011878	10011878	+	150	134	16
9	chr21.10011925	chr21	10011925	10011925	-	120	65	55
10	chr21.10011938	chr21	10011938	10011938	-	134	127	7

	coverage2	numCs2	numTs2	coverage3	numCs3	numTs3	coverage4	numCs4	numTs4
1	18	18	0	40	34	6	14	14	0
2	20	19	1	40	18	22	14	8	6
3	21	21	0	39	29	10	14	12	2
4	21	20	1	39	31	8	13	8	5
5	20	15	5	39	13	26	13	9	4
6	20	19	1	39	34	5	14	8	6
7	21	18	3	38	24	14	11	9	2
8	20	19	1	37	20	17	12	12	0
9	37	21	16	68	21	47	20	6	14
10	36	34	2	74	64	10	20	17	3

Slot "sample.ids":
[1] "test1" "test2" "ctrl1" "ctrl2"

Slot "assembly":
[1] "hg18"

Slot "context":
[1] "CpG"

Slot "treatment":
[1] 1 1 0 0

Slot "coverage.index":
[1] 6 9 12 15

Slot "numCs.index":
[1] 7 10 13 16

Slot "numTs.index":
[1] 8 11 14 17

Slot "destranded":
[1] FALSE

Slot "resolution":
[1] "base"


```
> select(myDiff,20:30) # select rows 10 of a methylDiff object
```

methylDiff object with 11 rows

```
-----
      id   chr   start   end strand   pvalue   qvalue
20 chr21.10012079 chr21 10012079 10012079   + 1.325366e-07 1.049731e-06
21 chr21.10012089 chr21 10012089 10012089   + 6.797159e-02 1.047612e-01
22 chr21.10012095 chr21 10012095 10012095   + 9.125016e-02 1.324085e-01
23 chr21.10012101 chr21 10012101 10012101   + 8.881784e-16 4.220791e-14
24 chr21.10012696 chr21 10012696 10012696   + 2.253460e-03 6.033165e-03
25 chr21.10012699 chr21 10012699 10012699   + 1.782895e-09 1.955228e-08
meth.diff
20 26.616915
21  9.564423
22  5.726470
23 39.807824
24  9.684982
25 44.703297
-----
sample.ids: test1 test2 ctrl1 ctrl2
destranded FALSE
assembly: hg18
context: CpG
treatment: 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

Answers to frequently asked questions and various how-tos can be reached at <http://zvfk.blogspot.com/search/label/methylKit>. In addition, <http://code.google.com/p/methylkit/> has online documentation and links to tutorials and other related material.

7 Acknowledgements

This package is developed at Weill Cornell Medical College by Altuna Akalin with important code contributions from Sheng Li and Matthias Kormaksson. 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 2.15.0 (2012-03-30)
Platform: x86_64-apple-darwin9.8.0/x86_64 (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] data.table_1.8.0 methylKit_0.5.2

loaded via a namespace (and not attached):
[1] BiocGenerics_0.2.0 GenomicRanges_1.8.6 IRanges_1.14.3
[4] KernSmooth_2.23-7  parallel_2.15.0     stats4_2.15.0
[7] tools_2.15.0
```

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.