Reading and filtering long-read modification data with bsseq

SrenBlikdal

3 February 2026

Abstract

This guide outlines a likelihood-based framework for preparing and filtering long-read data from Oxford Nanopore for analysis with the `bsseq` package. Unlike conversion-based methods, single-molecule sequencing captures both the nucleotide sequence and epigenetic modifications directly. This information enables detection of sample-specific CpG loci and supports effective data filtering prior to downstream modification analysis. The guide demonstrates how to process Oxford Nanopore data into bedMethyl format and filter data based on CpG likelihoods prior to modification analysis in `bsseq`.

Package

bsseq 1.48.0

Contents

1 Introduction

This guide outlines how to read and filter Oxford Nanopore sequencing data using the `bsseq` package in R. Designed for DNA methylation analysis, `bsseq` offers tools for reading, filtering, analyzing and visualizing modification data.

The first part focuses on preprocessing. It begins with processing raw `POD5` sequencing files into basecalled and modification-called `BAM` files using `dorado`. Next, these reads are mapped to a reference genome, while preserving modification information, using `SAMtools` and `Minimap2`. Finally, modification information is summarized in read-based `bedMethyl` files using `Modkit`.

The second part of the guide demonstrates how to read `bedMethyl` files as `MethylCounts` objects, and filtering them based on coverage and likelihood of representing a homozygous and/or heterozygous CpG loci. This workflow is illustrated using both single-sample and a multi-sample `MethylCounts` object.

1.1 Terminology

The following terms are used throughout this document:

Basecalling: Determining the sequenced nucleotide sequence (A,C,G or T) from the raw sequencing signals.
Modification calling: Identifying the modification state from the raw sequencing signal. In this guide, limited to C in CpG-context modifications: C, 5hmC or 5mC.
CpG site: A cytosine followed by a guanine in a DNA strand (5' to 3').
CpG locus: The combined term for the CpG site the forward and reverse strand in double stranded DNA.
CpG status: Inferring if a locus is a homozygous CpG, heterozygous CpG, a homozygous or heterozygous CpG or not a CpG at all based on sequencing data mapped to the position in the reference genome.
.99 homozygous CpG filtering: Filtering for loci with scaled likelihood above 99% of being a homozygous CpG locus given the data.
.99 heterozygous CpG filtering: Filtering for loci with scaled likelihood of above 99% of being a heterozygous CpG locus given the data.
.99 'allCpG' filtering: Filtering for loci with scaled likelihood of above 99% of being a homozygous CpG locus or heterozygous CpG locus given the data.
Reference-guided filtering: Restricting analysis to loci that overlap CpG loci in the reference genome (using the –cpg or –preset traditional in Modkit).

1.2 Citation

If you use the likelihood filtering, please cite our preprint [Hansen:2025], while the general use can be cited from the BSmooth paper [Hansen:2012].

1.3 Dependencies

```
library(bsseq)
library(tidyverse)
#Additional software modules needed for preprocessing:
#Dorado https://github.com/nanoporetech/dorado
```

```
#SAMtools https://github.com/samtools
#Minimap2 https://github.com/lh3/minimap2
#Modkit https://github.com/nanoporetech/modkit
```

2 Preprocessing

Before importing modification data from Oxford Nanopore sequencing into the `bsseq` package, the raw `POD5` data must be processed to generate base and modification called reads, mapped to a reference genome, and summarized in the pileup format, `bedMethyl`.

2.1 Base and modification calling with Dorado

To obtain the base and modification calls from the raw signal data, we use the `dorado` basecaller. The following command will base and modification call the `POD5` files in the input directory using a CpG-context model to call 5-methylcytosine and 5-hydroxymethylcytosine and output an unaligned `BAM` file. The unaligned `BAM` file will contain basecalls and modification calls for each read.

```
# Set input and output directories
input_directory= #/insert/input/directory/with/POD5/files/here
output_directory= #/insert/output/directory/for/unaligned/bam/files/here

# Run dorado basecaller with modification calling
dorado basecaller sup,5mCG_5hmCG \
$input_directory/ > $output_directory/unaligned.bam
```

2.2 Mapping with Minimap2 and SAMtools

To map the unaligned `BAM` file to a reference genome, we use `minimap2` and `samtools`. The following command converts the `BAM` to `FASTQ`, maps the reads, and output a sorted and indexed `BAM` file with modification tags.

```
# Set output directory and reference genome
output_directory= #/insert/output/directory/for/unaligned/bam/files/here
reference_genome= #/insert/reference/genome/here
cd $output_directory

# Map reads to the reference genome
samtools fastq -TMM,ML unaligned.bam | \
  minimap2 -ax map-ont -y $reference_genome - | \
  samtools view -bS -| \
  samtools sort - > aligned.bam

# Index the aligned BAM file
samtools index aligned.bam
```

2.3 Pileup modifications with Modkit

To generate a modification pileup from the aligned `BAM` file, we use `modkit`. The pileup can be read-based approach to include all the CpG loci observed in the reads **(recommended)**, or reference-guided and restricted to the CpG loci in the reference genome **(not recommended)**.

2.3.1 Read-based pileup

For read-based pileup we do not set any flags related to the reference genome CpG loci (–CpG or —–motif CG 0). However, since we used a CpG-context model for modification calling, only the reference positions with at least one mapped CpG site are included in the pileup.

```
# Set directory
output_directory= #/insert/output/directory/for/unaligned/bam/files/here
cd $output_directory

# Pileup modification from all CpG loci observed in the reads
modkit pileup aligned.bam all_GpG.bedMethyl
```

2.3.2 Reference-guided pileup

For reference-guided pileup we use the `-cpg` flag to restrict the analysis to the CpG loci in the reference genome.

```
# Set directory and reference genome
output_directory= #/insert/output/directory/for/unaligned/bam/files/here
reference_genome= #/insert/reference/genome/here
cd $output_directory

# Pileup modification from the CpG loci present in the reference genome
modkit pileup --$reference_genome --cpg aligned.bam ref_GpG.bedMethyl
```

The commands above represent the minimum requirements needed to process the raw Oxford Nanopore sequencing data into a `bedMethyl` file. We strongly recommend reading the documentation for each program to adjust the parameters for your project and data.

3 Reading bedMethyl files

The function `read.bedMethyl()` reads one or more `bedMethyl` file(s) and returns a `Methyl Counts` object which can be used for likelihood filtering, when setting `output` = `"Methyl Counts"`. Setting `strandCollapse` = `TRUE` merges data from the forward and reverse strand into a single representation.

3.1 Reading read-based pileup

The read-based bedMethyl files include all CpG loci observed in the reads should be read as `MethylCounts` objects as follows:

```
files<-list.files("~/Desktop/BSseq_long-read/data/silversides_chr24/modkit",
                  full.names=T)
mc_all<-read.bedMethyl(files, strandCollapse = T, output = "MethylCounts")
```

```
## Validating bedMethyl files and collecting metadata ...
mc_all
## An object of type 'MethylCounts' with
##   1153438 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

The `MethylCounts` object includes all the loci which are a CpG sites in at least one read mapped to the position. This includes non-reference CpG loci and false positive CpG loci introduced by sequencing and mapping errors.

3.2 Reading reference CpG loci only

The reference-guided bedMethyl files include only the CpG loci present in the reference genome, and can be read as `BSseq` objects or as `MethylCounts` objects as follows:

```
files<-list.files("~/Desktop/BSseq_long-read/data/silversides_chr24/modkit_cpg",
                  full.names=T)
mc_cpg<-read.bedMethyl(files=files, strandCollapse = T, output = "MethylCounts")
mc_cpg
## An object of type 'MethylCounts' with
##   341336 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

This object includes only loci that are a CpG site in at least one read, and are a CpG loci in the reference genome.

4 Filter a single sample

Modification analysis is often restricted to a subset of the loci in a sample. Filtering can be based on coverage thresholds or based on the likelihood of a locus being a homozygous or heterozygous CpG.

4.1 Coverage filtering

A `MethylCounts` object can be filtered using `getMethylCounts()`, where the coverage represent the number of times a CpG site is mapped at a specific locus.

4.1.1 All CpG loci

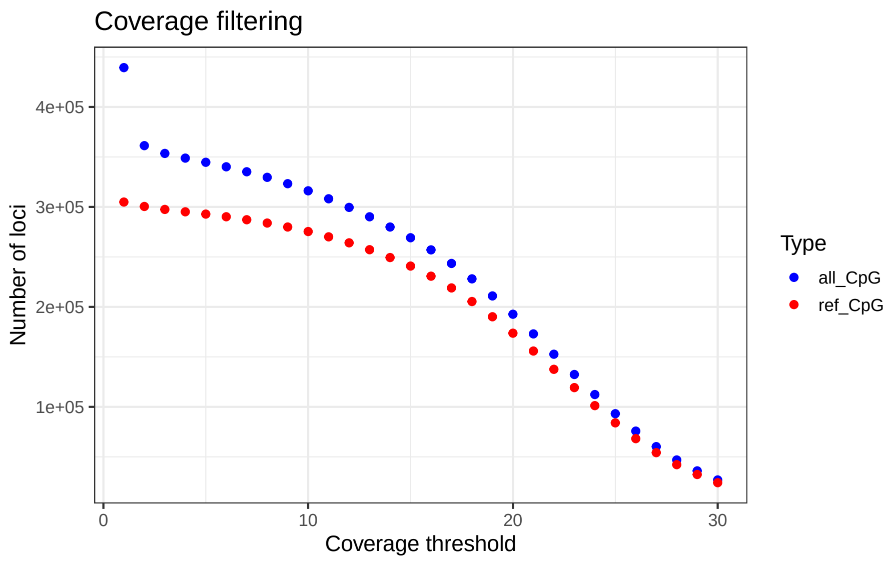
```
#get the first sample
mc_all_1 <- mc_all[,1]
#get the indices of loci with coverage >= 5
loci.idx <- which(getMethylCounts(mc_all_1, type="Cov")>= 5)
```

```
#filter the object to retain these loci
mc_all_1_filtered <- mc_all_1[loci.idx,]
mc_all_1_filtered
## An object of type 'MethylCounts' with
##   344661 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

4.1.2 Reference CpG loci

```
mc_cpg_1 <- mc_cpg[,1]
loci.idx <- which(getMethylCounts(mc_cpg_1, type="Cov")>= 5)
mc_cpg_1_filtered<-mc_cpg_1[loci.idx,]
mc_cpg_1_filtered
## An object of type 'MethylCounts' with
##   292828 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

In addition to the sample and the sequencing depth of a sample, the number of coverage filtered loci depend on whether non-reference CpG loci are included, and which coverage threshold is applied.



4.2 Likelihood filtering

A `MethylCounts` object imported using `read.bedMethyl()` includes both the CpG coverage and the non-CpG coverage all loci, which we use to estimate the error rate and call the CpG status i.e. determine if a locus is a homozygous CpG, a heterozygous CpG or not a CpG at all.

4.2.1 .99 “allCpG” filtering

To get the total (homozygous and heterozygous) CpG loci in a sample, one can use the `getCpGs()` function with `type` set to `"allCpG"`.

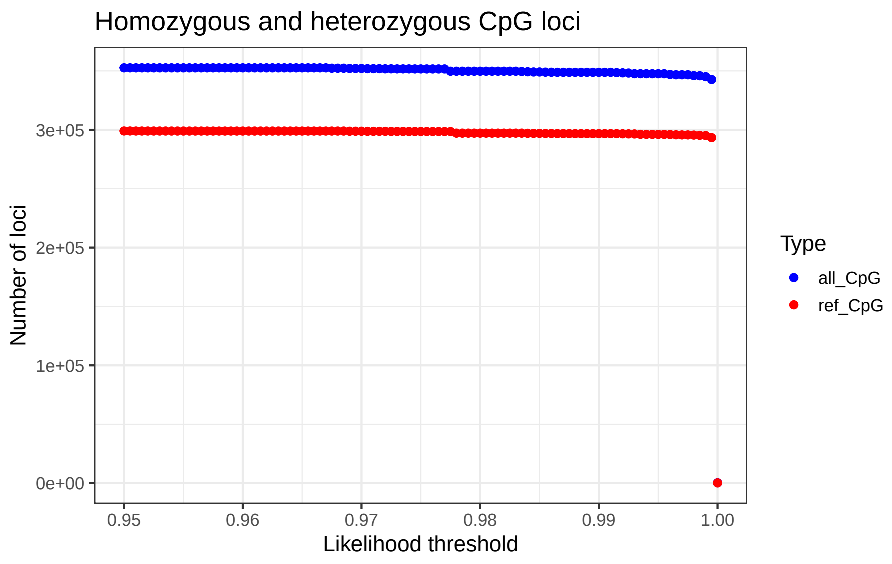
This can be for all the CpG loci observed in the reads:

```
#get the first sample
mc_all_1 <- mc_all[,1]
#get the indices of loci with scaled likelihood above 0.99 of being a "allCpG"
loci.idx <- getCpGs(mc_all_1, type = "allCpG", threshold = 0.99)
#filter the object to retain these loci
mc_all_1_filtered <- mc_all_1[loci.idx,]
mc_all_1_filtered
## An object of type 'MethylCounts' with
##   348807 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

Or for all the reference CpG loci observed in the reads:

```
mc_cpg_1 <- mc_cpg[,1]
loci.idx <- getCpGs(mc_cpg_1, type = "allCpG", threshold = 0.99)
mc_cpg_1_filtered <- mc_cpg_1[loci.idx,]
mc_cpg_1_filtered
## An object of type 'MethylCounts' with
##   296732 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

The number of “AllCpG” loci depends on the likelihood threshold and whether non-reference CpG loci are included.



4.2.2 .99 homozygous CpG

To get the homozygous CpG loci in a sample we use `getCpGs()` with `type = "homozygous"`.

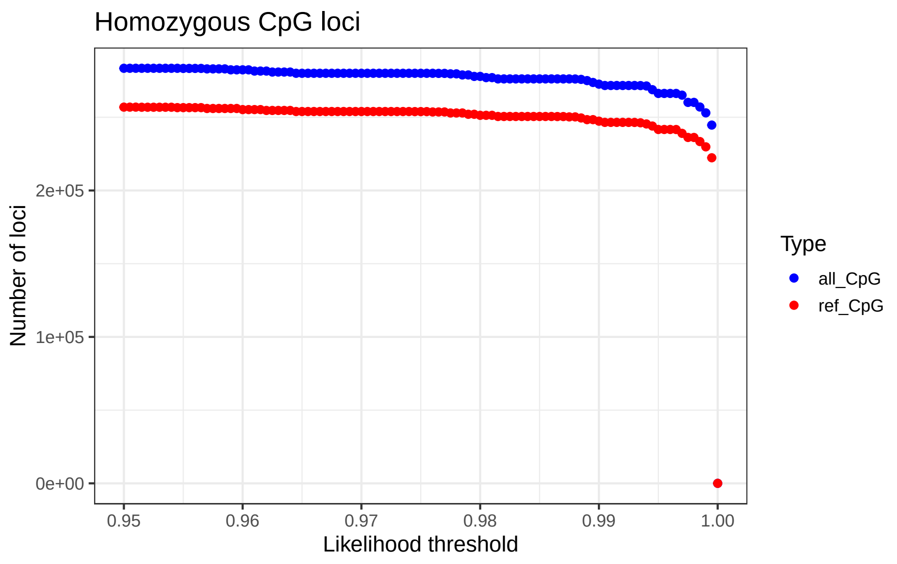
This can be for all the CpG loci observed in the reads:

```
#get the first sample
mc_all_1 <- mc_all[,1]
#get the indices of loci with scaled likelihood above 0.99 of being homozygous
loci.idx <- getCpGs(mc_all_1, type = "homozygous", threshold = 0.99)
#filter the object to retain these loci
mc_all_1_filtered <- mc_all_1[loci.idx,]
mc_all_1_filtered
## An object of type 'MethylCounts' with
##   272614 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

Or for all the reference CpG loci observed in the reads:

```
mc_cpg_1 <- mc_cpg[,1]
loci.idx <- getCpGs(mc_cpg_1, type = "homozygous", threshold = 0.99)
mc_cpg_1_filtered <- mc_cpg_1[loci.idx,]
mc_cpg_1_filtered
## An object of type 'MethylCounts' with
##   247335 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

The number of homozygous loci depends on the likelihood threshold and whether non-reference CpG loci are included.



4.2.3 .99 heterozygous CpG

To get the heterozygous CpG loci in a sample, we use the `getCpGs()` function with `type = "heterozygous"`.

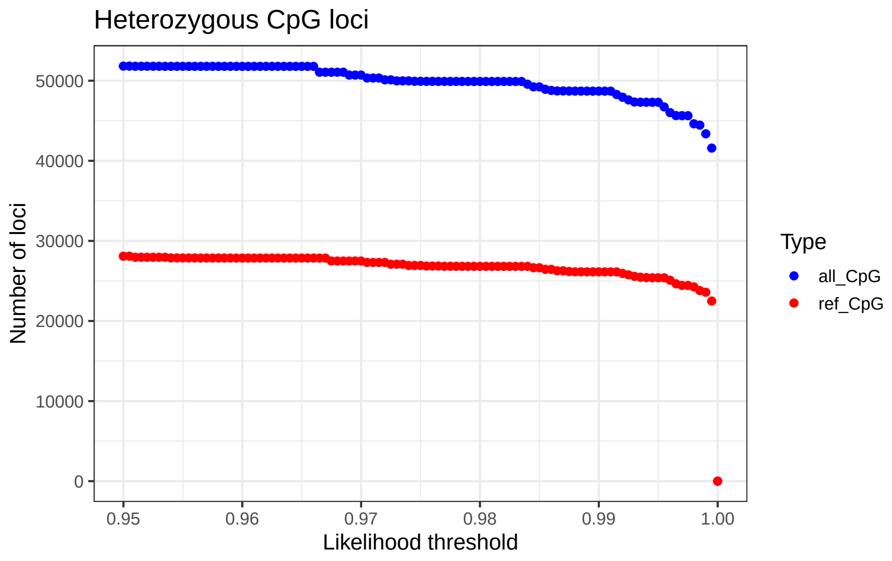
This can be for all the CpG loci observed in the reads:

```
#get the first sample
mc_all_1 <- mc_all[,1]
#get the indices of loci with scaled likelihood above 0.99 of being heterozygous
loci.idx <- getCpGs(mc_all_1, type = "heterozygous", threshold = 0.99)
#filter the object to retain these loci
mc_all_1_filtered <- mc_all_1[loci.idx,]
mc_all_1_filtered
## An object of type 'MethylCounts' with
##   48699 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

Or for all the reference CpG loci observed in the reads:

```
mc_cpg_1 <- mc_cpg[,1]
loci.idx <- getCpGs(mc_cpg_1, type = "heterozygous", threshold = 0.99)
mc_cpg_1_filtered <- mc_cpg_1[loci.idx,]
mc_cpg_1_filtered
## An object of type 'MethylCounts' with
##   26132 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

The number of heterozygous CpG loci depends on the likelihood threshold applied and whether non-reference CpG loci are retained. Reference-guided pileup generally removes ~half of the heterozygous loci in a sample.



5 Filter multiple samples

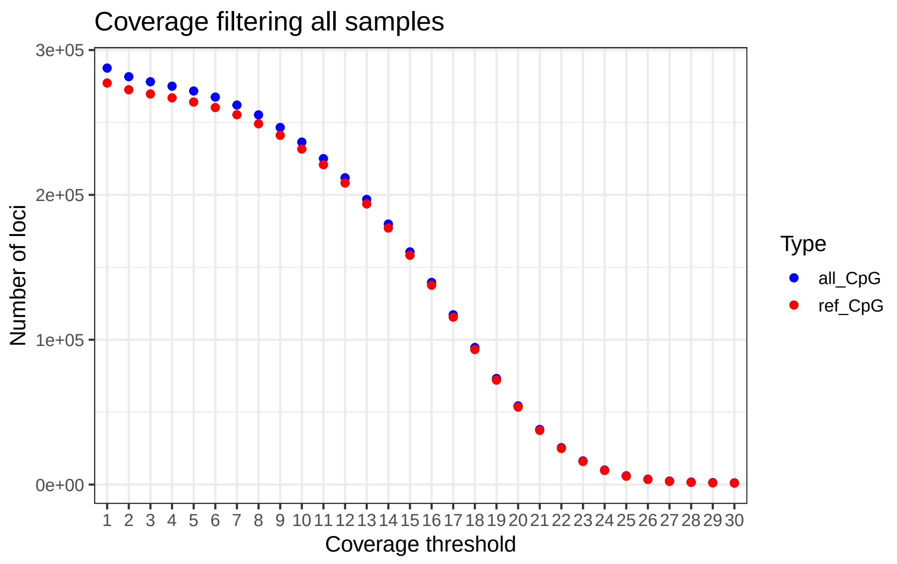
A project often includes multiple samples, and filtering can be applied to all samples in a `MethylCounts` object. Samples often share CpG loci and it is therefore advantageous to filter the loci in the multi-sample `MethylCounts` object to avoid false positive CpG loci introduced by sequencing and mapping errors.

5.1 Coverage filtering

A multi-sample `MethylCounts` object can be coverage filtered using `getMethylCounts()`.

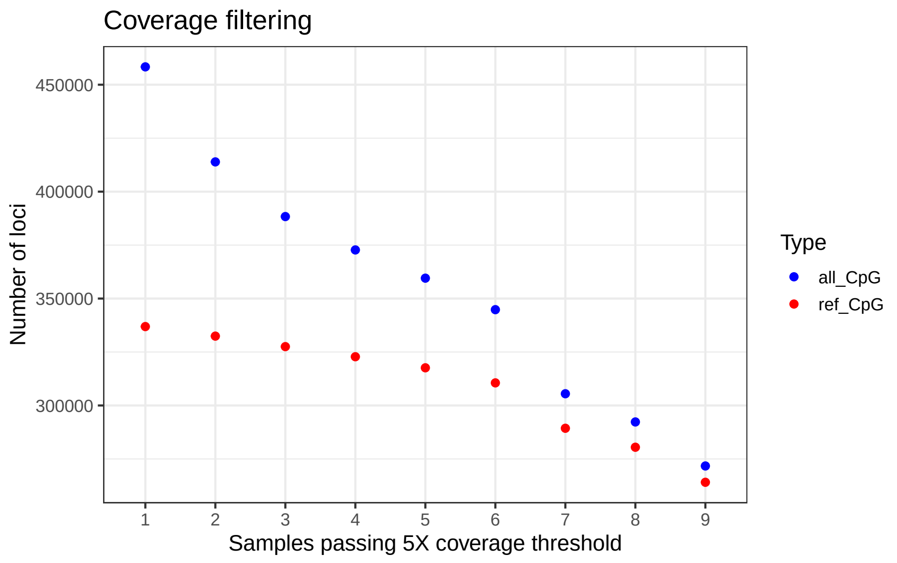
5.1.1 Coverage threshold

The coverage threshold can restrict the analysis to the loci with a coverage above a threshold in all samples (here, 9 samples):



5.1.2 Samples threshold

The coverage filtering can be less conservative by including all loci passing the threshold of 5X in some of the 9 samples:



5.1.3 Example

Filtering a `MethylCounts` object to retain only loci with a coverage of at least 5X in at least six of the nine samples can be done using this command:

```
loci.idx <- which(
  DelayedMatrixStats::rowSums2(getMethylCounts(mc_all, type="Cov")>= 5) >= 6)
mc_coverage_filtered <- mc_all[loci.idx,]
mc_coverage_filtered
```

```
## An object of type 'MethylCounts' with
##   344815 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

5.2 Likelihood filtering

For likelihood filtering of multiple samples, we recommend utilizing the functions `getCpGMatrix()` and `getMaxLikelihoodMatrix()` to obtain the CpG matrix and the maximum likelihood matrix for the samples.

The function `getCpGMatrix()` returns a matrix with the most likely CpG call for the loci and sample with the same dimensions as the `MethylCounts` object. In the the default setting homozygous CpG loci are represented by 0, heterozygous CpG loci by 1 and non-CpG loci by 2.

```
G_all <- getCpGMatrix(mc_all)
head(G_all)
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    2    2    0    2    2    2    2    2    2
## [2,]    2    2    2    2    2    2    2    2    2
## [3,]    1    0    0    0    2    1    2    0    1
## [4,]    1    0    2    0    2    1    2    1    0
## [5,]    2    2    2    2    2    1    2    2    2
## [6,]    1    2    2    2    0    2    2    2    2
```

The function `getMaxLikelihoodMatrix` returns a matrix with the corresponding scaled likelihood of most likely CpG call for the loci and sample with the same dimensions.

```
Q_all <- getMaxLikelihoodMatrix(mc_all)
head(round(Q_all,3))
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
## [1,] 0.333 0.333 0.990 0.333 0.333 0.333 0.333 0.333 0.333
## [2,] 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.995
## [3,] 0.969 1.000 0.875 0.997 0.333 1.000 0.333 0.997 1.000
## [4,] 0.941 1.000 0.333 0.997 0.333 1.000 0.333 0.989 1.000
## [5,] 0.333 0.333 0.333 0.333 0.333 1.000 0.333 0.333 0.333
## [6,] 0.941 0.990 0.652 0.333 0.650 0.333 0.333 0.333 0.333
```

Both functions can be run using `allCpG = TRUE`, where 0 in the CpG matrix represent homozygous or heterozygous CpG and the MaxLikelihoodMatrix represent the joint probability of homozygous or heterozygous CpG. This is useful for filtering the data based on likelihood thresholds.

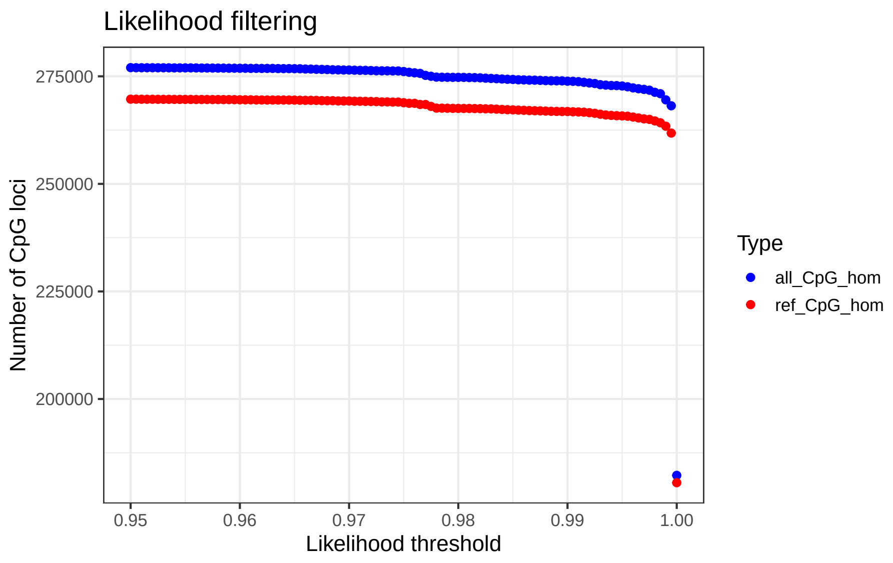
```
G_all <- getCpGMatrix(mc_all, allCpG = TRUE)
head(G_all)
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    2    2    0    2    2    2    2    2    2
## [2,]    2    2    2    2    2    2    2    2    2
## [3,]    0    0    0    0    2    0    2    0    0
## [4,]    0    0    2    0    2    0    2    0    0
## [5,]    2    2    2    2    2    0    2    2    2
## [6,]    0    2    2    2    0    2    2    2    2
```

```
Q_all <- getMaxLikelihoodMatrix(mc_all, allCpG = TRUE)
head(round(Q_all,3))
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
## [1,] 0.333 0.333 1.000 0.333 0.333 0.333 0.333 0.333 0.333
## [2,] 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.995
## [3,] 1.000 1.000 1.000 1.000 0.333 1.000 0.333 1.000 1.000
## [4,] 1.000 1.000 0.333 1.000 0.333 1.000 0.333 0.989 1.000
## [5,] 0.333 0.333 0.333 0.333 0.333 1.000 0.333 0.333 0.333
## [6,] 0.941 0.990 0.652 0.333 1.000 0.333 0.333 0.333 0.333
```

In both settings, locus with a coverage of 0 are represented as non-CpGs with likelihood of 1/3.

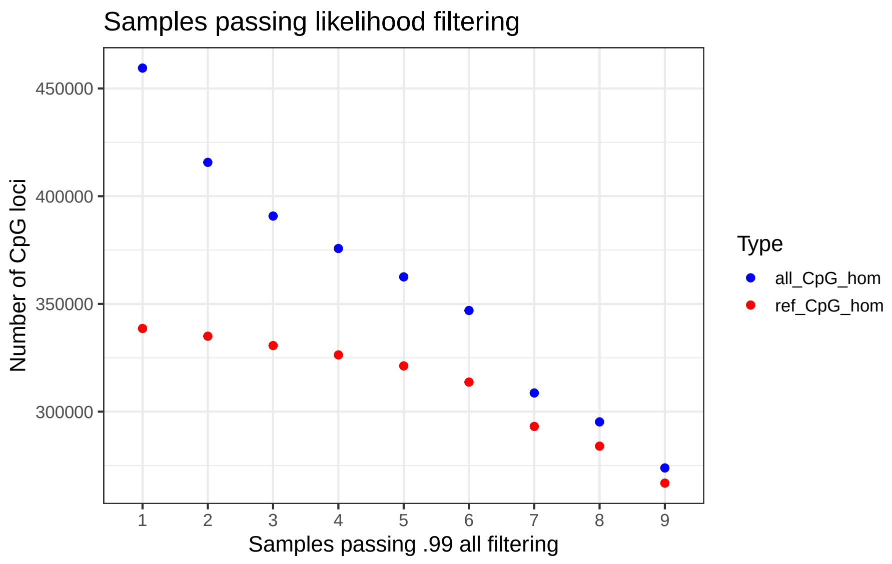
5.2.1 Likelihood threshold

The multi-sample `MethylCounts` object can be filtered for loci scaled likelihood of being a homozygous or heterozygous CpG loci in all nine samples above a certain likelihood threshold.



5.2.2 Samples threshold

The likelihood threshold can be combined with the samples threshold for more relaxed filtering for loci scaled likelihood of being a homozygous or heterozygous CpG loci in X out of all nine samples.



5.2.3 Example

The likelihood and samples thresholds can be combined to filter for e.g. loci with a likelihood above 0.99 of being a homozygous or heterozygous CpG loci in at least six of the nine samples:

```
G_all <- getCpGMatrix(mc_all, allCpG = TRUE)
Q_all <- getMaxLikelihoodMatrix(mc_all, allCpG = TRUE)

loci.idx <- which(
  DelayedMatrixStats::rowSums2(Q_all >= .99 & G_all==0) >= 6)
mc_filtered <- mc_all[loci.idx,]
mc_filtered
## An object of type 'MethylCounts' with
##   346944 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

6 HDF5 storage

The raw or filtered `MethylCounts` object can be saved as an HDF5 file using the `saveHDF5SummarizedExperiment()` function from the `HDF5Array` package. This allows for efficient storage and retrieval of large datasets.

```
HDF5Array::saveHDF5SummarizedExperiment(mc_filtered, "~/Desktop/mc_filtered.hdf5", replace = TRUE)
mc_filtered<-HDF5Array::loadHDF5SummarizedExperiment("~/Desktop/mc_filtered.hdf5")
mc_filtered
## An object of type 'MethylCounts' with
##   346944 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## Some assays are HDF5Array-backed
```

7 Converting to BSseq

For downstream analysis the filtered `MethylCounts` object can be converted to a `BSseq` object using the `BSseq()` constructor function.

```
# Filter and process a sample from the MethylCounts object
mc_sample <- mc_filtered[, 1]
mc_sample_filtered <- mc_sample[
  getCpGs(mc_sample, type = "homozygous", threshold = 0.99)]

# Display the filtered MethylCounts object
mc_sample_filtered
## An object of type 'MethylCounts' with
##   267767 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## Some assays are HDF5Array-backed
```

Depending on the analysis, you can specify the modification type (mods) as “5mC+5hmC” (default).

```
# Convert to BSseq with both 5mC and 5hmC data
bs <- BSseq(mc = mc_sample_filtered, mods = "5mC+5hmC")
bs
## An object of type 'BSseq' with
##   267767 loci
##   1 samples
##   5mC+5hmC values are stored in 'M'
## has not been smoothed
## Some assays are HDF5Array-backed

# Calculate the mean methylation using raw data
mean(getMeth(bs, type = "raw"))
## [1] 0.70369
```

Or only one modification type, e.g., “5mC”:

```
# Convert to BSseq with only 5mC data
bs_M <- BSseq(mc = mc_sample_filtered, mods = "5mC")
bs_M
## An object of type 'BSseq' with
##   267767 loci
##   1 samples
##   5mC values are stored in 'M'
## has not been smoothed
## Some assays are HDF5Array-backed
```

```
# Calculate the mean methylation for 5mC
mean(getMeth(bs_M, type = "raw"))
## [1] 0.6674152
```

or “5hmC”:

```
# Convert to BSseq with only 5hmC data
bs_H <- BSseq(mc = mc_sample_filtered, mods = "5hmC")
bs_H
## An object of type 'BSseq' with
##   267767 loci
##   1 samples
##   5hmC values are stored in 'M'
## has not been smoothed
## Some assays are HDF5Array-backed

# Calculate the mean methylation for 5hmC
mean(getMeth(bs_H, type = "raw"))
## [1] 0.0362748
```

8 Summary

In summary bedMethyl files from modkit can be imported in to bsseq and filtered for loci with a specific coverage or a specific likelihood of being a homozygous, heterozygous or homozygous/heterozygous CpG loci in all or a subset of samples. A import of a multi-sample project and filtering for loci with a scaled likelihood above 0.99 of being homozygous/heterozygous CpG loci in at least six samples and a coverage of at least 5 in at least six samples can be obtain with this command:

```
files<-list.files("~/Desktop/BSseq_long-read/data/silversides_chr24/modkit",
                  full.names=T)
mc_all<-read.bedMethyl(files, strandCollapse = T, output = "MethylCounts")

G <- getCpGMatrix(mc_all, allCpG = TRUE)
Q <- getMaxLikelihoodMatrix(mc_all, allCpG = TRUE)

loci.idx <- which(
  DelayedMatrixStats::rowSums2(Q >= .99 & G==0) >= 6 &
  DelayedMatrixStats::rowSums2(getMethylCounts(mc_all, type="Cov")>= 5) >= 6)
mc_filtered <- mc_all[loci.idx,]
mc_filtered
## An object of type 'MethylCounts' with
##   341389 loci
##   9 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

Specific samples in the BSseq object can be filtered for loci with a high probability of being e.g. homozygous CpG loci:

```
mc_sample<- mc_filtered[,1]
mc_sample_filtered<-mc_sample[
  getCpGs(mc_sample, type = "homozygous", threshold = 0.99)]
mc_sample_filtered
## An object of type 'MethylCounts' with
##   267180 loci
##   1 samples
##   5mC and 5hmC data from all samples
##   CG-context modification model detected for all samples
## All assays are in-memory
```

```
bs_sample_filtered<-BSseq(mc = mc_sample_filtered)
bs_sample_filtered
## An object of type 'BSseq' with
##   267180 loci
##   1 samples
##   5mC+5hmC values are stored in 'M'
## has not been smoothed
## All assays are in-memory
```

9 sessionInfo()

```
## R version 4.5.1 (2025-06-13)
## Platform: x86_64-apple-darwin20
## Running under: macOS Tahoe 26.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRlapack.dylib;  LAPACK versic
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Copenhagen
## tzcode source: internal
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] lubridate_1.9.4             forcats_1.0.0
##  [3] stringr_1.5.1               dplyr_1.1.4
##  [5] purrr_1.1.0                 readr_2.1.5
##  [7] tidyr_1.3.1                 tibble_3.3.0
##  [9] ggplot2_4.0.1               tidyverse_2.0.0
## [11] bsseq_1.48.0                SummarizedExperiment_1.39.1
## [13] Biobase_2.69.0              MatrixGenerics_1.21.0
## [15] matrixStats_1.5.0           GenomicRanges_1.61.1
## [17] Seqinfo_0.99.2              IRanges_2.43.0
## [19] S4Vectors_0.47.0            BiocGenerics_0.55.1
## [21] generics_0.1.4              BiocStyle_2.37.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.1          farver_2.1.2
##  [3] R.utils_2.13.0            Biostrings_2.77.2
##  [5] S7_0.2.1                  bitops_1.0-9
##  [7] fastmap_1.2.0             RCurl_1.98-1.17
##  [9] GenomicAlignments_1.45.2  XML_3.99-0.18
## [11] digest_0.6.37             timechange_0.3.0
## [13] lifecycle_1.0.4           statmod_1.5.0
## [15] magrittr_2.0.3            compiler_4.5.1
## [17] rlang_1.1.6               tools_4.5.1
## [19] yaml_2.3.10               data.table_1.17.8
## [21] rtracklayer_1.69.1        knitr_1.50
## [23] labeling_0.4.3            S4Arrays_1.9.1
## [25] curl_6.4.0                DelayedArray_0.35.2
## [27] RColorBrewer_1.1-3        abind_1.4-8
## [29] BiocParallel_1.43.4       HDF5Array_1.37.0
## [31] withr_3.0.2               R.oo_1.27.1
## [33] grid_4.5.1                beachmat_2.25.3
## [35] Rhdf5lib_1.31.0           scales_1.4.0
## [37] gtools_3.9.5              tinytex_0.58
## [39] cli_3.6.5                 rmarkdown_2.29
## [41] crayon_1.5.3              rstudioapi_0.17.1
## [43] tzdb_0.5.0                httr_1.4.7
## [45] rjson_0.2.23              DelayedMatrixStats_1.31.0
## [47] rhdf5_2.53.3              parallel_4.5.1
## [49] BiocManager_1.30.26       XVector_0.49.0
## [51] restfulr_0.0.16           vctrs_0.6.5
## [53] Matrix_1.7-3              bookdown_0.43
## [55] hms_1.1.3                 h5mread_1.1.1
## [57] locfit_1.5-9.12           limma_3.65.3
## [59] glue_1.8.0                codetools_0.2-20
## [61] stringi_1.8.7             gtable_0.3.6
## [63] BiocIO_1.19.0             pillar_1.11.0
## [65] htmltools_0.5.8.1         rhdf5filters_1.21.0
## [67] BSgenome_1.77.1           R6_2.6.1
## [69] sparseMatrixStats_1.21.0  evaluate_1.0.4
## [71] lattice_0.22-7            R.methodsS3_1.8.2
## [73] Rsamtools_2.25.2          Rcpp_1.1.0
## [75] SparseArray_1.9.1         permute_0.9-8
## [77] xfun_0.52                 pkgconfig_2.0.3
```

10 References