Analyzing RRBS methylation data with R

Basic analysis with R package methylKit

Altuna Akalın
ala2027@med.cornell.edu

EpiWorkshop 2013

Weill Cornell Medical College
Institute for Computational Biomedicine

Outline

Introduction

This tutorial will show how to analyze methylation data obtained from Reduced Representation Bisulfite Sequencing (RRBS) experiments. First, we will introduce:

- R basics
- Genomics and R: Bioconductor packages that will be useful when working with genomic intervals.
- How to use the package `methylKit` to analyze methylation data.

Before we start...

- Download and unzip the data folder `http://methylkit.googlecode.com/files/methylKitTutorialData_feb2012.zip`
- Make sure that you installed R 2.15.2 and RStudio
  - for Mac OSX : http://cran.r-project.org/bin/macosx/old/R-2.15.2.pkg
  - for windows :http://cran.r-project.org/bin/windows/base/old/2.15.2/
  - For linux : http://cran.r-project.org/src/base/R-2/R-2.15.2.tar.gz
  - http://www.rstudio.com/ide/download/

R Basics

Here are some basic R operations and data structures that will be good to know if you do not have prior experience with R.

Computations in R

R can be used as an ordinary calculator. There are a few examples:

```
2 + 3 * 5  # Note the order of operations.

## [1] 17

log(10)  # Natural logarithm with base e=2.718282

## [1] 2.303

4^2  # 4 raised to the second power

## [1] 16

3/2  # Division

## [1] 1.5

sqrt(16)  # Square root

## [1] 4
```

Vectors

R handles vectors easily and intuitively.

```
x <- c(1, 3, 2, 10, 5)  #create a vector x with 5 components
x

## [1]  1  3  2 10  5

y <- 1:5  #create a vector of consecutive integers
y

## [1] 1 2 3 4 5

y + 2  #scalar addition

## [1] 3 4 5 6 7

2 * y  #scalar multiplication

## [1]  2  4  6  8 10

y^2  #raise each component to the second power
```

Matrices

A matrix refers to a numeric array of rows and columns. One of the easiest ways to create a matrix is to combine vectors of equal length using cbind(), meaning "column bind":

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
m1 <- cbind(x, y)
m1

##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6

t(m1)  # transpose of m1

##   [,1] [,2] [,3]
## x    1    2    3
## y    4    5    6
```

Matrices

```
dim(m1)  # 2 by 3 matrix

## [1] 3 2
```

You can also directly list the elements and specify the matrix:

```
m2 <- matrix(c(1, 3, 2, 5, -1, 2, 2, 3, 9), nrow = 3)
m2

##      [,1] [,2] [,3]
## [1,]    1    5    2
## [2,]    3   -1    3
## [3,]    2    2    9
```

Data Frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.). Small to moderate size data frame can be constructed by data.frame() function. For example, we illustrate how to construct a data frame from genomic intervals or coordinates.

```
chr <- c("chr1", "chr1", "chr2", "chr2")
strand <- c("-", "-", "+", "+")
start <- c(200, 4000, 100, 400)
end <- c(250, 410, 200, 450)
mydata <- data.frame(chr, start, end, strand)
```

Data Frames

```
names(mydata) <- c("chr", "start", "end", "strand")  #variable names
mydata

##    chr start end strand
## 1 chr1   200 250      -
## 2 chr1  4000 410      -
## 3 chr2   100 200      +
## 4 chr2   400 450      +

# OR this will work too
mydata <- data.frame(chr = chr, start = start,
    end = end, strand = strand)
mydata

##    chr start end strand
## 1 chr1   200 250      -
## 2 chr1  4000 410      -
## 3 chr2   100 200      +
## 4 chr2   400 450      +
```

Data Frames

There are a variety of ways to identify the elements of a data frame .

```
mydata[2:4]  # columns 2,3,4 of data frame

##   start end strand
## 1   200 250      -
## 2  4000 410      -
## 3   100 200      +
## 4   400 450      +

# columns chr and start from data frame
mydata[c("chr", "start")]

##    chr start
## 1 chr1   200
## 2 chr1  4000
## 3 chr2   100
## 4 chr2   400
```

Data Frames

```
mydata$start  # variable start in the data frame

## [1]  200 4000  100  400
```

Lists

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name.

```
# example of a list, a string, a numeric
# vector, a matrix, and a scalar
w <- list(name = "Fred", mynumbers = c(1, 2, 3),
    mymatrix = matrix(1:4, ncol = 2), age = 5.3)
w

## $name
## [1] "Fred"
##
## $mynumbers
## [1] 1 2 3
##
## $mymatrix
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $age
## [1] 5.3
```

Lists

You can extract elements of a list using the [[]] convention.

```
w[[3]]  # 3rd component of the list

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

w[["mynumbers"]]  # component named mynumbers in list

## [1] 1 2 3
```

Graphics in R

R has great support for plotting and customizing plots. We will show only a few below.

```
x <- rnorm(50)  # sample 50 values from normal distr.
hist(x)  # plot the histogram of those values
hist(x, main = "Hello histogram!!!")  # change the title
y <- rnorm(50)  # randomly sample 50 points from normal distr.
plot(x, y)  #plot a scatter plot
y <- rnorm(50)  # randomly sample 50 points from normal distr.
boxplot(x, y, main = "boxplots of random samples")  #plot  boxplots
plot(x, y, main = "scatterplot of random samples")  #scatter plots
```

Graphics in R

Saving plots

```r
pdf(file = "mygraphs/myplot.pdf")
plot(x, y)
dev.off()
# OR
```

```r
plot(x, y)
dev.copy(pdf, file = "mygraphs/myplot.pdf")
dev.off()
```

Getting help on R functions/commands

Most R functions have great documentation on how to use them. Try `?` and `??`. `?` will pull the documentation on the functions and `??` will find help pages on a vague topic. Try on R terminal:
?hist
??histogram

Installing packages

Where R packages live...

- CRAN: http://cran.r-project.org/
- Bioconductor: http://bioconductor.org/
- R-forge: http://r-forge.r-project.org/
- Github: http://github.com/
- Googlecode: http://code.google.com/

How to install packages

- `install.packages("devtools")`
- `source("http://bioconductor.org/biocLite.R")`
  `biocLite("GenomicRanges")`
- Install from the source:
  `install.packages("devtools_1.1.tar.gz",repos=NULL, type="source")`

Genomics and R

Bioconductor and CRAN has many packages that will help analyze genomics data. Some of the most popular ones are GenomicRanges, IRanges, Rsamtools and BSgenome. Check out `http:\bioconductor.org` for packages that suits your purpose.

Reading the genomics data

Most of the genomics data are in the form of genomic intervals associated with a score. That means mostly the data will be in table format with columns denoting chromosome, start positions, end positions, strand and score. One of the popular formats is BED format. In R, you can easily read tabular format data with `read.table()` function. In addition, you can write data to a text file with `write.table()`.

```r
# read enhancer marker BED file
enh.df = read.table("subset.enhancers.bed", header = FALSE)
# read CpG island BED file
cpgi.df = read.table("subset.cpgi.hg18.bed.txt",
    header = FALSE)

# write data frames to text files
write.table(cpgi.df, "example.out.txt")
```

Reading the genomics data

```r
# check first lines to see how the data
# looks like
head(enh.df, 2)
head(cpgi.df, 2)
# get CpG islands on chr21
head(cpgi.df[cpgi.df$V1 == "chr21", ], 2)
```

Using GenomicRanges package for operations on genomic intervals

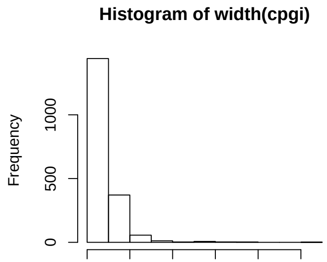
- One of the most useful operations when working with genomic intervals is the overlap operation
- bioconductor packages: IRanges and GenomicRanges provide efficient ways to handle genomic interval data.
- Below, we will show how to convert your data to GenomicRanges objects/data structures and do overlap between enhancers and CpG islands.

Using GenomicRanges package for operations on genomic intervals

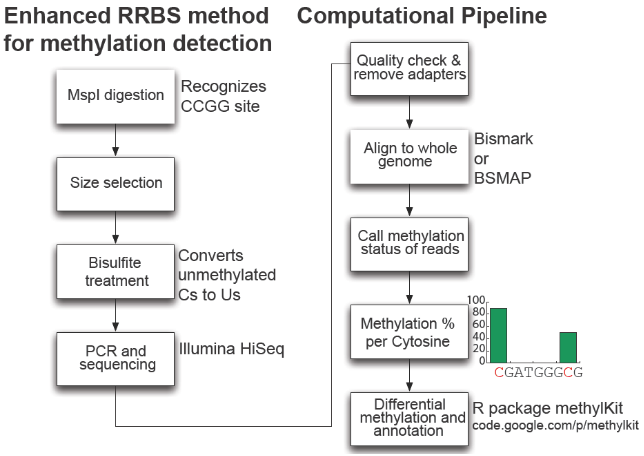
```
# covert enhancer data frame to GenomicRanges object
library(GenomicRanges) # load the package
enh <- GRanges(seqnames=enh.df$V1,
               ranges=IRanges(start=enh.df$V2,end=enh.df$V3)  )
cpgi = GRanges(seqnames=cpgi.df$V1,
               ranges=IRanges(start=cpgi.df$V2,end=cpgi.df$V3),
               ids=cpgi.df$V4  )
# find enhancers overlapping with CpG islands
cpg.enh=subsetByOverlaps(enh, cpgi)
```

Using GenomicRanges package for operations on genomic intervals

```
# number of enhancers overlapping with CpG
# islands
length(cpg.enh)
# number of all enhancers in the set
length(enh)
# plot histogram of lenths of CpG islands
hist(width(cpgi))
```



Methylation profile by Reduced Representation Bisulfite Sequencing (RRBS)


Enhanced RRBS method for methylation detection
MspI digestion
Recognizes CCGG site
Size selection
Bisulfite treatment
Converts unmethylated Cs to Us
PCR and sequencing
Illumina HiSeq
Computational Pipeline
Quality check & remove adapters
Align to whole genome
Bismark or BSMAP
Call methylation status of reads
Methylation % per Cytosine
100
80
60
40
20
0
CGATGGGCG
Differential methylation and annotation
R package methylKit
code.google.com/p/methylkit

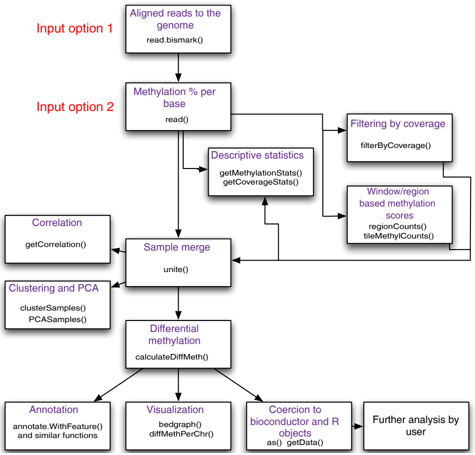
Installing methylKit

In R terminal, install the package and the dependencies by typing the following commands:

```
source("http://methylkit.googlecode.com/files/install.methylKit.R")
# installing the package with the
# dependencies
install.methylKit(ver = "0.5.6", dependencies = TRUE)
```

see
`https://code.google.com/p/methylkit/#Installation`
for more details on installing methylKit

Flowchart for capabilities of methylKit



Reading the methylation data

Now methylKit is installed, we will load the package and start by reading in the methylation call data. 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
## 1 chr20.12314 chr20 12314      F       21
## 2 chr20.12378 chr20 12378      R       13
## 3 chr20.12382 chr20 12382      R       13
## 4 chr20.12323 chr20 12323      F       21
## 5 chr20.55740 chr20 55740      R       53
##   freqC freqT
## 1 95.24  4.76
## 2 76.92 23.08
## 3 38.46 61.54
## 4 85.71 14.29
## 5 67.92 32.08
```

Reading the methylation data

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 as follows:

```
library(methylKit)
file.list = list("test1.CpG.txt", "test2.CpG.txt",
    "ctrl1.CpG.txt", "ctrl2.CpG.txt")
# 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")
```

Reading the methylation data

- Another way to read the methylation calls is directly from the SAM files. The SAM files must be sorted and only SAM files from Bismark aligner is supported at the moment. Check `read.bismark()` function help to learn more about this.

Quality check and basic features of the data

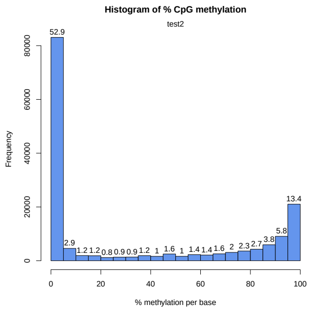
we can check the basic stats about the methylation data such as coverage and percent methylation. We now have a `methylRawList` object which contains methylation information per sample. The following command prints out percent methylation statistics for second sample: "test2"

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

## methylation statistics per base
## summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    0.00    0.00    2.48   34.00   81.80
##    Max.
##  100.00
## percentiles:
##      0%     10%     20%     30%     40%
##   0.000   0.000   0.000   0.000   0.000
##     50%     60%     70%     80%     90%
##   2.479  30.986  70.000  89.583 100.000
##     95%     99%   99.5%   99.9%    100%
## 100.000 100.000 100.000 100.000 100.000
```

Quality check and basic features of the data

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.

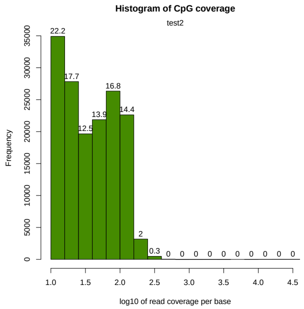
```
getMethylationStats(myobj[[2]], plot = T, both.strands = F)
```



Quality check and basic features of the data

We can also plot the read coverage per base information in a similar way. Experiments that are highly suffering from PCR duplication bias will have a secondary peak towards the right hand side of the histogram.

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



Filtering samples based on read coverage

It might be useful to filter samples based on coverage.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)
```

Sample Correlation
Uniting data sets

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.

```
meth = unite(myobj, destrand = FALSE)
# meth is a methylBase object
```

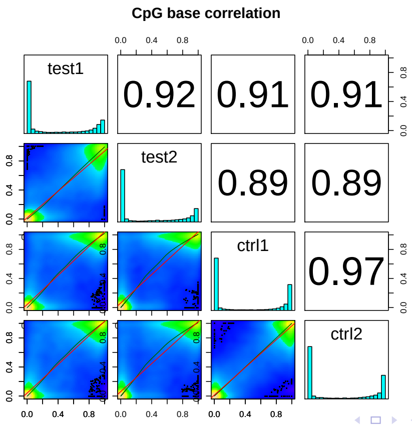
Let us take a look at the data content of methylBase object:

```
head(meth, 2)

##                  id   chr    start      end
## 1 chr20.10100840 chr20 10100840 10100840
## 2 chr20.10100844 chr20 10100844 10100844
##   strand coverage1 numCs1 numTs1 coverage2
## 1      +        29      2     27        11
## 2      +        29      2     27        11
##   numCs2 numTs2 coverage3 numCs3 numTs3
## 1      0     11        43      2     41
## 2      1     10        42      2     40
##   coverage4 numCs4 numTs4
```

Sample Correlation

This function will either plot scatter plot and correlation coefficients or just print a correlation matrix.

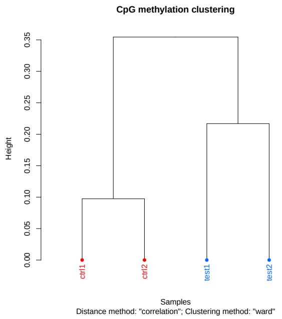
```
getCorrelation(meth, plot = T)
```



Clustering your samples based on the methylation profiles

`methylKit` can do hiearchical clustering.

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

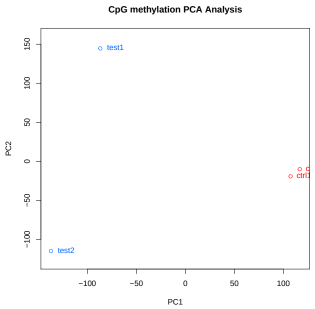


```
##
## Call:
```

Clustering your samples based on the methylation profiles

Principal Component Analysis (PCA) is another option. We can plot PC1 (principal component 1) and PC2 (principal component 2) axis and a scatter plot of our samples on those axis which will reveal how they cluster.

```
PCASamples(meth)
```



Clustering your samples based on the methylation profiles

clustering multiple conditions

It is possible to cluster samples from more than 2 conditions. The code below will not work since the tutorial does not have an example dataset, but if you happen to work with multiple conditions you can still cluster them in the way shown below.The members of the cluster will be color coded based on the "treatment" option.

```
file.list = list("condA_sample1.txt", "condA_sample2.txt",
    "condB_sample1.txt", "condB_sample1.txt",
    "condC_sample1.txt", "condC_sample1.txt")
clist = read(file.list, sample.id = list("A1",
    "A2", "B1", "B2", "C1", "C2"), assembly = "hg18",
    treatment = c(2, 2, 1, 1, 0, 0), context = "CpG")
newMeth = unite(clist)
clusterSamples(newMeth)
```

Getting differentially methylated bases

- `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.

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

`calculateDiffMeth()` can also use multiple cores for faster calculation.

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

Getting differentially methylated bases

Following bit selects the bases that have q-value<0.01 and percent methylation difference larger than 25%.

```
# 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)
```

Getting differentially methylated regions

`methylKit` can summarize methylation information over tiling windows or over a set of predefined regions (promoters, CpG islands, introns, etc.) rather than doing base-pair resolution analysis.

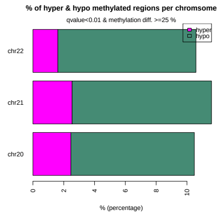
```
# you might get warnings here, ignore them
tiles = tileMethylCounts(myobj, win.size = 1000,
    step.size = 1000)

head(tiles[[1]], 2)

##                    id   chr start   end strand
## 1 chr20.12001.13000 chr20 12001 13000      *
## 2 chr20.55001.56000 chr20 55001 56000      *
##   coverage numCs numTs
## 1       68    53    15
## 2      212   192    20
```

Differential methylation events per chromosome

We can also visualize the distribution of hypo/hyper-methylated bases/regions per chromosome using the following function.

```
# if plot=FALSE a list having per chromosome
# differentially methylation events will be
# returned
diffMethPerChr(myDiff, plot = FALSE, qvalue.cutoff = 0.01,
    meth.cutoff = 25)
```

```
# if plot=TRUE a barplot will be plotted
diffMethPerChr(myDiff, plot = TRUE, qvalue.cutoff = 0.01,
    meth.cutoff = 25)
```



Annotating differential methylation events

We can annotate our differentially methylated regions/bases based on gene annotation. We need to read the gene annotation from a bed file and annotate our differentially methylated regions with that information. Similar gene annotation can be created using `GenomicFeatures` package available from Bioconductor.

```
gene.obj <- read.transcript.features("subset.refseq.hg18.bed.txt")
# annotate differentially methylated Cs with
# promoter/exon/intron using annotation data
annotate.WithGenicParts(myDiff25p, gene.obj)

## summary of target set annotation with genic parts
## 8009 rows in target set
## --------------
## --------------
## percentage of target features overlapping with annotation :
##   promoter       exon     intron intergenic
##      14.43      19.04      40.33      38.57
##
##
## percentage of target features overlapping with annotation (with pro
```

Annotating differential methylation events

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("subset.cpgi.hg18.bed.txt",
    feature.flank.name = c("CpGi", "shores"))
#
diffCpGann = annotate.WithFeature.Flank(myDiff25p,
    cpg.obj$CpGi, cpg.obj$shores, feature.name = "CpGi",
    flank.name = "shores")
```

Annotating differential methylation events

We can also read any BED file and annotate our differentially methylated bases/regions with them.

```
# read the enhancer marker bed file
enh.b = read.bed("subset.enhancers.bed")
#

diffCpG.enh = annotate.WithFeature(myDiff25p,
    enh.b, feature.name = "enhancers")
```

Working with annotated methylation events

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), 3)

##     target.row dist.to.feature feature.name
## 427          1         -121458    NM_000214
## 428          2          271458    NR_038972
## 310          3           -1871    NM_018354
##     feature.strand
## 427              -
## 428              -
## 310              -
```

Working with annotated methylation events

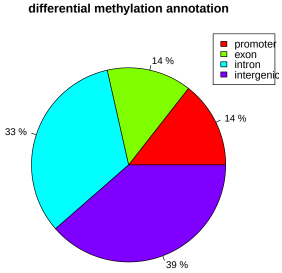
It is also desirable to get percentage/number of differentially methylated regions that overlap with intron/exon/promoters

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

##   promoter       exon     intron intergenic
##      14.43      14.11      32.89      38.57
```

Working with annotated methylation events

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

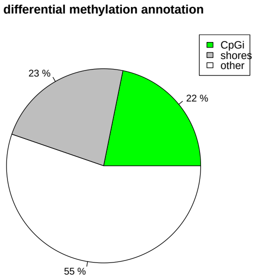
```
plotTargetAnnotation(diffAnn, precedence = TRUE,
    main = "differential methylation annotation")
```



Working with annotated methylation events

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")
```



Regional analysis

We can also summarize methylation information over a set of defined regions such as promoters. 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]], 3)

##                            id   chr    start
## 1 chr20.33505636.33507636.NA chr20 33505636
## 2   chr20.8060295.8062295.NA chr20  8060295
## 3   chr20.3137055.3139055.NA chr20  3137055
##        end strand coverage numCs numTs
## 1 33507636      +      727     4   723
## 2  8062295      +      954    15   939
## 3  3139055      +      553     2   551
```

methylKit convenience functions

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 flexiblity when customising their analyses.

```
class(meth)
as(meth, "GRanges")
class(myDiff)
as(myDiff, "GRanges")
```

methylKit convenience functions

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, 1:5)  # select first 5 rows
# OR
meth[1:5, ]  # select first 5 rows
```

methylKit convenience functions

Another useful function is `getData()`. You can use this function to extract data frames from methylRaw, methylBase and methylDiff objects.

```
# get data frame and show first rows
head(getData(meth))
# get data frame and show first rows
head(getData(myDiff))
```

Further information

information on RRBS and alike

- RRBS `http://www.nature.com/nprot/journal/v6/n4/abs/nprot.2010.190.html`
- Agilent methyl-seq capture `http://www.halogenomics.com/sureselect/methyl-seq`

Some of the aligners and pipelines

- Bismark `http://www.bioinformatics.bbsrc.ac.uk/projects/bismark/`
- AMP pipeline `http://code.google.com/p/amp-errbs/`
- BSMAP `http://code.google.com/p/bsmap/`
- other aligners and methods reviewed here `http://www.nature.com/nmeth/journal/v9/n2/full/nmeth.1828.html`

Further information

More info on methylKit

- The webpage for the package is `http://methylkit.googlecode.com/`
- Genome Biology paper for the package is at `http://genomebiology.com/2012/13/10/R87`
- Blog posts related to methylKit are at `http://zvfak.blogspot.com/search/label/methylKit`
- The slides for this presentation is here: `http://methylkit.googlecode.com/files/methylKitTutorialSlides_2013.pdf`
- The tutorial for the 2012 EpiWorkshop is here: `http://methylkit.googlecode.com/files/methylKitTutorial_feb2012.pdf`