# Package 'bsseq'

July 13, 2025

**Version** 1.45.2

**Encoding** UTF-8

**Title** Analyze, manage and store whole-genome methylation data

**Description** A collection of tools for analyzing and visualizing whole-genome methylation data from sequencing. This includes whole-genome bisulfite sequencing and Oxford nanopore data.

**Depends** R (>= 4.0), methods, BiocGenerics, GenomicRanges (>= 1.41.5), SummarizedExperiment (>= 1.19.5)

**Imports** IRanges (>= 2.23.9), Seqinfo, scales, stats, parallel, tools, graphics, Biobase, locfit, gtools, data.table (>= 1.11.8), S4Vectors (>= 0.27.12), R.utils (>= 2.0.0), DelayedMatrixStats (>= 1.5.2), permute, limma, DelayedArray (>= 0.15.16), Rcpp, BiocParallel, BSgenome, Biostrings, utils, HDF5Array (>= 1.19.11), rhdf5, beachmat (>= 2.23.2)

**Suggests** testthat, bsseqData, BiocStyle, rmarkdown, knitr, Matrix, doParallel, rtracklayer, BSgenome.Hsapiens.UCSC.hg38, batchtools

**Collate** utils.R hasGRanges.R BSseq-class.R BSseqTstat_class.R BSseq_utils.R combine.R read.bismark.R read.bedMethyl.R read.modbam2bed.R read.modkit.R BSmooth.R BSmooth.tstat.R dmrFinder.R gof_stats.R plotting.R fisher.R permutations.R BSmooth.fstat.R BSseqStat_class.R getStats.R hdf5_utils.R DelayedArray_utils.R collapseBSseq.R FWIRanges-class.R FWGRanges-class.R findLoci.R Likelihood_functions.R

**License** Artistic-2.0

**VignetteBuilder** knitr

**URL** https://github.com/kasperdanielhansen/bsseq

**BugReports** https://github.com/kasperdanielhansen/bsseq/issues

**biocViews** DNAMethylation

**LinkingTo** Rcpp, beachmat, assorthead (>= 1.1.4)

**SystemRequirements** C++17

**NeedsCompilation** yes

**RoxygenNote** 7.1.0

**git_url** https://git.bioconductor.org/packages/bsseq

**git_branch** devel

**git_last_commit** c673ef9

**git_last_commit_date** 2025-07-10

**Repository** Bioconductor 3.22

**Date/Publication** 2025-07-13

**Author** Kasper Daniel Hansen [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0086-0687>),
Peter Hickey [aut] (ORCID: <https://orcid.org/0000-0002-8153-6258>),
Hervé Pagès [ctb],
Aaron Lun [ctb]

**Maintainer** Kasper Daniel Hansen <kasperdanielhansen@gmail.com>

# Contents

BS.chr22 *Whole-genome bisulfite sequencing for chromosome 22 from Lister et al.*

## Description

This dataset represents chromosome 22 from the IMR90 cell line sequenced in Lister et al. Only CpG methylation are included (there were very few non-CpG loci). The two samples are two different extractions from the same cell line (ie. technical replicates), and are pooled in the analysis in the original paper.

## Usage

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

## Format

An object of class BSseq.

## Details

All coordinates are in hg18.

## Source

Obtained from http://neomorph.salk.edu/human_methylome/data.html specifically the files mc_imr90_r1.tar.gz and mc_imr90_r2.tar.gz. A script which downloads these files and constructs the BS.chr22 object may be found in 'inst/scripts/get_BS.chr22.R', see the example.

## References

R Lister et al. *Human DNA methylomes at base resolution show widespread epigenomic differences*. Nature (2009) 462, 315-322.

## Examples

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

script <- system.file("scripts", "get_BS.chr22.R", package = "bsseq")
script
readLines(script)
```

BSmooth *BSmooth, smoothing bisulfite sequence data*

## Description

This implements the BSmooth algorithm for estimating methylation levels from bisulfite sequencing data.

## Usage

```
BSmooth(BSseq,
        ns = 70,
        h = 1000,
        maxGap = 10^8,
        keep.se = FALSE,
        BPPARAM = bpparam(),
        chunkdim = NULL,
        level = NULL,
        verbose = getOption("verbose"))
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| ns | The minimum number of methylation loci in a smoothing window. |
| h | The minimum smoothing window, in bases. |
| maxGap | The maximum gap between two methylation loci, before the smoothing is broken across the gap. The default smoothes each chromosome separately. |
| keep.se | Should the estimated standard errors from the smoothing algorithm be kept. This will make the return object roughly 30 percent bigger and is currently not be used for anything in **bsseq**. |
| BPPARAM | An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation. Currently supported are SerialParam (Unix, Mac, Windows), MulticoreParam (Unix and Mac), SnowParam (Unix, Mac, and Windows, limited to single-machine clusters), and BatchtoolsParam (Unix, Mac, Windows, only with the in-memory realization backend). See sections 'Parallelization and progress monitoring' and 'Realization backends' for further details. |
| chunkdim | **Only applicable if** BACKEND == "HDF5Array"**.** The dimensions of the chunks to use for writing the data to disk. By default, getHDF5DumpChunkDim() using the dimensions of the returned BSseq object will be used. See ?{getHDF5DumpChunkDim} for more information. |
| level | **Only applicable if** BACKEND == "HDF5Array"**.** The compression level to use for writing the data to disk. By default, getHDF5DumpCompressionLevel() will be used. See ?getHDF5DumpCompressionLevel for more information. |
| verbose | A logical(1) indicating whether progress messages should be printed (default TRUE). |

## Details

ns and h are passed to the locfit function. The bandwidth used is the maximum (in genomic distance) of the h and a width big enough to contain ns number of methylation loci.

## Value

An object of class BSseq, containing coefficients used to fit smoothed methylation values and optionally standard errors for these.

## Realization backends

The BSmooth() function creates a new assay to store the coefficients used to construct the smoothed methylation estimates ((coef). An additional assay is also created if keep.se == TRUE (se.coef).

The choice of *realization backend* controls whether these assay(s) are stored in-memory as an ordinary matrix or on-disk as a HDF5Array, for example.

The choice of realization backend is controlled by the BACKEND argument, which defaults to the current value of DelayedArray::getAutoRealizationBackend().

BSmooth supports the following realization backends:

- NULL (in-memory): This stores each new assay in-memory using an ordinary matrix.
- HDF5Array (on-disk): This stores each new assay on-disk in a HDF5 file using an HDF5Matrix from **HDF5Array**.

Please note that certain combinations of realization backend and parallelization backend are currently not supported. For example, the HDF5Array realization backend is currently only compatible when used with a single-machine parallelization backend (i.e. it is not compatible with a SnowParam that specifies an *ad hoc* cluster of **multiple** machines). BSmooth() will issue an error when given such incompatible realization and parallelization backends. Furthermore, to avoid memory usage blow-ups, BSmooth() will issue an error if an in-memory realization backend is used when smoothing a disk-backed BSseq object.

Additional arguments related to the realization backend can be passed via the ... argument. These arguments must be named and are passed to the relevant RealizationSink constructor. For example, the ... argument can be used to specify the path to the HDF5 file to be used by BSmooth(). Please see the examples at the bottom of the page.

## Parallelization and progress monitoring

BSmooth() now uses the **BiocParallel** package to implement parallelization. This brings some notable improvements:

- Smoothed results can now be written directly to an on-disk realization backend by the worker. This dramatically reduces memory usage compared to previous versions of **bsseq** that required all results be retained in-memory.
- Parallelization is now supported on Windows through the use of a SnowParam object as the value of BPPARAM.
- Detailed and extensive job logging facilities.

All parallelization options are controlled via the BPPARAM argument. In general, we recommend that users combine multicore (single-machine) parallelization with an on-disk realization backend (see section, 'Realization backend'). For Unix and Mac users, this means using a MulticoreParam. For Windows users, this means using a single-machine SnowParam. Please consult the **BiocParallel** documentation to take full advantage of the more advanced features.

**Deprecated arguments:** `parallelBy`, `mc.cores`, and `mc.preschedule` are deprecated and will be removed in subsequent releases of **bsseq**. These arguments were necessary when `BSmooth()` used the **parallel** package to implement parallelization, but this functionality is superseded by the aforementioned use of **BiocParallel**. We recommend that users who previously relied on these arguments switch to `BPPARAM = MulticoreParam(workers = mc.cores, progressbar = TRUE)`.

**Progress monitoring:** A useful feature of **BiocParallel** are progress bars to monitor the status of long-running jobs, such as `BSmooth()`. Progress bars are controlled via the `progressbar` argument in the `BiocParallelParam` constructor. Progress bars replace the use of the deprecated `verbose` argument to print out information on the status of `BSmooth()`.

**BiocParallel** also supports extensive and detailed logging facilities. Please consult the **BiocParallel** documentation to take full advantage these advanced features.

## Author(s)

Method and original implementation by Kasper Daniel Hansen <khansen@jhsph.edu>. Updated implementation to support disk-backed BSseq objects and more general parallelization by Peter Francis Hickey.

## References

KD Hansen, B Langmead, and RA Irizarry. *BSmooth: from whole genome bisulfite sequencing reads to differentially methylated regions*. Genome Biology (2012) 13:R83. doi:10.1186/gb-2012-13-10-r83.

## See Also

`locfit` in the locfit package, as well as `BSseq`.

## Examples

```
## Not run:
  # Run BSmooth() on a matrix-backed BSseq object using an in-memory realization
  # backend with serial evaluation.
  data(BS.chr22)
  # This is a matrix-backed BSseq object.
  sapply(assays(BS.chr22, withDimnames = FALSE), class)
  BS.fit <- BSmooth(BS.chr22, BPPARAM = SerialParam(progressbar = TRUE))
  # The new 'coef' assay is an ordinary matrix.
  sapply(assays(BS.fit, withDimnames = FALSE), class)
  BS.fit

  # Run BSmooth() on a disk-backed BSseq object using the HDF5Array realization
  # backend (with data written to the file 'BSmooth_example.h5') with
  # multi-core parallel evaluation.
  BS.chr22 <- realize(BS.chr22, "HDF5Array")
  # This is a disk-backed BSseq object.
  sapply(assays(BS.chr22, withDimnames = FALSE), class)
  BS.fit <- BSmooth(BS.chr22,
              BPPARAM = MulticoreParam(workers = 2, progressbar = TRUE),
              BACKEND = "HDF5Array",
              filepath = "BSmooth_example.h5")
  # The new 'coef' assay is an HDF5Matrix.
  sapply(assays(BS.fit, withDimnames = FALSE), class)
  BS.fit
```

```
  # The new 'coef' assay is in the HDF5 file 'BSmooth_example.h5' (in the
  # current working directory).
  sapply(assays(BS.fit, withDimnames = FALSE), path)

## End(Not run)
```

---

`BSmooth.fstat` *Compute F-statistics based on smoothed whole-genome bisulfite sequencing data.*

---

## Description

Compute F-statistics based on smoothed whole-genome bisulfite sequencing data.

## Usage

```
BSmooth.fstat(BSseq, design, contrasts, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `BSseq` | An object of class `BSseq`. |
| `design` | The design matrix of the bisulfite-sequencing experiment, with rows corresponding to arrays and columns to coefficients to be estimated. |
| `contrasts` | Numeric matrix with rows corresponding to columns in `design` and columns containing contrasts. May be a vector if there is only one contrast. |
| `verbose` | Should the function be verbose? |

## Details

**TODO**

## Value

An object of class BSseqStat.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

`BSmooth` for the input object and BSseq for its class. BSseqStat describes the return class. This function is likely to be followed by the use of `smoothSds`, `computeStat`, and `dmrFinder`.

## Examples

```
if(require(bsseqData)) {
# limma required for makeContrasts()
library(limma)
data(keepLoci.ex)
data(BS.cancer.ex.fit)
BS.cancer.ex.fit <- updateObject(BS.cancer.ex.fit)
## Remember to subset the BSseq object, see vignette for explanation
## TODO: Kind of a forced example
design <- model.matrix(~0 + BS.cancer.ex.fit$Type)
colnames(design) <- gsub("BS\\.cancer\\.ex\\.fit\\$Type", "",
                         colnames(design))
contrasts <- makeContrasts(
    cancer_vs_normal = cancer - normal,
    levels = design
)
BS.stat <- BSmooth.fstat(BS.cancer.ex.fit[keepLoci.ex,],
                         design,
                         contrasts)
BS.stat

#------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#
library(HDF5Array)
# See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
hdf5_BS.cancer.ex.fit <- saveHDF5SummarizedExperiment(
    x = BS.cancer.ex.fit[keepLoci.ex, ],
    dir = tempfile())
hdf5_BS.stat <- BSmooth.fstat(hdf5_BS.cancer.ex.fit,
                              design,
                              contrasts)
hdf5_BS.stat
}
```

---

`BSmooth.tstat` *Compute t-statistics based on smoothed whole-genome bisulfite sequencing data.*

---

## Description

Compute t-statistics based on smoothed whole-genome bisulfite sequencing data.

## Usage

```
BSmooth.tstat(BSseq, group1, group2,
  estimate.var = c("same", "paired", "group2"), local.correct = TRUE,
  maxGap = NULL, qSd = 0.75, k = 21, mc.cores = 1, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `BSseq` | An object of class `BSseq`. |

| | |
|---|---|
| group1 | A vector of sample names or indexes for the 'treatment' group. |
| group2 | A vector of sample names or indexes for the 'control' group. |
| estimate.var | How is the variance estimated, see details. |
| local.correct | A logical; should local correction be used, see details. |
| maxGap | A scalar greater than 0, see details. |
| qSd | A scalar between 0 and 1, see details. |
| k | A positive scalar, see details. |
| mc.cores | The number of cores used. Note that setting mc.cores to a value greater than 1 is not supported on MS Windows, see the help page for mclapply. |
| verbose | Should the function be verbose? |

## Details

T-statistics are formed as the difference in means between group 1 and group 2 divided by an estimate of the standard deviation, assuming that the variance in the two groups are the same (same), that we have paired samples (paired) or only estimate the variance based on group 2 (group2). The standard deviation estimates are then smoothed (using a running mean with a width of k) and thresholded (using qSd which sets the minimum standard deviation to be the qSd-quantile). Optionally, the t-statistics are corrected for low-frequency patterns.

It is sometimes useful to use local.correct even if no large scale changes in methylation have been found; it makes the marginal distribution of the t-statistics more symmetric.

Additional details in the reference.

## Value

An object of class BSseqTstat.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## References

KD Hansen, B Langmead, and RA Irizarry. *BSmooth: from whole genome bisulfite sequencing reads to differentially methylated regions*. Genome Biology (2012) 13:R83. doi:10.1186/gb-2012-13-10-r83.

## See Also

BSmooth for the input object and BSseq for its class. BSseqTstat describes the return class. This function is likely to be followed by the use of dmrFinder. And finally, see the package vignette(s) for more information on how to use it.

## Examples

```
if(require(bsseqData)) {
  data(keepLoci.ex)
  data(BS.cancer.ex.fit)
  BS.cancer.ex.fit <- updateObject(BS.cancer.ex.fit)
  ## Remember to subset the BSseq object, see vignette for explanation
  BS.tstat <- BSmooth.tstat(BS.cancer.ex.fit[keepLoci.ex,],
```

```
                            group1 = c("C1", "C2", "C3"),
                            group2 = c("N1", "N2", "N3"),
                            estimate.var = "group2")
  BS.tstat
  ## This object is also stored as BS.cancer.ex.tstat in the
  ## bsseqData package

  #------------------------------------------------------------------------
  # An example using a HDF5Array-backed BSseq object
  #

    library(HDF5Array)
    # See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
    hdf5_BS.cancer.ex.fit <- saveHDF5SummarizedExperiment(
        x = BS.cancer.ex.fit[keepLoci.ex, ],
        dir = tempfile())
    hdf5_BS.tstat <- BSmooth.tstat(hdf5_BS.cancer.ex.fit,
                                   group1 = c("C1", "C2", "C3"),
                                   group2 = c("N1", "N2", "N3"),
                                   estimate.var = "group2")

}
```

---

BSseq *The constructor function for BSseq objects.*

---

## Description

The constructor function for BSseq objects.

## Usage

```
BSseq(M = NULL, Cov = NULL, Filtered = NULL, coef = NULL, se.coef = NULL,
  trans = NULL, parameters = NULL, pData = NULL, gr = NULL,
  pos = NULL, chr = NULL, sampleNames = NULL, rmZeroCov = FALSE)
```

## Arguments

| | |
|---|---|
| M | A matrix-like object of methylation evidence (see 'Details' below). |
| Cov | A matrix-like object of coverage (see 'Details' below)). |
| Filtered | A matrix-like object of ambiguous modification bases obtained from modbam2bed. |
| coef | A matrix-like object of smoothing estimates (see 'Details' below). |
| se.coef | A matrix-like object of smoothing standard errors (see 'Details' below). |
| trans | A smoothing transformation. |
| parameters | A list of smoothing parameters. |
| pData | An data.frame or DataFrame. |
| sampleNames | A vector of sample names. |
| gr | An object of type GRanges. |
| pos | A vector of locations. |
| chr | A vector of chromosomes. |
| rmZeroCov | Should genomic locations with zero coverage in all samples be removed. |

## Details

The 'M', 'Cov', 'coef', and 'se.coef' matrix-like objects will be coerced to DelayedMatrix objects; see DelayedMatrix in the **DelayedArray** package for the full list of supported matrix-like objects. We recommend using matrix objects for in-memory storage of data and HDF5Matrix for on-disk storage of data.

Genomic locations are specified either through `gr` or through `chr` and `pos` but not both. There should be the same number of genomic locations as there are rows in the `M` and `Cov` matrix.

The argument `rmZeroCov` may be useful in order to reduce the size of the return object be removing methylation loci with zero coverage.

In case one or more methylation loci appears multiple times, the `M` and `Cov` matrices are summed over rows linked to the same methylation loci. See the example below.

Users should never have to specify `coef`, `se.coef`, `trans`, and `parameters`, this is for internal use (they are added by `BSmooth`).

`phenoData` is a way to specify pheno data (as known from the `ExpressionSet` and `eSet` classes), at a minimum `sampleNames` should be given (if they are not present, the function uses `col.names(M)`).

## Value

An object of class `BSseq`.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

BSseq

## Examples

```
M <- matrix(0:8, 3, 3)
Cov <- matrix(1:9, 3, 3)
BS1 <- BSseq(chr = c("chr1", "chr2", "chr1"), pos = c(1,2,3),
             M = M, Cov = Cov, sampleNames = c("A","B", "C"))
BS1
BS2 <- BSseq(chr = c("chr1", "chr1", "chr1"), pos = c(1,1,1),
             M = M, Cov = Cov, sampleNames = c("A","B", "C"))
BS2

#------------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#

library(HDF5Array)
hdf5_M <- realize(M, "HDF5Array")
hdf5_Cov <- realize(Cov, "HDF5Array")
hdf5_BS1 <- BSseq(chr = c("chr1", "chr2", "chr1"),
                  pos = c(1, 2, 3),
                  M = hdf5_M,
                  Cov = hdf5_Cov,
                  sampleNames = c("A", "B", "C"))
hdf5_BS1
hdf5_BS2 <- BSseq(chr = c("chr1", "chr1", "chr1"),
```

```
                    pos = c(1, 1, 1),
                    M = hdf5_M,
                    Cov = hdf5_Cov,
                    sampleNames = c("A", "B", "C"))
hdf5_BS2
```

---

`BSseq-class` *Class BSseq*

---

## Description

A class for representing whole-genome or capture bisulfite sequencing data.

## Objects from the Class

An object from the class links together several pieces of information. (1) genomic locations stored as a `GRanges` object, a location by samples matrix of M values, a location by samples matrix of Cov (coverage) values, a location by samples matrix of Filtered (ambiguous modification status) values, and phenodata information.In addition, there are slots for representing smoothed data. This class is an extension of RangedSummarizedExperiment from the **SummarizedExperiment** package.

## Slots

`trans`**:** Object of class `function`. This function transforms the `coef` slot from the scale the smoothing was done to the 0-1 methylation scale.

`parameters`**:** Object of class `list`. A list of parameters representing for example how the data was smoothed.

## Methods

**[** `signature(x = "BSseq")`: Subsetting by location (using integer indices) or sample (using integers or sample names).

**length** Unlike for `RangedSummarizedExperiment`, `length()` is the number of methylation loci (equal to `length(granges(x))`).

**sampleNames,sampleNames<-** Sample names and its replacement function for the object. This is an alias for `colnames`.

**pData,pData<-** Obtain and replace the `pData` slot of the `phenoData` slot. This is an alias for `colData`.

**show** The show method.

**combine** This function combines two `BSSeq` objects. The genomic locations of the new object is the union of the genomic locations of the individual objects. In addition, the methylation data matrices are placed next to each other (as appropriate wrt. the new genomic locations) and zeros are entered into the matrices as needed.

## Utilities

This class extends RangedSummarizedExperiment from the **SummarizedExperiment** package and therefore inherits a number of useful GRanges methods that operate on the rowRanges slot, used for accessing and setting the genomic locations and also do subsetByOverlaps.

There are a number of almost methods-like functions for operating on objects of class BSseq, including getBSseq, collapseBSseq, and orderBSseq. They are detailed below.

`collapseBSseq(BSseq, columns)` is used to collapse an object of class BSseq. By collapsing we simply mean that certain columns (samples) are merge together by summing up the methylation evidence and coverage. This is a useful function if you start by reading in a dataset based on say flowcells and you (after QC) want to simply add a number of flowcells into one sample. The argument columns specify which samples are to be merged, in the following way: it is a character vector of new sample names, and the names of the column vector indicates which samples in the BSseq object are to be collapsed. If columns have the same length as the number of rows of BSseq (and has no names) it is assumed that the ordering corresponds to the sample ordering in BSseq.

`orderBSseq(BSseq, seqOrder = NULL)` simply orders an object of class BSseq according to (increasing) genomic locations. The seqOrder vector is a character vector of seqnames(BSseq) describing the order of the chromosomes. This is useful for ordering chr1 before chr10.

`chrSelectBSseq(BSseq, seqnames = NULL, order = FALSE)` subsets and optionally reorders an object of class BSseq. The seqnames vector is a character vector of seqnames(BSseq) describing which chromosomes should be retained. If order is TRUE, the chromosomes are also re-ordered using orderBSseq.

`getBSseq(BSseq, type = c("Cov", "M", "gr", "coef", "se.coef", "trans", "parameters"))` is a general accessor: is used to obtain a specific slot of an object of class BSseq. It is primarily intended for internal use in the package, for users we recommend granges to get the genomic locations, getCoverage to get the coverage slots and getMeth to get the smoothed values (if they exist).

`hasBeenSmoothed(BSseq)` This function returns a logical depending on whether or not the BSseq object has been smoothed using BSmooth.

`combineList(list, BACKEND = NULL)` This function function is a faster way of using combine on multiple BSseq objects. The input is a list, with each component an object of class BSseq. The (slower) alternative is to use Reduce(combine, list).

The BACKEND argument determines which backend should be used for the 'M' and 'Cov' matrices and, if present, the 'coef' and 'se.coef' matrices (the latter two can only be combined if all objects have the same rowRanges). The default, BACKEND = NULL, corresponds to using matrix objects. See setAutoRealizationBackend (in the **DelayedArray** package) for alternative backends.

`strandCollapse(BSseq, shift = TRUE)` This function operates on a BSseq objects which has stranded loci (i.e. loci where the strand is one of '+' or '-'). It will collapse the methylation and coverage information across the two strands, unstranding the loci in the process and potentially re-ordering them.

The argument shift indicates whether the positions for the loci on the reverse strand should be shifted one (i.e. the positions for these loci are the positions of the 'G' in the 'CpG'; this is the case for Bismark output for example).

## Coercion

Package versions 1.5.2 and 1.11.1 introduced a new version of representing 'BSseq' objects. You can update old serialized (saved) objects by invoking `x <- updateObject(x)`.

### Assays

This class overrides the default implementation of `assays` to make it faster. Per default, no names are added to the returned data matrices.

Assay names can conveniently be obtained by the function `assayNames(x)`

### Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

### See Also

The package vignette. `BSseq` for the constructor function. `RangedSummarizedExperiment` (in the **SummarizedExperiment** package) for the underlying class. `getBSseq`, `getCoverage`, and `getMeth` for accessing the data stored in the object and finally `BSmooth` for smoothing the bisulfite sequence data.

### Examples

```
M <- matrix(1:9, 3,3)
colnames(M) <- c("A1", "A2", "A3")
BStest <- BSseq(pos = 1:3, chr = c("chr1", "chr2", "chr1"), M = M, Cov = M + 2)
chrSelectBSseq(BStest, seqnames = "chr1", order = TRUE)
collapseBSseq(BStest, group = c("A", "A", "B"))

#------------------------------------------------------------------------------
# An example using a HDF5-backed BSseq object
#
hdf5_BStest <- realize(BStest, "HDF5Array")
chrSelectBSseq(hdf5_BStest, seqnames = "chr1", order = TRUE)
collapseBSseq(
    BSseq = hdf5_BStest,
    group = c("A", "A", "B"),
    BACKEND = "HDF5Array",
    type = "integer")
```

---

## BSseqStat-class *Class BSseqStat*

---

### Description

A class for representing statistics for smoothed whole-genome bisulfite sequencing data.

### Usage

```
BSseqStat(gr = NULL, stats = NULL, parameters = NULL)
```

### Arguments

| | |
|---|---|
| `gr` | The genomic locations as an object of class `GRanges`. |
| `stats` | The statistics, as a list of matrix-like objects (see 'Details' below). |
| `parameters` | A list of parameters. |

## Details

The matrix-like elements of the list in the 'stats' slot will be coerced to DelayedMatrix objects; see DelayedMatrix in the **DelayedArray** package for the full list of supported matrix-like objects. We recommend using matrix objects for in-memory storage of data and HDF5Matrix for on-disk storage of data.

## Objects from the Class

Objects can be created by calls of the form BSseqStat(...). However, usually objects are returned by BSmooth.fstat(...) and not constructed by the user.

## Slots

stats: This is a list of DelayedMatrix objects with list elements representing various statistics for methylation loci along the genome.

parameters: Object of class list. A list of parameters representing how the statistics were computed.

gr: Object of class GRanges giving genomic locations.

## Extends

Class hasGRanges, directly.

## Methods

[ The subsetting operator; one may only subset in one dimension, corresponding to methylation loci.

**show** The show method.

## Utilities

This class extends hasGRanges and therefore inherits a number of useful GRanges methods that operate on the gr slot, used for accessing and setting the genomic locations and also do subsetByOverlaps.

## Coercion

Package version 1.11.1 introduced a new version of representing 'BSseqStat' objects. You can update old serialized (saved) objects by invoking x <- updateObject(x).

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

hasGRanges for accessing the genomic locations. BSmooth.fstat for a function that returns objects of class BSseqStat, and smoothSds, computeStat and dmrFinder for functions that operate based on these statistics. Also see the more specialised BSseqTstat.

BSseqTstat-class *Class BSseqTstat*

## Description

A class for representing t-statistics for smoothed whole-genome bisulfite sequencing data.

## Usage

```
BSseqTstat(gr = NULL, stats = NULL, parameters = NULL)
```

## Arguments

| | |
|---|---|
| gr | The genomic locations as an object of class GRanges. |
| stats | The statistics, as a matrix-like object (see 'Details' below). |
| parameters | A list of parameters. |

## Details

The 'stats' matrix-like object will be coerced to a DelayedMatrix objects; see DelayedMatrix in the **DelayedArray** package for the full list of supported matrix-like objects. We recommend using matrix objects for in-memory storage of data and HDF5Matrix for on-disk storage of data.

## Objects from the Class

Objects can be created by calls of the form BSseqTstat(...). However, usually objects are returned by BSmooth.tstat(...) and not constructed by the user..

## Slots

stats: This is a DelayedMatrix object with columns representing various statistics for methylation loci along the genome.

parameters: Object of class list. A list of parameters representing how the t-statistics were computed.

gr: Object of class GRanges giving genomic locations.

## Extends

Class hasGRanges, directly.

## Methods

[ The subsetting operator; one may only subset in one dimension, corresponding to methylation loci.

**show** The show method.

## Utilities

This class extends hasGRanges and therefore inherits a number of useful GRanges methods that operate on the gr slot, used for accessing and setting the genomic locations and also do subsetByOverlaps.

### Coercion

Package version 1.11.1 introduced a new version of representing 'BSseqTstat' objects. You can update old serialized (saved) objects by invoking `x <- updateObject(x)`.

### Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

### See Also

The package vignette(s). `hasGRanges` for accessing the genomic locations. `BSmooth.tstat` for a function that returns objects of class `BSseqTstat`, and `dmrFinder` for a function that computes DMRs based on the t-statistics. Also see `BS.cancer.ex.tstat` for an example of the class in the **bsseqData** package.

### Examples

```
if(require(bsseqData)) {
    data(BS.cancer.ex.tstat)
    dmrFinder(BS.cancer.ex.tstat)
}
```

---

## computeStat — *Compute a test statistic based on smoothed whole-genome bisulfite sequencing data.*

---

### Description

Compute a test statistic based on smoothed whole-genome bisulfite sequencing data.

### Usage

```
computeStat(BSseqStat, coef = NULL)
```

### Arguments

| | |
|---|---|
| `BSseqStat` | An object of class `BSseqStat`, typically an object returned by `smoothSds(...)` and not constructed by the user. |
| `coef` | A vector indicating for which coefficients the statistic is to be computed (`coef = NULL` corresponds to testing all coefficients). If the length of the `coef` is 1 then the corresponding t-statistic is computed, otherwise the corresponding F-statistic is computed. |

### Details

**TODO**

### Value

An object of class BSseqStat. More speciically, the input BSseqStat object with the computed statistics added to the `stats` slot (accessible with `getStats`).

### Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

### See Also

smoothSds for the function to create the appropriate BSseqStat input object. BSseqStat also describes the return class. This function is likely to be followed by the use of dmrFinder.

### Examples

```
if(require(bsseqData)) {
    data(keepLoci.ex)
    data(BS.cancer.ex.fit)
    BS.cancer.ex.fit <- updateObject(BS.cancer.ex.fit)
    ## Remember to subset the BSseq object, see vignette for explanation
    ## TODO: Kind of a forced example
    design <- model.matrix(~0 + BS.cancer.ex.fit$Type)
    colnames(design) <- gsub("BS\\.cancer\\.ex\\.fit\\$Type", "",
                             colnames(design))
    contrasts <- makeContrasts(
        cancer_vs_normal = cancer - normal,
        levels = design
    )
    BS.stat <- BSmooth.fstat(BS.cancer.ex.fit[keepLoci.ex,],
                             design,
                             contrasts)
    BS.stat <- smoothSds(BS.stat)
    BS.stat <- computeStat(BS.stat)
    BS.stat
}
```

---

## data.frame2GRanges *Converts a data frame to a GRanges.*

---

### Description

Converting a data.frame to a GRanges object. The data.frame needs columns like chr, start and end (strand is optional). Additional columns may be kept in the GRanges object.

### Usage

```
data.frame2GRanges(df, keepColumns = FALSE, ignoreStrand = FALSE)
```

### Arguments

| | |
|---|---|
| df | A data.frame with columns chr or seqnames, start, end and optionally a strand column. |
| keepColumns | In case df has additional columns, should these columns be stored as metadata columns on the return GRanges or should they be discarded. |
| ignoreStrand | In case df has a strand column, should this column be ignored. |

## Value

An object of class GRanges

## Note

In case df has rownames, they will be used as names for the return object.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## Examples

```
df <- data.frame(chr = "chr1", start = 1:3, end = 2:4,
                 strand = c("+","-","+"))
data.frame2GRanges(df, ignoreStrand = TRUE)
```

---

# dmrFinder *Finds differentially methylated regions for whole genome bisulfite sequencing data.*

---

## Description

Finds differentially methylated regions for whole genome bisulfite sequencing data. Essentially identifies regions of the genome where all methylation loci have an associated t-statistic that is beyond a (low, high) cutoff.

## Usage

```
dmrFinder(bstat, cutoff = NULL, qcutoff = c(0.025, 0.975),
  maxGap=300, stat = "tstat.corrected", verbose = TRUE)
```

## Arguments

| | |
|---|---|
| bstat | An object of class BSseqStat or BSseqTstat. |
| cutoff | The cutoff of the t-statistics. This should be a vector of length two giving the (low, high) cutoff. If NULL, see qcutoff. |
| qcutoff | In case cutoff is NULL, compute the cutoff using these quantiles of the t-statistic. |
| maxGap | If two methylation loci are separated by this distance, break a possible DMR. This guarantees that the return DMRs have CpGs that are this distance from each other. |
| stat | Which statistic should be used? |
| verbose | Should the function be verbose? |

## Details

The workhorse function is BSmooth.tstat which sets up a t-statistic for a comparison between two groups.

Note that post-processing of these DMRs are likely to be necessary, filtering for example for length (or number of loci).

## Value

A `data.frame` with columns

`start`, `end`, `width`, `chr`
: genomic locations and width.

`n`
: The number of methylation loci.

`invdensity`
: Average length per loci.

`group1.mean`
: The mean methylation level across samples and loci in 'group1'.

`group2.mean`
: The mean methylation level across samples and loci in 'group2'.

`meanDiff`
: The mean difference in methylation level; the difference between `group1.mean` and `group2.mean`.

`idxStart`, `idxEnd`, `cluster`
: Internal use.

`areaStat`
: The 'area' of the t-statistic; equal to the sum of the t-statistics for the individual methylation loci.

`direction`
: either 'hyper' or 'hypo'.

`areaStat.corrected`
: Only present if `column = "tstat.corrected"`, contains the area of the corrected t-statistics.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>.

## References

KD Hansen, B Langmead, and RA Irizarry. *BSmooth: from whole genome bisulfite sequencing reads to differentially methylated regions*. Genome Biology (2012) 13:R83. doi:10.1186/gb-2012-13-10-r83.

## See Also

`BSmooth.tstat` for the function constructing the input object, and `BSseqTstat` for its class. In the example below, we use `BS.cancer.ex.tstat` as the actual input object. Also see the package vignette(s) for a detailed example.

## Examples

```
if(require(bsseqData)) {
  dmrs0 <- dmrFinder(BS.cancer.ex.tstat, cutoff = c(-4.6, 4.6), verbose = TRUE)
  dmrs <- subset(dmrs0, abs(meanDiff) > 0.1 & n >= 3)
}
```

`estimateErrorRate` *Estimate CpG-specific error rate from BSseq object.*

## Description

This function estimates the CpG-specific error rate from a single sample BSseq object generated using `read.bedMethyl`.

## Usage

```
estimateErrorRate(BSseq, minCov = 10, maxCov = 100, minRatio = 0.8, plotErrorProfile = FALSE)
```

## Arguments

| | |
|---|---|
| `BSseq` | A single sample object of class `BSseq`. |
| `minCov` | A non-negative integer specifying the minimum coverage required for CpG loci to be considered. |
| `maxCov` | A non-negative integer specifying the maximum coverage allowed for CpG loci to be considered. |
| `minRatio` | A numeric value between 0 and 1 specifying the minimum ratio of CpG sites to non-CpG sites for a loci to be considered. |
| `plotErrorProfile` | A logical value indicating whether to plot the CpG to non-CpG ratio distribution for the filtered sites. |

## Value

A numeric value representing the estimated CpG-specific error rate the BSseq object.

## Author(s)

Søren Blikdal Hansen (soren.blikdal.hansen@sund.ku.dk)

## See Also

`BSseq` for the `BSseq` class, `read.bedMethyl` for details on reading data into a `BSseq` object.

## Examples

```
# Example input files
infiles <- c(system.file("extdata/HG002_nanopore_test.bedMethyl.gz",
                         package = "bsseq"),
             system.file("extdata/HG002_pacbio_test.bedMethyl.gz",
                         package = "bsseq"))

# Run the function to import data
bsseq <- read.bedMethyl(files = infiles,
                        colData = DataFrame(row.names = c("test_nanopore",
                                                          "test_pacbio")),
                        strandCollapse = TRUE,
                        verbose = TRUE)
```

```
# Estimate error rate
estimateErrorRate(bsseq[, 1], plotErrorProfile = FALSE)
```

---

findLoci *Find methylation loci in a genome*

---

## Description

This is a convenience function to find methylation loci, such as CpGs, in a reference genome. The result is useful as the value of the `loci` argument for `read.bismark()`.

## Usage

```
findLoci(pattern,
         subject,
         include = seqlevels(subject),
         strand = c("*", "+", "-"),
         fixed = "subject",
         resize = TRUE)
```

## Arguments

| | |
|---|---|
| `pattern` | A string specifying the pattern to search for, e.g. `"CG"`. Can contain IUPAC ambiguity codes, e.g., `"CH"`. |
| `subject` | A string containing a file path to the genome sequence, in FASTA or 2bit format, to be searched. Alternatively, a `BSgenome` or `DNAStringSet` object storing the genome sequence to be searched. |
| `include` | A character vector indicating the seqlevels of `subject` to be used. |
| `strand` | A character scaler specifying the strand of `subject` to be used. If `strand = "*"`, then both the positive (`strand = "+"`) and negative (`strand = "-"` strands will be searched.) It is assumed that `subject` contains the sequence with respect to the + strand. |
| `fixed` | If `"subject"` (the default), IUPAC ambiguity codes in the pattern only are interpreted as wildcards, e.g., a `pattern` containing `CH` matches a `subject` containing `CA` but not vice versa. See `?Biostrings::` `lowlevel-matching` for more information |
| `resize` | A logical scalar specifying whether the ranges should be shifted to have width 1 and anchored by the start of the locus, e.g., resize a CpG dinucleotide to give the co-ordinates of the cytosine. |

## Details

This function provides a convenience wrapper for finding methylation loci in a genome, based on running `vmatchPattern()`. Users requiring finer-grained control should directly use the `vmatchPattern()` function and coerce the result to a `GRanges` object.

## Value

A `GRanges` object storing the found loci.

### Author(s)

Peter F. Hickey

### See Also

- Biostrings::vmatchPattern()
- ?BSgenome::`BSgenome-utils`

### Examples

```
library(Biostrings)
# Find CpG dinucleotides on the both strands of an artificial sequence
my_seq <- DNAStringSet("ATCAGTCGC")
names(my_seq) <- "test"
findLoci(pattern = "CG", subject = my_seq)
# Find CHG trinucleotides on the both strands of an artificial sequence
findLoci(pattern = "CHG", subject = my_seq)

# Find CpG dinucleotides on the + strand of chr17 from the human genome (hg38)
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg38")) {
  findLoci(
      pattern = "CG",
      subject = BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38,
      include = "chr17",
      strand = "+")
}
```

---

## fisherTests *Compute Fisher-tests for a BSseq object*

---

### Description

A function to compute Fisher-tests for an object of class BSseq.

### Usage

```
fisherTests(BSseq, group1, group2, lookup = NULL,
  returnLookup = TRUE, mc.cores = 1, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| group1 | A vector of sample names or indexes for the 'treatment' group. |
| group2 | A vector of sample names or indexes for the 'control' group. |
| lookup | A 'lookup' object, see details. |
| returnLookup | Should a 'lookup' object be returned, see details. |
| mc.cores | The number of cores used. Note that setting mc.cores to a value greater than 1 is not supported on MS Windows, see the help page for mclapply. |
| verbose | Should the function be verbose. |

## Details

This function computes row-wise Fisher's exact tests. It uses an internal lookup table so rows which forms equivalent 2x2 tables are group together and only a single test is computed. If `returnLookup` is `TRUE` the return object contains the lookup table which may be feed to another call to the function using the `lookup` argument.

If `group1`, `group2` designates more than 1 sample, the samples are added together before testing.

This function can use multiple cores on the same computer.

This test cannot model biological variability.

## Value

if `returnLookup` is `TRUE`, a list with components `results` and `lookup`, otherwise just the `results` component. The `results` (component) is a matrix with the same number of rows as the `BSseq` argument and 2 columns `p.value` (the unadjusted p-values) and `log2OR` (log2 transformation of the odds ratio).

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

fisher.test for information about Fisher's test. `mclapply` for the `mc.cores` argument.

## Examples

```
M <- matrix(1:9, 3,3)
colnames(M) <- c("A1", "A2", "A3")
BStest <- BSseq(pos = 1:3, chr = c("chr1", "chr2", "chr1"),
                M = M, Cov = M + 2)
results <- fisherTests(BStest, group1 = "A1", group2 = "A2",
                       returnLookup = TRUE)
results

#------------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#
library(HDF5Array)
# See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
hdf5_BStest <- saveHDF5SummarizedExperiment(x = BStest,
                                            dir = tempfile())
results <- fisherTests(hdf5_BStest,
                       group1 = "A1",
                       group2 = "A2",
                       returnLookup = TRUE)
results
```

## FWGRanges-class *Classes FWIRanges and FWGRanges*

### Description

Classes for fixed-width IRanges and GRanges, ie. objects where all ranges have the same width. The intention is for these classes to be added to GenomicRanges. Documented here temporarily.

### Details

See description. Otherwise works like IRanges and GRanges, except there are many unimplemented methods.

This is really a private class, with private methods, but R's NAMESPACE handling means they get unintentionally exported. Hence this documentation.

### Examples

```
showClass("FWIRanges")
```

## getCoverage *Obtain coverage for BSseq objects.*

### Description

Obtain coverage for BSseq objects.

### Usage

```
getCoverage(BSseq, regions = NULL, type = c("Cov", "M"),
  what = c("perBase", "perRegionAverage", "perRegionTotal"),
  withDimnames = TRUE)
```

### Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| regions | An optional data.frame or GenomicRanges object specifying a number of genomic regions. |
| type | This returns either coverage or the total evidence for methylation at the loci. |
| what | The type of return object, see details. |
| withDimnames | A logical(1), indicating whether dimnames should be applied to extracted coverage elements. Setting withDimnames = FALSE increases the speed and memory efficiency with which coverage is extracted. |

## Value

**NOTE:** The return type of `getCoverage` varies depending on its arguments.

If `regions` are not specified (`regions = NULL`) a `DelayedMatrix` object (`what = "perBase"`) is returned. This will either contain the per-base coverage, the average coverage, or the genome total coverage (depending on value of `what`).

If `what = "perBase"` and `regions` are specified, a list is returned. Each element of the list is a `DelayedMatrix` object corresponding to the genomic loci inside the region. It is conceptually the same as splitting the coverage by region.

If `what = "perRegionAverage"` or `what = "perRegionTotal"` and `regions` are specified the return value is a `DelayedMatrix` object. Each row of the `DelayedMatrix` corresponds to a region and contains either the average coverage or the total coverage in the region.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>.

## See Also

`BSseq` for the `BSseq` class.

## Examples

```
data(BS.chr22)
head(getCoverage(BS.chr22, type = "M"))
reg <- GRanges(seqnames = c("chr22", "chr22"),
  ranges = IRanges(start = c(1, 2*10^7), end = c(2*10^7 +1, 4*10^7)))
getCoverage(BS.chr22, regions = reg, what = "perRegionAverage")
  cList <- getCoverage(BS.chr22, regions = reg)
length(cList)
head(cList[[1]])

#------------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#

library(HDF5Array)
# See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
hdf5_BS.chr22 <- saveHDF5SummarizedExperiment(x = BS.chr22,
                                              dir = tempfile())
head(getCoverage(hdf5_BS.chr22, type = "M"))
reg <- GRanges(seqnames = c("chr22", "chr22"),
               ranges = IRanges(start = c(1, 2 * 10 ^ 7),
               end = c(2 * 10 ^ 7 + 1, 4 * 10 ^ 7)))
getCoverage(hdf5_BS.chr22, regions = reg, what = "perRegionAverage")
hdf5_cList <- getCoverage(hdf5_BS.chr22, regions = reg)
length(hdf5_cList)
head(hdf5_cList[[1]])
```

getCpGMatrix *Generate a matrix of the most likely CpG status for a multi-sample BSseq object.*

## Description

This function generates a matrix of the most likely CpG status for all loci and samples in a BSseq object. Each element of the matrix represents the most likely CpG status (0 for homozygous CpG, 1 for heterozygous CpG, and 2 for non-CpG or if allCpG = TRUE 0 for homozygous or heterozygous CpG, and 2 for non-CpG) for a specific locus and sample.

## Usage

```
getCpGMatrix(BSseq, e = NULL, allCpG = FALSE)
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| e | An optional numeric vector representing error rates for each sample. If NULL, the error rate for each sample is estimated using estimateErrorRate. |
| allCpG | A logical value indicating whether to classify loci as allCpG (i.e. combine homozygous or heterozygous CpG) and non-CpG based on their likelihoods. Should be the same for getCpGMatrix and getMaxLikelihoodMatrix |

## Value

A numeric matrix where each row represents a locus, and each column represents a sample, and the values correspond to the CpG status (same order as the BSseq object in input).

## Author(s)

Søren Blikdal Hansen (soren.blikdal.hansen@sund.ku.dk)

## See Also

BSseq for the BSseq class, read.bedMethyl for details on reading data into a BSseq object, estimateErrorRate for estimating the CpG-specific error rate. getCpGs for filtering a single-sample BSseg object. getMaxLikelihoodMatrix for generating a matrix with the maximum scaled likelihoods matching the CpGMatrix.

## Examples

```
# Example input files
infiles <- c(system.file("extdata/HG002_nanopore_test.bedMethyl.gz",
                         package = "bsseq"),
             system.file("extdata/HG002_pacbio_test.bedMethyl.gz",
                         package = "bsseq"))

# Run the function to import data
bsseq <- read.bedMethyl(files = infiles,
                        colData = DataFrame(row.names = c("test_nanopore",
                                                          "test_pacbio")),
```

```
                        strandCollapse = TRUE,
                        verbose = TRUE)

# Single samples can be filtered using the getCpGs function
bsseq_nano <- bsseq[, 1]
bsseq_nano_99All_filtered <- bsseq[getCpGs(bsseq_nano,
                                           type = "allCpG", threshold = 0.99)]

bsseq_pacbio <- bsseq[, 2]
bsseq_pacbio_99All_filtered <- bsseq[getCpGs(bsseq_pacbio,
                                             type = "allCpG", threshold = 0.99)]

# For filtering multiple samples, we can use a CpGMatrix and a MaxLikelihoodMatrix
# Construct the CpGMatrix and getMaxLikelihoodMatrix for the bsseq object
CpGMatrix <- getCpGMatrix(bsseq, allCpG = TRUE)
MaxLikelihoodMatrix <- getMaxLikelihoodMatrix(bsseq, allCpG = TRUE)

# Filter for allCpG loci with a likelihood > 0.99 in both samples
bsseq_combined_99All_filtered <- bsseq[which(rowAlls(CpGMatrix == 0)
                                         & rowMins(MaxLikelihoodMatrix) > 0.99)]
```

---

`getCpGs` *Get CpG (and non-CpG) loci from a single sample BSseq object.*

---

## Description

This function identifies CpG (and non-CpG) loci from a single sample BSseq object using scaled likelihoods computed from the count of CpG sites and Non-CpG sites mapped to the loci, based on the specified type, and minimum scaled likelihood (threshold). The function only work for BSseq objects generated using read.bedMethyl.

## Usage

```
getCpGs(BSseq, type = c("homozygous", "heterozygous", "allCpG", "nonCpG"), threshold = 0.99, e = NUL
```

## Arguments

| | |
|---|---|
| `BSseq` | An single sample object of class `BSseq`. |
| `type` | A character string specifying the type of loci to extract. Must be one of `"homozygous"`, `"heterozygous"`, `"allCpG"`, or `"nonCpG"`. |
| `threshold` | A numeric value between 0 and 1 specifying the minimum likelihood threshold required for loci to be included. |
| `e` | An optional numeric value representing the error rate. If `NULL`, the error rate is estimated using `estimateErrorRate`. |

## Value

An integer vector of indices representing the loci that match the criteria.

## Author(s)

Søren Blikdal Hansen (soren.blikdal.hansen@sund.ku.dk)

## See Also

BSseq for the BSseq class, read.bedMethyl for details on reading data into a BSseq object, estimateErrorRate for estimating the CpG-specific error rate.

## Examples

```
# Example input files
infiles <- c(system.file("extdata/HG002_nanopore_test.bedMethyl.gz",
                         package = "bsseq"),
             system.file("extdata/HG002_pacbio_test.bedMethyl.gz",
                         package = "bsseq"))

# Run the function to import data
bsseq <- read.bedMethyl(files = infiles,
                        colData = DataFrame(row.names = c("test_nanopore",
                                                          "test_pacbio")),
                        strandCollapse = TRUE,
                        verbose = TRUE)

# Filter CpG sites for the Nanopore dataset
bsseq_nano <- bsseq[, 1]
bsseq_nano_99All_filtered <- bsseq[getCpGs(bsseq_nano,
                                           type = "allCpG", threshold = 0.99)]

# Filter CpG sites for the PacBio dataset
bsseq_pacbio <- bsseq[, 2]
bsseq_pacbio_99All_filtered <- bsseq[getCpGs(bsseq_pacbio,
                                             type = "allCpG", threshold = 0.99)]
```

---

# getMaxLikelihoodMatrix

*Generate a matrix of the scaled likelihood of most likely CpG status for a multi-sample BSseq object.*

---

## Description

This function generates a matrix of the scaled likelihoods for most likely CpG status for a multi-sample BSseq object. Each element of the matrix represents the scaled likelihood of the most likely CpG status for the locus in the sample. If no data is available for a locus in a sample, the entry in the CpGMatrix is 2 (nonCpG) and the corresponding MaxLikelihood is 1/3.

## Usage

```
getMaxLikelihoodMatrix(BSseq, e = NULL, allCpG = FALSE)
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| e | An optional numeric vector representing error rates for each sample. If NULL, the error rate for each sample is estimated using estimateErrorRate. |
| allCpG | A logical value indicating whether to classify loci as allCpG and non-CpG loci and sum the scaled likelihood of homozygous CpG and heterozygous CpG. Should be the same for getMaxLikelihoodMatrix and getCpGMatrix |

## Value

A numeric matrix where each row represents a locus, each column represents a sample, and the values correspond to the quality scores.

## Author(s)

Søren Blikdal Hansen (soren.blikdal.hansen@sund.ku.dk)

## See Also

BSseq for the BSseq class, read.bedMethyl for details on reading data into a BSseq object, estimateErrorRate for estimating the CpG-specific error rate. getCpGs for filtering a single-sample BSseg object. getCpGMatrix for generating a matrix with the most likely CpG status matching the getMaxLikelihoodMatrix.

## Examples

```
# Example input files
infiles <- c(system.file("extdata/HG002_nanopore_test.bedMethyl.gz",
                         package = "bsseq"),
             system.file("extdata/HG002_pacbio_test.bedMethyl.gz",
                         package = "bsseq"))

# Run the function to import data
bsseq <- read.bedMethyl(files = infiles,
                        colData = DataFrame(row.names = c("test_nanopore",
                                                          "test_pacbio")),
                        strandCollapse = TRUE,
                        verbose = TRUE)

# Single samples can be filtered using the getCpGs function
bsseq_nano <- bsseq[, 1]
bsseq_nano_99All_filtered <- bsseq[getCpGs(bsseq_nano,
                                           type = "allCpG", threshold = 0.99)]

bsseq_pacbio <- bsseq[, 2]
bsseq_pacbio_99All_filtered <- bsseq[getCpGs(bsseq_pacbio,
                                             type = "allCpG", threshold = 0.99)]

# For filtering multiple samples, we can use a CpGMatrix and a MaxLikelihoodMatrix
# Construct the CpGMatrix and QualityMatrix for the bsseq object
CpGMatrix <- getCpGMatrix(bsseq, allCpG = TRUE)
MaxLikelihoodMatrix <- getMaxLikelihoodMatrix(bsseq, allCpG = TRUE)

# Filter for allCpG loci with a likelihood > 0.99 in both samples
bsseq_combined_99All_filtered <- bsseq[which(rowAlls(CpGMatrix == 0)
                                          & rowMins(MaxLikelihoodMatrix) > 0.99)]
```

---

getMeth *Obtain methylation estimates for BSseq objects.*

---

## Description

Obtain methylation estimates for BSseq objects, both smoothed and raw.

## Usage

```
getMeth(BSseq, regions = NULL, type = c("smooth", "raw"),
  what = c("perBase", "perRegion"), confint = FALSE, alpha = 0.95,
  withDimnames = TRUE)
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| regions | An optional data.frame or GenomicRanges object specifying a number of genomic regions. |
| type | This returns either smoothed or raw estimates of the methylation level. |
| what | The type of return object, see details. |
| confint | Should a confidence interval be return for the methylation estimates (see below). This is only supported if what is equal to perBase. |
| alpha | alpha value for the confidence interval. |
| withDimnames | A logical(1), indicating whether dimnames should be applied to extracted coverage elements. Setting withDimnames = FALSE increases the speed and memory efficiency with which coverage is extracted. |

## Value

**NOTE:** The return type of getMeth varies depending on its arguments.

If region = NULL the what argument is ignored. This is also the only situation in which confint = TRUE is supported. The return value is either a DelayedMatrix (confint = FALSE or a list with three DelayedMatrix components confint = TRUE (meth, upper and lower), giving the methylation estimates and (optionally) confidence intervals.

Confidence intervals for type = "smooth" is based on standard errors from the smoothing algorithm (if present). Otherwise it is based on pointwise confidence intervals for binomial distributions described in Agresti (see below), specifically the score confidence interval.

If regions are specified, what = "perBase" will make the function return a list, each element of the list being a DelayedMatrix corresponding to a genomic region (and each row of the DelayedMatrix being a loci inside the region). If what = "perRegion" the function returns a DelayedMatrix, with each row corresponding to a region and containing the average methylation level in that region.

## Note

A BSseq object needs to be smoothed by the function BSmooth in order to support type = "smooth".

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>.

## References

A Agresti and B Coull. *Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions*. The American Statistician (1998) 52:119-126.

## See Also

BSseq for the BSseq class and BSmooth for smoothing such an object.

## Examples

```
data(BS.chr22)
head(getMeth(BS.chr22, type = "raw"))
reg <- GRanges(seqnames = c("chr22", "chr22"),
  ranges = IRanges(start = c(1, 2*10^7), end = c(2*10^7 +1, 4*10^7)))
head(getMeth(BS.chr22, regions = reg, type = "raw", what = "perBase"))

#------------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#

library(HDF5Array)
# See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
hdf5_BS.chr22 <- saveHDF5SummarizedExperiment(x = BS.chr22,
                                              dir = tempfile())
head(getMeth(hdf5_BS.chr22, type = "raw"))
head(getMeth(hdf5_BS.chr22, regions = reg, type = "raw", what = "perBase"))
```

---

# getStats *Obtain statistics from a BSseqTstat object*

---

## Description

Essentially an accessor function for the statistics of a BSseqTstat object.

## Usage

```
getStats(bstat, regions = NULL, ...)
```

## Arguments

| | |
|---|---|
| bstat | An object of class BSseqStat or BSseqTstat. |
| regions | An optional data.frame or GenomicRanges object specifying a number of genomic regions. |
| ... | Additional arguments passed to the different backends based on the class of bstat; see Details. |

## Details

Additional argument when the bstat object is of class BSseqTstat:

**stat** Which statistics column should be obtained.

## Value

An object of class data.frame possible restricted to the regions specified.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

BSseqTstat for the BSseqTstat class, and getCoverage and getMeth for similar functions, operating on objects of class BSseq.

## Examples

```
if(require(bsseqData)) {
  data(BS.cancer.ex.tstat)
  head(getStats(BS.cancer.ex.tstat))
  reg <- GRanges(seqnames = c("chr22", "chr22"),
     ranges = IRanges(start = c(1, 2*10^7), end = c(2*10^7 +1, 4*10^7)))
  head(getStats(BS.cancer.ex.tstat, regions = reg))
}
```

---

# GoodnessOfFit *Binomial and poisson goodness of fit statistics for BSSeq objects*

---

## Description

Binomial and poisson goodness of fit statistics for BSSeq objects, including plotting capability.

## Usage

```
poissonGoodnessOfFit(BSseq, nQuantiles = 10^5)
binomialGoodnessOfFit(BSseq, method = c("MLE"), nQuantiles = 10^5)
## S3 method for class 'chisqGoodnessOfFit'
print(x, ...)
## S3 method for class 'chisqGoodnessOfFit'
plot(x, type = c("chisq", "pvalue"), plotCol = TRUE, qqline = TRUE,
  pch = 16, cex = 0.75, ...)
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| x | A chisqGoodnessOfFit object (as produced by poissonGoodnessOfFit or binomialGoodnessOfFit |
| nQuantiles | The number of (evenly-spaced) quantiles stored in the return object. |
| method | How is the parameter estimated. |
| type | Are the chisq or the p-values being plotted. |
| plotCol | Should the extreme quantiles be colored. |
| qqline | Add a qqline. |
| pch, cex | Plotting symbols and size. |
| ... | Additional arguments being passed to qqplot (for plot) or ignored (for print). |

## Details

These functions compute and plot goodness of fit statistics for BSseq objects. For each methylation loci, the Poisson goodness of fit statistic tests whether the coverage (at that loci) is independent and identically Poisson distributed across the samples. In a similar fashion, the Binomial goodness of fit statistic tests whether the number of reads supporting methylation are independent and identically binomial distributed across samples (with different size parameters given by the coverage vector).

These functions do not handle NA values.

## Value

The plotting method is invoked for its side effect. Both `poissonGoodnessOfFit` and `binomialGoodnessOfFit` returns an object of class `chisqGoodnessOfFit` which is a list with components

| | |
|---|---|
| `chisq` | a vector of Chisq values. |
| `quantiles` | a vector of quantiles (of the chisq values). |
| `df` | degress of freedom |

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

For the plotting method, see `qqplot`.

## Examples

```
if(require(bsseqData)) {
  data(BS.cancer.ex)
  BS.cancer.ex <- updateObject(BS.cancer.ex)
  gof <- poissonGoodnessOfFit(BS.cancer.ex)
  plot(gof)

#------------------------------------------------------------------------------
# An example using a HDF5Array-backed BSseq object
#

    library(HDF5Array)

    # See ?SummarizedExperiment::saveHDF5SummarizedExperiment for details
    hdf5_BS.cancer.ex <- saveHDF5SummarizedExperiment(x = BS.cancer.ex,
                                                      dir = tempfile())
    hdf5_gof <- poissonGoodnessOfFit(hdf5_BS.cancer.ex)
    plot(hdf5_gof)

}
```

---

# hasGRanges-class *Class hasGRanges*

---

## Description

A class with a GRanges slot, used as a building block for other classes. Provides basic accessor functions etc.

## Objects from the Class

Objects can be created by calls of the form `new("hasGRanges", ...)`.

## Slots

**`gr`:** Object of class `GRanges`.

### Methods

**"["** Subsets a single dimension.

**granges** Get the GRanges object representing genomic locations.

**start,start<-,end,end<-,width,width<-** Start, end and width for the genomic locations of the object, also replacement functions. This accessor functions operate directly on the gr slot.

**strand,strand<-** Getting and setting the strand of the genomic locations (the gr slot).

**seqlengths,seqlengths<-** Getting and setting the seqlengths of the genomic locations (the gr slot).

**seqlevels,seqlevels<-** Getting and setting the seqlevels of the genomic locations (the gr slot).

**seqnames,seqnames<-** Getting and setting the seqnames of the genomic locations (the gr slot).

**show** The show method.

**findOverlaps** (query = "hasGRanges", subject = "hasGRanges"): finds overlaps between the granges() of the two objects.

**findOverlaps** (query = "GenomicRanges", subject = "hasGRanges"): finds overlaps between query and the granges() of the subject.

**findOverlaps** (query = "hasGRanges", subject = "GenomicRanges"): finds overlaps between the granges() of the query and the subject.

**subsetByOverlaps** (query = "hasGRanges", subject = "hasGRanges"): Subset the query, keeping the genomic locations that overlaps the subject.

**subsetByOverlaps** (query = "hasGRanges", subject = "GenomicRanges"): Subset the query, keeping the genomic locations that overlaps the subject.

**subsetByOverlaps** (query = "GenomicRanges", subject = "hasGRanges"): Subset the query, keeping the genomic locations that overlaps the subject.

### Note

If you extend the hasGRanges class, you should consider writing a subset method ([), and a show method. If the new class supports single index subsetting, the subsetByOverlaps methods will automatically extend.

### Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

### Examples

```
showClass("hasGRanges")
```

---

## Internals *Internals*

---

### Description

The matrixOrNULL class is a class union, representing a slot which is either a matrix or NULL (representing missing information).

### Value

Undocumented

plotRegion *Plotting BSmooth methylation estimates*

## Description

Functions for plotting BSmooth methylation estimates. Typically used to display differentially methylated regions.

## Usage

```
plotRegion(BSseq, region = NULL, extend = 0, main = "",
  addRegions = NULL, annoTrack = NULL, cex.anno = 1,
  geneTrack = NULL, cex.gene = 1.5, col = NULL, lty = NULL,
  lwd = NULL, BSseqStat = NULL, stat = "tstat.corrected",
  stat.col = "black", stat.lwd = 1, stat.lty = 1, stat.ylim = c(-8, 8),
  mainWithWidth = TRUE, regionCol = alpha("red", 0.1), addTicks = TRUE,
  addPoints = FALSE, pointsMinCov = 5, highlightMain = FALSE)

plotManyRegions(BSseq, regions = NULL, extend = 0, main = "",
  addRegions = NULL, annoTrack = NULL, cex.anno = 1,
  geneTrack = NULL, cex.gene = 1.5, col = NULL, lty = NULL,
  lwd = NULL, BSseqStat = NULL, stat = "tstat.corrected",
  stat.col = "black", stat.lwd = 1, stat.lty = 1, stat.ylim = c(-8, 8),
  mainWithWidth = TRUE, regionCol = alpha("red", 0.1), addTicks = TRUE,
  addPoints = FALSE, pointsMinCov = 5, highlightMain = FALSE,
  verbose = TRUE)
```

## Arguments

| | |
|---|---|
| BSseq | An object of class BSseq. |
| region | A data.frame (with start, end and chr columns) with 1 row or GRanges of length 1. If region is NULL the entire BSseq argument is plotted. |
| regions | A data.frame (with start, end and chr columns) or GRanges. |
| extend | Describes how much the plotting region should be extended in either direction. The total width of the plot is equal to the width of the region plus twice extend. |
| main | The plot title. The default is to construct a title with information about which genomic region is being plotted. |
| addRegions | A set of additional regions to be highlighted on the plots. As the regions argument. |
| annoTrack | A named list of GRanges objects. Each component is a track and the names of the list are the track names. Each track will be plotted as solid bars, and we routinely display information such as CpG islands, exons, etc. |
| cex.anno | cex argument when plotting annoTrack. |
| geneTrack | **EXPERIMENTAL**: A data.frame with columns: chr, start, end, gene_ID, exon_number, strand, gene_name, isoforms. This interface is under active development and subject to change. |
| cex.gene | cex argument when plotting geneTrack. |
| col | The color of the methylation estimates, see details. |

| | |
|---|---|
| lty | The line type of the methylation estimates, see details. |
| lwd | The line width of the methylation estimates, see details. |
| BSseqStat | An object of class BSseqStat. If present, a new panel will be shown with the t-statistics. |
| stat | Which statistics will be plotted (only used is BSseqStat is not NULL.) |
| stat.col | color for the statistics plot. |
| stat.lwd | line width for the statistics plot. |
| stat.lty | line type for the statistics plot. |
| stat.ylim | y-limits for the statistics plot. |
| mainWithWidth | Should the default title include information about width of the plot region. |
| regionCol | The color used for highlighting the region. |
| addTicks | Should tick marks showing the location of methylation loci, be added? |
| addPoints | Should the individual unsmoothed methylation estimates be plotted. This usually leads to a very confusing plot, but may be useful for diagnostic purposes. |
| pointsMinCov | The minimum coverage a methylation loci need in order for the raw methylation estimates to be plotted. Useful for filtering out low coverage loci. Only used if addPoints = TRUE. |
| highlightMain | Should the plot region be highlighted? |
| verbose | Should the function be verbose? |

## Details

The correct choice of aspect ratio depends on the width of the plotting region. We tend to use width = 10, height = 5.

plotManyRegions is used to plot many regions (hundreds or thousands), and is substantially quicker than repeated calls to plotRegion.

This function has grown to be rather complicated over time. For custom plotting, it is sometimes useful to use the function definition as a skeleton and directly modify the code.

## Value

This function is invoked for its side effect: producing a plot.

## Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

## See Also

The package vignette has an extended example.

`read.bedMethyl` *Parsing bedMethyl output from modkit pileup.*

## Description

Parsing bedMethyl output from modkit pileup.

## Usage

```
read.bedMethyl(files,
               loci = NULL,
               colData = NULL,
               rmZeroCov = TRUE,
               strandCollapse = TRUE,
               BPPARAM = bpparam(),
               BACKEND = NULL,
               dir = tempfile("BSseq"),
               replace = FALSE,
               chunkdim = NULL,
               level = NULL,
               nThread = 1L,
               verbose = getOption("verbose"))
```

## Arguments

| | |
|---|---|
| `files` | The path to the files created by running modkit pileup, one sample per file. See the methods section of [link to preprint] for validated output. |
| `loci` | `NULL` (default) or a `GenomicRanges` instance containing methylation loci (all with width equal to 1). If `loci = NULL`, then `read.bedMethyl()` will perform a first pass over the bedMethyl files to identify candidate loci. If `loci` is a `GenomicRanges` instance, then these form the candidate loci. The candidate loci will be collapsed if `strandCollapse = TRUE`. |
| `colData` | An optional `DataFrame` describing the samples. Row names, if present, become the column names of the BSseq object. If `NULL`, then a `DataFrame` will be created with `files` used as the row names. |
| `rmZeroCov` | A `logical(1)` indicating whether methylation loci that have zero coverage in all samples should be removed. Default setting is `rmZeroCov = TRUE` |
| `strandCollapse` | A `logical(1)` indicating whether strand-symmetric methylation loci (i.e. CpGs) should be collapsed across strands. |
| `BPPARAM` | An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation. |
| `BACKEND` | `NULL` or a single string specifying the name of the realization backend. Currently, the backend is not supported for downstream applications. |
| `dir` | **Only applicable if** `BACKEND == "HDF5Array"`**.** The path (as a single string) to the directory where to save the HDF5-based BSseq object. |
| `replace` | **Only applicable if** `BACKEND == "HDF5Array"`**.** If the directory `dir` already exists, should it be replaced with a new one? |
| `chunkdim` | **Only applicable if** `BACKEND == "HDF5Array"`**.** The dimensions of the chunks to use for writing the data to disk. |

| | |
|---|---|
| level | The compression level to use for writing the data to disk. |
| nThread | The number of threads used by fread when reading the files. |
| verbose | A logical(1) indicating whether progress messages should be printed (default TRUE). |

## File formats

The format of each file should be similar to the examples in [link to preprint]. Files ending in .gz, .bz2, .xz, or .zip will be automatically decompressed to tempdir().

**Supported file formats:** Modkit bedMethyl files from modkit pileup. For downstream likelihood functions we recommend running modkit pileup on output from bam files modification/basecalled using a CG context model and not using a reference genome for pileup.

**Unsupported file formats:** Other types of output.

**One-based vs. zero-based genomic co-ordinates:** The genomic co-ordinates of bedMethyl files are zero-based. Since Bioconductor packages typically use one-based co-ordinates, the co-ordinates from the bedMethyl files are converted to one-based in the BSseq object.

## Author(s)

Søren Blikdal Hansen (soren.blikdal.hansen@sund.ku.dk)

## Examples

```
# Example: Reading bedMethyl files included in the bsseq package
# Paths to example bedMethyl files in the package's extdata directory
infiles <- c(system.file("extdata/HG002_nanopore_test.bedMethyl.gz",
                         package = "bsseq"),
             system.file("extdata/HG002_pacbio_test.bedMethyl.gz",
                         package = "bsseq"))

# Run the function to import data
bsseq <- read.bedMethyl(files = infiles,
                        colData = DataFrame(row.names = c("test_nanopore",
                                                          "test_pacbio")),
                        rmZeroCov = FALSE,
                        strandCollapse = TRUE,
                        verbose = TRUE)

# View the resulting BSseq object
bsseq
```

---

# read.bismark *Parsing output from the Bismark alignment suite.*

---

## Description

Parsing output from the Bismark alignment suite.

## Usage

```
read.bismark(files,
             loci = NULL,
             colData = NULL,
             rmZeroCov = FALSE,
             strandCollapse = TRUE,
             BPPARAM = bpparam(),
             BACKEND = NULL,
             dir = tempfile("BSseq"),
             replace = FALSE,
             chunkdim = NULL,
             level = NULL,
             nThread = 1L,
             verbose = getOption("verbose"))
```

## Arguments

| | |
|---|---|
| files | The path to the files created by running Bismark's methylation extractor, one sample per file. Files ending in .gz or .bz2 will be automatically decompressed to tempfile(). We strongly recommend you use the 'genome wide cytosine report' output files. See section 'File formats' for further details. |
| loci | NULL (default) or a GenomicRanges instance containing methylation loci (all with width equal to 1). If loci = NULL, then read.bismark() will perform a first pass over the Bismark file to identify candidate loci. If loci is a GenomicRanges instance, then these form the candidate loci. In either case, the candidate loci will be filtered if rmZeroCov = TRUE and collapsed if strandCollapse = TRUE to form the final set of methylation loci that form the rowRanges of the returned BSseq object. See section 'Efficient use of read.bismark()' for further details. |
| colData | An optional DataFrame describing the samples. Row names, if present, become the column names of the BSseq object. If NULL, then a DataFrame will be created with files used as the row names. |
| rmZeroCov | A logical(1) indicating whether methylation loci that have zero coverage in all samples be removed. For several reasons, the default rmZeroCov = FALSE is recommended even in cases where you ultimately want to remove such loci. See section 'Efficient use of read.bismark()' for further details. |
| strandCollapse | A logical(1) indicating whether strand-symmetric methylation loci (i.e. CpGs) should be collapsed across strands. This is only applicable for stranded methylation loci, e.g., loci extracted from 'genome wide cytosine reports' (see section 'File formats' for further details). |
| BPPARAM | An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation. Currently supported are SerialParam (Unix, Mac, Windows), MulticoreParam (Unix and Mac), SnowParam (Unix, Mac, and Windows, limited to single-machine clusters), and BatchtoolsParam (Unix, Mac, Windows, only with the in-memory realization backend). See sections 'Parallelization and progress monitoring' and 'Realization backends' for further details. |
| BACKEND | NULL or a single string specifying the name of the realization backend. When the backend is set to NULL, the M and Cov assays are realized in memory as ordinary matrices, otherwise these are realized with the given BACKEND. See section 'Realization backends' for further details. |

| | |
|---|---|
| dir | **Only applicable if** BACKEND == "HDF5Array". The path (as a single string) to the directory where to save the HDF5-based BSseq object. The directory will be created so should not already exist, unless replace is set to TRUE. |
| replace | **Only applicable if** BACKEND == "HDF5Array". If directory dir already exists, should it be replaced with a new one? The content of the existing directory will be lost! |
| chunkdim | **Only applicable if** BACKEND == "HDF5Array". The dimensions of the chunks to use for writing the data to disk. By default, getHDF5DumpChunkDim() using the dimensions of the returned BSseq object will be used. See ?getHDF5DumpChunkDim for more information. |
| level | **Only applicable if** BACKEND == "HDF5Array". The compression level to use for writing the data to disk. By default, getHDF5DumpCompressionLevel() will be used. See ?getHDF5DumpCompressionLevel for more information. |
| nThread | The number of threads used by fread when reading the files. Be careful when combining a parallel backend specified with BPPARAM with nThread > 1 because each worker will use nThread. |
| verbose | A logical(1) indicating whether progress messages should be printed (default TRUE). |

## Value

A BSseq object.

## File formats

The format of each file is automatically detected using the internal function bsseq:::.guessBismarkFileType(). Files ending in .gz, .bz2, .xz, or .zip will be automatically decompressed to tempdir().

**Supported file formats:** Bismark's 'genome wide cytosine report' (https://github.com/FelixKrueger/Bismark/tree/master/Docs#the-genome-wide-cytosine-report-optional-is-tab-delimite and 'coverage' (https://github.com/FelixKrueger/Bismark/tree/master/Docs#the-coverage-output-look formats are both supported. If setting loci = NULL, then we strongly recommend using the 'genome wide cytosine report' output format because this includes strand information for each locus. The 'coverage' output does not contain strand information and so the strand of the returned BSseq object will be set to * unless stranded loci are supplied.

**Unsupported file formats:** Neither the 'bedGraph' output format (https://github.com/FelixKrueger/Bismark/tree/master/Docs#the-bedgraph-output-optional-looks-like-this-tab-delimit nor the 'bismark_methylation_extractor' output format (https://github.com/FelixKrueger/Bismark/tree/master/Docs#the-bismark_methylation_extractor-output-is-in-the-form-tab-delimited are supported. The former does not include the required counts of methylated and unmethylated reads hile the is an intermediate file containing read-level, rather than locus-level, data on methylation.

**One-based vs. zero-based genomic co-ordinates:** The genomic co-ordinates of the Bismark output files may be zero-based or one-based depending on whether the --zero_based argument was used when running Bismark's methylation extractor. Furthermore, the default co-ordinate counting system varies by version of Bismark. **bsseq** makes no assumptions about the basis of the genomic co-ordinates and it is left to the user to ensure that the appropriate basis is used in the analysis of their data.

Since Bioconductor packages typically use one-based co-ordinates, we strongly recommend that your Bismark files are also one-based.

**Efficient use of** `read.bismark()`

We recommend the following to achieve fast and efficient importing of Bismark files:

- Specify the set of methylation loci via the `loci` argument.
- Use Bismark files in the 'coverage' output format.
- Leave `rmZeroCov = FALSE`.
- Use a `BPPARAM` with a moderate number of workers (cores).
- Use `BACKEND = "HDF5Array"`.
- Use multiple threads per worker (i.e. `nThread` > 1).

Each point is discussed below.

**Specifying `loci`:** Specifying the set of methylation loci via the `loci` argument means that `read.bismark()` does not need to first parse all `files` to identify the set of candidate loci. Provided that `rmZeroCov = FALSE`, this means that each file is only read once. This may be a considerable saving when there are a large number of `files`.

**If you are unsure whether the below-described shortcuts apply to your data, leave `loci = NULL` and let `read.bismark()` identify the set of candidate loci from `files`.**

You may wish to use the `findLoci()` function to find all methylation loci of interest in your reference genome (e.g., all CpGs) and then pass the result via the `loci` argument.

Alternatively, if all `files` are 'genome wide cytosine reports' for samples aligned to the same reference genome, then all `files` contain the exact same set of methylation loci. In this case, you may wish to first construct `loci` using the internal function `bsseq:::.readBismarkAsFWGRanges()` applied to a single file, e.g., `loci = bsseq:::.readBismarkAsFWGRanges(files[1], rmZeroCov, strandCollapse)`.

**Using the 'coverage' Bismark files:** It will generally be faster to parse Bismark files in the 'coverage' output format than those in the 'genome wide cytosine report' format This is because the former only includes loci with non-zero coverage and so the file size is often considerably smaller, particularly for shallowly sequenced samples (e.g., those from single-cell bisulfite sequencing).

**Leaving `rmZeroCov = FALSE`:** If you set `rmZeroCov = TRUE`, then `read.bismark()` must first parse all the `files` to identify which loci have zero coverage in all samples and then filter these out from the set of candidate loci. **This will happen even if you supply `loci` with a `GenomicRanges` of candidate loci.**

Furthermore, any coverage-based filtering of methylation loci is best left until you have constructed your final BSseq object. In our experience, the final BSseq object is often the product of combining multiple BSseq objects, each constructed with a separate call to `read.bismark()`. In such cases, it is premature to use `rmZeroCov = TRUE` when running each `read.bismark()`; regretably, combining these objects will often then lead to an inefficiently stored BSseq object.

**Using a `BPPARAM` with a moderate number of workers (cores):**

Each file can be processed on its own, so you can process in parallel as many files as you have workers. However, if using the HDF5Array backend, then writing to the HDF5 file cannot be performed in parallel and so becomes the bottleneck. Nonetheless, by using a moderate number of workers (2 - 10), we can ensure there is processed data available to write to disk as soon as the current write is completed.

**Using `BACKEND = "HDF5Array"`:** By using the HDF5Array realization backend from **HDF5Array**, we reduce the amount of data that is kept in-memory at any one time. Once each file is parsed, the data are written to the HDF5 file and are no longer needed in-memory. When combined with

multiple workers (cores), this means that each file will only need to read and retain in-memory 1 sample's worth of data at a time.

Conversely, if you opt for all data to be in-memory (via BACKEND = NULL), then each worker will pass each file's data back to the main process and memory usage will steadily accumulate to often unreasonable levels.

**Using nThread > 1:** read.bismark uses data.table::fread to read each file, which supports threaded-parallisation. Depending on the system, increasing nThread can achieve near-linear speed-ups in the number of threads for reading each file. Care needs to be taken when nThread > 1 is used in conjunction with a parallel backend via BPPARAM to ensure the system isn't overloaded. For example, using BPPARAM = MulticoreParam(workers = 10) with nThread = 4 may use up to 40 workers simultaneously.

### Realization backends

The read.bismark() function creates a BSseq object with two assays, M and Cov. The choice of *realization backend* controls whether these assays are stored in-memory as an ordinary matrix or on-disk as a HDF5Array, for example. The choice of realization backend is controlled by the BACKEND argument, which defaults to the current value of DelayedArray::getAutoRealizationBackend().

read.bismark() supports the following realization backends:

- NULL (in-memory): This stores each new assay in-memory using an ordinary matrix.
- HDF5Array (on-disk): This stores each new assay on-disk in a HDF5 file using an HDF5Matrix from **HDF5Array**.

Please note that certain combinations of realization backend and parallelization backend are currently not supported. For example, the HDF5Array realization backend is currently only compatible when used with a single-machine parallelization backend (i.e. it is not compatible with a SnowParam that specifies an *ad hoc* cluster of **multiple** machines). BSmooth() will issue an error when given such incompatible realization and parallelization backends.

Additional arguments related to the realization backend can be passed via the ... argument. These arguments must be named and are passed to the relevant RealizationSink constructor. For example, the ... argument can be used to specify the path to the HDF5 file to be used by BSmooth(). Please see the examples at the bottom of the page.

### Parallelization, progress monitoring, and logging

read.bismark() now uses the **BiocParallel** package to implement parallelization. This brings some notable improvements:

- Imported files can now be written directly to an on-disk realization backend by the worker. This dramatically reduces memory usage compared to previous versions of **bsseq** that required all results be retained in-memory.
- Parallelization is now supported on Windows through the use of a SnowParam object as the value of BPPARAM.
- Detailed and extensive job logging facilities.

All parallelization options are controlled via the BPPARAM argument. In general, we recommend that users combine multicore (single-machine) parallelization with an on-disk realization backend (see section, 'Realization backend'). For Unix and Mac users, this means using a MulticoreParam. For Windows users, this means using a single-machine SnowParam. Please consult the **BiocParallel** documentation to take full advantage of the more advanced features.

A useful feature of **BiocParallel** are progress bars to monitor the status of long-running jobs, such as `BSmooth()`. Progress bars are controlled via the `progressbar` argument in the `BiocParallelParam` constructor.

**BiocParallel** also supports extensive and detailed logging facilities. Please consult the **BiocParallel** documentation to take full advantage these advanced features.

## Author(s)

Peter Hickey <peter.hickey@gmail.com>

## See Also

- `collapseBSseq()` for collapsing (aggregating) data from sample's with multiple Bismark methylation extractor files (e.g., technical replicates).

## Examples

```
# Run read.bismark() on a single sample to construct a matrix-backed BSseq
# object.
infile <- system.file("extdata/test_data.fastq_bismark.bismark.cov.gz",
                      package = "bsseq")
bsseq <- read.bismark(files = infile,
                      colData = DataFrame(row.names = "test_data"),
                      rmZeroCov = FALSE,
                      strandCollapse = FALSE,
                      verbose = TRUE)
# This is a matrix-backed BSseq object.
sapply(assays(bsseq, withDimnames = FALSE), class)
bsseq

## Not run:
# Run read.bismark() on a single sample to construct a HDF5Array-backed BSseq
# object (with data written to 'test_dir')
test_dir <- tempfile("BSseq")
bsseq <- read.bismark(files = infile,
                      colData = DataFrame(row.names = "test_data"),
                      rmZeroCov = FALSE,
                      strandCollapse = FALSE,
                      BACKEND = "HDF5Array",
                      dir = test_dir,
                      verbose = TRUE)
# This is a HDF5Array-backed BSseq object.
sapply(assays(bsseq, withDimnames = FALSE), class)
# The 'M' and 'Cov' assays are in the HDF5 file 'assays.h5' (in 'test_dir').
sapply(assays(bsseq, withDimnames = FALSE), path)

## End(Not run)
```

---

# read.modbam2bed *Construct BSseq objects from nanopore BED files*

---

## Description

Construct BSseq objects from nanopore BED files

## Usage

```
read.modbam2bed(
    files,
    colData = NULL,
    rmZeroCov = FALSE,
    strandCollapse = TRUE
)
```

## Arguments

| | |
|---|---|
| files | vector, BED files |
| colData | data frame, phenotypic data with samples as rows and variables as columns |
| rmZeroCov | A logical (1) indicating whether methylation loci that have zero coverage in all samples be removed |
| strandCollapse | A logical (1) indicating whether stand-symmetric methylation loci (i.e. CpGs) should be collapsed across strands |

## Details

This function reads in nanopore sequencing modified BED files to Bsseq objects. Nanopore sequencing data (i.e. aggregated modified base counts) is stored in modified-base BAM files. These modified-base BAM files are converted to bedMethyl (BED) files using modbam2bed.

**Details for using modbam2bed:** After installing modbam2bed, a conda environment is activated. Index files for BAM files are created using `samtools index`. The code requires aligned reads with the Mm and Ml tags (MM and ML also supported), and the reference sequence used for alignment (<reference.fasta>).

- `-e, --extended` to output canonical, modified, and filtered bases;
- `-m, --mod_base=BASE` to output modified base of interest, one of: 5mC, 5hmC, 5fC, 5caC, 5hmU, 5fU, 5caU, 6mA, 5oxoG, Xao. (Or modA, modC, modG, modT, modU, modN for generic modified base);
- `-r, --region=chr:start-end` to output chromosome or genomic region of interest;
- `-f, --threshold=THRESHOLD` to output filtered bases for probability lower than threshold (default = 0.66)

**modbam2bed to Bsseq object:** After creating BED files using modbam2bed, the BED files are read in and the Bsseq object is constructed via `read.modbam2bed()` function. The function reads in BED files, extract genomic regions, methylation, coverage, ambiguous modification status data and sample information and then construct Bsseq object using `BSseq` function within the package.

## Value

BSseq object

## Examples

```
files <- c(system.file("extdata/modbam2bed/ctr1.chr10.chr11.bed.gz", package = "bsseq"),
    system.file("extdata/modbam2bed/ctr2.chr10.chr11.bed.gz", package = "bsseq"),
    system.file("extdata/modbam2bed/ctr3.chr10.chr11.bed.gz", package = "bsseq"),
    system.file("extdata/modbam2bed/tret1.chr10.chr11.bed.gz", package = "bsseq"),
    system.file("extdata/modbam2bed/tret2.chr10.chr11.bed.gz", package = "bsseq"),
    system.file("extdata/modbam2bed/tret3.chr10.chr11.bed.gz", package = "bsseq"))
```

```
pd <- data.frame(condition = rep(c("control", "treatment"), each = 3),
    replicate = rep(c("rep1", "rep2", "rep3"), times = 2))
bsseq_nano <- bsseq::read.modbam2bed(files,colData=pd,rmZeroCov = FALSE,
    strandCollapse=TRUE)
```

---

`read.modkit` *Construct BSseq objects from nanopore BED files*

---

## Description

Construct BSseq objects from nanopore BED files

## Usage

```
read.modkit(
    files,
    colData = NULL,
    rmZeroCov = FALSE,
    strandCollapse = TRUE
)
```

## Arguments

| | |
|---|---|
| `files` | vector, BED files |
| `colData` | data frame, phenotypic data with samples as rows and variables as columns |
| `rmZeroCov` | A logical (1) indicating whether methylation loci that have zero coverage in all samples be removed |
| `strandCollapse` | A logical (1) indicating whether stand-symmetric methylation loci (i.e. CpGs) should be collapsed across strands |

## Details

This function reads in nanopore sequencing modified BED files to Bsseq objects. Nanopore sequencing data (i.e. aggregated modified base counts) is stored in modified-base BAM files. These modified-base BAM files are converted to bedMethyl (BED) files using modkit.

**Details for modkit:** Modkit outputs modified reads, unmodified reads, ambiguous modification reads (reads where the probability was below the threshold and usually failing the lowest 10th percentile), and other modified reads.

**modkit to Bsseq object:** After creating BED files using modkit, the BED files are read in and the Bsseq object is constructed via `read.modkit()` function. The function reads in BED files, extract genomic regions, methylation, coverage, ambiguous modification status data and sample information and then construct Bsseq object using `BSseq` function within the package. Other modification bases such as hydroxymethylation are extracted and added to the methylation matrix when present.

## Value

BSseq objects

### Examples

```
# No other modification present
files <- c(system.file("extdata/modkit/chr21.chr22.HG002.top1000.bed.gz", package = "bsseq"))
bsseq_nano <- read.modkit(files, rmZeroCov = FALSE, strandCollapse=FALSE)

# Other modification present
files <- c(system.file("extdata/modkit/Hypo1.first50Bed.txt",package = "bsseq"))
bsseq_nano <- read.modkit(files, rmZeroCov = FALSE, strandCollapse=FALSE)
```

---

## smoothSds — *Smooth the standard deviations using a thresholded running mean based on smoothed whole-genome bisulfite sequencing data.*

---

### Description

Smooth the standard deviations using a thresholded running mean based on smoothed whole-genome bisulfite sequencing data.

### Usage

```
smoothSds(BSseqStat, k = 101, qSd = 0.75, mc.cores = 1, maxGap = 10^8,
          verbose = TRUE)
```

### Arguments

| | |
|---|---|
| BSseqStat | An object of class BSseqStat, typically an object returned by BSmooth.fstat(...) and not constructed by the user. |
| k | A positive scalar, see details. |
| qSd | A scalar between 0 and 1, see details. |
| mc.cores | The number of cores used. Note that setting mc.cores to a value greater than 1 is not supported on MS Windows, see the help page for mclapply. |
| maxGap | A scalar greater than 0, see details. |
| verbose | Should the function be verbose? |

### Details

The standard deviation estimates are smoothed using a running mean with a width of k and thresholded using qSd which sets the minimum standard deviation to be the qSd-quantile.

### Value

An object of class BSseqStat. More speciically, the input BSseqStat object with the computed statistics added to the stats slot (accessible with getStats).

### Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>

### See Also

BSmooth.fstat for the function to create the appropriate BSseqStat input object. BSseqStat also describes the return class. This function is likely to be followed by the use of computeStat.

## Examples

```
if(require(bsseqData)) {
    # library(limma) required for makeContrasts()
    library(limma)
    data(keepLoci.ex)
    data(BS.cancer.ex.fit)
    BS.cancer.ex.fit <- updateObject(BS.cancer.ex.fit)
    ## Remember to subset the BSseq object, see vignette for explanation
    ## TODO: Kind of a forced example
    design <- model.matrix(~0 + BS.cancer.ex.fit$Type)
    colnames(design) <- gsub("BS\\.cancer\\.ex\\.fit\\$Type", "",
                             colnames(design))
    contrasts <- makeContrasts(
        cancer_vs_normal = cancer - normal,
        levels = design
    )
    BS.stat <- BSmooth.fstat(BS.cancer.ex.fit[keepLoci.ex,],
                             design,
                             contrasts)
    BS.stat <- smoothSds(BS.stat)
    ## Comparing the raw standard deviations to the smoothed standard
    ## deviations
    summary(getStats(BS.stat, what = "rawSds"))
    summary(getStats(BS.stat, what = "smoothSds"))
}
```

# Index