Package 'epialleleR'

November 11, 2026

Title Fast, Accurate, Epiallele-Aware Methylation Caller and Reporter

Version 1.20.0

Description Epialleles are specific DNA methylation patterns that are mitotically and/or meiotically inherited. This package calls and reports cytosine methylation as well as frequencies of hypermethylated epialleles at the level of genomic regions or individual cytosines in next-generation sequencing data using binary alignment map (BAM) files as an input. Among other things, this package can also extract and visualise methylation patterns and assess allele specificity of methylation.

SystemRequirements C++17, GNU make

NeedsCompilation yes

Depends R (>= 4.1)

Imports stats, methods, utils, data.table, BiocGenerics, GenomicRanges, Rcpp

LinkingTo Rcpp, BH, Rhtslib

Suggests GenomeInfoDb, SummarizedExperiment, VariantAnnotation, RUnit, knitr, rmarkdown, ggplot2

License Artistic-2.0

URL https://github.com/BBCG/epialleleR

BugReports https://github.com/BBCG/epialleleR/issues

Encoding UTF-8

biocViews DNAMethylation, Epigenetics, MethylSeq, LongRead

RoxygenNote 7.3.3

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/epialleleR

git_branch RELEASE_3_23

git_last_commit ad311c0

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Oleksii Nikolaienko [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5910-4934>)

Maintainer Oleksii Nikolaienko <oleksii.nikolaienko@gmail.com>

Contents

callMethylation 2
extractPatterns 4
generateBedEcdf 7
generateBedReport 9
generateCytosineReport 15
generateMhlReport 19
generateVcfReport 23
plotPatterns 28
preprocessBam 31
preprocessGenome 36
simulateBam 37

Index **40**

`callMethylation` *callMethylation*

Description

This function calls cytosine methylation and stores calls in BAM files.

Usage

```
callMethylation(
  input.bam.file,
  output.bam.file,
  genome,
  nthreads = 1,
  verbose = TRUE
)
```

Arguments

`input.bam.file`	input BAM file location string.
`output.bam.file`	output BAM file location string.
`genome`	reference (genomic) sequences file location string or an output of `preprocessGenome`.
`nthreads`	non-negative integer for the number of additional HTSlib threads to be used during file decompression (default: 1).
`verbose`	boolean to report progress and timings (default: TRUE).

Details

The function makes cytosine methylation calls for short-read methylation (bisulfite/enzymatic) sequencing alignments from input BAM file and writes them in the XM tag of the output BAM file. Calls are made on the basis of reference (e.g., genomic) sequence and observed sequence and cytosine context of reads. Data reading/processing is done by means of HTSlib, therefore it is possible to significantly (>5x) speed up the calling using several (4-8) HTSlib decompression threads.

Methylation calling with this function is only possible for sequencing data obtained using either bisulfite or other similar sequencing method (enzymatic methylation sequencing). Cytosine methylation in long-read, native DNA sequencing alignments should be called using other, appropriate tools.

It is a requirement that the genomic strand the read was aligned to is known. This information is typically stored in XG tag of Bismark/Illumina BAM files, or in YD tag of BWA-meth alignment files, or in ZS tag of BSMAP alignment files. ‘epialleleR‘ is aware of that and will use the whichever tag is available.

The sequence context of cytosines (h/H for "CHH", x/X for "CHG", z/Z for "CG") is determined based on the actual (observed) sequence of the read. E.g., if read "ACGT" was aligned to the forward strand of reference sequence "ACaaGT" with the CIGAR string "2M2D2M" (2 bases match, 2 reference bases are deleted, 2 bases match), then methylation call string will be ".Z.." (in contrast to the reference's one of ".H...."). This makes cytosine calls nearly identical to ones produced by Bismark Bisulfite Read Mapper and Methylation Caller or Illumina DRAGEN Bio IT Platform, however with one important distinction: ‘epialleleR‘ reports sequence context of cytosines followed by unknown bases ("CNN") as "H.." instead of "U.." (unknown; as for example Illumina DRAGEN Bio IT Platform does). Similarly, forward strand context of "CNG" is reported as "X..", forward strand context of "CGN" -> "Z..", reverse strand context of "NNG" -> "..H", reverse strand context of "CNG" -> "..X", reverse strand context of "NCG" -> "..Z". Both lowercase and uppercase ACGTN symbols in reference sequence are allowed and correctly recognised, however all the other symbols (e.g., extended IUPAC symbols, MRSVWYHKDB) within sequences are converted to N.

As a reference sequence, the function expects either location of (preferably ‘bgzip‘ped) FASTA file or an object obtained by `preprocessGenome`. The latter is recommended if methylation calling is to be performed on multiple BAM files.

The alignment records of the output BAM file will contain additional XM tag with the methylation call string for every mapped read which did not have XM tag available. Besides that, XG tag with reference sequence strand ("CT" or "GA") is added to such reads in case it wasn't present.

Please note that for the purpose of methylation calling, the very same reference genome must be used for both alignment (when BAM is produced) and calling cytosine methylation by `callMethylation` method. Exception is thrown if reference sequence header of BAM file doesn't match reference sequence data provided (this matching is performed on the basis of names and lengths of reference sequences).

Value

list object with simple statistics of processed ("nrecs") records and calls made ("ncalled"). Even though "ncalled" can be less than "nrecs" (e.g., because not all reads are mapped), all records from the input BAM are written to the output BAM.

See Also

preprocessGenome for preloading reference sequences and 'epialleleR' vignettes for the description of usage and sample data.

Bismark Bisulfite Read Mapper and Methylation Caller, bwa-meth for fast and accurate alignment of long bisulfite-seq reads, BSMAP: whole genome bisulfite sequence MAPping program, or info on Illumina DRAGEN Bio IT Platform.

Examples

```
callMethylation(
 input.bam.file=system.file("extdata", "test", "dragen-se-unsort-xg.bam", package="epialleleR"),
  output.bam.file=tempfile(pattern="output-", fileext=".bam"),
  genome=system.file("extdata", "test", "reference.fasta.gz", package="epialleleR")
)
```

---

extractPatterns *extractPatterns*

Description

This function extracts methylation patterns (epialleles) for a given genomic region of interest.

Usage

```
extractPatterns(
  bam,
  bed,
  bed.row = 1,
  zero.based.bed = FALSE,
  match.min.overlap = 1,
  extract.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  min.context.freq = 0.01,
  clip.patterns = FALSE,
  strand.offset = c(CG = 1, CHG = 2, CHH = 0, CxG = 0, CX = 0)[extract.context],
  highlight.positions = c(),
  ...,
  verbose = TRUE
)
```

Arguments

bam	BAM file location string OR preprocessed output of preprocessBam function. Read more about BAM file requirements and BAM preprocessing at preprocessBam.
bed	Browser Extensible Data (BED) file location string OR object of class GRanges holding genomic coordinates for regions of interest. It is used to match sequencing reads to the genomic regions for pattern extraction. The style of seqlevels of BED file/object must match the style of seqlevels of the BAM file/object used.

The BED/GRanges rows are **not** sorted internally. As of now, the strand information is ignored and patterns matching both strands are extracted.

`bed.row` single non-negative integer specifying what 'bed' region should be included in the output (default: 1).

`zero.based.bed` boolean defining if BED coordinates are zero based (default: FALSE).

`match.min.overlap`
integer for the smallest overlap between read's and BED/GRanges start or end positions during matching of capture-based NGS reads (default: 1).

`extract.context`
string defining cytosine methylation context used to report:

- "CG" (the default) – CpG cytosines (called as zZ)
- "CHG" – CHG cytosines (xX)
- "CHH" – CHH cytosines (hH)
- "CxG" – CG and CHG cytosines (zZxX)
- "CX" – all cytosines

`min.context.freq`
real number in the range [0;1] (default: 0.01). Genomic positions that are covered by smaller fraction of patterns (e.g., with erroneous context) won't be included in the report.

`clip.patterns` boolean if patterns should not extend over the edge of 'bed' region (default: FALSE).

`strand.offset` single non-negative integer specifying the offset of bases at the reverse (-) strand compared to the forward (+) strand. Allows to "merge" genomic positions when methylation is symmetric (in CG and CHG contexts). By default, equals 1 for 'extract.context'=="CG", 2 for "CHG", or 0 otherwise.

`highlight.positions`
integer vector with genomic positions of bases to include in every overlapping pattern. Allows to visualize the distribution of single-nucleotide variations (SNVs) among methylation patterns. 'highlight.positions' takes precedence if any of these positions overlap with within-the-context positions of methylation pattern.

`...` other parameters to pass to the `preprocessBam` function. Options have no effect if preprocessed BAM data was supplied as an input.

`verbose` boolean to report progress and timings (default: TRUE).

Details

The function matches reads (for paired-end sequencing alignment files - read pairs as a single entity) to the genomic region provided in a BED file/GRanges object, extracts methylation statuses of bases within those reads, and returns a data frame which can be used for further analysis and/or plotting of DNA methylation patterns by `plotPatterns` function.

Value

`data.table` object containing per-read (pair) base methylation information for the genomic region of interest. The report columns are:

- seqnames – read (pair) reference sequence name
- strand – read (pair) strand
- start – start of the read (pair)
- end – end of the read (pair)
- nbase – number of within-the-context bases for this read (pair)
- beta – beta value of this read (pair), calculated as a ratio of the number of methylated within-the-context bases to the total number of within-the-context bases
- pattern – hex representation of 64-bit FNV-1a hash calculated for all reported base positions and bases in this read (pair). This hash value depends only on included genomic positions and their methylation call string chars (hHxXzZ) or nucleotides (ACGT, for highlighted bases only), thus it is expected to be unique for every methylation pattern, although equal for identical methylation patterns independently on read (pair) start, end, or strand (when correct 'strand.offset' is given)
- ... – columns for each genomic position that hold corresponding methylation call string char, or NA if position is not present in the read (pair)

See Also

plotPatterns for pretty plotting of the output, preprocessBam for preloading BAM data, generateCytosineReport for methylation statistics at the level of individual cytosines, generateBedReport for genomic region-based statistics, generateVcfReport for evaluating epiallele-SNV associations, generateBedEcdf for analysing the distribution of per-read beta values, and 'epialleleR' vignettes for the description of usage and sample data.

Examples

```
# amplicon data
amplicon.bam <- system.file("extdata", "amplicon010meth.bam",
                            package="epialleleR")
amplicon.bed <- system.file("extdata", "amplicon.bed",
                            package="epialleleR")

# extract patterns
patterns <- extractPatterns(bam=amplicon.bam, bed=amplicon.bed, bed.row=3)

# and then plot them
plotPatterns(patterns)

# patterns from long reads, clipped to two narrow areas of interest
long.bam <- system.file("extdata", "longread.bam", package="epialleleR")
long.bed <- as(c("chr17:43125000-43125999", "chr17:43126001-43126999"),
               "GRanges")
long.data <- preprocessBam(
  bam=long.bam, targets=long.bed, clip.to.targets=TRUE,
  min.mapq=30, min.baseq=20, min.prob=178
)
plotPatterns(
  extractPatterns(bam=long.data,
                  bed=as("chr17:43125000-43127000", "GRanges")),
```

```
  npatterns.per.bin=Inf
)
```

---

generateBedEcdf *generateBedEcdf*

Description

This function computes empirical cumulative distribution functions (eCDF) for per-read beta values of the sequencing reads.

Usage

```
generateBedEcdf(
  bam,
  bed,
  bed.type = c("amplicon", "capture"),
  bed.rows = c(1),
  zero.based.bed = FALSE,
  match.tolerance = 1,
  match.min.overlap = 1,
  ecdf.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  ...,
  verbose = TRUE
)
```

Arguments

bam	BAM file location string OR preprocessed output of preprocessBam function. Read more about BAM file requirements and BAM preprocessing at preprocessBam.
bed	Browser Extensible Data (BED) file location string OR object of class GRanges holding genomic coordinates for regions of interest. It is used to match sequencing reads to the genomic regions prior to eCDF computation. The style of seqlevels of BED file/object must match the style of seqlevels of the BAM file/object used.
bed.type	character string for the type of assay that was used to produce sequencing reads: • "amplicon" (the default) – used for amplicon-based next-generation sequencing when exact coordinates of sequenced fragments are known. Matching of reads to genomic ranges are then performed by the read's start or end positions, either of which should be no further than 'match.tolerance' bases away from the start or end position of genomic ranges given in BED file/GRanges object • "capture" – used for capture-based next-generation sequencing when reads partially overlap with the capture target regions. Read is considered to match the genomic range when their overlap is more or equal to 'match.min.overlap'. If read matches two or more BED genomic regions, only the first match is taken (input GRanges are **not** sorted internally)

`bed.rows` integer vector specifying what 'bed' regions should be included in the output. If 'c(1)' (the default), then function returns eCDFs for the first region of 'bed', if NULL - eCDF functions for all 'bed' genomic regions as well as for the reads that didn't match any of the regions (last element of the return value; only if there are such reads).

`zero.based.bed` boolean defining if BED coordinates are zero based (default: FALSE).

`match.tolerance` integer for the largest difference between read's and BED `GRanges` start or end positions during matching of amplicon-based NGS reads (default: 1).

`match.min.overlap` integer for the smallest overlap between read's and BED `GRanges` start or end positions during matching of capture-based NGS reads (default: 1). If read matches two or more BED genomic regions, only the first match is taken (input `GRanges` are **not** sorted internally).

`ecdf.context` string defining cytosine methylation context used for computing within-the-context and out-of-context eCDFs:

- "CG" (the default) – within-the-context: CpG cytosines (called as zZ), out-of-context: all the other cytosines (hHxX)
- "CHG" – within-the-context: CHG cytosines (xX), out-of-context: hHzZ
- "CHH" – within-the-context: CHH cytosines (hH), out-of-context: xXzZ
- "CxG" – within-the-context: CG and CHG cytosines (zZxX), out-of-context: CHH cytosines (hH)
- "CX" – all cytosines are considered within-the-context

`...` other parameters to pass to the `preprocessBam` function. Options have no effect if preprocessed BAM data was supplied as an input.

`verbose` boolean to report progress and timings (default: TRUE).

Details

The function matches reads (for paired-end sequencing alignment files - read pairs as a single entity) to the genomic regions provided in a BED file/`GRanges` object, computes average per-read beta values according to the cytosine context parameter 'ecdf.context', and returns a list of eCDFs for within- and out-of-context average per-read beta values, which can be used for plotting.

The resulting eCDFs and their plots can be used to characterise the methylation pattern of a particular genomic region, e.g. if reads that match to that region are methylated in an "all-CpGs-or-none" manner or if some intermediate methylation levels are more frequent.

Value

list with a number of elements equal to the length of 'bed.rows' (if not NULL), or to the number of genomic regions within 'bed' (if 'bed.rows==NULL') plus one item for all reads not matching 'bed' genomic regions (if any). Every list item is a list on it's own, consisting of two eCDF functions for within- and out-of-context per-read beta values.

See Also

preprocessBam for preloading BAM data, generateCytosineReport for methylation statistics at the level of individual cytosines, generateBedReport for genomic region-based statistics, generateVcfReport for evaluating epiallele-SNV associations, extractPatterns for exploring methylation patterns and plotPatterns for pretty plotting of its output, and 'epialleleR' vignettes for the description of usage and sample data.

Examples

```
# amplicon data
amplicon.bam <- system.file("extdata", "amplicon010meth.bam",
                            package="epialleleR")
amplicon.bed <- system.file("extdata", "amplicon.bed",
                            package="epialleleR")

# let's compute eCDF
amplicon.ecdfs <- generateBedEcdf(bam=amplicon.bam, bed=amplicon.bed,
                                  bed.rows=NULL)

# there are 5 items in amplicon.ecdfs, let's plot them all
par(mfrow=c(1,length(amplicon.ecdfs)))

# cycle through items
for (x in 1:length(amplicon.ecdfs)) {
  # four of them have names corresponding to amplicon.bed genomic regions,
  # fifth - NA for all the reads that don't match to any of those regions
  main <- if (is.na(names(amplicon.ecdfs[x]))) "unmatched"
          else names(amplicon.ecdfs[x])

  # plotting eCDF for within-the-context per-read beta values (in red)
  plot(amplicon.ecdfs[[x]]$context, col="red", verticals=TRUE,
       do.points=FALSE, xlim=c(0,1), xlab="per-read beta value",
       ylab="cumulative density", main=main)

  # adding eCDF for out-of-context per-read beta values (in blue)
  plot(amplicon.ecdfs[[x]]$out.of.context, add=TRUE, col="blue",
       verticals=TRUE, do.points=FALSE)
}

# recover default plotting parameters
par(mfrow=c(1,1))
```

---

generateBedReport *generateBedReport*

Description

'generateBedReport', 'generateAmpliconReport', 'generateCaptureReport' – these functions match BAM reads to the set of genomic locations and return the fraction of reads with an average methylation level passing certain threshold.

Usage

```
generateAmpliconReport(
  bam,
  bed,
  report.file = NULL,
  zero.based.bed = FALSE,
  match.tolerance = 1,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  filter.reads = TRUE,
  min.context.sites = 0,
  max.outofcontext.beta = 0.1,
  threshold.reads = TRUE,
  min.context.beta = 0.5,
  ...,
  gzip = FALSE,
  verbose = TRUE
)

generateCaptureReport(
  bam,
  bed,
  report.file = NULL,
  zero.based.bed = FALSE,
  match.min.overlap = 1,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  filter.reads = TRUE,
  min.context.sites = 0,
  max.outofcontext.beta = 0.1,
  threshold.reads = TRUE,
  min.context.beta = 0.5,
  ...,
  gzip = FALSE,
  verbose = TRUE
)

generateBedReport(
  bam,
  bed,
  report.file = NULL,
  zero.based.bed = FALSE,
  bed.type = c("amplicon", "capture"),
  match.tolerance = 1,
```

```
  match.min.overlap = 1,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  filter.reads = TRUE,
  min.context.sites = 0,
  max.outofcontext.beta = 0.1,
  threshold.reads = TRUE,
  min.context.beta = 0.5,
  ...,
  gzip = FALSE,
  verbose = TRUE
)
```

Arguments

bam	BAM file location string OR preprocessed output of preprocessBam function. Read more about BAM file requirements and BAM preprocessing at preprocessBam.
bed	Browser Extensible Data (BED) file location string OR object of class GRanges holding genomic coordinates for regions of interest. The style of seqlevels of BED file/object must be the same as the style of seqlevels of BAM file/object used. The BED/GRanges rows are **not** sorted internally. As of now, the strand information is ignored and reads (read pairs) matching both strands are separately counted and reported.
report.file	file location string to write the BED report. If NULL (the default) then report is returned as a data.table object.
zero.based.bed	boolean defining if BED coordinates are zero-based (default: FALSE).
match.tolerance	integer for the largest difference between read's and BED GRanges start or end positions during matching of amplicon-based NGS reads (default: 1).
cytosine.context	string defining cytosine methylation context used for filtering and/or thresholding the reads: • "CG" (the default) – within-the-context: CpG cytosines (called as zZ), out-of-context: all the other cytosines (hHxX) • "CHG" – within-the-context: CHG cytosines (xX), out-of-context: hHzZ • "CHH" – within-the-context: CHH cytosines (hH), out-of-context: xXzZ • "CxG" – within-the-context: CG and CHG cytosines (zZxX), out-of-context: CHH cytosines (hH) • "CX" – all cytosines are considered within-the-context, this effectively results in no thresholding
filter.reads	boolean defining if sequence reads with too few context bases or too high out-of-context cytosine methylation should be filtered out (e.g., reads resulting from incompletely bisulfite-converted templates). Default: TRUE. Filtering is strongly recommended for short-read sequencing (bisulfite or enzymatic) because it removes reads from incompletely converted DNA molecules.
min.context.sites	non-negative integer for minimum number of cytosines within the 'cytosine.context' (default: 0, i.e., all reads will satisfy this criterion). When 'min.context.sites'>0,

reads containing **fewer** within-the-context cytosines will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`max.outofcontext.beta`
: real number in the range [0;1] (default: 0.1). Reads with average beta value for out-of-context cytosines **above** this threshold will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`threshold.reads`
: boolean defining if sequence reads should be thresholded before counting reads belonging to variant epialleles (default: TRUE). Disabling thresholding is possible but makes no sense in the context of this function, because all the reads will be assigned to the variant epiallele, which will result in VEF==1 (in such case 'NA' VEF values are returned in order to avoid confusion).

`min.context.beta`
: real number in the range [0;1] (default: 0.5). Reads with average beta value for within-the-context cytosines **below** this threshold are considered completely unmethylated (thus belonging to the reference epiallele). This option has no effect when read thresholding is disabled.

`...`
: other parameters to pass to the `preprocessBam` function. Options have no effect if preprocessed BAM data was supplied as an input.

`gzip`
: boolean to compress the report (default: FALSE).

`verbose`
: boolean to report progress and timings (default: TRUE).

`match.min.overlap`
: integer for the smallest overlap between read's and BED `GRanges` start or end positions during matching of capture-based NGS reads (default: 1). If read matches two or more BED genomic regions, only the first match is taken (input `GRanges` are **not** sorted internally).

`bed.type`
: character string for the type of assay that was used to produce sequencing reads:

  - "amplicon" (the default) – used for amplicon-based next-generation sequencing when exact coordinates of sequenced fragments are known. Matching of reads to genomic ranges are then performed by the read's start or end positions, either of which should be no further than 'match.tolerance' bases away from the start or end position of genomic ranges given in BED file/`GRanges` object
  - "capture" – used for capture-based next-generation sequencing when reads partially overlap with the capture target regions. Read is considered to match the genomic range when their overlap is more or equal to 'match.min.overlap'. If read matches two or more BED genomic regions, only the first match is taken (input `GRanges` are **not** sorted internally)

Details

Functions report hypermethylated variant epiallele frequencies (VEF) per genomic region of interest using BAM and BED files as input. Reads (for paired-end sequencing alignment files - read pairs as a single entity) are matched to genomic locations by exact coordinates ('generateAmpliconReport'

or ‘generateBedReport‘ with an option bed.type="amplicon") or minimum overlap (‘generateCaptureReport‘ or ‘generateBedReport‘ with an option bed.type="capture") – the former to be used for amplicon-based NGS data, while the latter – for the capture-based NGS data. The function’s logic is explained below.

NB: you can modify and/or run this example – see Examples section at the bottom of this page.

Suppose there is a BAM file with four reads, all mapped to the "+" strand of chromosome 1, positions 1-16. The genomic range is supplied as a parameter ‘bed = as("chr1:1-100", "GRanges")‘. Assuming the following values for the filtering and thresholding parameters (cytosine.context = "CG", filter.reads = TRUE, min.context.sites = 2, max.outofcontext.beta = 0.1, threshold.reads = TRUE, min.context.beta = 0.5), the input and the output results will look as follows:

methylation string	filter	threshold	explained
...Z..x+.h..x..h.	excluded	<NA>	min.context.sites < 2 (only one zZ base)
...Z..z.h..x..h.	pass	above	pass all criteria
...Z..z.h..X..h.	excluded	<NA>	max.outofcontext.beta > 0.1 (1XH / 3xXhH = 0.33)
...Z..z.h..z-.h.	pass	below	min.context.beta < 0.5 (1Z / 3zZ = 0.33)

Since the reads number one and three are filtered out, and only the second read will satisfy the thresholding criteria, the following BED report will be produced (again, given that all reads map to chr1:+:1-16):

seqnames	start	end	width	strand	nreads+	nreads-	nfiltered	VEF
chr1	1	100	100	*	2	0	2	0.5

Please note, that read thresholding by an average methylation level (as explained above) makes little sense for long-read sequencing alignments, as such reads can cover multiple regions with very different DNA methylation properties. If necessary, one could either clip long sequencing reads to narrow ‘targets‘ in `preprocessBam` function during BAM loading or use `extractPatterns`, limiting pattern output to the region of interest only.

Value

`data.table` object containing VEF report for BED `GRanges` or NULL if report.file was specified. If BAM file contains reads that would not match to any of BED `GRanges`, the last row in the report will contain information on such reads (with seqnames, start and end equal to NA). The report columns are:

- seqnames – reference sequence name
- start – start of genomic region
- end – end of genomic region
- width – width of genomic region
- strand – strand
- ... – other columns that were present in BED or metadata columns of `GRanges` object
- nreads+ – number of valid reads (pairs) mapped to the forward ("+") strand
- nreads- – number of valid reads (pairs) mapped to the reverse ("-") strand
- nfiltered – number of invalid reads (pairs) filtered out due to too few context bases or too high out-of-context cytosine methylation (‘NA‘ if filtering was disabled)
- VEF – fraction of valid reads passing the threshold (‘NA‘ if thresholding was disabled)

See Also

preprocessBam for preloading BAM data, generateCytosineReport for methylation statistics at the level of individual cytosines, generateVcfReport for evaluating epiallele-SNV associations, extractPatterns for exploring methylation patterns and plotPatterns for pretty plotting of its output, generateBedEcdf for analysing the distribution of per-read beta values, and 'epialleleR' vignettes for the description of usage and sample data.

GRanges class for working with genomic ranges, seqlevelsStyle function for getting or setting the seqlevels style.

Examples

```
# amplicon data
amplicon.bam    <- system.file("extdata", "amplicon010meth.bam",
                               package="epialleleR")
amplicon.bed    <- system.file("extdata", "amplicon.bed",
                               package="epialleleR")
amplicon.report <- generateAmpliconReport(bam=amplicon.bam,
                                          bed=amplicon.bed)
plotPatterns(
  extractPatterns(
    bam=amplicon.bam, bed=amplicon.bed, match.min.overlap=100
  ), npatterns.per.bin=Inf
)

# capture NGS
capture.bam    <- system.file("extdata", "capture.bam",
                              package="epialleleR")
capture.bed    <- system.file("extdata", "capture.bed",
                              package="epialleleR")
capture.report <- generateCaptureReport(bam=capture.bam, bed=capture.bed)

# generateAmpliconReport and generateCaptureReport are just aliases
# of the generateBedReport
bed.report <- generateBedReport(bam=capture.bam, bed=capture.bed,
                                bed.type="capture")
identical(capture.report, bed.report)

# long-read data can be used if clipped to a narrow target area(s)
long.bam <- system.file("extdata", "longread.bam", package="epialleleR")
long.bed <- as("chr17:43124909-43125554", "GRanges")
long.data <- preprocessBam(
  bam=long.bam, targets=long.bed, clip.to.targets=TRUE,
  min.mapq=30, min.baseq=20, min.prob=178
)
long.report <- generateBedReport(
  bam=long.data, bed=long.bed, bed.type="capture", filter.reads=FALSE
)
plotPatterns(
  extractPatterns(bam=long.data, bed=long.bed),
  npatterns.per.bin=Inf
)
```

```
# toy example from the description
temp.bam <- tempfile(fileext=".bam")
temp.bed <- as("chr1:1-100", "GRanges")
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            seq=c("AGACGTTAGTAATAGTA", "AAACGTTGTAATAGTA",
                  "AGACGTTGTAACAGTA",  "AAACGTTGTAATGTA"),
            XM=c( "...Z..x+.h..x..h.", "...Z..z.h..x..h.",
                  "...Z..z.h..X..h.",  "...Z..z.h..z.h."),
            cigar=c("7M1I9M", "16M", "16M", "12M1D3M"))
# with read filtering
generateBedReport(bam=temp.bam, bed=temp.bed, min.context.sites=2)
# without read filtering
generateBedReport(bam=temp.bam, bed=temp.bed, filter.reads=FALSE)
# patterns plotted
plotPatterns(
  extractPatterns(bam=temp.bam, bed=temp.bed, extract.context="CX"),
  plot.context="CX", npatterns.per.bin=Inf
)
```

---

generateCytosineReport

generateCytosineReport

Description

This function counts methylated and unmethylated DNA bases taking into the account average methylation level of the entire sequence read.

Usage

```
generateCytosineReport(
  bam,
  report.file = NULL,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  filter.reads = TRUE,
  min.context.sites = 0,
  max.outofcontext.beta = 0.1,
  threshold.reads = TRUE,
  min.context.beta = 0.5,
  report.context = cytosine.context,
  ...,
  gzip = FALSE,
  verbose = TRUE
)
```

Arguments

`bam` BAM file location string OR preprocessed output of `preprocessBam` function. Read more about BAM file requirements and BAM preprocessing at `preprocessBam`.

`report.file` file location string to write the cytosine report. If NULL (the default) then report is returned as a `data.table` object.

`cytosine.context`
string defining cytosine methylation context used for filtering and/or thresholding the reads:

- "CG" (the default) – within-the-context: CpG cytosines (called as zZ), out-of-context: all the other cytosines (hHxX)
- "CHG" – within-the-context: CHG cytosines (xX), out-of-context: hHzZ
- "CHH" – within-the-context: CHH cytosines (hH), out-of-context: xXzZ
- "CxG" – within-the-context: CG and CHG cytosines (zZxX), out-of-context: CHH cytosines (hH)
- "CX" – all cytosines are considered within-the-context, this effectively results in no thresholding

`filter.reads` boolean defining if sequence reads with too few context bases or too high out-of-context cytosine methylation should be filtered out (e.g., reads resulting from incompletely bisulfite-converted templates). Default: TRUE. Filtering is strongly recommended for short-read sequencing (bisulfite or enzymatic) because it removes reads from incompletely converted DNA molecules.

`min.context.sites`
non-negative integer for minimum number of cytosines within the 'cytosine.context' (default: 0, i.e., all reads will satisfy this criterion). When 'min.context.sites'>0, reads containing **fewer** within-the-context cytosines will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`max.outofcontext.beta`
real number in the range [0;1] (default: 0.1). Reads with average beta value for out-of-context cytosines **above** this threshold will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`threshold.reads`
boolean defining if sequence reads (read pairs) should be thresholded before counting methylated cytosines (default: TRUE). Disabling thresholding (together with filtering) makes the report virtually indistinguishable from the ones generated by other software, such as Bismark or Illumina DRAGEN Bio IT Platform. Thresholding is **not** recommended for long-read sequencing data because long reads might cover multiple regions with very different cytosine methylation.

`min.context.beta`
real number in the range [0;1] (default: 0.5). Reads with average beta value for within-the-context cytosines **below** this threshold are considered completely unmethylated (all C are counted as T). This option has no effect when read thresholding is disabled.

report.context	string defining cytosine methylation context to report (default: value of 'cytosine.context').
...	other parameters to pass to the preprocessBam function. Options have no effect if preprocessed BAM data was supplied as an input.
gzip	boolean to compress the report (default: FALSE).
verbose	boolean to report progress and timings (default: TRUE).

Details

The function reports cytosine methylation information using BAM file or data as an input. In contrast to the other currently available software, reads (for paired-end sequencing alignment files - read pairs as a single entity) can be thresholded by their average methylation level before counting methylated bases, effectively resulting in hypermethylated variant epiallele frequency (VEF) being reported instead of beta value. The function's logic is explained below.

NB: you can modify and/or run this example – see Examples section at the bottom of this page.

Let's suppose we have a BAM file with four reads, all mapped to the "+" strand of chromosome 1, positions 1-16. Assuming the following values for the filtering and thresholding parameters (cytosine.context = "CG", filter.reads=TRUE, min.context.sites = 2, max.outofcontext.beta = 0.1, threshold.reads = TRUE, min.context.beta = 0.5), the input and results will look as following:

methylation string	filter	threshold	explained	methylation reported
...Z..x+.h..x..h.	excluded	<NA>	min.context.sites < 2 (only one zZ base)	all cytosines unmethylated
...Z..z.h..x..h.	pass	above	pass all criteria	only C4 (Z at position 4) is
...Z..z.h..X..h.	excluded	<NA>	max.outofcontext.beta > 0.1 (1XH / 3xXhH = 0.33)	read excluded from reporti
...Z..z.h..z-.h.	pass	below	min.context.beta < 0.5 (1Z / 3zZ = 0.33)	all cytosines unmethylated

Since the reads number one and three are filtered out, and only the second read will satisfy the thresholding criteria, the following CX report will be produced (given that all reads map to chr1:+:1-16):

rname	strand	pos	context	meth	unmeth
chr1	+	4	CG	1	1
chr1	+	7	CG	0	2
chr1	+	9	CHH	0	2
chr1	+	15	CHH	0	2

(Disclaimer: the cytosine base at position 12 is absent in the report, because its context cannot be determined based on two reads that passed the filtering: in one of the reads this base is in CG context and in the other in CHG context. Since none of these contexts is present in **more** than a half of the reads, the base is skipped.)

With the read filtering and thresholding disabled (filter.reads=FALSE, threshold.reads = FALSE) all reads will be included and all methylated bases will retain their status, so the CX report will be very similar (nearly identical) to the reports produced by other methylation callers (such as Bismark or Illumina DRAGEN Bio IT Platform):

rname	strand	pos	context	meth	unmeth

chr1	+	4	CG	4	0
chr1	+	7	CG	0	3
chr1	+	9	CHH	0	4
chr1	+	12	CHG	1	2
chr1	+	15	CHH	0	4

Other notes:

Methylation string bases in unknown context ("uU") are simply ignored, which, to the best of our knowledge, is consistent with the behaviour of other tools.

In order to mitigate the effect of sequencing errors (leading to rare variations in the methylation context, as in reads 1 and 4 above), the context present in more than 50% of the reads is assumed to be correct, while all bases at the same position but having other methylation context are simply ignored. This allows reports to be prepared without using the reference genome sequence.

The downside of not using the reference genome sequence is the inability to determine the actual sequence of triplet for every base in the cytosine report. Therefore this sequence is not reported, and this won't change until such information will be considered as worth adding.

Please also note, that read thresholding by an average methylation level (as explained above) makes little sense for long-read sequencing alignments, as such reads can cover multiple regions with very different DNA methylation properties.

Value

`data.table` object containing cytosine report in Bismark-like format or NULL if report.file was specified. The report columns are:

- rname — reference sequence name (as in BAM)
- strand — strand
- pos — cytosine position
- context — methylation context
- meth — number of methylated cytosines
- unmeth — number of unmethylated cytosines

See Also

'values' vignette for a comparison and visualisation of epialleleR output values for various input files. 'epialleleR' vignette for the description of usage and sample data.

`preprocessBam` for preloading BAM data, `generateBedReport` for genomic region-based statistics, `generateVcfReport` for evaluating epiallele-SNV associations, `extractPatterns` for exploring methylation patterns and `plotPatterns` for pretty plotting of its output, `generateBedEcdf` for analysing the distribution of per-read beta values.

Examples

```
capture.bam <- system.file("extdata", "capture.bam", package="epialleleR")

# CpG report with filtering and thresholding
```

```
cg.report <- generateCytosineReport(capture.bam)

# CX report with filtering but without thresholding
cx.report <- generateCytosineReport(capture.bam, threshold.reads=FALSE,
             report.context="CX")

# Long-read sequencing with both filtering and thresholding disabled
long.bam <- system.file("extdata", "longread.bam", package="epialleleR")
long.data <- preprocessBam(bam=long.bam, min.mapq=30, min.baseq=20,
                           min.prob=178)
cg.report <- generateCytosineReport(bam=long.data, filter.reads=FALSE,
                                    threshold.reads=FALSE)
plot(cg.report[, .(pos, beta=data.table::frollmean(meth/(meth+unmeth), 100))], type="l")

# toy example from the description
temp.bam <- tempfile(fileext=".bam")
temp.bed <- as("chr1:1-100", "GRanges")
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            seq=c("AGACGTTAGTAATAGTA", "AAACGTTGTAATAGTA",
                  "AGACGTTGTAACAGTA",  "AAACGTTGTAATGTA"),
            XM=c( "...Z..x+.h..x..h.", "...Z..z.h..x..h.",
                  "...Z..z.h..X..h.",  "...Z..z.h..z.h."),
            cigar=c("7M1I9M", "16M", "16M", "12M1D3M"))
# with read filtering and thresholding (default)
generateCytosineReport(bam=temp.bam, report.context="CX")
# with read filtering (nondefault min.context.sites) and thresholding
generateCytosineReport(bam=temp.bam, report.context="CX",
                       min.context.sites=2)
# without read filtering
generateCytosineReport(bam=temp.bam, report.context="CX",
                       filter.reads=FALSE)
# without read thresholding
generateCytosineReport(bam=temp.bam, report.context="CX",
                       threshold.reads=FALSE)
# both read filtering and thresholding disabled = similar to other software
generateCytosineReport(bam=temp.bam, report.context="CX",
                       filter.reads=FALSE, threshold.reads=FALSE)
# patterns plotted
plotPatterns(
  extractPatterns(bam=temp.bam, bed=temp.bed, extract.context="CX"),
  plot.context="CX", npatterns.per.bin=Inf
)
```

`generateMhlReport` *generateMhlReport*

Description

This function computes *Linearised* Methylated Haplotype Load ($lMHL$) per genomic position.

Usage

```
generateMhlReport(
  bam,
  report.file = NULL,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  max.haplotype.window = 0,
  filter.reads = TRUE,
  min.haplotype.length = 0,
  max.outofcontext.beta = 0.1,
  ...,
  gzip = FALSE,
  verbose = TRUE
)
```

Arguments

`bam` BAM file location string OR preprocessed output of `preprocessBam` function. Read more about BAM file requirements and BAM preprocessing at `preprocessBam`.

`report.file` file location string to write the $lMHL$ report. If NULL (the default) then report is returned as a `data.table` object.

`cytosine.context`
string for a cytosine context that defines a haplotype:

- "CG" (the default) – CpG cytosines only (called as zZ)
- "CHG" – CHG cytosines only (xX)
- "CHH" – CHH cytosines only (hH)
- "CxG" – CG and CHG cytosines (zZxX)
- "CX" – all cytosines; this, as well as the other non-CG contexts, may have little sense but still included for consistency

If $lMHL$ calculations are needed for all three possible cytosine contexts *independently*, one has to run this function for each required 'cytosine.context' separately, because 'cytosine.context'=="CX" assumes that *any* cytosine context is allowed within the same haplotype. This behaviour may change in the future.

`max.haplotype.window`
non-negative integer for maximum value of L' in $lMHL$ formula. When 0 (the default), calculations are performed for the full haplotype length ($L' = L$, although the maximum value is currently limited to 65535). Having no length restrictions make sense for short-read sequencing when the length of the read is comparable to the length of a typical methylated block, the depth of coverage is high, and the lengths of all reads are roughly equal. However, calculations using non-restricted haplotype length are meaningless for long-read sequencing — when the same read may cover a number of regions with very different methylation properties, and reads themselves can be of a very different length. In the latter case it is advised to limit the 'max.haplotype.window' to a number of cytosines in a typical hypermethylated region. For thorough explanation and more examples, see Details section and vignette.

`filter.reads` boolean defining if sequence reads with too high out-of-context cytosine methylation (specified by 'max.outofcontext.beta') or too few within-the-context bases (specified by 'min.haplotype.length') should be filtered out. Default: TRUE. Filtering is strongly recommended for short-read sequencing (bisulfite or enzymatic) because it removes reads from incompletely converted DNA molecules.

`min.haplotype.length` non-negative integer for minimum length of a haplotype (default: 0 will include haplotypes of any length). When 'min.haplotype.length'>0, reads (read pairs) with fewer than 'min.haplotype.length' cytosines within the 'cytosine.context' are skipped. This option has no effect when read filtering is disabled.

`max.outofcontext.beta` real number in the range [0;1] (default: 0.1). Reads (read pairs) with average beta value for out-of-context cytosines **above** this threshold (e.g., reads resulting from incompletely bisulfite-converted templates) are skipped. Value of 1 disables filtering by out-of-context methylation. This option has no effect when read filtering is disabled.

`...` other parameters to pass to the `preprocessBam` function. Options have no effect if preprocessed BAM data was supplied as an input.

`gzip` boolean to compress the report (default: FALSE).

`verbose` boolean to report progress and timings (default: TRUE).

Details

The function reports *Linearised* Methylated Haplotype Load ($lMHL$) at the level of individual cytosines using BAM file location or preprocessed data as an input. Function uses the following formula:

$$lMHL = \frac{\sum_{i=1}^{L'} w_i \times MH_i}{\sum_{i=1}^{L'} w_i \times H_i}$$

where L' is the length of a calculation window (e.g., number of CpGs; $L' \leq L$, where L is the length of a haplotype covering current genomic position), MH_i is a number of fully successive methylated stretches with i loci within a methylated stretch that overlaps current genomic position, H_i is a number of fully successive stretches with i loci, w_i is a weight for i-locus haplotype ($w_i = i$).

This formula is a modification of the original Methylated Haplotype Load (MHL) formula that was first described by Guo et al., 2017 (doi: 10.1038/ng.3805):

$$MHL = \frac{\sum_{i=1}^{L} w_i \times \frac{MH_i}{H_i}}{\sum_{i=1}^{L} w_i}$$

where L is the length of a longest haplotype covering current genomic position, $\frac{MH_i}{H_i} = P(MH_i)$ is the fraction of fully successive methylated stretches with i loci, w_i is a weight for i-locus haplotype ($w_i = i$).

The modifications to original formula are made in order to:

- **provide granularity of values** — the original MHL formula gives the same MHL value for every cytosine of a partially methylated haplotype (e.g., MHL=0.358 for each cytosine within a read with methylation call string "zZZZ"). In contrast, $lMHL$==0 for the non-methylated cytosines (e.g., $lMHL$==c(0, 0.5, 0.5, 0.5) for cytosines within a read with methylation call string "zZZZ").
- **enable calculations for long-read sequencing alignments** — $lMHL$ calculation window can be limited to a particular number of cytosines. This allows to use the formula for very long haplotypes as well as to compare values for sequencing data of varying read length.
- **reduce the complexity of MHL calculation** for data of high breadth and depth — $lMHL$ values for all genomic positions can be calculated using a single pass (cycling through reads just once) as the linearised calculations of numerator and denominator for $lMHL$ do not require prior knowledge on how many reads cover a particular position. This is achieved by moving H_i multiplier to the denominator of the $lMHL$ formula.

These modifications make $lMHL$ calculation similar though *non-equivalent* to the original MHL. However, the most important property of MHL — emphasis on hypermethylated blocks — is retained. And in return, $lMHL$ gets better applicability for analysis of sequencing data of varying depth and read length.

Other notes on function's behaviour:

Methylation string bases in unknown context ("uU") are simply ignored, which, to the best of our knowledge, is consistent with the behaviour of other tools.

Cytosine context present in more than 50% of the reads is assumed to be correct, while all bases at the same position but having other methylation context are simply ignored. This allows reports to be prepared without using the reference genome sequence.

Value

`data.table` object containing $lMHL$ report or NULL if report.file was specified. The report columns are:

- rname – reference sequence name (as in BAM)
- strand – strand
- pos – cytosine position
- context – methylation context
- coverage – number of reads (read pairs) that include this position
- length – average length of a haplotype, i.e., average number of cytosines within 'cytosine.context' for reads (read pairs) that include this position
- lmhl – $lMHL$ value

See Also

'values' vignette for a comparison and visualisation of epialleleR output values for various input files. 'epialleleR' vignette for the description of usage and sample data.

`preprocessBam` for preloading BAM data, `generateCytosineReport` for other methylation statistics at the level of individual cytosines, `generateBedReport` for genomic region-based statistics, `generateVcfReport` for evaluating epiallele-SNV associations, `extractPatterns` for exploring methylation patterns and `plotPatterns` for pretty plotting of its output, `generateBedEcdf` for analysing the distribution of per-read beta values.

Examples

```
capture.bam <- system.file("extdata", "capture.bam", package="epialleleR")

# lMHL report
mhl.report <- generateMhlReport(capture.bam)

# lMHL report with a `max.haplotype.window` of 1 is identical to a
# conventional cytosine report (or nearly identical when sequencing errors
# are present)
mhl.report <- generateMhlReport(capture.bam, max.haplotype.window=1)
cg.report  <- generateCytosineReport(capture.bam, threshold.reads=FALSE)
identical(
  mhl.report[, .(rname, strand, pos, context, value=lmhl)],
  cg.report[ , .(rname, strand, pos, context, value=meth/(meth+unmeth))]
)

# Long-read sequencing with filtering disabled, using window of 10 CpGs
long.bam <- system.file("extdata", "longread.bam", package="epialleleR")
long.data <- preprocessBam(bam=long.bam, min.mapq=30, min.baseq=20,
                           min.prob=178)
mhl.report <- generateMhlReport(bam=long.data, max.haplotype.window=10,
                                filter.reads=FALSE)
plot(mhl.report[, .(pos, lmhl=data.table::frollmean(lmhl, 100))], type="l")

## toy examples to illustrate the logic of computations
temp.bam <- tempfile(fileext=".bam")

# case 1: fully methylated haplotype
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            XM="h..Z..Z.Z..Z...Z.h.")
generateMhlReport(temp.bam)

# case 2: incompletely methylated haplotype
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            XM="h..Z..Z.z..Z...Z.h.")
generateMhlReport(temp.bam)

# case 3: hypermethylated read and hypomethylated read
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            XM=c("h..Z..Z.Z..Z...Z.h.", "h..z..Z.z..Z...z.h."))
generateMhlReport(temp.bam)

# case 4: incompletely bisulfite-converted read and hypomethylated read
simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
            XM=c("H..Z..Z.Z..Z...Z.h.", "h..z..Z.z..Z...z.h."))
generateMhlReport(temp.bam)
```

---

generateVcfReport *generateVcfReport*

Description

This function reports base frequencies at particular genomic positions and tests their association with the methylation status of the sequencing reads.

Usage

```
generateVcfReport(
  bam,
  vcf,
  vcf.style = NULL,
  bed = NULL,
  report.file = NULL,
  zero.based.bed = FALSE,
  cytosine.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  filter.reads = TRUE,
  min.context.sites = 0,
  max.outofcontext.beta = 0.1,
  threshold.reads = TRUE,
  min.context.beta = 0.5,
  ...,
  gzip = FALSE,
  verbose = TRUE
)
```

Arguments

bam	BAM file location string OR preprocessed output of preprocessBam function. Read more about BAM file requirements and BAM preprocessing at preprocessBam.
vcf	Variant Call Format (VCF) file location string OR a VCF object returned by readVcf function. If VCF object is supplied, the style of its seqlevels must match the style of seqlevels of the BAM file/object used.
vcf.style	string for the seqlevels style of the VCF file, if different from BED file/object. Only has effect when 'vcf' parameter points to the VCF file location and 'bed' is not NULL. Possible values: • NULL (the default) – seqlevels in BED file/object and VCF file are the same • "NCBI", "UCSC", ... – valid parameters of seqlevelsStyle function
bed	Browser Extensible Data (BED) file location string OR object of class GRanges holding genomic coordinates for regions of interest. It is used to include only the specific genomic ranges when the VCF file is loaded. This option has no effect when VCF object is supplied as a 'vcf' parameter. The style of seqlevels of BED file/object must match the style of seqlevels of the BAM file/object used.
report.file	file location string to write the VCF report. If NULL (the default) then report is returned as a data.table object.
zero.based.bed	boolean defining if BED coordinates are zero based (default: FALSE).

`cytosine.context`
: string defining cytosine methylation context used for filtering and/or thresholding the reads:
  - "CG" (the default) – within-the-context: CpG cytosines (called as zZ), out-of-context: all the other cytosines (hHxX)
  - "CHG" – within-the-context: CHG cytosines (xX), out-of-context: hHzZ
  - "CHH" – within-the-context: CHH cytosines (hH), out-of-context: xXzZ
  - "CxG" – within-the-context: CG and CHG cytosines (zZxX), out-of-context: CHH cytosines (hH)
  - "CX" – all cytosines are considered within-the-context, this effectively results in no thresholding

`filter.reads`
: boolean defining if sequence reads with too few context bases or too high out-of-context cytosine methylation should be filtered out (e.g., reads resulting from incompletely bisulfite-converted templates). Default: TRUE. Filtering is strongly recommended for short-read sequencing (bisulfite or enzymatic) because it removes reads from incompletely converted DNA molecules.

`min.context.sites`
: non-negative integer for minimum number of cytosines within the 'cytosine.context' (default: 0, i.e., all reads will satisfy this criterion). When 'min.context.sites'>0, reads containing **fewer** within-the-context cytosines will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`max.outofcontext.beta`
: real number in the range [0;1] (default: 0.1). Reads with average beta value for out-of-context cytosines **above** this threshold will not be thresholded and will be ignored in further computations. This option has no effect when read filtering is disabled.

`threshold.reads`
: boolean defining if sequence reads should be thresholded before counting bases in reference and variant epialleles (default: TRUE). Disabling thresholding is possible but makes no sense in the context of this function, because all the reads will be assigned to the variant epiallele, which will result in Fisher's Exact test p-value of 1 (in columns 'FEp+' and 'FEP-').

`min.context.beta`
: real number in the range [0;1] (default: 0.5). Reads with average beta value for within-the-context cytosines **below** this threshold are considered completely unmethylated (thus belonging to the reference epiallele). This option has no effect when read thresholding is disabled.

`...`
: other parameters to pass to the `preprocessBam` function. Options have no effect if preprocessed BAM data was supplied as an input.

`gzip`
: boolean to compress the report (default: FALSE).

`verbose`
: boolean to report progress and timings (default: TRUE).

Details

Using BAM reads and sequence variation information as an input, 'generateVcfReport' function filters and thresholds the reads (for paired-end sequencing alignment files - read pairs as a single

entity) according to supplied parameters and calculates the occurrence of **Ref**erence and **Alt**ernative bases within reads, taking into the account DNA strand the read mapped to and average methylation level (epiallele status) of the read.

The information on sequence variation can be supplied as a Variant Call Format (VCF) file location or an object of class VCF, returned by the `readVcf` function call. As whole-genome VCF files can be extremely large, it is strongly advised to use only relevant subset of their data, prefiltering the VCF object manually before calling 'generateVcfReport' or specifying 'bed' parameter when 'vcf' points to the location of such large VCF file. Please note that all the BAM, BED and VCF files must use the same style for seqlevels (i.e. chromosome names).

After counting, function checks if certain bases occur more often within reads belonging to certain epialleles using Fisher Exact test (HTSlib's own implementation) and reports separate p-values for reads mapped to **"+"** (forward) and **"-"** (reverse) DNA strands.

Please note that the final report currently includes only the VCF entries with single-base REF and ALT alleles. Also, the default ('min.baseq=0') output of 'generateVcfReport' is equivalent to the one of 'samtools mplieup -Q 0 ...', and therefore may result in false SNVs caused by misalignments. Remember to increase 'min.baseq' ('samtools mpileup -Q' default value is 13) to obtain higher-quality results.

Read thresholding by an average methylation level used in this function makes little sense for long-read sequencing alignments, as such reads can cover multiple regions with very different DNA methylation properties. If necessary, one could either clip long sequencing reads to narrow 'targets' in `preprocessBam` function during BAM loading or use `extractPatterns`, limiting pattern output to the region of interest only.

Value

`data.table` object containing VCF report or NULL if report.file was specified. The report columns are:

- name – variation identifier (e.g. "rs123456789")
- seqnames – reference sequence name
- range – genomic coordinates of the variation
- REF – base at the reference allele
- ALT – base at the alternative allele
- nfiltered – number of filtered out reads
- [M|U][+|-][Ref|Alt] – number of **Ref**erence or **Alt**ernative bases that were found at this particular position within **M**ethylated (above threshold) or **U**nmethylated (below threshold) reads that were mapped to **"+"** (forward) or **"-"** (reverse) DNA strand. NA values mean that it is not possible to determine the number of bases due to the bisulfite conversion-related limitations (C->T variants on "+" and G->A on "-" strands)
- SumRef – sum of all **Ref**erence base counts
- SumAlt – sum of all **Alt**ernative base counts
- FEp+ – Fisher Exact test p-value for association of a variation with methylation status of the reads that map to the **"+"** (forward) DNA strand. Calculated using following contingency table:

M+Ref	M+Alt
U+Ref	U+Alt

- FEp- – Fisher Exact test p-value for association of a variation with methylation status of the reads that map to the **"-"** (reverse) DNA strand. Calculated using following contingency table:

M-Ref	M-Alt
U-Ref	U-Alt

See Also

preprocessBam for preloading BAM data, generateCytosineReport for methylation statistics at the level of individual cytosines, generateBedReport for genomic region-based statistics, extractPatterns for exploring methylation patterns and plotPatterns for pretty plotting of its output, generateBedEcdf for analysing the distribution of per-read beta values, and 'epialleleR' vignettes for the description of usage and sample data.

GRanges class for working with genomic ranges, readVcf function for loading VCF data, seqlevelsStyle function for getting or setting the seqlevels style.

Examples

```
capture.bam <- system.file("extdata", "capture.bam", package="epialleleR")
capture.bed <- system.file("extdata", "capture.bed", package="epialleleR")
capture.vcf <- system.file("extdata", "capture.vcf.gz",
                           package="epialleleR")

# VCF report
vcf.report <- generateVcfReport(bam=capture.bam, bed=capture.bed,
                                vcf=capture.vcf)

# toy example to illustrate the logic of computations
if (requireNamespace("VariantAnnotation", quietly=TRUE)) {
  # simulate toy BAM
  temp.bam <- tempfile(fileext=".bam")
  simulateBam(output.bam.file=temp.bam, rname="chr1", XG="CT",
              seq=c("AGACGTTAGTAATAGTA", "AGACGTTGTAATAGTA",
                    "AAACGTTGTAACAGTA",  "AAACGTTGTAATGTA"),
              XM=c( "...Z..x+.h..x..h.", "...Z..z.h..x..h.",
                    "...Z..z.h..X..h.",  "...Z..z.h..z.h."),
              cigar=c("7M1I9M", "16M", "16M", "12M1D3M"))
  # toy VCF
  vcf <- VariantAnnotation::VCF(rowRanges=as("chr1:2", "GRanges"),
                                collapsed=FALSE)
  VariantAnnotation::ref(vcf) <- as("A", "DNAStringSet")
  VariantAnnotation::alt(vcf) <- as("G", "DNAStringSet")

  # when default values of filtering and thresholding parameters are used,
  # read filtering will exclude the third read from this BAM file
  # because it has too many outside-of-context methylated cytosines.

  # results with read filtering and thresholding
```

```
  generateVcfReport(bam=temp.bam, vcf=vcf)
  # results without read filtering
  generateVcfReport(bam=temp.bam, vcf=vcf, filter.reads=FALSE)
}
```

---

plotPatterns *plotPatterns*

Description

This convenience function plots methylation patterns (epialleles) previously extracted by `extractPatterns`.

Usage

```
plotPatterns(
  patterns,
  order.by = c("beta", "count"),
  beta.range = c(0, 1),
  bin.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  nbins = 10,
  npatterns.per.bin = 2,
  plot.context = c("CG", "CHG", "CHH", "CxG", "CX"),
  genomic.scale = c("continuous", "discrete"),
  breaks = "auto",
  marginal = c("density", "count"),
  marginal.position = c("left", "right"),
  marginal.transform = c("identity", "log10"),
  marginal.limits = NULL,
  marginal.size = 0.25,
  ...,
  tag = c("none", "count", "beta", "pattern"),
  tag.size = 2.5,
  tag.colour = "#87654c",
  tag.fill = "lemonchiffon",
  title = TRUE,
  subtitle = TRUE,
  context.size = c(1, 2, 3),
  base.size = 3,
  methylation.fill = c("grey97", "grey10"),
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

patterns	output of `preprocessBam` function (methylation patterns as a `data.table` object).

`order.by` string defining order of patterns on the plot (default order by: "beta").

`beta.range` numeric vector of length 2 for the range of average pattern beta values represented on the plot (default: [0;1]).

`bin.context` string defining cytosine methylation context used to calculate average beta value of a pattern that is further used to assign patterns to bins:

- "CG" (the default) – CpG cytosines (called as zZ)
- "CHG" – CHG cytosines (xX)
- "CHH" – CHH cytosines (hH)
- "CxG" – CG and CHG cytosines (zZxX)
- "CX" – all cytosines

`nbins` a single integer defining the number of bins (i.e., intervals within 'beta.range'). Default: 10.

`npatterns.per.bin` integer vector for the number of the most abundant patterns selected from each bin (default: 2). When of length 1, the same number of patterns will be taken. When of length 'nbins', allows to fine-tune the number of selected patterns from each bin. Setting to 'Inf' effectively results in plotting all patterns.

`plot.context` string defining methylation context of cytosines included in the plot (default: "CG"; for the range of available values, see 'bin.context' above).

`genomic.scale` string for the type of genomic position scale of the plot: either "continuous" (the default) or "discrete".

`breaks` a vector of breaks for the genomic position scale of the plot. If "auto" (the default), breaks for continuous scale are computed by the default ggplot2 routines, while breaks for discrete scale are a subset of plotted positions selected using `pretty`. Possible values: `ggplot2::waiver()` for ggplot2 defaults, integer vector of breaks for continuous scale, or character vector of breaks for discrete scale.

`marginal` string for the type of marginal plot: either "density" (probability density of average beta values of all patterns; the default) or "count" (counts of plotted patterns). "none" is not implemented yet; create an issue if interested.

`marginal.position` string for the position of marginal plot: either "left" (the default) or "right" (not implemented yet; create an issue if interested).

`marginal.transform` string for the transformation of marginal scale (default: "identity"). Check `ggplot2::scale_x_continuous()` for more details.

`marginal.limits` limits of marginal scale (default: NULL). Check `ggplot2::scale_x_continuous()` for more details.

`marginal.size` numeric in range (0;1) for the relative width of the marginal plot (default: 0.25).

`...` additional arguments passed to `stats::density()` call used in marginal density plot. Possible value: `adjust=0.25`.

`tag` string for optional tagging of patterns with their count ("count"), average beta value ("beta"), or pattern ID ("pattern"). Default: "none".

tag.size	numeric for the font size of the tag text (in millimetres; default: 2.5).
tag.colour	string for the colour of of the tag text. Default: "#87654c".
tag.fill	string for the colour of of the tag background. Default: "lemonchiffon".
title	the title of the plot. When 'TRUE' (the default), a genomic region from which patterns were extracted. Other possible values: anything that can be converted to string, or 'NULL' for no title.
subtitle	the subtitle of the plot. When 'TRUE' (the default), a number of patterns plotted. Other possible values: anything that can be converted to string, or 'NULL' for no subtitle.
context.size	a numeric vector with sizes of circles representing cytosines within each of three contexts: CHH, CHG, and CG (default: c(1, 2, 3)).
base.size	numeric for the font size of the text for highlighted bases (in millimetres; default: 3).
methylation.fill	a vector of length 2 for colours representing unmethylated and methylated cytosines, respectively. These colours are also mapped to the lowest (0) and highest (1) possible beta values to represent average beta values of methylation patterns and create a gradient fill of a marginal density plot. Default: c("grey97", "grey10").
plot	boolean. If 'TRUE' (the default), patterns are plotted, and the selected ones are silently returned as a data.table object. If 'FALSE', the grob table object is returned instead.
verbose	boolean to report basic info on input and output.

Details

As the number of methylation patterns can be quite large, by default, the function plots **the most abundant unique patterns** only. The complete logic is as follows:

- from the input methylation patterns, all unique patterns are extracted and counted
- unique patterns are split in bins by their average beta value
- most abundant unique methylation patterns from each bin are plotted and silently returned

On the resulting plot, each cytosine is shown as a circle, where the size of that circle represents cytosine context and the fill encodes methylation status. If available, highlighted bases are shown as labels of different colours.

Value

the plot and (silently) the data.table object containing plotted methylation patterns (if 'plot==TRUE'), or grob table object (if 'plot==FALSE').

See Also

extractPatterns for extracting methylation patterns, preprocessBam for preloading BAM data, generateCytosineReport for methylation statistics at the level of individual cytosines, generateBedReport for genomic region-based statistics, generateVcfReport for evaluating epiallele-SNV associations, generateBedEcdf for analysing the distribution of per-read beta values, and 'epialleleR' vignettes for the description of usage and sample data.

Examples

```
# amplicon data
amplicon.bam <- system.file("extdata", "amplicon010meth.bam",
                            package="epialleleR")
custom.range <- as("chr17:43124861-43125150", "GRanges")

# let's get our patterns
patterns <- extractPatterns(bam=amplicon.bam, bed=custom.range)

# default plot + silently returned plotted patterns
selected.patterns <- plotPatterns(patterns)

# all unique patterns with their counts as a margin, categorical positions,
# tagged with pattern IDs, returned as a `gtable` object
tbl <- plotPatterns(patterns, npatterns.per.bin=Inf, marginal="count",
                    genomic.scale="discrete", tag="pattern", plot=FALSE)

# which can be plotted later
grid::grid.newpage()
grid::grid.draw(tbl)
```

`preprocessBam` *preprocessBam*

Description

This function reads and preprocesses BAM file.

Usage

```
preprocessBam(
  bam.file,
  paired = NULL,
  override.check = FALSE,
  min.mapq = 0,
  min.baseq = 0,
  min.prob = -1,
  highest.prob = TRUE,
  skip.duplicates = FALSE,
  skip.secondary = TRUE,
  skip.qcfail = TRUE,
  skip.supplementary = TRUE,
  trim = 0,
  targets = NULL,
  zero.based.targets = FALSE,
  clip.to.targets = FALSE,
  nthreads = 1,
```

```
  verbose = TRUE
)
```

Arguments

bam.file	BAM file location string.
paired	boolean for expected alignment endness: 'TRUE' for paired-end, 'FALSE' for single-end, or 'NULL' for auto detect (the default).
override.check	boolean to use supplied endness ('paired' parameter) even if it is different from the autodetected one (default: FALSE).
min.mapq	non-negative integer threshold for minimum read mapping quality. Default is 0, however a higher value (e.g., 30) is recommended.
min.baseq	non-negative integer threshold for minimum nucleotide base quality. Default is 0, however a higher value (e.g., at least 13 as in 'samtools mpileup') is recommended.
min.prob	integer threshold for minimum scaled probability of modification (methylation) to consider. Affects processing of long-read sequencing alignments only. According to SAM/BAM specification, the continuous base modification probability range 0.0 to 1.0 is remapped in equal sized portions to the discrete integers 0 to 255 inclusively. Default is -1, however a higher value (e.g., 178 which corresponds to a continuous base modification probability of ~0.7) is strongly recommended. When default (-1), all C+m and G-m cytosine methylation modifications recorded in MM/Mm tag will be included, even if ML/Ml tag with probabilities is absent (in such case, probability of modification equals -1). If a base has at least one modifications, and neither the probability of any of modifications nor the probability of the conventional base reach the 'min.prob', this base will be masked in this particular read and not counted neither as modified nor as conventional.
highest.prob	boolean defining if methylation modification must have the highest probability among all modifications at a particular base to be considered in the analyses (default: TRUE). Affects processing long-read sequencing alignments only. If default (TRUE) and ML/Ml tag with probability scores is absent, then cytosines with more than one modification will be omitted (as the probability of all modifications will be equal).
skip.duplicates	boolean defining if duplicate aligned reads should be skipped (default: FALSE). Option has no effect if duplicate reads were not marked by alignment software.
skip.secondary	boolean defining if secondary alignments should be skipped (default: TRUE). Do not change.
skip.qcfail	boolean defining if alignments failing QC should be skipped (default: TRUE). Do not change.
skip.supplementary	boolean defining if supplementary alignments should be skipped (default: TRUE). Do not change.
trim	non-negative integer or vector of length 2 for the number of nucleotide bases to be trimmed from 5' and 3' ends of a template (i.e., read pair for paired-end

BAM or read for single-end BAM). Default: 0 for no trimming. Specifying 'trim=1' will result in removing of a single base from both ends, while specifying 'trim=c(1,2)' will result in removing of a single base from 5' end and 2 bases from 3' end.

`targets` Browser Extensible Data (BED) file location string OR object of class `GRanges` holding genomic coordinates for regions of interest. The style of seqlevels of BED file/object must be the same as the style of seqlevels of BAM file. During targets loading, the genomic regions are reduced (i.e., overlapping regions are merged), and the strand information is discarded. These reduced regions are then used to only load overlapping reads (merged read pairs for paired-end sequencing).

`zero.based.targets`
boolean defining if BED file coordinates are zero-based (default: FALSE).

`clip.to.targets`
boolean defining if overlapping reads (merged read pairs in case of paired-end sequencing) should be clipped to retain only fragments overlapping with 'targets' (default: FALSE). If a read overlaps with two or more targets, all overlapping fragments will be retained.

`nthreads` non-negative integer for the number of additional HTSlib threads to be used during BAM file decompression (default: 1). Two threads (and usually no more than four) make sense for files larger than 100 MB.

`verbose` boolean to report progress and timings (default: TRUE).

Details

The function loads and preprocesses BAM file, saving time when multiple analyses are to be performed on large input files. Currently, HTSlib is used to read the data, therefore it is possible to speed up the loading by means of HTSlib decompression threads.

This function is also called internally when BAM file location is supplied as an input for other 'epialleleR' methods.

'preprocessBam' currently allows to load both short-read (e.g., bisulfite) and long-read (native) sequencing alignments. Specific requirements for these types of data are given below.

Value

`data.table` object containing preprocessed BAM data.

NB: most of the BAM data is stored not in the data.table *per se*, but as an external object linked to this data.table. Therefore, saving/loading this data.table cannot be used to save/recover preprocessed BAM data.

Short-read sequencing

For preprocessing of short reads (and therefore for all reporting methods), 'epialleleR' requires genomic strand (XG tag) and a methylation call string (XM tag) to be present in a BAM file - i.e., methylation calling must be performed after read mapping/alignment by your software of choice. It is the case for BAM files produced by Bismark Bisulfite Read Mapper and Methylation Caller, Illumina DRAGEN, Illumina Cloud analysis solutions, as well as contemporary Illumina sequencing

instruments with on-board read mapping/alignment (NextSeq 1000/2000, NovaSeq X), therefore such files can be analysed without additional steps. For alignments produced by other tools, e.g., BWA-meth or BSMAP, methylation calling must be performed prior to BAM loading / reporting, by means of callMethylation.

Long-read sequencing

For preprocessing of long reads, ‘epialleleR‘ requires presence of MM (Mm) and ML (Ml) tags that hold information on base modifications and related probabilities, respectively. These are standard tags described in SAM/BAM format specification, therefore relevant tools for analysis and alignment of long sequencing reads should be able to produce them.

Other details

‘preprocessBam‘ always tests if BAM file is paired- or single-ended and has all the necessary tags available. It is recommended to use ‘verbose‘ processing and check messages for correct identification of alignment endness. Otherwise, if the ‘paired‘ parameter is set explicitly, exception is thrown when expected endness differs from the autodetected one. It is nevertheless possible to override the autodetected endness and load BAM as specified in ‘paired‘ by setting the ‘override.check‘ to TRUE.

During preprocessing of paired-end alignments, paired reads are merged according to their base quality: nucleotide base with the highest value in the QUAL string is taken, unless its quality is less than ‘min.baseq‘, which results in no information for that particular position ("-"/"N"). These merged reads are then processed as a single entity in all ‘epialleleR‘ methods. Due to merging, overlapping bases in read pairs are counted only once, and the base with the highest quality is taken.

It is currently a requirement that paired-end BAM file must be sorted by QNAME instead of genomic location (i.e., "unsorted") to perform merging of paired-end reads. Error message is shown if it is sorted by genomic location, in this case please sort it by QNAME using ’samtools sort -n -o out.bam in.bam’.

During preprocessing of single-end alignments, no read merging is performed. Only bases with quality of at least ‘min.baseq‘ are considered. Lower base quality results in no information for that particular position ("-"/"N").

For RRBS-like protocols, it is possible to trim alignments from one or both ends. Trimming is performed during BAM loading and will therefore influence results of all downstream ‘epialleleR‘ methods. Internally, trimming is performed at the level of a template (i.e., read pair for paired-end BAM or individual read for single-end BAM). This ensures that only necessary parts (real ends of sequenced fragment) are removed for paired-end sequencing reads.

It is also possible to load only a subset of reads (read pairs) of interest or only fragments of such reads by supplying a list of targets (see description of ‘targets‘ and other related options). The subsetting is performed without using BAM index.

Specific considerations for long-read sequencing data

Any location not reported is implicitly assumed to contain no modification.

According to SAM format specification, MM base modification tags are allowed to list modifications observed not only on the original sequenced strand (e.g., ‘C+m‘) but also on the opposite strand (e.g., ‘G-m‘). The logic of their processing is as follows (with the examples given below):

- if an alignment record has no methylation modifications (neither 'C+m', nor 'G-m' are present), this record is, naturally, considered to be a single read with no cytosines methylated
- if an alignment record has 'C+m' modification (base modifications on the original sequenced strand), then this record is, naturally, considered to be a single read with cytosine modifications on the sequenced strand
- if an alignment record has 'G-m' modification (base modifications on the strand opposite to sequenced), then this record is treated as two reads, with the original sequenced strand having no modifications, while the opposite strand having cytosine modifications
- if both 'C+m' and 'G-m' are present, then this record is treated as two reads, with both strands having cytosine modifications

See Also

`preprocessGenome` for preloading reference sequences and `callMethylation` for methylation calling.

`generateCytosineReport` for methylation statistics at the level of individual cytosines, `generateBedReport` for genomic region-based statistics, `generateVcfReport` for evaluating epiallele-SNV associations, `extractPatterns` for exploring methylation patterns and `plotPatterns` for pretty plotting of its output, `generateBedEcdf` for analysing the distribution of per-read beta values, and 'epialleleR' vignettes for the description of usage and sample data.

Sequence Alignment/Map format specifications, specifications for optional SAM tags, duplicate alignments marking by Samtools and Illumina DRAGEN Bio IT Platform.

Examples

```
# short-read sequencing
capture.data <- preprocessBam(
  system.file("extdata", "capture.bam", package="epialleleR"),
  targets=as("chr17:43120000-43130000", "GRanges")
)
generateCytosineReport(capture.data, threshold.reads=TRUE)

# long-read sequencing
longread.data <- preprocessBam(
  system.file("extdata", "longread.bam", package="epialleleR"),
  min.mapq=30, min.baseq=20, min.prob=178
)
generateCytosineReport(longread.data, threshold.reads=FALSE)

# Specifics of long-read alignment processing
out.bam <- tempfile(pattern="out-", fileext=".bam")

simulateBam(
  seq=c("ACGCCATYCGCGCCA"),
  Mm=c("C+m,0,2,0;"),
  Ml=list(as.integer(c(102,128,153))),
  output.bam.file=out.bam
)
generateCytosineReport(out.bam, threshold.reads=FALSE, report.context="CX")
```

```
simulateBam(
  seq=c("ACGCCATYCGCGCCA"),
  Mm=c("G-m,0,0,0;"),
  Ml=list(as.integer(c(138,101,96))),
  output.bam.file=out.bam
)
generateCytosineReport(out.bam, threshold.reads=FALSE, report.context="CX")

simulateBam(
  seq=c("ACGCCATYCGCGCCA"),
  Mm=c("C+m,0,2,0;G-m,0,0,0;"),
  Ml=list(as.integer(c(102,128,153,138,101,96))),
  output.bam.file=out.bam
)
generateCytosineReport(out.bam, threshold.reads=FALSE, report.context="CX")
```

`preprocessGenome` *preprocessGenome*

Description

This function reads and preprocesses (optionally 'bgzip'ped) FASTA file with reference sequences.

Usage

```
preprocessGenome(genome.file, nthreads = 1, verbose = TRUE)
```

Arguments

`genome.file`	reference (genomic) sequences file location string.
`nthreads`	non-negative integer for the number of additional HTSlib threads to be used during file decompression (default: 1).
`verbose`	boolean to report progress and timings (default: TRUE).

Details

The function loads and preprocesses reference (genomic) sequences, saving time when methylation calling needs to be performed on multiple BAM files. Currently, reading the data is done by means of HTSlib, therefore it is possible to speed up the loading by means of HTSlib decompression threads when FASTA file is compressed by 'bgzip'.

This function is also called internally when file location is supplied as an input for `callMethylation` method.

'preprocessGenome' checks if index file is present, and if not, creates it automatically. It is possible and recommended to use compressed FASTA file as an input, but the file must be compressed by 'bgzip' (part of samtools/HTSlib). When FASTA file is compressed, faster loading can be achieved using (typically one) additional HTSlib decompression thread.

During loading, both lowercase and uppercase ACGTN symbols are allowed and correctly recognised, however all the other symbols (e.g., extended IUPAC symbols, MRSVWYHKDB) within sequences are converted to N.

Please also note that for the purpose of methylation calling, the very same reference genome must be used for both alignment (when BAM is produced) and calling cytosine methylation by `callMethylation` method.

Value

list object containing preprocessed reference sequence data.

See Also

`callMethylation` for methylation calling, and 'epialleleR' vignettes for the description of usage and sample data.

Block compression/decompression utility bgzip.

Examples

```
genome.file <- system.file("extdata", "test", "reference.fasta.gz", package="epialleleR")
 genome.data <- preprocessGenome(genome.file)
```

`simulateBam` *simulateBam*

Description

This function creates sample BAM files given mandatory and optional BAM fields.

Usage

```
simulateBam(
  output.bam.file = NULL,
  qname = NULL,
  flag = NULL,
  rname = NULL,
  pos = NULL,
  mapq = NULL,
  cigar = NULL,
  rnext = NULL,
  pnext = NULL,
  tlen = NULL,
  seq = NULL,
  qual = NULL,
  ...,
  verbose = TRUE
)
```

Arguments

`output.bam.file`
: output BAM file location string. If NULL (default), records are not written to BAM but returned as a `data.table` object for review.

`qname`
: character vector of query names. When default (NULL), names like "q0001".."qNNNN" will be assigned.

`flag`
: integer vector of bitwise flags (a combination of the BAM_F* constants). When default (NULL), zero (i.e., unique, valid, single-end, aligned read) is assigned for every record.

`rname`
: character vector of chromosome (reference) names. When default (NULL), "chrS" is assigned for every record.

`pos`
: integer vector of 1-based leftmost coordinates of the queries. When default (NULL), 1 is assigned for every record.

`mapq`
: integer vector of mapping qualities. When default (NULL), 60 is assigned for every record.

`cigar`
: character vector of CIGAR strings. When default (NULL), "lM" is assigned for every record, where 'l' is the length of the query ('seq').

`rnext`
: character vector of chromosome (reference) names for next read in template. When default (NULL), "chrS" is assigned for every record.

`pnext`
: integer vector of 1-based leftmost coordinates of next read in template. When default (NULL), 1 is assigned for every record.

`tlen`
: integer vector of observed template lengths. When default (NULL), the length of the corresponding query ('seq') is assigned for every record.

`seq`
: character vector of query sequences. When default (NULL), random sequence is assigned. The lengths of these random sequences equal to the lengths of methylation call strings from the 'XM' optional parameter (if supplied), or to the 'tlen' parameter (if defined). If none of these parameters is supplied, length of every 'seq' will equal 10.

`qual`
: query sequence quality strings (ASCII of base QUALity plus 33). When default (NULL), quality of every base is assigned to "F" (QUALity of 47 + 33). The lengths of these quality strings equal to the length of the corresponding query sequences ('seq') for every record.

`...`
: optional tags to add to the records, in the form 'tag=value'. Value can be either:

  - an integer vector to create a tag with a single integer value per alignment record (e.g., "NM" tag),
  - or a float vector to create a tag with a single float value per alignment record,
  - or a character vector (e.g., "XM" tag for methylation call string, "XG"/"YD"/"ZS" tag for reference strand read was aligned to)
  - or a list of numeric vectors to create tags array holding arrays of numeric values.

`verbose`
: boolean to report progress and timings (default: TRUE).

Details

The function creates sample alignment records and saves them in BAM file. Output can be used to test epialleleR methods as well as other tools for methylation analysis. This method can significantly simplify calculation of methylation metrics on example data (beta, VEF, and lMHL values of epialleleR; methylation heterogeneity metrics of other tools).

The number of records written will be equal to the largest length of any supplied (nondefault) parameter or 1 if no parameters were supplied. If lengths of supplied parameters differ, shorter vectors will be recycled (a whole number of times or with remainder if necessary).

Please note that function performs almost no validity checks for supplied fields. In particular, be extra careful constructing paired-end BAM alignments, and if necessary use 'samtools' to perform validity check or manual editing after BAM->SAM conversion.

Value

number of BAM records written (if 'output.bam.file' is not NULL) or `data.table` object containing final records prepared for writing. NB: this object has 0-based coordinates and numerically encoded reference names.

See Also

`generateCytosineReport` and `generateMhlReport` for methylation reports at the level of individual cytosines, as well as 'epialleleR' vignettes for the description of usage and sample data.

Samtools for viewing BAM files. SAMv1 file format specifications. Specifications of optional SAM tags. metheor for ultrafast DNA methylation heterogeneity calculation from bisulfite alignments.

Examples

```
out.bam <- tempfile(pattern="simulated", fileext=".bam")
simulateBam(
  output.bam.file=out.bam,
  pos=c(1, 2),
  XM=c("ZZZzzZZZ", "ZZzzzzZZ"),
  XG=c("CT", "AG"),
  xi=5:6,
  xf=0.05,
  ai=list(as.integer(c(1:3)), as.integer(c(4:6))),
  af=list(seq(-1, 1, 0.5))
)
generateCytosineReport(out.bam, threshold.reads=FALSE)
# check this BAM with `samtools view` or using `output.bam.file=NULL`
```

Index

callMethylation, 2, *3*, *34–37*

data.table, *5*, *11*, *13*, *16*, *18*, *20*, *22*, *24*, *26*, *28*, *30*, *33*, *38, 39*

extractPatterns, 4, *9*, *13, 14*, *18*, *22*, *26–28*, *30*, *35*

generateAmpliconReport *(*generateBedReport*)*, 9
generateBedEcdf, *6*, 7, *14*, *18*, *22*, *27*, *30*, *35*
generateBedReport, *6*, *9*, 9, *18*, *22*, *27*, *30*, *35*
generateCaptureReport *(*generateBedReport*)*, 9
generateCytosineReport, *6*, *9*, *14*, 15, *22*, *27*, *30*, *35*, *39*
generateMhlReport, 19, *39*
generateVcfReport, *6*, *9*, *14*, *18*, *22*, 23, *30*, *35*
ggplot2::scale_x_continuous(), *29*
ggplot2::waiver(), *29*
GRanges, *4, 5*, *7, 8*, *11–14*, *24*, *27*, *33*

plotPatterns, *5, 6*, *9*, *14*, *18*, *22*, *27*, 28, *35*
preprocessBam, *4–9*, *11–14*, *16–18*, *20–22*, *24–28*, *30*, 31
preprocessGenome, *2–4*, *35*, 36
pretty, *29*

readVcf, *24*, *26, 27*

seqlevelsStyle, *14*, *24*, *27*
simulateBam, *37*
stats::density(), *29*