The bsseq user's guide

Kasper Daniel Hansen
kasperdanielhansen@gmail.com

Modified: June 10, 2015. Compiled: May 15, 2016

Capabilities

The packages required for this vignette are

```
library(bsseq)
```

Introduction

This *R* package is the reference implementation of the BSmooth algorithm for analyzing whole-genome bisulfite sequencing (WGBS) data. This package does *not* contain alignment software, which is available from http://rafalab.jhsph.edu/bsmooth. This package is not intended for analysis of single-loci bisulfite sequencing (typically Sanger bisulfite sequencing or pyro bisulfite sequencing).

The package has been used to analyze capture bisulfite sequencing data. For this type of data, the most useful parts of the package is its data-handling functions. The BSmooth algorithm itself may not be useful for a standard capture bisulfite sequencing experiment, since it relies heavily on smoothing, which again requires that we have measured methylation in bigger regions of the genome.

The BSmooth algorithm is described in detail in [**?**]. It was applied to human cancer data in [**?**] and we have also used it to analyze data from Rhesus Macaque [**?**]. Specifically, the algorithm uses smoothing to get reliable semi-local methylation estimates from low-coverage bisulfite sequencing. After smoothing it uses biological replicates to estimate biological variation and identify differentially methylated regions (DMRs). The smoothing portion could be used even on a single sample, but we believe that variation between individuals is an important aspect of DNA methylation and needs to be accounted for, see also [**?**] for a relevant discussion.

The main focus for our development has been the analysis of CpG methylation in humans, and we have successfully used it in other primates [**?**]. It is highly likely that the approach will work in non-human organisms, but care must be taken: the smoothing parameters (which we have selected based on human data) should be evaluated carefully. Furthermore, it may not be suitable at all for organisms with vastly different (from humans) methylation structures.

With respect to non-CpG methylation, the situation is mixed. We have never used the algorithm to analyze non-CpG methylation, but it should be straightforward to do so. However, the data structures used in the current code might not conveniently scale from the 28.2M CpGs in the human genome to the roughly 2x585M Cs (it may well be possible to do an separate analysis for each chromosome). This should be less of an issue for organisms with smaller genomes. We are considering changing these underlying data structures to allow for easier analysis of non-CpG methylation in humans.

System Requirements

The package requires that all the data is loaded into system memory. By "data" we do not mean the individual reads (which is big for a whole-genome experiment). Instead, what we need are summarized data given us the number of reads supporting methylation as well as the total number of reads at each loci. Focusing on human data, we need to work with objects with a maximum of 28.2 million entries, per sample (since there are roughly 28.2 millions CpGs in the human genome). This provides us with an upper limit on the data object.

Based on this, the 8 samples from [**?**] including the smoothed values, take up roughly 1.2GB of RAM, meaning an analysis can easily be done with 8GB of RAM. In order to improve speed, the package allows for easy parallel processing of samples/chromosomes. This will require multiple copies of the data object for each core used, increasing the memory usage substantially to perhaps even 32GB. This can be avoided by processing the samples sequentially at the cost of speed.

On a 64GB node, the 8 samples from [**?**] takes roughly one hour to process in parallel using 8 cores (using much less than 64GB of RAM in total). This does not including parsing the data from the alignment output files.

Some terminology

Because not all methylation happens at CpG sites, we have tried to use the term "methylation loci" instead of CpG. We use this term to refer to a single-base methylation site.

Some standard terms from the DNA methylation field: differentially methylated region (DMR), hyper methylation (methylation that is higher in one condition/sample than in another), hypo methylation (methylation that is lower in one condition/sample than in another), and finally differentially methylated position (DMP) referring to a single loci.

Citation

If you use this package, please cite our BSmooth paper [**?**].

Overview

The package assumes that the following data has been extracted from alignments:

1. genomic positions, including chromosome and location, for methylation loci.
2. a (matrix) of M (Methylation) values, describing the number of read supporting methylation covering a single loci. Each row in this matrix is a methylation loci and each column is a sample.
3. a (matrix) of Cov (Coverage) values, describing the number of read supporting methylation covering a single loci. Each row in this matrix is a methylation loci and each column is a sample.

We can have a look at some data from [**?**], specifically chromosome 22 from the IMR90 cell line.

```
data(BS.chr22)
BS.chr22
```

```
## An object of type 'BSseq' with
##   494728 methylation loci
##   2 samples
## has not been smoothed
```

The genomic positions are stored as a `GRanges` object `GRanges` are general genomic regions; we represent a single base methylation loci as an interval of width 1 (which may seem a bit strange, but there are good reasons for this). For example, the first 4 loci in the Lister data are

```
head(granges(BS.chr22), n = 4)
```

```
## GRanges object with 4 ranges and 0 metadata columns:
##       seqnames              ranges strand
##          <Rle>           <IRanges>  <Rle>
##   [1]    chr22 [14430632, 14430632]      *
##   [2]    chr22 [14430677, 14430677]      *
##   [3]    chr22 [14430687, 14430687]      *
##   [4]    chr22 [14430702, 14430702]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We also have the M and Cov matrices

```
head(getCoverage(BS.chr22, type = "M"), n = 4)
```

```
##      [,1] [,2]
## [1,]   17   20
## [2,]    4   20
## [3,]    6   19
## [4,]    2    4
```

```
head(getCoverage(BS.chr22), n = 4)
```

```
##      [,1] [,2]
## [1,]   18   23
```

```
## [2,]   11   28
## [3,]   10   25
## [4,]    8   21
```

Since CpG methylation is symmetric on the two strands of a chromosome, we aggregated reads on the forward and reverse strand to form a single value, and we assume the genomic position points to the C of the CpG. It is not crucial in any way to do this, one may easily analyze each strand separately, but CpG methylation is symmetric and this halves the number of loci.

How to input your methylation data into this data structure (called a `BSseq` object) is described in a section below. We also have a section on how to operate on these types of objects.

An analysis typically consists of the following steps.

1. Smoothing, using the function `BSmooth`.
2. Compute t-statistics, using the function `BSmooth.tstat`. This converts the `BSseq` object into a `BSseqTstat` object.
3. Threshold these t-statistics to identify DMRs, using the function `dmrFinder`, returning a simple `data.frame`.

It is usually a good idea to look at the smoothed data either before or after identifying DMRs. This can be done using the functions `plotRegion` and `plotManyRegions`.

We also have functions for assessing goodness of fit for binomial and poison models; this is like to be of marginal interest to most users. See the man page for `binomialGoodnessOfFit`.

We also allow for easy computation of Fisher's exact test for each loci, using the function `fisherTests`.

Using objects of class BSseq

Basic operations

Objects of class `BSseq` contains a `GRanges` object with the genomic locations. This `GRanges` object can be obtained by `granges`. A number of standard `GRanges` methods works directly on the `BSseq` object, such as `start`, `end`, `seqnames` (chromosome names) etc.

These objects also contains a `phenoData` object for sample pheno data. Useful methods are `sampleNames`, `pData`.

Finally, we have methods such as `dim`, `ncol` (number of columns; number of samples), `nrow` (number of rows; number of methylation loci).

Objects can be subsetted using two indicies `BSseq[i,j]` with the first index being methylation loci and the second index being samples. Another very useful way of subsetting the object is by using the method `subsetByOverlaps`. This selects all methylation loci inside a set of genomic intervals (there is a difference between first and second argument and either can be `BSseq` or `GRanges`).

Examples:

```
head(start(BS.chr22), n = 4)
```

```
## [1] 14430632 14430677 14430687 14430702
```

```
head(seqnames(BS.chr22), n = 4)
```

```
## factor-Rle of length 4 with 1 run
##   Lengths:     4
##   Values : chr22
## Levels(1): chr22
```

```
sampleNames(BS.chr22)
```

```
## [1] "r1" "r2"
```

```
pData(BS.chr22)
```

```
## DataFrame with 2 rows and 1 column
##            Rep
##    <character>
## r1  replicate1
## r2  replicate2
```

```
dim(BS.chr22)
```

```
## [1] 494728      2
```

```
BS.chr22[1:6,1]
```

```
## An object of type 'BSseq' with
##   6 methylation loci
##   1 samples
## has not been smoothed
```

```
subsetByOverlaps(BS.chr22,
                 GRanges(seqnames = "chr22",
                         ranges = IRanges(start = 1, end = 2*10^7)))
```

```
## An object of type 'BSseq' with
##   67082 methylation loci
##   2 samples
## has not been smoothed
```

Data handling

We have a number of functions for manipulating one or more `BSseq` objects.

`BSseq` instantiates an object of class `BSseq`. Genomic locations are passed in, either as a `GRanges` object (argument `gr`) or as chromosome and location vectors (arguments `chr` and `pos`). The arguments `M` and `Cov` accepts matrices, and it is possible to directly give it a `phenoData` object.

```
M <- matrix(0:8, 3, 3)
Cov <- matrix(1:9, 3, 3)
BStmp <- BSseq(chr = c("chr1", "chrX", "chr1"), pos = 1:3,
               M = M, Cov = Cov, sampleNames = c("A1","A2", "B"))
```

A BSseq object may be ordered by orderBSseq. This ensures that data from a single chromosome appears in an ordered, contiguous block. There is also the possibility for specifying the chromosome order (this is less important). The smoothing functions assumes that the underlying BSseq has been ordered.

```
granges(BStmp)
```

```
## GRanges object with 3 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1    [1, 1]      *
##   [2]     chr1    [3, 3]      *
##   [3]     chrX    [2, 2]      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
BStmp <- orderBSseq(BStmp, seqOrder = c("chr1", "chrX"))
granges(BStmp)
```

```
## GRanges object with 3 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1    [1, 1]      *
##   [2]     chr1    [3, 3]      *
##   [3]     chrX    [2, 2]      *
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

chrSelectBSseq performs the often useful task of selecting one or more chromosomes and can also order the output. In case order = TRUE, the output is ordered according to the order of the seqnames argument.

```
chrSelectBSseq(BStmp, seqnames = "chr1", order = TRUE)
```

```
## An object of type 'BSseq' with
##   2 methylation loci
##   3 samples
## has not been smoothed
```

Of course, in this case the function does little, since BS.chr22 already only contains data from chromosome 22.

combine combines two BSseq objects in the following way: the samples for the return objects is the union of the samples from the two objects, and the methylation loci are the union of the two methylation

loci. The two objects do not need to have measured the same loci (in the example below, BStmp has data on chromosome 1 and X)..

```
BStmp2 <- combine(BStmp, BS.chr22[1:3,])
```

```
## Warning in .Seqinfo.mergexy(x, y):  The 2 combined objects have no sequence levels
in common.  (Use
##  suppressWarnings() to suppress this warning.)
```

```
granges(BStmp2)
```

```
## GRanges object with 6 ranges and 0 metadata columns:
##       seqnames              ranges strand
##          <Rle>           <IRanges>  <Rle>
##   [1]     chr1 [       1,        1]      *
##   [2]     chr1 [       3,        3]      *
##   [3]     chrX [       2,        2]      *
##   [4]    chr22 [14430632, 14430632]      *
##   [5]    chr22 [14430677, 14430677]      *
##   [6]    chr22 [14430687, 14430687]      *
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

```
getCoverage(BStmp2)
```

```
##      A1 A2 B r1 r2
## [1,]  1  4 7  0  0
## [2,]  3  6 9  0  0
## [3,]  2  5 8  0  0
## [4,]  0  0 0 18 23
## [5,]  0  0 0 11 28
## [6,]  0  0 0 10 25
```

collapseBSseq performs the often useful task of adding several columns of a BSseq object. This is often used in the beginning of an analysis where each column may correspond to a lane and several such columns represents the data for a single biological sample.

```
collapseBSseq(BStmp, columns = c("A", "A", "B"))
```

```
## An object of type 'BSseq' with
##   3 methylation loci
##   2 samples
## has not been smoothed
```

Obtaining coverage (methylation)

Coverage, either Cov or M values, are obtained by getCoverage, using the type argument:

```
head(getCoverage(BS.chr22, type = "Cov"), n = 4)
```

```
##      [,1] [,2]
## [1,]   18   23
## [2,]   11   28
## [3,]   10   25
## [4,]    8   21
```

```
head(getCoverage(BS.chr22, type = "M"), n = 4)
```

```
##      [,1] [,2]
## [1,]   17   20
## [2,]    4   20
## [3,]    6   19
## [4,]    2    4
```

This will return a – possibly very large – matrix. It is also possible to get region based coverage by using the `regions` argument to the function. This argument is either a `data.frame` (with columns `chr`, `start` and `end`) or a `GRanges` object. Let us do an example

```
regions <- GRanges(seqnames = c("chr22", "chr22"),
                   ranges = IRanges(start = 1.5 * 10^7 + c(0,200000),
                                    width = 1000))
getCoverage(BS.chr22, regions = regions, what = "perRegionTotal")
```

```
##      r1 r2
## [1,] 30 38
## [2,] NA NA
```

When `what` is `perRegionTotal` the return value is the total coverage of each region (and note that regions without coverage return `NA`). Similarly, `perRegionAverage` yields the average coverage in the region. However, it is often useful to get the actual values, like

```
getCoverage(BS.chr22, regions = regions, what = "perBase")
```

```
## [[1]]
##      [,1] [,2]
## [1,]   21   30
## [2,]    2    8
## [3,]    7    0
##
## [[2]]
## NULL
```

This is the default behaviour, and it returns a list where each element corresponds to a region. Note that regions with no coverage gets a `NULL`.

Methylation estimates can be obtained in the same way, using the function `getMeth`. If `type` is set to `raw` the function returns simple single-loci methylation estimates (which are M/Cov). To get smoothed

estimates, the BSseq object needs to have been smoothed using Bsmooth, and type set to smooth (default). The getCoverage, getMeth has a regions and a what argument. For getMeth the what argument can be perBase or perRegion (the later is really the per-region average methylation). Additionally, confidence intervals can be computed using a method taking possibly low coverage loci into account as well as loci where the methylation percentage might be close to 0 or 1 [**?**]. Currently, confidence intervals cannot be computed for region-level summary estimates. Examples

```
getMeth(BS.chr22, regions, type = "raw")
```

```
## [[1]]
##           [,1]      [,2]
## [1,] 0.1904762 0.1666667
## [2,] 1.0000000 0.7500000
## [3,] 0.1428571       NaN
##
## [[2]]
## NULL
```

```
getMeth(BS.chr22, regions[2], type = "raw", what = "perBase")
```

```
## [[1]]
## NULL
```

Reading data

Alignment output from the BSmooth alignment suite

It is straightforward to read output files (summarized evidence) from the BSmooth alignment suite, using read.bsmooth. This function takes as argument a number of directories, usually corresponding to samples, with the alignment output. Sometimes, one might want to read only certain chromosomes, which can be accomplished by the seqnames argument. Also, the default values of qualityCutoff = 20 ensures that we do not use methylation evidence with a base quality strictly lower than 20 (since we may not be confident whether the read really supports methylation or not). The function read.bsmooth allows for both gzipped and non-gzipped input files. It is faster to read gzipped files, so we recommend gzipping post-alignment.

During the development of BSmooth we experimented with a number of different output formats. These legacy formats can be read with read.umtab and read.umtab2.

Alignment output from other aligners

We are interested in adding additional parsers to the package; if your favorite alignment software is not supported, feel free to get in touch.

In general, we need summarized methylation evidence. In short, for each methylation loci, we need to know how many reads cover the loci and how many of those reads support methylation.

As an example, consider the Lister data. The files posted online looks like

```
> head mc_imr90_r1_22
assembly        position        strand  class   mc      h
22      14430632        +       CG      9       10
22      14430633        -       CG      8       8
22      14430677        +       CG      1       1
22      14430678        -       CG      3       10
22      14430688        -       CG      6       10
22      14430703        -       CG      2       8
22      14431244        +       CG      5       10
22      14431245        -       CG      5       11
```

For these files, the evidence is split by strand. It is often easiest to read in one sample at a time, instantiate a `BSseq` object and the use `combine` and `collapseBSseq` to combine the samples, since these functions deal objects that have different sets of CpGs. In the Lister data, note that the position is the position of the "C", so basically, if you want to combine the evidence from both strands, CpGs on the "-" strand needs to have 1 subtracted from their position. A full script showing how these files are used to create `BS.chr22` can be found in `inst/scripts/get_BS.chr22.R` in the *bsseq* package. The path of this file may be found by

```
system.file("scripts", "get_BS.chr22.R", package = "bsseq")
```

Analysis

Computing smoothed methylation levels is simple. We subset `BS.chr22` so we only smooth 1 sample, for run-time purpose

```
BS.chr22.1 <- BSmooth(BS.chr22[,"r1"], verbose = TRUE)
```

```
## [BSmooth] preprocessing ... done in 0.6 sec
## [BSmooth] smoothing by 'sample' (mc.cores = 1, mc.preschedule = FALSE)
## [BSmooth] smoothing done in 55.1 sec
```

```
BS.chr22.1
```

```
## An object of type 'BSseq' with
##   494728 methylation loci
##   1 samples
## has been smoothed with
##   BSmooth (ns = 70, h = 1000, maxGap = 100000000)
```

A number of arguments pertains to using multiple cores. This is useful if you have multiple samples or chromosomes. `mc.cores` tells `BSmooth` how many cores to use and `parallelBy` describes whether

the parallelization is over samples or chromosomes. If the parallelization is over samples, each core will compute the smoothing of the entire genome. And if it is over samples, each cores will smooth all samples for one or more chromosomes. The appropriate choice depends on the number of samples and cpu cores available. If you have the exact same number of samples as cores, the fastest is `parallelBy = "sample"`. If you have less samples than cores, the fastest is `parallelBy = "chromosome"`. The argument `mc.preshedule` should not need to be changed (unless perhaps if a small value of `maxGap` is uses); see the man page for `parallel::mclapply`. Note that setting `mc.cores` to a value greater than 1 is not support on MS Windows due to a limitation of the operating system.

For a more detailed discussion of the analysis tools, read the companion vignette "Analyzing WGBS with the bsseq package", which also finding DMRs and plotting them.

Fisher's exact tests may be efficiently computed using the function `fisherTests`.

Binomial and poisson goodness of fit tests statistics may be computed using `binomialGoodnessOfFit` and `poissonGoodnessOfFit`.

SessionInfo

- R version 3.3.0 (2016-05-03), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.32.0, BiocGenerics 0.18.0, GenomeInfoDb 1.8.2, GenomicRanges 1.24.0, IRanges 2.6.0, S4Vectors 0.10.0, SummarizedExperiment 1.2.2, bsseq 1.8.2, limma 3.28.4
- Loaded via a namespace (and not attached): BiocStyle 2.0.2, R.methodsS3 1.7.1, R.oo 1.20.0, R.utils 2.3.0, Rcpp 0.12.5, XVector 0.12.0, chron 2.3-47, colorspace 1.2-6, data.table 1.9.6, evaluate 0.9, formatR 1.4, grid 3.3.0, gtools 3.5.0, highr 0.6, knitr 1.13, lattice 0.20-33, locfit 1.5-9.1, magrittr 1.5, matrixStats 0.50.2, munsell 0.4.3, permute 0.9-0, plyr 1.8.3, scales 0.4.0, stringi 1.0-1, stringr 1.0.0, tools 3.3.0, zlibbioc 1.18.0