Analyzing RRBS methylation data with methylKit

Altuna Akalın
ala2027@med.cornell.edu

February 21, 2012

Contents

1 Introduction

This tutorial will show how to analyze methylation data obtained from Reduced Representation Bisulfite Sequencing (RRBS) experiments. First, we will introduce R basics and some Bioconductor packages that will be useful when working with genomic intervals. Following that, we will show how to use the package `methylKit` to analyze methylation data.

2 The Data

In the tutorial, we will use an example dataset that have a set of test and control samples. Each sample file contains percent methylation values per base, and we will use them when we compare methylation profiles of the test and control cases. We assume that reader already aligned the reads and called percent methylation values per base. There are number of ways to align bisulfite converted reads and call percent methylation values per base, see `methylKit` package vignette for methods directly supported by `methylKit`. The example dataset we will be working on also includes multiple genome annotation files such as CpG islands and gene annotation files. The annotation files will be used when we annotate methylation events. The example dataset is located at : `http://methylkit.googlecode.com/files/methylKitTutorialData_feb2012.zip`. Unzip the data and start R in the unzipped folder or set unzipped folder as the working directory using `setwd` in R.

3 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.

3.1 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
```

```
abs(3 - 7)  # Absolute value of 3-7
```

```
## [1] 4
```

```
pi  # The mysterious number
```

```
## [1] 3.142
```

```
exp(2)  # exponential function
```

```
## [1] 7.389
```

```
# This is a comment line
```

3.2 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
```

```
## [1]  1  4  9 16 25
```

```
2^y  #raise 2 to the first through fifth power
```

```
## [1]  2  4  8 16 32
```

```
y  #y itself has not been unchanged
```

```
## [1] 1 2 3 4 5
```

```
y <- y * 2
y  #it is now changed
```

```
## [1]  2  4  6  8 10
```

3.3 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, 4)
y <- c(4, 5, 6, 7)
m1 <- cbind(x, y)
m1
```

```
##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
## [4,] 4 7
```

```
t(m1)  # transpose of m1
```

```
##   [,1] [,2] [,3] [,4]
## x    1    2    3    4
## y    4    5    6    7
```

```
dim(m1)  # 2 by 5 matrix
```

```
## [1] 4 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
```

3.4 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)
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      +
```

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      +
```

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

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

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

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

3.5 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 with 4 components -
# a string, a numeric vector, a matrix, and a scaler
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
##
```

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
```

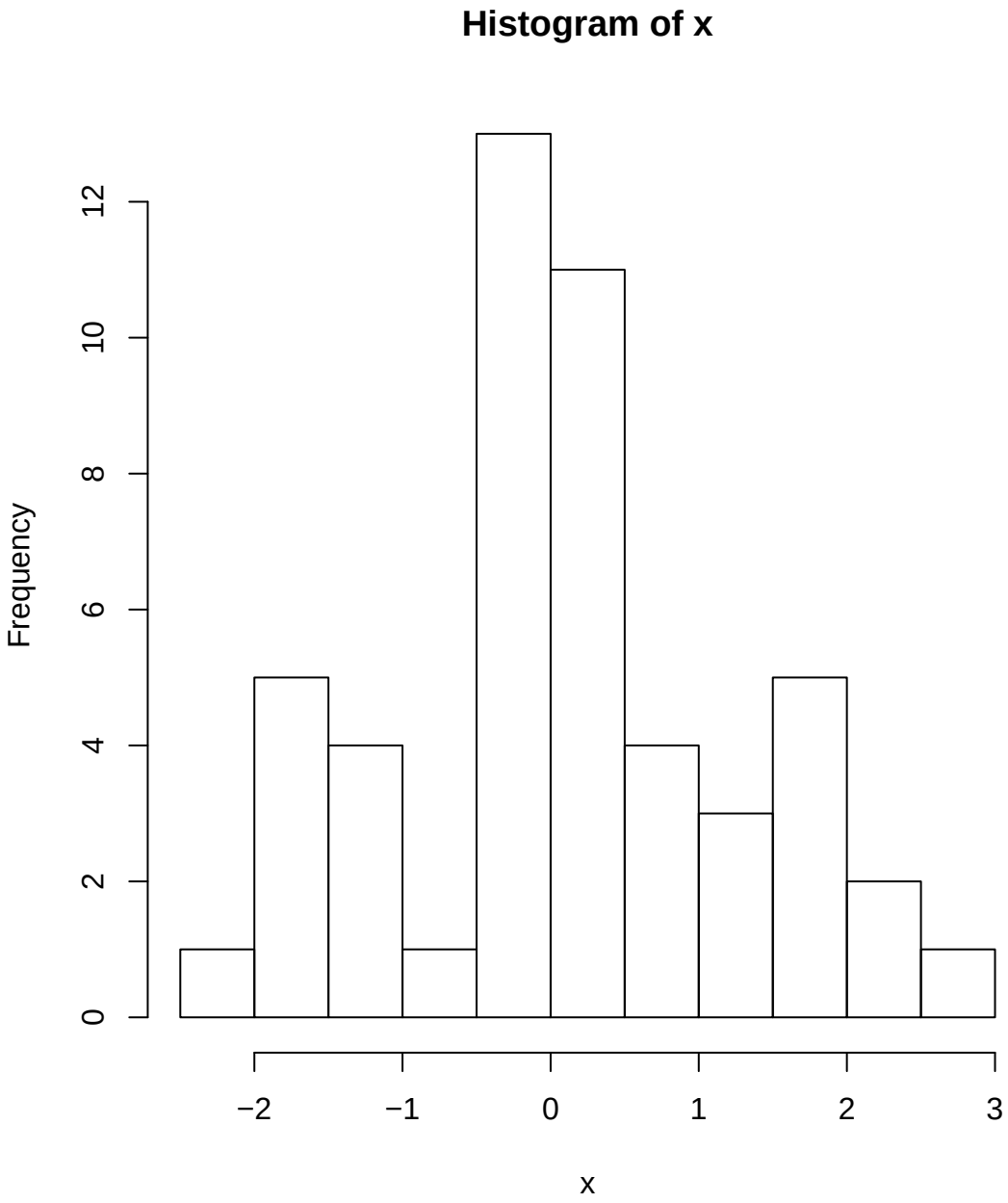
```
w[["mynumbers"]]  # component named mynumbers in list
```

```
## [1] 1 2 3
```

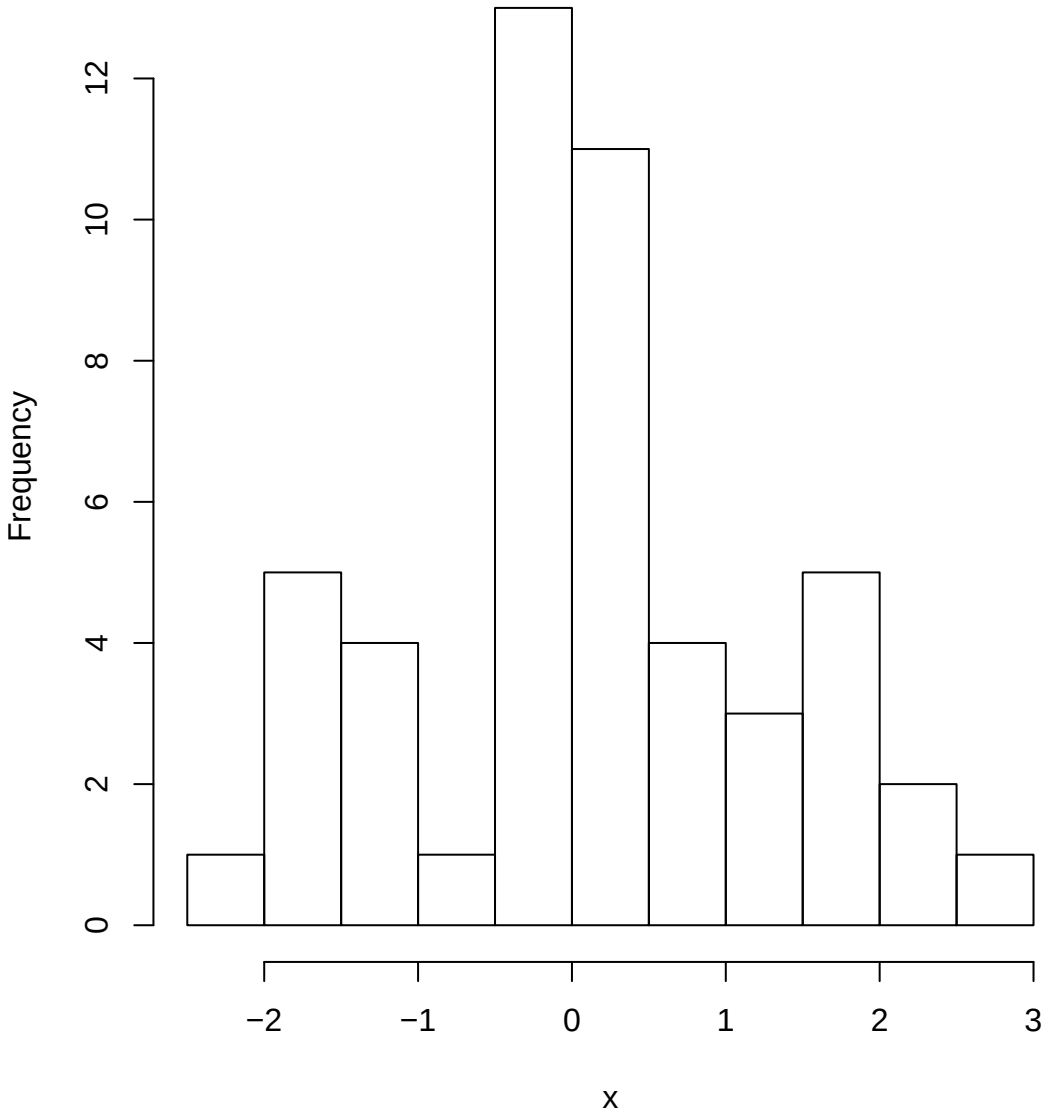
3.6 plotting

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

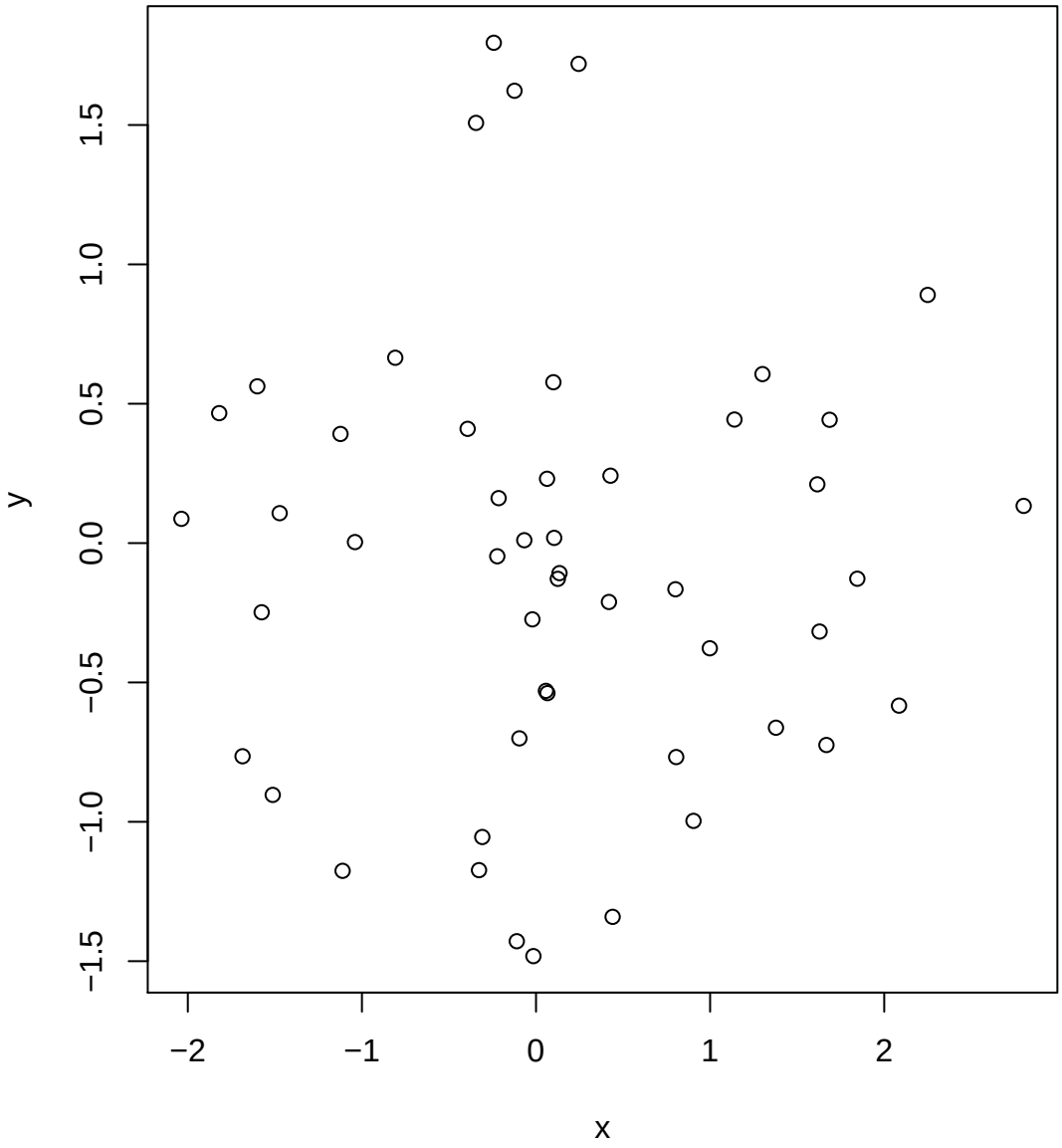
```
x <- rnorm(50)  # sample 50 values from normal distribution and store them in
vectorx
hist(x)  # plot the histogram of those values
```



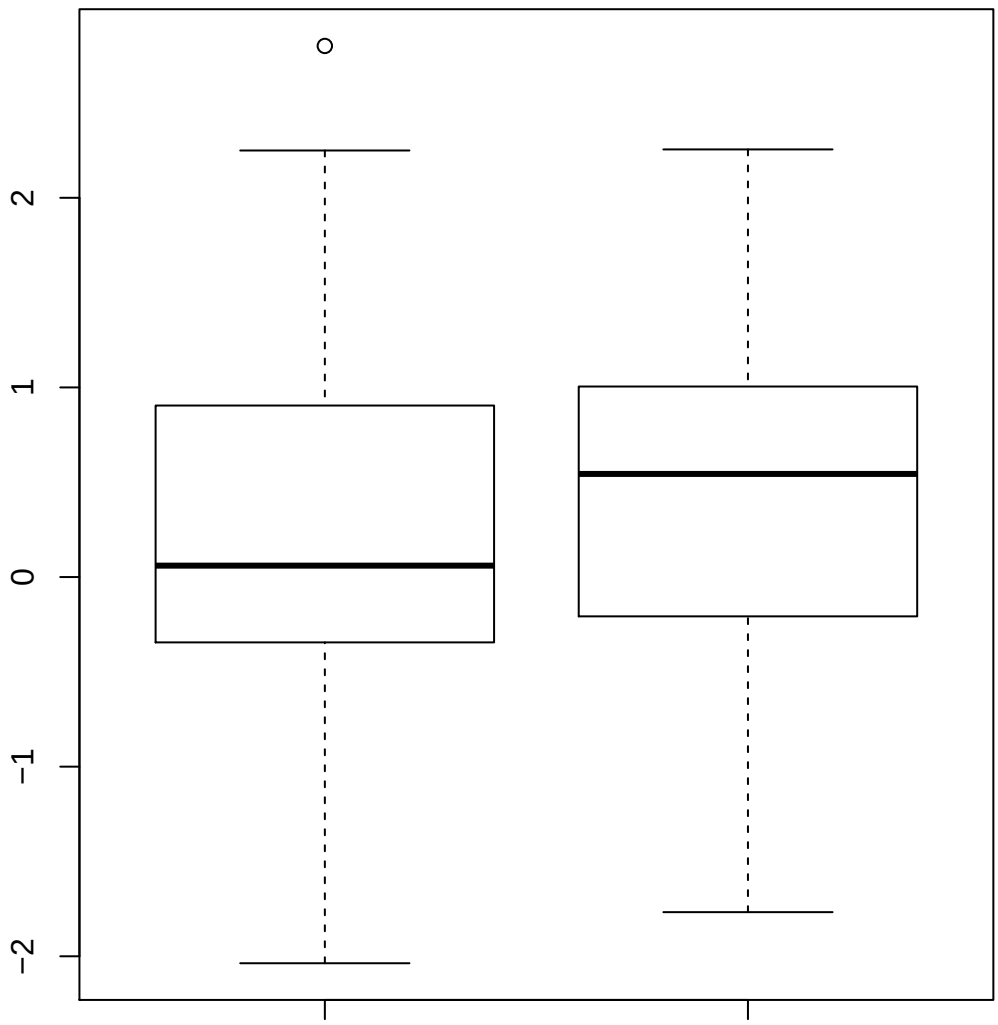

```r
hist(x, main = "Hello histogram!!!")  # change the title
```



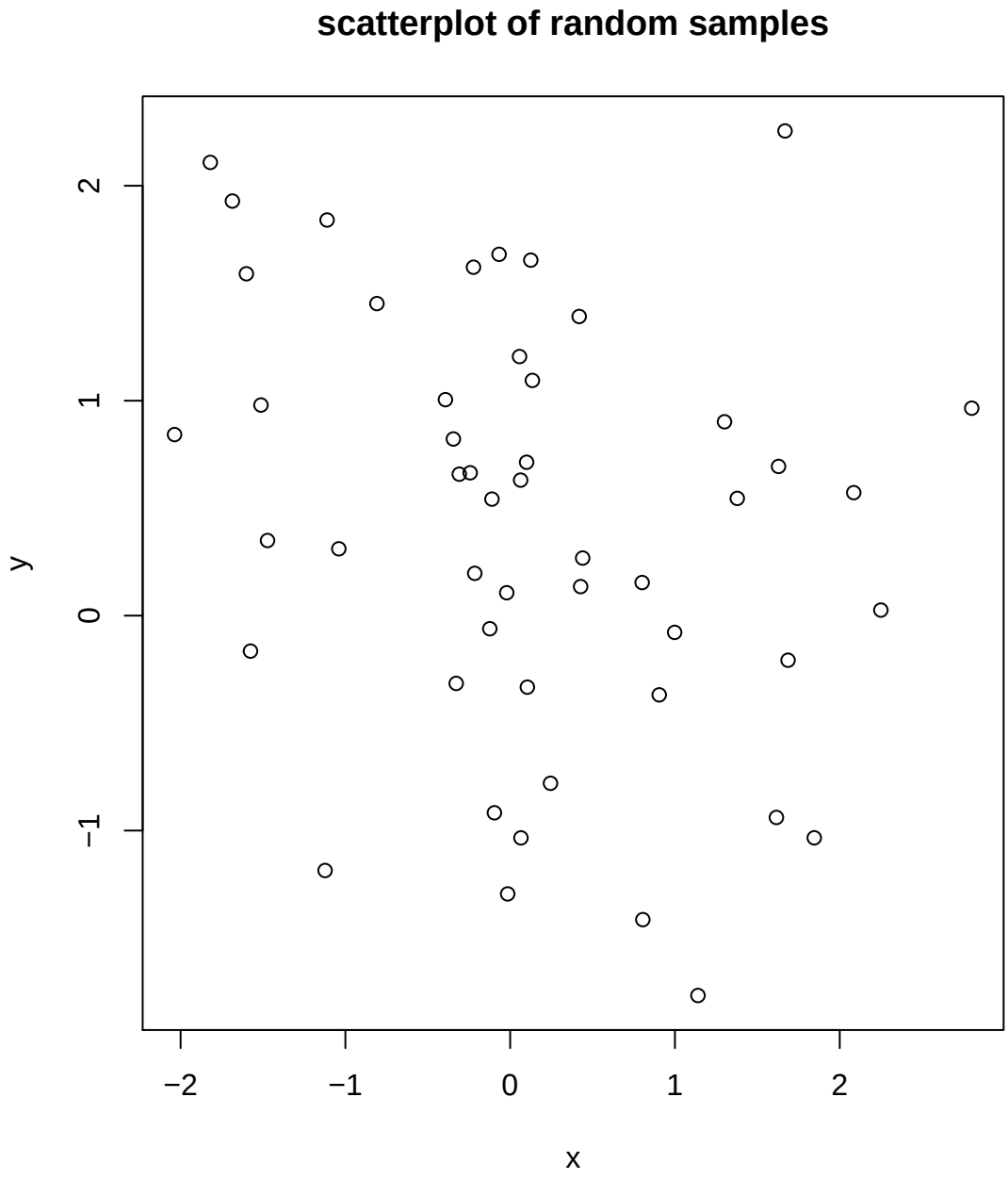

```
y <- rnorm(50)  # randomly sample 50 points from normal distribution
plot(x, y)  #plot a scatter plot
```




```r
y <- rnorm(50)  # randomly sample 50 points from normal distribution
boxplot(x, y, main = "boxplots of random samples")  #plot  boxplots
```




```
plot(x, y, main = "scatterplot of random samples")  #scatter plots with title
```



Saving plots

```
pdf("mygraphs/myplot.pdf")
plot(x, y)
dev.off()
```

3.7 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

4 Genomics and R

Bioconductor has many packages that will help analyze genomics data. Some of the most popular ones are GenomicRanges, IRanges, Rsamtools and BSgenome. If you are interested you can check bioconductor.org for packages that suit your purpose. Next subsections will show how to use R and bioconductor to read-in and manipulate genomic intervals. Knowing more about R and Bioconductor packages will be useful if you want to customize or enhance your methylation data analysis.

4.1 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 used primarly by UCSC genome browser but most other genome browsers and tools will support BED format. We have all the annotation data in BED format. In R, you can easily read tabular format data with read.table() function.

```
enh.df <- read.table("subset.enhancers.bed", header = FALSE)  # read enhancer marker
BED file
cpgi.df <- read.table("subset.cpgi.hg18.bed.txt", header = FALSE)  # read CpG island
BED file
head(enh.df)  # check first lines to see how the data looks like
```

```
##      V1     V2     V3 V4   V5 V6    V7    V8 V9
## 1 chr20 266275 267925  . 1000  .  9.11 13.17 -1
## 2 chr20 287400 294500  . 1000  . 10.53 13.02 -1
## 3 chr20 300500 302500  . 1000  .  9.10 13.39 -1
## 4 chr20 330400 331800  . 1000  .  6.39 13.51 -1
## 5 chr20 341425 343400  . 1000  .  6.20 12.99 -1
## 6 chr20 437975 439900  . 1000  .  6.31 13.52 -1
```

```
head(cpgi.df)
```

```
##      V1     V2     V3       V4
## 1 chr20 195575 195851  CpG:_28
## 2 chr20 207789 208148  CpG:_32
## 3 chr20 219055 219437  CpG:_33
## 4 chr20 225831 227155 CpG:_135
## 5 chr20 252826 256323 CpG:_286
## 6 chr20 275376 276977 CpG:_116
```

```
# get CpG islands on chr21
head(cpgi.df[cpgi.df$V1 == "chr21", ])
```

```
##        V1       V2       V3       V4
## 800 chr21  9906603  9906958  CpG:_46
## 801 chr21  9917382  9917652  CpG:_30
## 802 chr21 10011784 10013284 CpG:_152
## 803 chr21 10128589 10129003  CpG:_38
## 804 chr21 13331283 13332372  CpG:_73
## 805 chr21 13957814 13958107  CpG:_24
```

4.2 Using GenomicRanges package for operations on genomic intervals

One of the most useful operations when working with genomic intervals is the overlap operation. For example, we may want to know how many of enhancers overlap with CpG islands or how many

of the binding sites overlap with promoters, etc. Unfortunately, basic R functions are not designed to deal with such problems, however bioconductor packages: IRanges and GenomicRanges provide efficient ways to handle genomic interval data and provide many functions for operating on genomic intervals. Below, we will show how to convert your data to GenomicRanges objects/data structures and do overlap between enhancers and CpG islands.

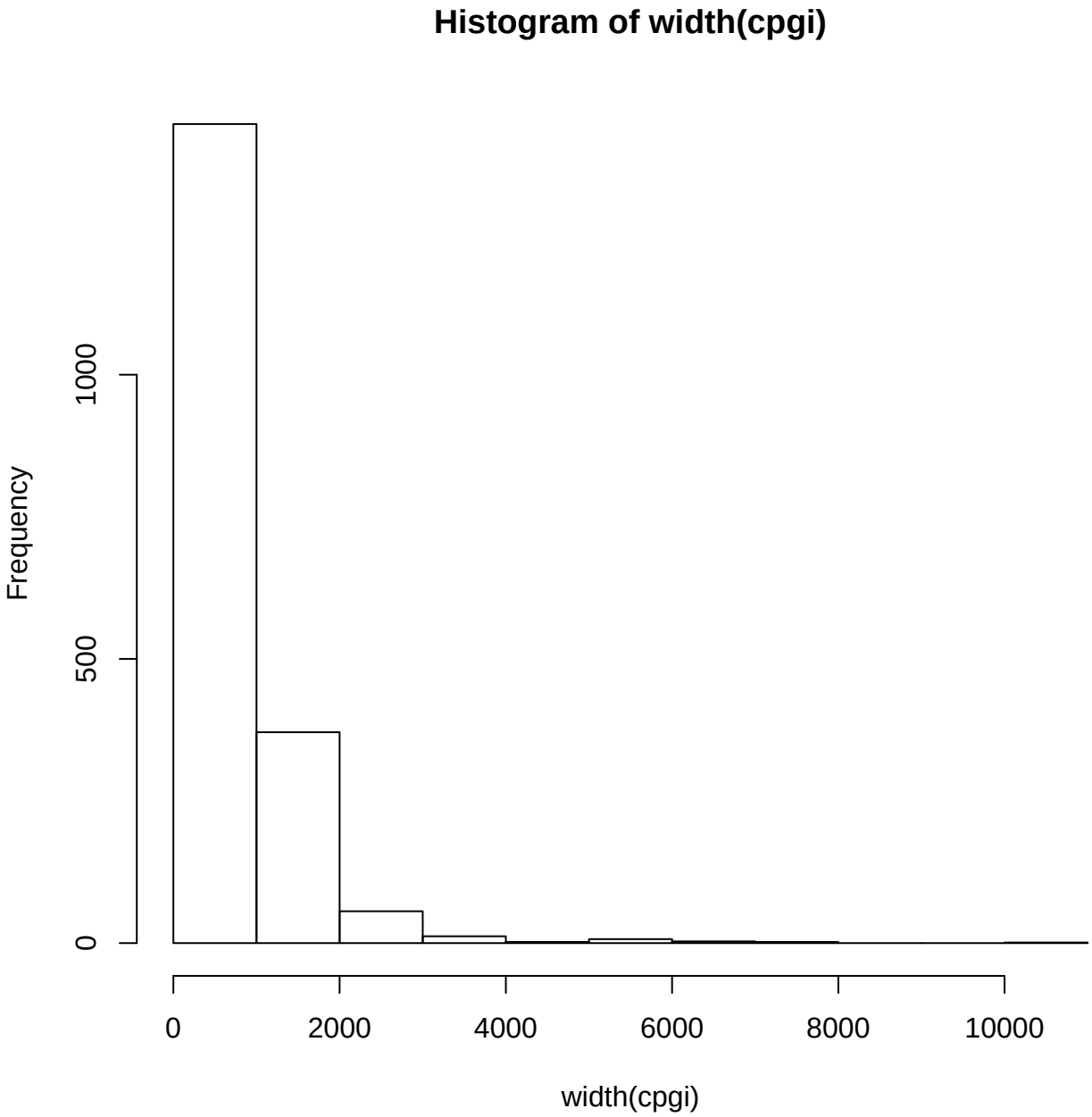
```
# covert enhancer data frame to GenomicRanges object
library(GenomicRanges)  # load the package
enh <- GRanges(seqnames = enh.df$V1, ranges = IRanges(start = enh.df$V2,
    end = enh.df$V3))
cpgi <- GRanges(seqnames = cpgi.df$V1, ranges = IRanges(start = cpgi.df$V2,
    end = cpgi.df$V3), ids = cpgi.df$V4)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

```
## [1] 1062
```

```
# number of all enhancers in the set
length(enh)
```

```
## [1] 50416
```

```
# plot histogram of lenths of CpG islands
hist(width(cpgi))
```



5 Methylation profile by Reduced Representation Bisulfite Sequencing (RRBS)

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. In Reduced Representation Bisulfite Sequencing, only a subset of the genome is analyzed. First, the DNA is digested with a methylation-insensitive restriction enzyme (MspI). Next they bisulfite sequence only the fragments of a specified length, which consist of mostly methylated DNA. By using such a method the you ensure you are looking at CpG rich areas of the genome (such as CpG islands) since MspI cuts from CCGG sites. After sequencing, the reads are aligned using special aligners that can handle bisulfite converted reads and percent methylation values per base are called. We reccomend using Bismark aligner `http://www.bioinformatics.bbsrc.ac.uk/projects/bismark/` or our Bismark based pipeline `http://code.google.com/p/amp-errbs/`. However, other aligners such as BSMAP `http://code.google.com/p/bsmap/` can also produce percent methylation calls per base and is potentially usable with methylKit and will be fully supported in later versions of methylKit.

5.1 Installing methylKit

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

```
install.packages("data.table")
source("http://www.bioconductor.org/biocLite.R")
biocLite("GenomicRanges")
```

Download latest version of methylKit from ''`http://code.google.com/p/methylKit/`''

```
# install the downloaded *.tar.gz source file with the
#   following command
# REMEMBER to put the correct *.tar.gz file
install.packages("methylKit_0.4.1.tar.gz", repos = NULL,
    type = "source")
```

5.2 Reading the methylation data

Now methylKit is installed, we will load the package and 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 chr20.12314 chr20 12314      F       21 95.24  4.76
## 2 chr20.12378 chr20 12378      R       13 76.92 23.08
## 3 chr20.12382 chr20 12382      R       13 38.46 61.54
## 4 chr20.12323 chr20 12323      F       21 85.71 14.29
## 5 chr20.55740 chr20 55740      R       53 67.92 32.08
```

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")
```

5.3 Quality check and basic features of the data

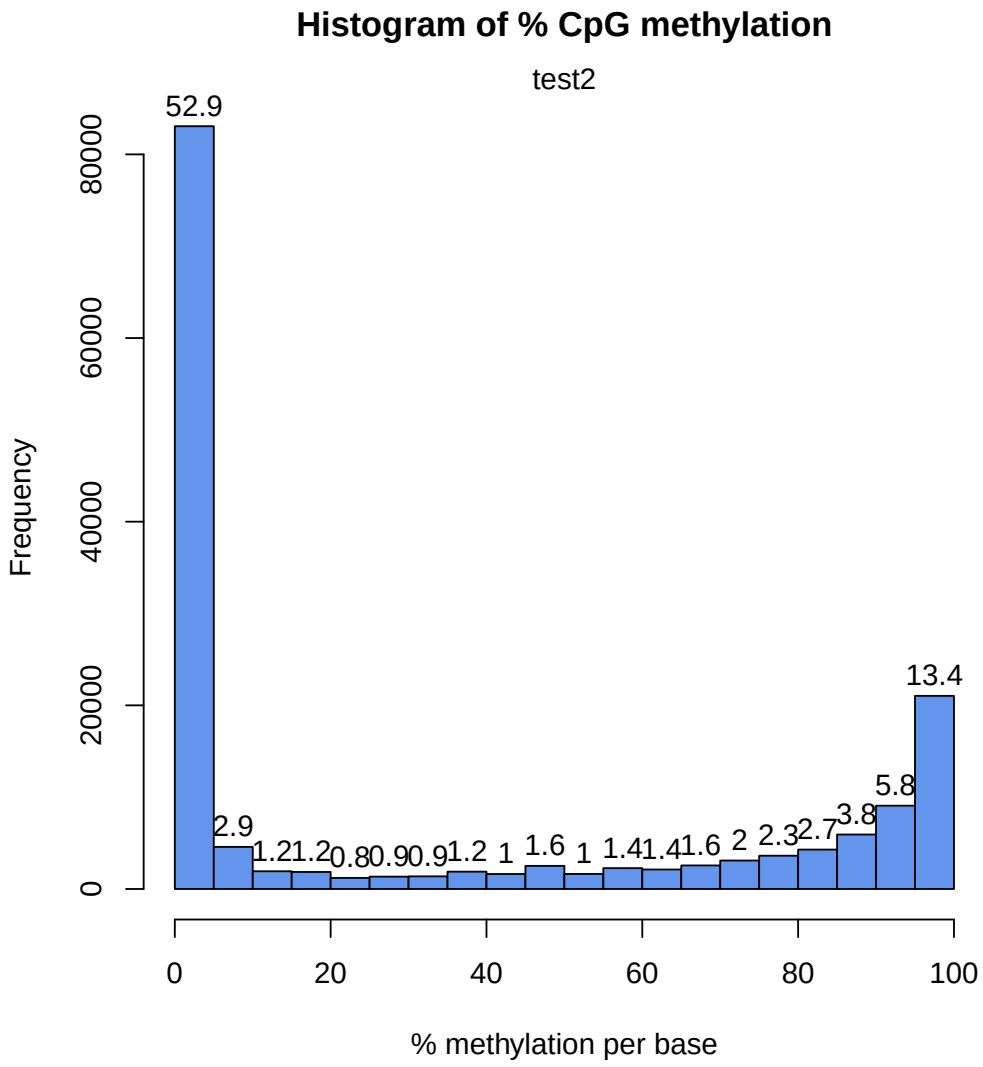
Since we read the methylation data now, we can check the basic stats about the methylation data such as coverage and percent methylation. We now have a `methylRawList` object which contains methylation information per sample. The following command prints out percent methylation statistics for second 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    0.00    2.48   34.00   81.80  100.00
## percentiles:
##      0%     10%     20%     30%     40%     50%     60%     70%     80%     90%
##   0.000   0.000   0.000   0.000   0.000   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
##
```

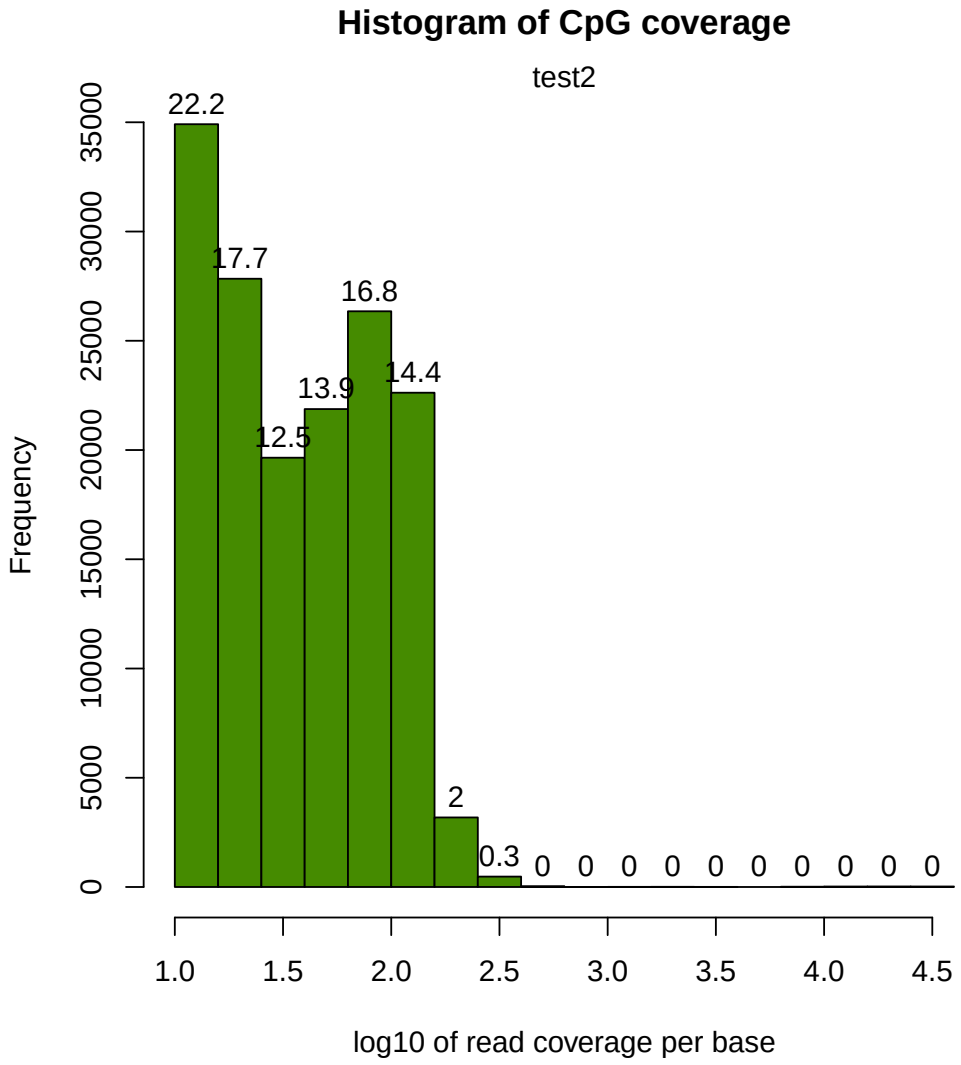
The following command plots the histogram for percent methylation distribution.The figure below is the histogram and numbers on bars denote what percentage of locations are contained in that bin. Typically, percent methylation histogram should have two peaks on both ends. In any given cell, any given 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.

```
library("graphics")
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.

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



5.4 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)
```

5.5 Sample Correlation

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:

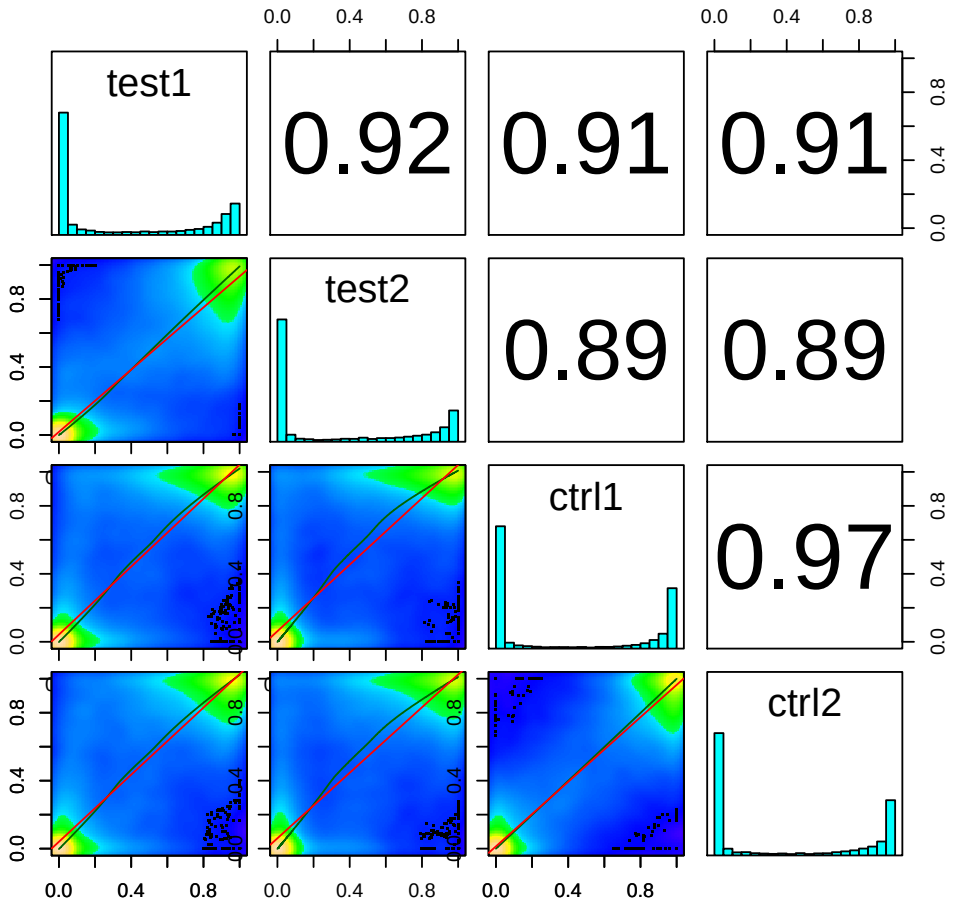
```
head(meth)
```

```
##               id   chr    start      end strand coverage1 numCs1 numTs1 coverage2
## 1 chr20.10100840 chr20 10100840 10100840      +        29      2     27        11
## 2 chr20.10100844 chr20 10100844 10100844      +        29      2     27        11
## 3 chr20.10100852 chr20 10100852 10100852      +        29      0     29        11
## 4 chr20.10146236 chr20 10146236 10146236      -        22      2     20        26
## 5 chr20.10146246 chr20 10146246 10146246      -        21      0     21        23
## 6 chr20.10146248 chr20 10146248 10146248      -        21      0     21        27
##   numCs2 numTs2 coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
## 1      0     11        43      2     41        10      0     10
## 2      1     10        42      2     40        10      0     10
## 3      0     11        43      0     43        10      0     10
## 4      5     21        16      0     16        10      1      9
## 5      0     23        17      0     17        10      1      9
## 6      1     26        17      2     15        10      1      9
```

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)
```

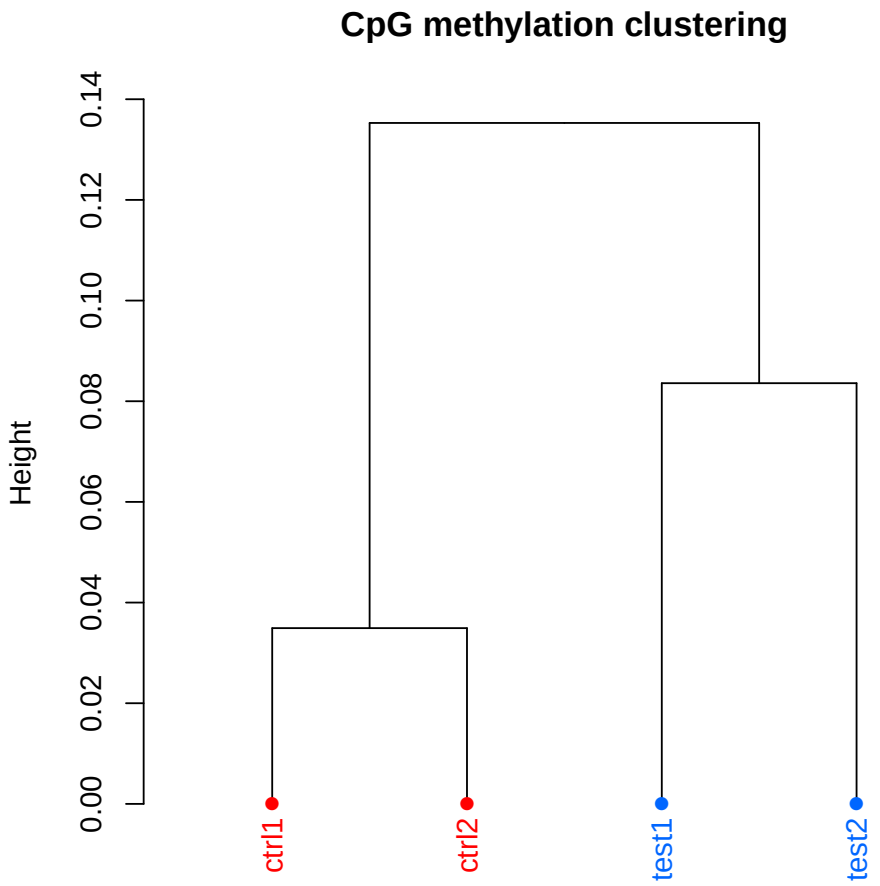
```
##        test1  test2  ctrl1  ctrl2
## test1 1.0000 0.9164 0.9124 0.9142
## test2 0.9164 1.0000 0.8923 0.8919
## ctrl1 0.9124 0.8923 1.0000 0.9651
## ctrl2 0.9142 0.8919 0.9651 1.0000
```



5.6 Clustering your samples based on the methylation profiles

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)
```



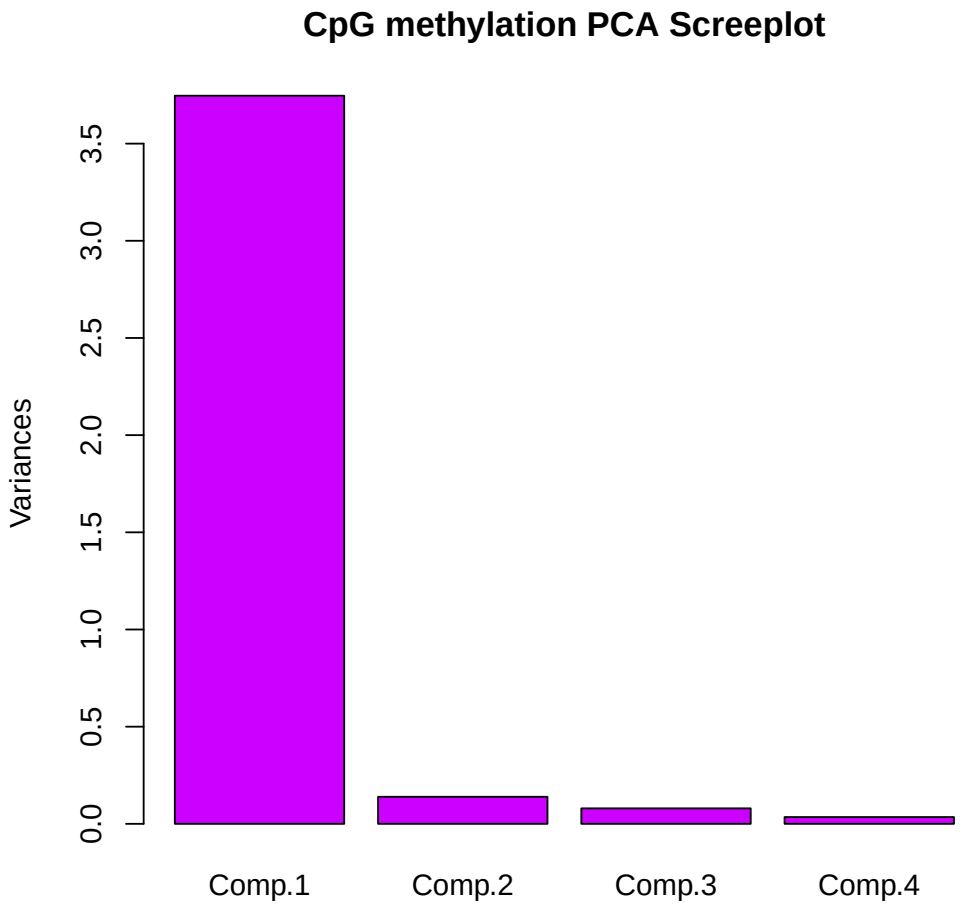

```
##
## Call:
## hclust(d = d, method = HCLUST.METHODS[hclust.method])
##
## Cluster method   : ward
## Distance         : pearson
## Number of objects: 4
##
```

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.

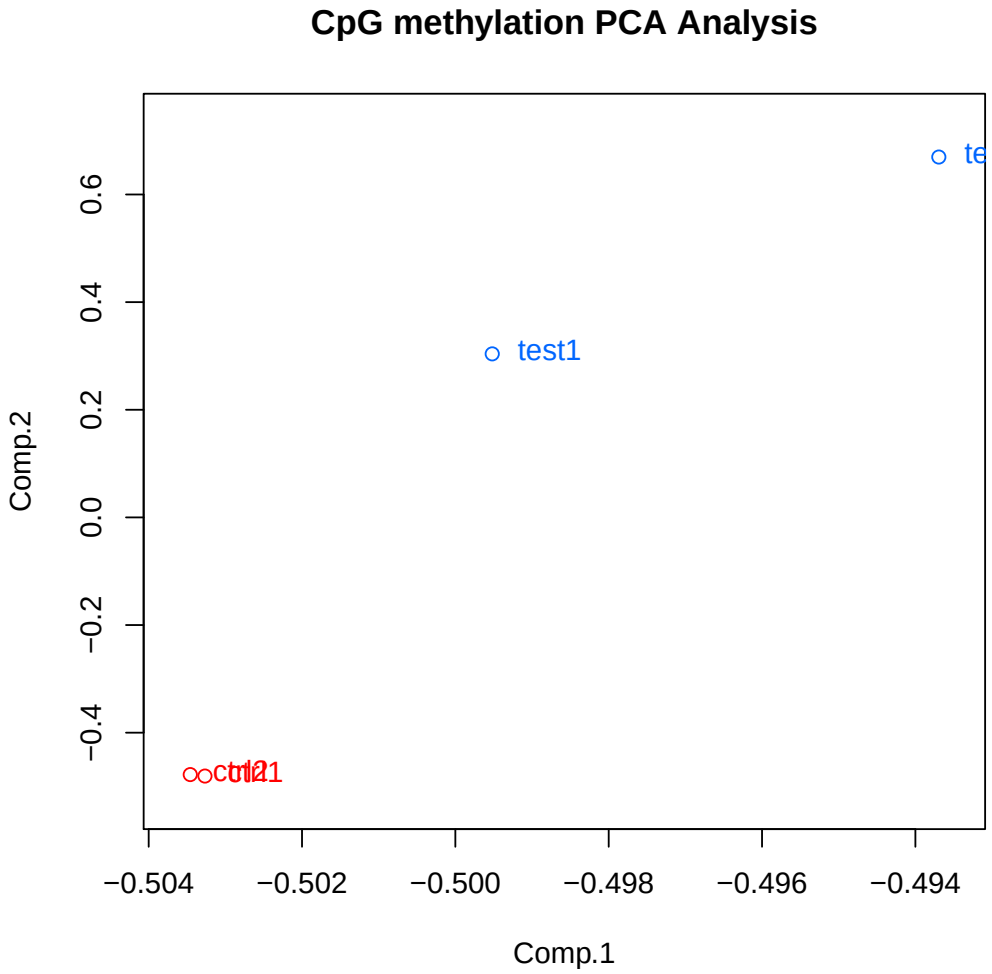
```
PCASamples(meth, screeplot = TRUE)
```




```
## Importance of components:
##                         Comp.1  Comp.2  Comp.3   Comp.4
## Standard deviation     1.9356 0.37297 0.28208 0.186784
## Proportion of Variance 0.9366 0.03478 0.01989 0.008722
## Cumulative Proportion  0.9366 0.97138 0.99128 1.000000
```

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)
```




```
## Importance of components:
##                          Comp.1  Comp.2  Comp.3   Comp.4
## Standard deviation     1.9356 0.37297 0.28208 0.186784
## Proportion of Variance 0.9366 0.03478 0.01989 0.008722
## Cumulative Proportion  0.9366 0.97138 0.99128 1.000000
```

5.6.1 clustering samples from 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)
```

5.7 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)
```

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)
```

5.7.1 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)
```

5.8 Getting differentially methylated regions

For some situations, it might be desirable to 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. `methylKit` provides functionality to do such analysis.

5.8.1 Tiling windows 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.

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

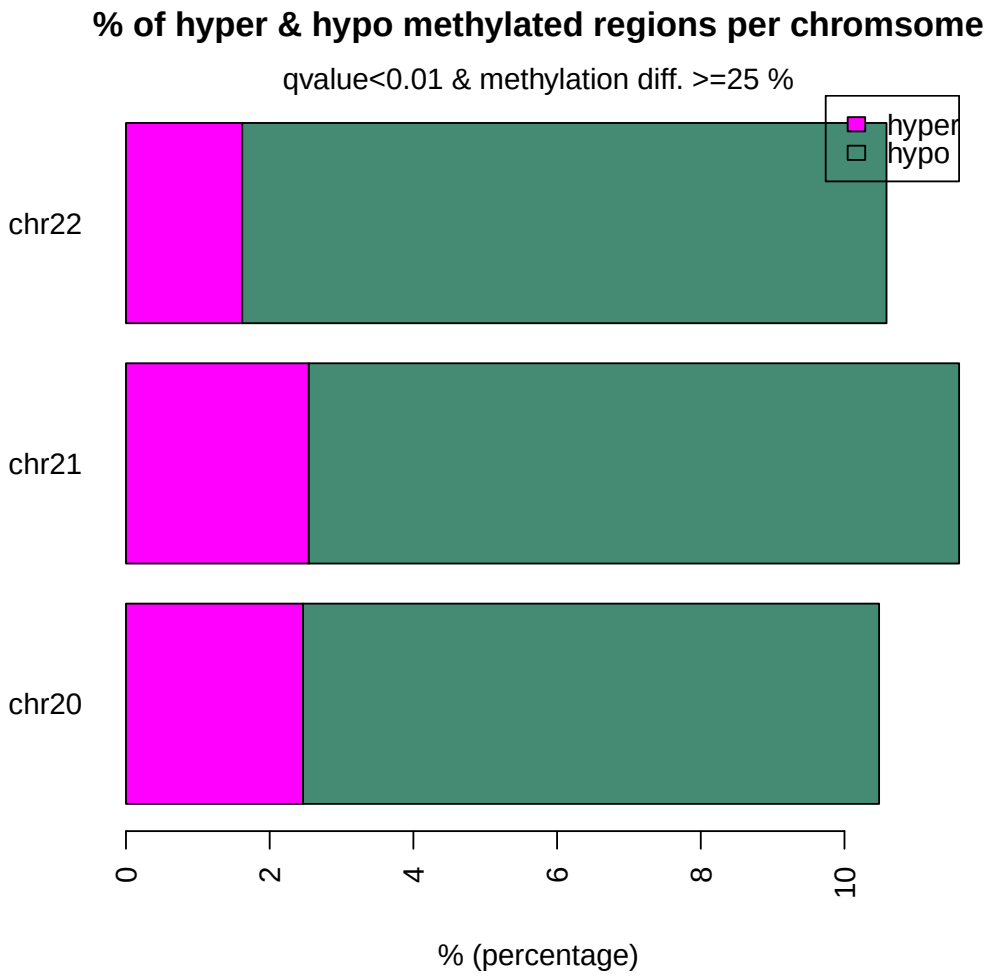
5.9 Differential methylation events per chromosome

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.

```
# 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)
```

```
## $diffMeth.per.chr
##     chr number.of.hypomethylated percentage.of.hypomethylated
## 1 chr20                     2498                         8.017
## 2 chr21                     1316                         9.051
## 3 chr22                     2638                         8.967
##   number.of.hypermethylated percentage.of.hypermethylated
## 1                       768                         2.465
## 2                       370                         2.545
## 3                       476                         1.618
##
## $diffMeth.all
##   percentage.of.hypermethylated number.of.hypermethylated
## 1                         2.149                      1614
##   percentage.of.hypomethylated number.of.hypomethylated
## 1                        8.589                     6452
##
```

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



5.10 Annotating differential methylation events

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("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 promoter>exon>intron precedence) :
##   promoter       exon     intron intergenic
##      14.43      14.11      32.89      38.57
##
##
## percentage of annotation boundaries with feature overlap :
## promoter     exon   intron
##   20.423    3.851   10.507
##
##
## summary of distances to the nearest TSS :
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0    2440   10200   30800   32300  952000
```

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")
```

5.10.1 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 feature.strand
## 427          1         -121458    NM_000214              -
## 428          2          271458    NR_038972              -
## 310          3           -1871    NM_018354              -
```

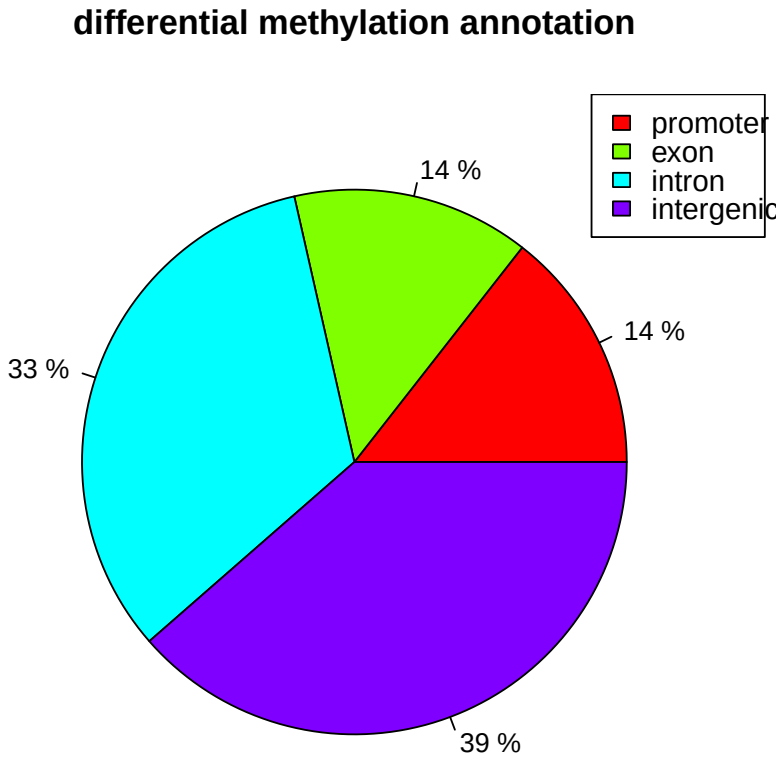
It is also desirable to get percentage/number of differentially methylated regions that overlap with intron/exon/promoters

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

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

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

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



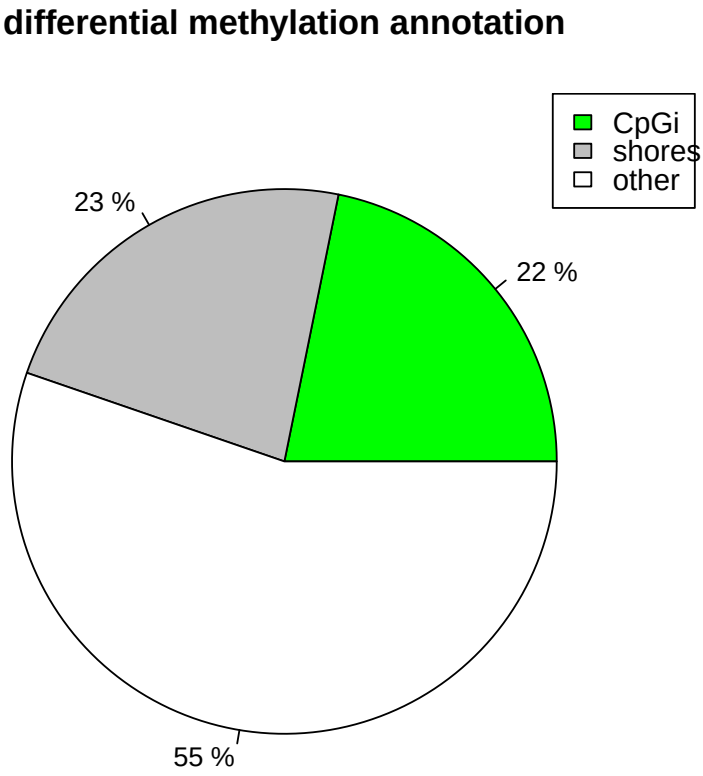
If you need to get the annotation table which shows overlap status of the methylation events with the annotation such as promoter/exon/intron, you can use `getMembers()` function.

```
head(getMembers(diffAnn))
```

```
##      prom exon intron
## [1,]    0    0      0
## [2,]    0    0      0
## [3,]    0    0      0
## [4,]    0    0      0
## [5,]    0    0      0
## [6,]    1    0      1
```

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
##   20.423    3.851   10.507
```

5.11 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 numTs
## 1 chr20.33505636.33507636.NA chr20 33505636 33507636      +      727     4   723
```

```
## 2   chr20.8060295.8062295.NA chr20  8060295  8062295      +      954    15   939
## 3   chr20.3137055.3139055.NA chr20  3137055  3139055      +      553     2   551
## 4   chr20.1046239.1048239.NA chr20  1046239  1048239      +      535     4   531
## 5 chr20.13923145.13925145.NA chr20 13923145 13925145      +      534    10   524
## 6 chr20.10362950.10364950.NA chr20 10362950 10364950      +      165     0   165
```

5.12 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)
```

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

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

```
## GRanges with 75116 ranges and 13 elementMetadata values:
##           seqnames               ranges strand   |             id coverage1
##              <Rle>            <IRanges>  <Rle>   |       <factor> <integer>
##       [1]    chr20 [10100840, 10100840]      +   | chr20.10100840        29
##       [2]    chr20 [10100844, 10100844]      +   | chr20.10100844        29
##       [3]    chr20 [10100852, 10100852]      +   | chr20.10100852        29
##       [4]    chr20 [10146236, 10146236]      -   | chr20.10146236        22
##       [5]    chr20 [10146246, 10146246]      -   | chr20.10146246        21
##       [6]    chr20 [10146248, 10146248]      -   | chr20.10146248        21
##       [7]    chr20 [10146438, 10146438]      -   | chr20.10146438        25
##       [8]    chr20 [10146449, 10146449]      -   | chr20.10146449        25
##       [9]    chr20 [10146463, 10146463]      -   | chr20.10146463        25
##       ...      ...                 ...     ... ...            ...       ...
##   [75108]    chr22 [49568898, 49568898]      -   | chr22.49568898        13
##   [75109]    chr22 [49568930, 49568930]      +   | chr22.49568930        25
##   [75110]    chr22 [49568935, 49568935]      +   | chr22.49568935        25
##   [75111]    chr22 [49568940, 49568940]      +   | chr22.49568940        25
##   [75112]    chr22 [49568942, 49568942]      +   | chr22.49568942        25
##   [75113]    chr22 [49568948, 49568948]      +   | chr22.49568948        25
##   [75114]    chr22 [49568959, 49568959]      +   | chr22.49568959        25
##   [75115]    chr22 [49568965, 49568965]      +   | chr22.49568965        25
##   [75116]    chr22 [49568978, 49568978]      +   | chr22.49568978        25
##              numCs1    numTs1 coverage2    numCs2    numTs2 coverage3    numCs3
##           <numeric> <numeric> <integer> <numeric> <numeric> <integer> <numeric>
##       [1]         2        27        11         0        11        43         2
##       [2]         2        27        11         1        10        42         2
##       [3]         0        29        11         0        11        43         0
##       [4]         2        20        26         5        21        16         0
##       [5]         0        21        23         0        23        17         0
##       [6]         0        21        27         1        26        17         2
##       [7]         0        25       104         4       100        43         1
##       [8]         1        24       104         3       101        42         0
##       [9]         0        25       105         0       105        43         0
##       ...       ...       ...       ...       ...       ...       ...       ...
##   [75108]         0        13        53         0        53        41         0
##   [75109]         0        25       103         0       103        26         0
```

```
##   [75110]         0        25       108         0       108        26         0
##   [75111]         0        25       109         0       109        26         0
##   [75112]         0        25       107         0       107        26         0
##   [75113]         0        25       106         0       106        26         0
##   [75114]         0        25       105         0       105        26         0
##   [75115]         0        25       106         0       106        25         0
##   [75116]         0        25       106         0       106        24         0
##              numTs3 coverage4    numCs4    numTs4
##           <numeric> <integer> <numeric> <numeric>
##       [1]        41        10         0        10
##       [2]        40        10         0        10
##       [3]        43        10         0        10
##       [4]        16        10         1         9
##       [5]        17        10         1         9
##       [6]        15        10         1         9
##       [7]        42        30         0        30
##       [8]        42        30         2        28
##       [9]        43        30         0        30
##       ...       ...       ...       ...       ...
##   [75108]        41        17         0        17
##   [75109]        26        23         0        23
##   [75110]        26        23         0        23
##   [75111]        26        23         0        23
##   [75112]        26        23         0        23
##   [75113]        26        23         0        23
##   [75114]        26        23         0        23
##   [75115]        25        23         0        23
##   [75116]        24        23         0        23
##   ---
##   seqlengths:
##    chr20 chr21 chr22
##       NA    NA    NA
```

```
class(myDiff)
```

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

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

```
## GRanges with 75116 ranges and 3 elementMetadata values:
##           seqnames              ranges strand   |             id    qvalue
##              <Rle>           <IRanges>  <Rle>   |       <factor> <numeric>
##       [1]    chr20 [10100840, 10100840]      +   | chr20.10100840    0.6995
##       [2]    chr20 [10100844, 10100844]      +   | chr20.10100844    0.5694
##       [3]    chr20 [10100852, 10100852]      +   | chr20.10100852    0.6995
##       [4]    chr20 [10146236, 10146236]      -   | chr20.10146236    0.2366
##       [5]    chr20 [10146246, 10146246]      -   | chr20.10146246    0.2844
##       [6]    chr20 [10146248, 10146248]      -   | chr20.10146248    0.2025
##       [7]    chr20 [10146438, 10146438]      -   | chr20.10146438    0.5537
##       [8]    chr20 [10146449, 10146449]      -   | chr20.10146449    0.6995
##       [9]    chr20 [10146463, 10146463]      -   | chr20.10146463    0.6995
##       ...      ...                 ...    ... ...            ...       ...
##   [75108]    chr22 [49568898, 49568898]      -   | chr22.49568898    0.6995
##   [75109]    chr22 [49568930, 49568930]      +   | chr22.49568930    0.6995
##   [75110]    chr22 [49568935, 49568935]      +   | chr22.49568935    0.6995
```

```
##   [75111]    chr22 [49568940, 49568940]      +   | chr22.49568940    0.6995
##   [75112]    chr22 [49568942, 49568942]      +   | chr22.49568942    0.6995
##   [75113]    chr22 [49568948, 49568948]      +   | chr22.49568948    0.6995
##   [75114]    chr22 [49568959, 49568959]      +   | chr22.49568959    0.6995
##   [75115]    chr22 [49568965, 49568965]      +   | chr22.49568965    0.6995
##   [75116]    chr22 [49568978, 49568978]      +   | chr22.49568978    0.6995
##           meth.diff
##           <numeric>
##       [1]     1.226
##       [2]     3.654
##       [3]     0.000
##       [4]    10.737
##       [5]    -3.704
##       [6]    -9.028
##       [7]     1.731
##       [8]     0.323
##       [9]     0.000
##       ...       ...
##   [75108]         0
##   [75109]         0
##   [75110]         0
##   [75111]         0
##   [75112]         0
##   [75113]         0
##   [75114]         0
##   [75115]         0
##   [75116]         0
##   ---
##   seqlengths:
##    chr20 chr21 chr22
##       NA    NA    NA
```

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
```

```
##                id   chr    start      end strand coverage1 numCs1 numTs1 coverage2
## 1  chr20.10100840 chr20 10100840 10100840      +        29      2     27        11
## 2  chr20.10100844 chr20 10100844 10100844      +        29      2     27        11
## 3  chr20.10100852 chr20 10100852 10100852      +        29      0     29        11
## 4  chr20.10146236 chr20 10146236 10146236      -        22      2     20        26
## 5  chr20.10146246 chr20 10146246 10146246      -        21      0     21        23
## 6  chr20.10146248 chr20 10146248 10146248      -        21      0     21        27
## 7  chr20.10146438 chr20 10146438 10146438      -        25      0     25       104
## 8  chr20.10146449 chr20 10146449 10146449      -        25      1     24       104
## 9  chr20.10146463 chr20 10146463 10146463      -        25      0     25       105
## 10 chr20.10146472 chr20 10146472 10146472      -        24      0     24       101
##    numCs2 numTs2 coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
## 1       0     11        43      2     41        10      0     10
## 2       1     10        42      2     40        10      0     10
## 3       0     11        43      0     43        10      0     10
## 4       5     21        16      0     16        10      1      9
## 5       0     23        17      0     17        10      1      9
## 6       1     26        17      2     15        10      1      9
## 7       4    100        43      1     42        30      0     30
## 8       3    101        42      0     42        30      2     28
```

```
## 9       0    105        43      0     43        30      0     30
## 10      1    100        43      1     42        24      0     24
```

```
select(myDiff, 20:30)  # select rows 10
```

```
##                 id   chr    start      end strand  pvalue  qvalue meth.diff
## 20 chr20.10148116 chr20 10148116 10148116      + 0.47835 0.59398     1.136
## 21 chr20.10148136 chr20 10148136 10148136      + 1.00000 0.69954     0.000
## 22 chr20.10148138 chr20 10148138 10148138      + 1.00000 0.69954     0.000
## 23 chr20.10148155 chr20 10148155 10148155      - 1.00000 0.69954     0.000
## 24 chr20.10148166 chr20 10148166 10148166      - 1.00000 0.69954     0.000
## 25 chr20.10148183 chr20 10148183 10148183      - 1.00000 0.69954     0.000
## 26 chr20.10148190 chr20 10148190 10148190      - 1.00000 0.69954     0.000
## 27 chr20.10148194 chr20 10148194 10148194      - 1.00000 0.69954     0.000
## 28 chr20.10148199 chr20 10148199 10148199      - 1.00000 0.69954     0.000
## 29 chr20.10148203 chr20 10148203 10148203      + 0.25736 0.39650     1.538
## 30 chr20.10148204 chr20 10148204 10148204      - 0.01307 0.04123    -5.085
```

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))
```

```
##                 id   chr    start      end strand coverage1 numCs1 numTs1 coverage2
## 1 chr20.10100840 chr20 10100840 10100840      +        29      2     27        11
## 2 chr20.10100844 chr20 10100844 10100844      +        29      2     27        11
## 3 chr20.10100852 chr20 10100852 10100852      +        29      0     29        11
## 4 chr20.10146236 chr20 10146236 10146236      -        22      2     20        26
## 5 chr20.10146246 chr20 10146246 10146246      -        21      0     21        23
## 6 chr20.10146248 chr20 10146248 10146248      -        21      0     21        27
##   numCs2 numTs2 coverage3 numCs3 numTs3 coverage4 numCs4 numTs4
## 1      0     11        43      2     41        10      0     10
## 2      1     10        42      2     40        10      0     10
## 3      0     11        43      0     43        10      0     10
## 4      5     21        16      0     16        10      1      9
## 5      0     23        17      0     17        10      1      9
## 6      1     26        17      2     15        10      1      9
```

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

```
##                 id   chr    start      end strand pvalue qvalue meth.diff
## 1 chr20.10100840 chr20 10100840 10100840      + 0.7739 0.6995     1.226
## 2 chr20.10100844 chr20 10100844 10100844      + 0.4457 0.5694     3.654
## 3 chr20.10100852 chr20 10100852 10100852      + 1.0000 0.6995     0.000
## 4 chr20.10146236 chr20 10146236 10146236      - 0.1261 0.2366    10.737
## 5 chr20.10146246 chr20 10146246 10146246      - 0.1618 0.2844    -3.704
## 6 chr20.10146248 chr20 10146248 10146248      - 0.1020 0.2025    -9.028
```

6 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, Chrisopher Mason and Ari Melnick for their contribution of ideas and support. Their support and discussions lead to development of methylKit.

7 R session info

```
sessionInfo()
```

```
## R version 2.14.1 (2011-12-22)
## 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] methylKit_0.4.1     GenomicRanges_1.6.4 IRanges_1.12.5      knitr_0.1
##
## loaded via a namespace (and not attached):
##  [1] codetools_0.2-8   data.table_1.7.7  digest_0.5.1      evaluate_0.4.1
##  [5] formatR_0.3-4     highlight_0.3.1   KernSmooth_2.23-7 parallel_2.14.1
##  [9] parser_0.0-14     plyr_1.7.1        Rcpp_0.9.9        stringr_0.6
## [13] tools_2.14.1
```