Package ‘methylKit’

March 31, 2025

Type Package

Title DNA methylation analysis from high-throughput bisulfite sequencing results

Version 1.32.1

Author Altuna Akalin [aut, cre], Matthias Kormaksson [aut], Sheng Li [aut], Arsene Wabo [ctb], Adrian Bierling [aut], Alexander Blume [aut], Katarzyna Wreczycka [ctb]

Maintainer Altuna Akalin `<aakalin@gmail.com>`, Alexander Blume `<alex.gos90@gmail.com>`

Description methylKit is an R package for DNA methylation analysis and annotation from high-throughput bisulfite sequencing. The package is designed to deal with sequencing data from RRBS and its variants, but also target-capture methods and whole genome bisulfite sequencing. It also has functions to analyze base-pair resolution 5hmC data from experimental protocols such as oxBS-Seq and TAB-Seq. Methylation calling can be performed directly from Bismark aligned BAM files.

License Artistic-2.0

URL `https://github.com/al2na/methylKit`

BugReports `https://github.com/al2na/methylKit/issues`

LazyLoad yes

NeedsCompilation yes

LinkingTo Rcpp, Rhtslib (>= 1.13.1), zlibbioc

SystemRequirements GNU make

biocViews DNAMethylation, Sequencing, MethylSeq

Depends R (>= 3.5.0), GenomicRanges (>= 1.18.1), methods

Imports IRanges, data.table (>= 1.9.6), parallel, S4Vectors (>= 0.13.13), GenomeInfoDb, KernSmooth, qvalue, emdbook, Rsamtools, gtools, fastseg, rtracklayer, mclust, mgcv, Rcpp, R.utils, limma, grDevices, graphics, stats, utils

Suggests testthat (>= 2.1.0), knitr, rmarkdown, genomation, BiocManager

VignetteBuilder knitr

Collate 'methylKit.R' 'backbone.R' 'diffMeth.R' 'clusterSamples.R' 'regionalize.R' 'processBismarkAln.R' 'RcppExports.R' 'document_data.R' 'bedgraph.R' 'reorganize.R' 'percMethylation.R' 'normalizeCoverage.R' 'pool.R' 'adjustMethylC.R' 'updateMethObject.R' 'batchControl.R' 'dataSim.R' 'methylDBClasses.R' 'methylDBFunctions.R' 'tabix.functions.R' 'methSeg.R' 'diffMethDSS.R' 'deprecated_defunct.R' 'onUnload.R'

RoxygenNote 7.1.1

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

git_branch RELEASE_3_20

git_last_commit d374f65

git_last_commit_date 2025-02-28

Repository Bioconductor 3.20

Date/Publication 2025-03-31

Contents

`adjustMethylC` *Adjust measured 5mC levels using 5hmC levels*

Description

Measured 5mC levels via bisulfite sequencing might be a combination of 5hmC and 5mC levels since bisulfite sequencing can not distinguish between the two. This function can adjust 5mC levels of a bisulfite sequencing experiment if the user supplies corresponding 5hmC levels from the same sample.

Usage

```
adjustMethylC(mc,hmc,save.db,...,chunk.size)

## S4 method for signature 'methylRaw,methylRaw'
adjustMethylC(mc, hmc, save.db = FALSE, ..., chunk.size = 1e+06)

## S4 method for signature 'methylRawList,methylRawList'
adjustMethylC(mc, hmc, save.db = FALSE, ..., chunk.size = 1e+06)

## S4 method for signature 'methylRawDB,methylRawDB'
adjustMethylC(mc, hmc, save.db = TRUE, ..., chunk.size = 1e+06)

## S4 method for signature 'methylRawListDB,methylRawListDB'
adjustMethylC(mc, hmc, save.db = TRUE, ..., chunk.size = 1e+06)
```

Arguments

`mc`	a `methylRawList`, `methylRaw`, `methylRawDB` or `methylRawListDB` containing 5mC levels of a sample or set of samples
`hmc`	a `methylRawList`, `methylRaw`, `methylRawDB` or `methylRawListDB` containing 5hmC levels of a sample or set of samples. If a `methylRawList` or `methylRawListDB` given the sample order should be same as "mc" `methylRawList` or `methylRawListDB` object.
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_filtered" to current filename if database already exists or generate new file with filename "sampleID_filtered" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawDB` or `methylRawListDB` objects (default: 1e6)

Value

returns adjusted 5-methyl cytosine levels in the form of `methylRawList`, `methylRaw`, `methylRawDB` or `methylRawListDB` object depending on the input object

Details

The parameter `chunk.size` is only used when working with `methylRawDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawDB` and `methylRawListDB`, while being per default FALSE for `methylRaw` and `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might change the value of this parameter.

References

1. Booth, Branco, et al. (2012). Quantitative Sequencing of 5-Methylcytosine and 5-Hydroxymethylcytosine at Single-Base Resolution. Science, 934

2. Yu, Hon, et al. (2012). Base-resolution analysis of 5-hydroxymethylcytosine in the Mammalian genome. Cell, 149(6), 1368-80.

Examples

```
# read 5hmC and 5mC files
hmc.file=system.file("extdata", "test1.myCpG.txt", package = "methylKit")
mc.file =system.file("extdata", "test2.myCpG.txt", package = "methylKit")
```

```
my5hmC=methRead( hmc.file,sample.id="hmc",assembly="hg18")
my5mC =methRead( mc.file,sample.id="mc",assembly="hg18")

# adjusting the 5mC levels using 5hmC levels
adjusted.5mC=adjustMethylC(my5mC,my5hmC)
```

`assocComp` *Associate principal components with sample annotations*

Description

This function associates principal components with sample annotations such as age, gender, batch_id. Can be used to detect which batch effects are associated with the variation in the methylation values.

Usage

```
assocComp(mBase, sampleAnnotation)
```

Arguments

`mBase`	`methylBase` or `methylBaseDB` object with no NA values in the data part.
`sampleAnnotation`	a data frame where columns are different annotations and rows are the samples, in the same order as in the methylBase object.

Value

a named list of principal component matrix (named 'pcs'), by principal compopents (named 'vars') and a p-value matrix showing association p-values between sample annotations and principal components (named 'association').

Author(s)

Altuna Akalin

Examples

```
data(methylKit)
sampleAnnotation=data.frame(batch_id=c("a","a","b","b"),age=c(19,34,23,40))
as=assocComp(mBase=methylBase.obj,sampleAnnotation)
```

---

bedgraph *Get bedgraph from methylRaw, methylRawList and methylDiff objects*

Description

The function converts `methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylDiff` or `methylDiffDB` object into a bedgraph format. It either writes as a file or returns a `data.frame`

Usage

```
bedgraph(
  methylObj,
  file.name = NULL,
  col.name,
  unmeth = FALSE,
  log.transform = FALSE,
  negative = FALSE,
  add.on = "",
  chunk.size = 1e+06
)

## S4 method for signature 'methylDiff'
bedgraph(
  methylObj,
  file.name,
  col.name,
  unmeth,
  log.transform,
  negative,
  add.on
)

## S4 method for signature 'methylRaw'
bedgraph(
  methylObj,
  file.name,
  col.name,
  unmeth,
  log.transform,
  negative,
  add.on
)

## S4 method for signature 'methylRawList'
bedgraph(
  methylObj,
  file.name,
  col.name,
  unmeth,
  log.transform,
  negative,
```

```
  add.on
)

## S4 method for signature 'methylRawDB'
bedgraph(
  methylObj,
  file.name = NULL,
  col.name,
  unmeth = FALSE,
  log.transform = FALSE,
  negative = FALSE,
  add.on = "",
  chunk.size = 1e+06
)

## S4 method for signature 'methylRawListDB'
bedgraph(
  methylObj,
  file.name = NULL,
  col.name,
  unmeth = FALSE,
  log.transform = FALSE,
  negative = FALSE,
  add.on = "",
  chunk.size = 1e+06
)

## S4 method for signature 'methylDiffDB'
bedgraph(
  methylObj,
  file.name,
  col.name,
  log.transform,
  negative,
  add.on,
  chunk.size
)
```

Arguments

`methylObj`	a `methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylDiff` or `methylDiffDB` object
`file.name`	Default: NULL. if a string is given a bedgraph file will be written, if NULL a data.frame or a list of data frames will be returned
`col.name`	name of the column in `methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylDiff` or `methylDiffDB` objects to be used as a score for the bedgraph. For `methylDiff` or `methylDiffDB`, `col.name` must be one of the following 'pvalue','qvalue', 'meth.diff'. For `methylRaw`, `methylRawDB`, `methylRawList` and `methylRawListDB` it must be one of the following 'coverage', 'numCs','numTs', 'perc.meth'
`unmeth`	when working with `methylRaw`, `methylRawDB`, `methylRawList` and `methylRawListDB` objects should you output unmethylated C percentage this makes it easier to

	see the unmethylated bases because their methylation percentage values will be zero. Only invoked when file.name is not NULL.
`log.transform`	Default FALSE, If TRUE the score column of the bedgraph wil be in log10 scale. Ignored when col.name='perc.meth'
`negative`	Default FALSE, If TRUE, the score column of the bedgraph will be multiplied by -1. Ignored when col.name='perc.meth'
`add.on`	additional string to be add on the track line of bedgraph. can be viewlimits,priority etc. Check bedgraph track line options at UCSC browser
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawDB`, `methylRawListDB` or `methylDiffDB` objects, default: 1e6

Value

Returns a `data.frame` or list of data.frames if `file.name=NULL`, if a file.name given appropriate bed file will be written to that file

Details

The parameter `chunk.size` is only used when working with `methylRawDB`, `methylRawListDB` or `methylDiffDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Examples

```
data(methylKit)

# getting a bedgraph file from a methylDiff object containing differential
# methylation percentages
bedgraph(methylDiff.obj, file.name="test.bed", col.name="meth.diff",
         unmeth=FALSE,log.transform=FALSE,negative=FALSE,add.on="")

# remove the file
unlink("test.bed")

# getting a bedgraph file from a methylBase object containing percent
#methylation values
bedgraph(methylRawList.obj[[1]], file.name="test2.bed", col.name="perc.meth",
         unmeth=FALSE,log.transform=FALSE,negative=FALSE,add.on="")

unlink("test2.bed") # remove the file
```

`calculateDiffMeth` *Calculate differential methylation statistics*

Description

The function calculates differential methylation statistics between two groups of samples. The function uses either logistic regression test or Fisher's Exact test to calculate differential methylation. See the rest of the help page and references for detailed explanation on statistics.

Usage

```
calculateDiffMeth(
  .Object,
  covariates = NULL,
  overdispersion = c("none", "MN", "shrinkMN"),
 adjust = c("SLIM", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none", "qvalue"),
  effect = c("wmean", "mean", "predicted"),
  parShrinkMN = list(),
  test = c("F", "Chisq", "fast.fisher", "midPval"),
  mc.cores = 1,
  slim = TRUE,
  weighted.mean = TRUE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylBaseDB'
calculateDiffMeth(
  .Object,
  covariates = NULL,
  overdispersion = c("none", "MN", "shrinkMN"),
 adjust = c("SLIM", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none", "qvalue"),
  effect = c("wmean", "mean", "predicted"),
  parShrinkMN = list(),
  test = c("F", "Chisq", "fast.fisher", "midPval"),
  mc.cores = 1,
  slim = TRUE,
  weighted.mean = TRUE,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)
```

Arguments

`.Object`	a `methylBase` or `methylBaseDB` object to calculate differential methylation
`covariates`	a `data.frame` containing covariates, which should be included in the test.
`overdispersion`	If set to "none"(default), no overdispersion correction will be attempted. If set to "MN", basic overdispersion correction, proposed by McCullagh and Nelder (1989) will be applied.This correction applies a scaling parameter to variance estimated by the model. EXPERIMENTAL: If set to "shrinkMN", scaling parameter will be shrunk towards a common value (not thoroughly tested as of yet).
`adjust`	different methods to correct the p-values for multiple testing. Default is "SLIM" from methylKit. For "qvalue" please see `qvalue` and for all other methods see `p.adjust`.
`effect`	method to calculate the mean methylation different between groups using read coverage as weights (default). When set to "mean", the generic mean is applied

	and when set to "predicted", predicted means from the logistic regression model is used for calculating the effect.
`parShrinkMN`	a list for squeezeVar(). (NOT IMPLEMENTED)
`test`	the statistical test used to determine the methylation differences. The Chisq-test is used by default for more than two groups, while the F-test can be chosen if overdispersion control is applied. If there is one sample per group the Fisher's exact test will be applied using "fast.fisher", while "midPval" can be choosen to boost calculation speed. See details section for more information.
`mc.cores`	integer denoting how many cores should be used for parallel differential methylation calculations (can only be used in machines with multiple cores).
`slim`	If set to FALSE, `adjust` will be set to "BH" (default behaviour of earlier versions)
`weighted.mean`	If set to FALSE, `effect` will be set to "mean" (default behaviour of earlier versions)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` objects (default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details section.
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: The default suffix is a 13-character random string appended to the fixed prefix “methylDiff”, e.g. “methylDiff_16d3047c1a254.txt.bgz”. `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

a `methylDiff` object containing the differential methylation statistics and locations for regions or bases

Details

Covariates can be included in the analysis. The function will then try to separate the influence of the covariates from the treatment effect via a linear model.
The Chisq-test is used per default only when no overdispersion correction is applied. If overdispersion correction is applied, the function automatically switches to the F-test. The Chisq-test can be manually chosen in this case as well, but the F-test only works with overdispersion correction switched on.
If there is one sample in each group, e.g. after applying the pooling samples, the Fisher's exact test will be applied for differential methylation. methyKit offers two implementations to perform this test, which yield slightly different results but differ much in computation time. "fast.fisher" is a cut down version ‘fisher.test()‘ that should produce the exact same results as the base implementation, while "midPval" will produce marginaly different p-values, but offers a large boost in calculation speed.

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database.

Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylBaseDB`, while being per default FALSE for `methylBase`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

References

Altuna Akalin, Matthias Kormaksson, Sheng Li, Francine E. Garrett-Bakelman, Maria E. Figueroa, Ari Melnick, Christopher E. Mason. (2012). "methylKit: A comprehensive R package for the analysis of genome-wide DNA methylation profiles." Genome Biology.

McCullagh and Nelder. (1989). Generalized Linear Models. Chapman and Hall. London New York.

Barnard. (1989). On alleged gains in power from lower P-values. Statistics in Medicine. Armitage and Berry. (1994) Statistical Methods in Medical Research (3rd edition). Blackwell.

See Also

`pool`, `reorganize` `dataSim`

Examples

```
data(methylKit)

# The Chisq-test will be applied when no overdispersion control is chosen.
my.diffMeth=calculateDiffMeth(methylBase.obj,covariates=NULL,
                              overdispersion=c("none"),
                              adjust=c("SLIM"))

# pool samples in each group
pooled.methylBase=pool(methylBase.obj,sample.ids=c("test","control"))

# After applying the pool() function, there is one sample in each group.
# The Fisher's exact test will be applied for differential methylation.
my.diffMeth2=calculateDiffMeth(pooled.methylBase,covariates=NULL,
                               overdispersion=c("none"),
                               adjust=c("SLIM"))

# Covariates and overdispersion control:
# generate a methylBase object with age as a covariate
covariates=data.frame(age=c(30,80,30,80))
sim.methylBase<-dataSim(replicates=4,sites=1000,treatment=c(1,1,0,0),
                        covariates=covariates,
                        sample.ids=c("test1","test2","ctrl1","ctrl2"))

# Apply overdispersion correction and include covariates
# in differential methylation calculations.
my.diffMeth3<-calculateDiffMeth(sim.methylBase,
                                covariates=covariates,
                                overdispersion="MN",test="Chisq",mc.cores=1)
```

`calculateDiffMethDSS` *calculate Differential Methylation with DSS*

Description

This function provides an interface to the beta-binomial model from DSS package by Hao Wu. It calculates the differential methylation statistics using a beta-binomial model with parameter shrinkage. See the reference for details.

Usage

```
calculateDiffMethDSS(
  meth,
 adjust = c("SLIM", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none", "qvalue"),
  mc.cores = 1
)

## S4 method for signature 'methylBase'
calculateDiffMethDSS(
  meth,
 adjust = c("SLIM", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none", "qvalue"),
  mc.cores = 1
)
```

Arguments

`meth`	a methylBase object
`adjust`	different methods to correct the p-values for multiple testing. Default is "SLIM" from methylKit. For "qvalue" please see `qvalue` and for all other methods see `p.adjust`.
`mc.cores`	integer denoting how many cores should be used for parallel differential methylation calculations (can only be used in machines with multiple cores).

Value

a methylDiff object

References

Feng H, Conneely K and Wu H (2014). A bayesian hierarchical model to detect differentially methylated loci from single nucleotide resolution sequencing data. Nucleic acids research

Examples

```
data(methylKit)

dssDiffay <- calculateDiffMethDSS(methylBase.obj, adjust="SLIM", mc.cores=1)
```

`clusterSamples`	*Hierarchical Clustering using methylation data The function clusters samples using* `hclust` *function and various distance metrics derived from percent methylation per base or per region for each sample.*

Description

Hierarchical Clustering using methylation data

The function clusters samples using `hclust` function and various distance metrics derived from percent methylation per base or per region for each sample.

Usage

```
clusterSamples(.Object, dist="correlation", method="ward",
                        sd.filter=TRUE,sd.threshold=0.5,
                        filterByQuantile=TRUE, plot=TRUE,chunk.size)

## S4 method for signature 'methylBase'
clusterSamples(
  .Object,
  dist,
  method,
  sd.filter,
  sd.threshold,
  filterByQuantile,
  plot
)

## S4 method for signature 'methylBaseDB'
clusterSamples(
  .Object,
  dist = "correlation",
  method = "ward",
  sd.filter = TRUE,
  sd.threshold = 0.5,
  filterByQuantile = TRUE,
  plot = TRUE,
  chunk.size = 1e+06
)
```

Arguments

`.Object`	a `methylBase` or `methylBaseDB` object
`dist`	the distance measure to be used. This must be one of "`correlation`", "`euclidean`", "`maximum`", "`manhattan`", "`canberra`", "`binary`" or "`minkowski`". Any unambiguous abbreviation can be given. (default:"`correlation`")
`method`	the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "`ward.D`", "`ward.D`", "`single`", "`complete`", "`average`", "`mcquitty`", "`median`" or "`centroid`". (default:"`ward`")

`sd.filter`	If `TRUE`, the bases/regions with low variation will be discarded prior to clustering (default:TRUE)
`sd.threshold`	A numeric value. If `filterByQuantile` is `TRUE`, features whose standard deviations is less than the quantile denoted by `sd.threshold` will be removed. If `filterByQuantile` is `FALSE`, then features whose standard deviations is less than the value of `sd.threshold` will be removed.(default:0.5)
`filterByQuantile`	A logical determining if `sd.threshold` is to be interpreted as a quantile of all Standard Deviation values from bases/regions (the default), or as an absolute value
`plot`	a logical value indicating whether to plot hierarchical clustering. (default:TRUE)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` objects, default: 1e6

Value

a `tree` object of a hierarchical cluster analysis using a set of dissimilarities for the n objects being clustered.

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Examples

```
data(methylKit)

clusterSamples(methylBase.obj, dist="correlation", method="ward", plot=TRUE)
```

`dataSim` *Simulate DNA methylation data*

Description

The function simulates DNA methylation data from multiple samples. See references for detailed explanation on statistics.

Usage

```
dataSim(
  replicates,
  sites,
  treatment,
  percentage = 10,
  effect = 25,
```

```
    alpha = 0.4,
    beta = 0.5,
    theta = 10,
    covariates = NULL,
    sample.ids = NULL,
    assembly = "hg18",
    context = "CpG",
    add.info = FALSE
)
```

Arguments

`replicates`	the number of samples that should be simulated.
`sites`	the number of CpG sites per sample.
`treatment`	a vector containing treatment information.
`percentage`	the proportion of sites which should be affected by the treatment.
`effect`	a number between 0 and 100 specifying the effect size of the treatment. This is essentially describing the average percent methylation difference between differentially methylated bases.See 'Examples' and 'Details'.
`alpha`	shape1 parameter for beta distribution (used for initial sampling of methylation proportions)
`beta`	shape2 parameter for beta distribution (used for initial sampling of methylation proportions)
`theta`	dispersion parameter for beta distribution (initial sampling of methylation proportions)
`covariates`	a data.frame containing covariates (optional)
`sample.ids`	will be generated automatically from `treatment`, but can be overwritten by a character vector containing sample names.
`assembly`	the assembly description (e.g. "hg18").Only needed for book keeping.
`context`	the experimanteal context of the data (e.g. "CpG"). Only needed for book keeping.
`add.info`	if set to TRUE, the output will be a list with the first element being the methylbase object and a vector of indices that indicate which CpGs should be differentially methylated. This vector can be used to subset simulated methylBase or methylDiff object with differentially methylated bases.

Value

a methylBase object containing simulated methylation data, or if add.info=TRUE a list containing the methylbase object and the indices of all treated sites (differentially methylated bases or regions) as the second element.

Details

The function uses a Beta distribution to simulate the methylation proportion background across all samples. The parameters `alpha`, `beta` used in a beta distribution to draw methylation proportions,μ, from a typical bimodal distribution. For each initial methylation proportion drawn using the parameters above, a range of methylation proportions is distributed around the original μ with overdispersion parameter θ, this is using an alternative parameterization of Beta distribution: $Beta(\mu, \theta)$.

The parameters `percentage` and `effect` determine the proportion of sites that are affected by the treatment (meaning differential sites) and the strength of this influence, respectively. `effect` is added on top of μ for the CpGs that are affected by the treament. The affected group of samples for that particular CpG will now be distributed by $Beta(\mu + effect, \theta)$. The coverage is modeled with a negative binomial distribution, using `rnbinom` function with `size=1` and `prob=0.01`. The additional information needed for a valid methylBase object, such as CpG start, end and strand, is generated as "dummy values", but can be overwritten as needed.

Examples

```
data(methylKit)

# Simulate data for 4 samples with 20000 sites each.
# The methylation in 10% of the sites are elevated by 25%.
my.methylBase=dataSim(replicates=4,sites=2000,treatment=c(1,1,0,0),
percentage=10,effect=25)
```

`diffMethPerChr` *Get and plot the number of hyper/hypo methylated regions/bases per chromosome*

Description

This function gets number of hyper/hypo methylated regions/bases from `methylDiff` object. It can also plot percentages of differentially methylated bases per chromosome.

Usage

```
diffMethPerChr(
  x,
  plot = TRUE,
  qvalue.cutoff = 0.01,
  meth.cutoff = 25,
  exclude = NULL,
  keep.empty.chrom = FALSE,
  ...
)

## S4 method for signature 'methylDiff'
diffMethPerChr(
  x,
  plot = TRUE,
  qvalue.cutoff = 0.01,
  meth.cutoff = 25,
  exclude = NULL,
  keep.empty.chrom = FALSE,
  ...
)
```

```
## S4 method for signature 'methylDiffDB'
diffMethPerChr(
  x,
  plot = TRUE,
  qvalue.cutoff = 0.01,
  meth.cutoff = 25,
  exclude = NULL,
  keep.empty.chrom = FALSE,
  ...
)
```

Arguments

`x`	a `methylDiff` object
`plot`	TRUE|FALSE. If TRUE horizontal barplots for proportion of hypo/hyper methylated bases/regions
`qvalue.cutoff`	cutoff for q-value
`meth.cutoff`	cutoff for percent methylation difference
`exclude`	names of chromosomes to be excluded from plot
`keep.empty.chrom`	keep chromosome in list / plot, even if it contains no hyper/hypo sites
`...`	extra graphical parameters to be passed to `barplot` function

Value

plots a piechart or a barplot for percentage of the target features overlapping with annotation

Examples

```
data(methylKit)

# get a list of differentially methylated bases/regions per chromosome and overall
diffMethPerChr(methylDiff.obj, plot=FALSE,qvalue.cutoff=0.01,
               meth.cutoff=25,exclude=NULL)
```

`extract` *extract parts of methylRaw,methylRawDB,methylBase,methylBaseDB and methylDiff data*

Description

The function extracts part of the data and returns a new object.

Usage

```
## S4 method for signature 'methylRaw,ANY,ANY,ANY'
x[i, j]

## S4 method for signature 'methylBase,ANY,ANY,ANY'
x[i, j]

## S4 method for signature 'methylDiff,ANY,ANY,ANY'
x[i, j]

## S4 method for signature 'methylRawDB,ANY,ANY,ANY'
x[i, j]

## S4 method for signature 'methylBaseDB,ANY,ANY,ANY'
x[i, j]

## S4 method for signature 'methylDiffDB,ANY,ANY,ANY'
x[i, j]
```

Arguments

`x`	an `methylBase`,`methylBaseDB`, `methylRaw`,`methylRawDB` or `methylDiff` object
`i`	a numeric or logical vector. This vector corresponds to bases or regions contained in `methylKit` objects.The vector is used to subset the data.
`j`	This argument can not be used for the extraction of columns. As unintentional extraction of the columns will cause an error in the downstream analysis. Using this argument will cause an error. Use `getData` to access the data part of the objects.

Examples

```
data(methylKit)

# selects first hundred rows, returns a methylRaw object
subset1=methylRawList.obj[[1]][1:100]

# selects first hundred rows, returns a methylBase object
subset2=methylBase.obj[1:100,]

# selects first hundred rows, returns a methylDiff object
subset3=methylDiff.obj[1:100,]

# This will get chromomsomes, will return a factor
# That means the resulting object will ceases to be a methylKit object
chrs=methylDiff.obj[[2]]
```

`filterByCoverage` *Filter methylRaw, methylRawDB, methylRawList and methylRawListDB object based on read coverage*

Description

This function filters `methylRaw`, `methylRawDB`, `methylRawList` and `methylRawListDB` objects. You can filter based on lower read cutoff or high read cutoff. Higher read cutoff is usefull to eliminate PCR effects Lower read cutoff is usefull for doing better statistical tests.

Usage

```
filterByCoverage(
  methylObj,
  lo.count = NULL,
  lo.perc = NULL,
  hi.count = NULL,
  hi.perc = NULL,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRaw'
filterByCoverage(
  methylObj,
  lo.count = NULL,
  lo.perc = NULL,
  hi.count = NULL,
  hi.perc = NULL,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawList'
filterByCoverage(
  methylObj,
  lo.count = NULL,
  lo.perc = NULL,
  hi.count = NULL,
  hi.perc = NULL,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawDB'
filterByCoverage(
  methylObj,
  lo.count = NULL,
```

```
  lo.perc = NULL,
  hi.count = NULL,
  hi.perc = NULL,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylRawListDB'
filterByCoverage(
  methylObj,
  lo.count = NULL,
  lo.perc = NULL,
  hi.count = NULL,
  hi.perc = NULL,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)
```

Arguments

`methylObj`	a `methylRaw`, `methylRawDB`, `methylRawList` or `methylRawListDB` object
`lo.count`	An integer for read counts.Bases/regions having lower coverage than this count is discarded
`lo.perc`	A double [0-100] for percentile of read counts. Bases/regions having lower coverage than this percentile is discarded
`hi.count`	An integer for read counts. Bases/regions having higher coverage than this is count discarded
`hi.perc`	A double [0-100] for percentile of read counts. Bases/regions having higher coverage than this percentile is discarded
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawDB` or `methylRawListDB` objects, default: 1e6
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_filtered" to current filename if database already exists or generate new file with filename "sampleID_filtered" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to original directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

`methylRaw`, `methylRawDB`, `methylRawList` or `methylRawListDB` object depending on input object

Details

The parameter `chunk.size` is only used when working with `methylRawDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawDB` and `methylRawListDB`, while being per default FALSE for `methylRaw` and `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might change the value of this parameter.

Examples

```
data(methylKit)

# filter out bases with covereage above 500 reads
filtered1=filterByCoverage(methylRawList.obj,lo.count=NULL,lo.perc=NULL,
hi.count=500,hi.perc=NULL)

# filter out bases with cread coverage above 99.9th percentile of coverage
# distribution
filtered2=filterByCoverage(methylRawList.obj,lo.count=NULL,lo.perc=NULL,
hi.count=NULL,hi.perc=99.9)

# filter out bases with covereage above 500 reads and save to database
# "test1_max500.txt.bgz"
# in directory "methylDB", filtered3 now becomes a \code{methylRawDB} object
filtered3=filterByCoverage(methylRawList.obj[[1]], lo.count=NULL,lo.perc=NULL,
                           hi.count=500, hi.perc=NULL, save.db=TRUE,
                           suffix="max500", dbdir="methylDB")

# tidy up
rm(filtered3)
unlink("methylDB",recursive=TRUE)
```

`getAssembly` *get assembly of the genome*

Description

The function returns the genome assembly stored in any of the `methylBase`,`methylBaseDB`,`methylRaw`, `methylRawDB`,`methylDiff` objects

Usage

```
getAssembly(x)

## S4 method for signature 'methylBase'
getAssembly(x)

## S4 method for signature 'methylRaw'
```

```
getAssembly(x)

## S4 method for signature 'methylDiff'
getAssembly(x)

## S4 method for signature 'methylRawDB'
getAssembly(x)

## S4 method for signature 'methylBaseDB'
getAssembly(x)

## S4 method for signature 'methylDiffDB'
getAssembly(x)
```

Arguments

`x`	an `methylBase`,`methylBaseDB`, `methylRaw`,`methylRawDB` or `methylDiff` object

Value

the assembly string for the object

Examples

```
data(methylKit)

getAssembly(methylBase.obj)
getAssembly(methylDiff.obj)
getAssembly(methylRawList.obj[[1]])
```

`getContext` *get the context of methylation*

Description

The function returns the context of methylation. For example: "CpG","CHH" or "CHG"

Usage

```
getContext(x)

## S4 method for signature 'methylBase'
getContext(x)

## S4 method for signature 'methylRaw'
getContext(x)

## S4 method for signature 'methylDiff'
getContext(x)
```

```
## S4 method for signature 'methylRawDB'
getContext(x)

## S4 method for signature 'methylBaseDB'
getContext(x)

## S4 method for signature 'methylDiffDB'
getContext(x)
```

Arguments

x	an `methylBase`,`methylBaseDB`, `methylRaw`,`methylRawDB` or an `methylDiff` object

Value

a string for the context methylation

Examples

```
data(methylKit)

getContext(methylBase.obj)
getContext(methylDiff.obj)
getContext(methylRawList.obj[[1]])
```

`getCorrelation` *get correlation between samples in methylBase or methylBaseDB object*

Description

The functions returns a matrix of correlation coefficients and/or a set of scatterplots showing the relationship between samples. The scatterplots will contain also fitted lines using `lm()` for linear regression and `lowess` for polynomial regression.

Usage

```
getCorrelation(object,method="pearson",plot=FALSE,nrow)

## S4 method for signature 'methylBase'
getCorrelation(object, method = c("pearson", "kendall", "spearman"), plot)

## S4 method for signature 'methylBaseDB'
getCorrelation(object, method = "pearson", plot = FALSE, nrow = 2e+06)
```

Arguments

`object`	a methylBase or methylBaseDB object
`method`	a character string indicating which correlation coefficient (or covariance) is to be computed (default:"pearson", other options are "kendall" and "spearman")
`plot`	scatterPlot if TRUE (default:FALSE)
`nrow`	a numeric giving the number of lines to read in of methylBaseDB object, defaults to 2e6

Value

a correlation matrix object and plot scatterPlot

Details

The argument 'nrow' is only evaluated if the input is a `methylBaseDB` object. If 'nrow' is not specified `getCorrelation` will read the first 2M records of the given object, but if you want to read all records 'nrow' has to be NULL. You should change 'nrow' if using `getCorrelation` with all records of the methylBaseDB object would take too long.

If the scatter plot is plotted, the red line in the plot is from linear regression fit and the green line is from polynomial regression fit with `stats::lowess`.

Examples

```
data(methylKit)

getCorrelation(methylBase.obj,method="pearson",plot=FALSE)

# create methylBaseDB
methylBaseDB.obj <- unite(methylRawList.obj,save.db=TRUE,dbdir="methylDB")

getCorrelation(methylBaseDB.obj,method="pearson",plot=FALSE,nrow=10000)




# remove Database again
rm(methylBaseDB.obj)
unlink("methylDB",recursive=TRUE)
```

`getCoverageStats` *get coverage stats from methylRaw object*

Description

The function returns basic statistics about read coverage per base. It can also plot a histogram of read coverage values.

Usage

```
getCoverageStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)

## S4 method for signature 'methylRaw'
getCoverageStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)

## S4 method for signature 'methylRawDB'
getCoverageStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)
```

Arguments

`object`	a `methylRaw` or `methylRawDB` object
`plot`	plot a histogram of coverage if TRUE (default:FALSE)
`both.strands`	do stats and plot for both strands if TRUE (default:FALSE)
`labels`	should the bars of the histrogram have labels showing the percentage of values in each bin (default:TRUE)
`...`	options to be passed to `hist` function
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawDB` objects (default: 1e6)

Value

a summary of coverage statistics or plot a histogram of coverage

Details

The parameter `chunk.size` is only used when working with `methylRawDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Examples

```
data(methylKit)

# gets coverage stats for the first sample in methylRawList.obj object
getCoverageStats(methylRawList.obj[[1]],plot=TRUE,
both.strands=FALSE,labels=TRUE)
```

`getData` *get the data slot from the methylKit objects*

Description

The functions retrieves the table containing methylation information from `methylKit` Objects. The data retrived from this function is of a `data.frame`. This is basically containing all relevant methylation information per genomic region or base.

Usage

```
getData(x)

## S4 method for signature 'methylBase'
getData(x)

## S4 method for signature 'methylRaw'
getData(x)

## S4 method for signature 'methylDiff'
getData(x)

## S4 method for signature 'methylRawDB'
getData(x)

## S4 method for signature 'methylBaseDB'
getData(x)

## S4 method for signature 'methylDiffDB'
getData(x)
```

Arguments

x	an `methylBase`,`methylBaseDB`, `methylRaw`,`methylRawDB` or `methylDiff` object

Value

data frame for methylation events

Examples

```
data(methylKit)

# following commands show first lines of returned
# data.frames from getData() function
head(
getData(methylBase.obj)
)

head( getData(methylDiff.obj))

head(getData(methylRawList.obj[[1]]))
```

---

getDBPath *Get path to database of the methylDB objects*

Description

The function returns the path to the flat file database that stores the data of the `methylRawDB`, `methylRawListDB`, `methylBaseDB` or `methylDiffDB` objects.

Usage

```
getDBPath(x)

## S4 method for signature 'methylRawListDB'
getDBPath(x)

## S4 method for signature 'methylBaseDB'
getDBPath(x)

## S4 method for signature 'methylRawDB'
getDBPath(x)

## S4 method for signature 'methylDiffDB'
getDBPath(x)
```

Arguments

x	an `methylBaseDB`,`methylRawDB`, `methylRawListDB` or `methylDiffDB` object

Examples

```
data(methylKit)

methylBaseDB.obj <- unite(methylRawList.obj,save.db=TRUE,dbdir="methylDB")


#The path to the database is returned
getDBPath(methylBaseDB.obj)
```

```
# remove Database again
rm(methylBaseDB.obj)
unlink("methylDB",recursive=TRUE)
```

---

getMethylationStats *get Methylation stats from methylRaw or methylRawDB object*

Description

The function returns basic statistics about It can also plot a histogram of

Usage

```
getMethylationStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)

## S4 method for signature 'methylRaw'
getMethylationStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)

## S4 method for signature 'methylRawDB'
getMethylationStats(
  object,
  plot = FALSE,
  both.strands = FALSE,
  labels = TRUE,
  ...,
  chunk.size = 1e+06
)
```

Arguments

`object`	a `methylRaw` or `methylRawDB` object
`plot`	plot a histogram of Methylation if TRUE (deafult:FALSE)
`both.strands`	do plots and stats for both strands seperately if TRUE (deafult:FALSE)
`labels`	should the bars of the histrogram have labels showing the percentage of values in each bin (default:TRUE)

`...`	options to be passed to `hist` function.
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawDB` objects (default: 1e6)

Value

a summary of Methylation statistics or plot a histogram of coverage

Details

The parameter `chunk.size` is only used when working with `methylRawDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Examples

```
data(methylKit)

# gets Methylation stats for the first sample in methylRawList.obj object
getMethylationStats(methylRawList.obj[[1]],plot=TRUE,
both.strands=FALSE,labels=TRUE)
```

`getMethylDiff` *get differentially methylated regions/bases based on cutoffs*

Description

The function subsets a `methylDiff` or `methylDiffDB` object in order to get differentially methylated bases/regions satisfying thresholds.

Usage

```
getMethylDiff(
  .Object,
  difference = 25,
  qvalue = 0.01,
  type = "all",
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylDiff'
getMethylDiff(
  .Object,
  difference = 25,
  qvalue = 0.01,
  type = "all",
  chunk.size = 1e+06,
```

```
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylDiffDB'
getMethylDiff(
  .Object,
  difference = 25,
  qvalue = 0.01,
  type = "all",
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)
```

Arguments

`.Object`	a `methylDiff` or `methylDiffDB` object
`difference`	cutoff for absolute value of methylation percentage change between test and control (default:25)
`qvalue`	cutoff for qvalue of differential methylation statistic (default:0.01)
`type`	one of the "hyper","hypo" or "all" strings. Specifies what type of differentially menthylated bases/regions should be returned. For retrieving Hyper-methylated regions/bases type="hyper", for hypo-methylated type="hypo" (default:"all")
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylDiffDB` objects (default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: see Details
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_type" to current filename if database already exists or generate new file with filename "methylDiff_type" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

a methylDiff or methylDiffDB object containing the differential methylated locations satisfying the criteria

Details

The parameter `chunk.size` is only used when working with `methylDiffDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylDiffDB`, while being per default FALSE for `methylDiff`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Examples

```
data(methylKit)

# get differentially methylated bases/regions with specific cutoffs
all.diff=getMethylDiff(methylDiff.obj,difference=25,qvalue=0.01,type="all")

# get hyper-methylated
hyper=getMethylDiff(methylDiff.obj,difference=25,qvalue=0.01,type="hyper")

# get hypo-methylated
hypo=getMethylDiff(methylDiff.obj,difference=25,qvalue=0.01,type="hypo")
```

`getSampleID` *Get or Set Sample-IDs of the methylKit objects*

Description

The function returns or replaces the sample-ids stored in any of the following `methylKit` objects: `methylRaw`, `methylRawDB`, `methylBase`, `methylBaseDB`, `methylRawList`, `methylRawListDB`, `methylDiff`, `methylDiffDB`.

Usage

```
getSampleID(x)
getSampleID(x) <- value

getSampleID(x) <- value

## S4 method for signature 'methylRawList'
getSampleID(x)

## S4 replacement method for signature 'methylRawList'
getSampleID(x) <- value

## S4 method for signature 'methylBase'
getSampleID(x)

## S4 replacement method for signature 'methylBase'
getSampleID(x) <- value

## S4 method for signature 'methylRaw'
getSampleID(x)

## S4 replacement method for signature 'methylRaw'
getSampleID(x) <- value

## S4 method for signature 'methylDiff'
```

```
getSampleID(x)

## S4 replacement method for signature 'methylDiff'
getSampleID(x) <- value

## S4 method for signature 'methylRawListDB'
getSampleID(x)

## S4 replacement method for signature 'methylRawListDB'
getSampleID(x) <- value

## S4 method for signature 'methylBaseDB'
getSampleID(x)

## S4 replacement method for signature 'methylBaseDB'
getSampleID(x) <- value

## S4 method for signature 'methylRawDB'
getSampleID(x)

## S4 replacement method for signature 'methylRawDB'
getSampleID(x) <- value

## S4 method for signature 'methylDiffDB'
getSampleID(x)

## S4 replacement method for signature 'methylDiffDB'
getSampleID(x) <- value
```

Arguments

`x`	an `methylBaseDB`,`methylRawListDB` or `methylDiffDB` object
`value`	a valid replacement vector for the sample-ids of the object

Examples

```
data(methylKit)

#The Sample-Ids can be printed ..
getSampleID(methylBase.obj)

# .. or replaced.
newObj <- methylBase.obj
getSampleID(newObj) <- c("sample1","sample2","sample3","sample4")
getSampleID(newObj)
```

`getTreatment` *Get or Set treatment vector of methylKit object*

Description

The function returns or replaces the treatment vector stored in any of the following methylKit objects: `methylBase`,`methylRawList`,`methylBaseDB`, `methylRawListDB`,`methylDiff`,`methylDiffDB`.

Usage

```
getTreatment(x)
getTreatment(x) <- value

getTreatment(x) <- value

## S4 method for signature 'methylRawList'
getTreatment(x)

## S4 replacement method for signature 'methylRawList'
getTreatment(x) <- value

## S4 method for signature 'methylBase'
getTreatment(x)

## S4 replacement method for signature 'methylBase'
getTreatment(x) <- value

## S4 method for signature 'methylDiff'
getTreatment(x)

## S4 replacement method for signature 'methylDiff'
getTreatment(x) <- value

## S4 method for signature 'methylRawListDB'
getTreatment(x)

## S4 replacement method for signature 'methylRawListDB'
getTreatment(x) <- value

## S4 method for signature 'methylBaseDB'
getTreatment(x)

## S4 replacement method for signature 'methylBaseDB'
getTreatment(x) <- value

## S4 method for signature 'methylDiffDB'
getTreatment(x)

## S4 replacement method for signature 'methylDiffDB'
getTreatment(x) <- value
```

Arguments

`x`	a `methylKit` object
`value`	a valid replacement for the treatment vector of the object

Examples

```
data(methylKit)

# The treatment vector can be printed ..
getTreatment(methylBase.obj)

# .. or replaced with a new one
newObj <- methylBase.obj
getTreatment(newObj) <- c(1,2,3,4)
getTreatment(newObj)
```

`joinSegmentNeighbours` *Join directly neighbouring segments produced by methSeg*

Description

Segmentation and clustering are two distinct steps in methSeg(), leading to adjacent segments of the same class. This leads to a bias segment length distributions, which is removed by joining those neighbours.

Usage

```
joinSegmentNeighbours(res)
```

Arguments

`res`	A `GRanges` object with segment classification and information prudoced by `methSeg`

Value

A `GRanges` object with segment classification and information.

See Also

`methSeg`

`makeMethylDB` *coerce methylKit objects from memory to flat file database objects*

Description

The function converts in-memory methylKit objects to methylDB objects

Usage

```
makeMethylDB(obj, dbdir = getwd())

## S4 method for signature 'methylBase'
makeMethylDB(obj, dbdir = getwd())

## S4 method for signature 'methylRaw'
makeMethylDB(obj, dbdir = getwd())

## S4 method for signature 'methylDiff'
makeMethylDB(obj, dbdir = getwd())

## S4 method for signature 'methylRawList'
makeMethylDB(obj, dbdir = getwd())
```

Arguments

`obj`	an `methylBase`, `methylRaw`, `methylRawList` or an `methylDiff` object
`dbdir`	directory where flat file database(s) should be stored, defaults to getwd(), working directory.

Value

an `methylBaseDB`,`methylRawDB`, `methylRawListDB` or an `methylDiffDB` object

Examples

```
## Not run:
data(methylKit)


makeMethylDB(methylBase.obj,"my/path")

## End(Not run)
```

`methRead` *read file(s) to methylRaw or methylRawList objects*

Description

The function reads a list of files or single files with methylation information for bases/region in the genome and creates a methylrawList or methylraw object. The information can be stored as flat file database by creating a methylrawlistDB or methylrawDB object.

Usage

```
methRead(
  location,
  sample.id,
  assembly,
```

```
  dbtype = NA,
  pipeline = "amp",
  header = TRUE,
  skip = 0,
  sep = "\t",
  context = "CpG",
  resolution = "base",
  treatment = NA,
  dbdir = getwd(),
  mincov = 10
)

## S4 method for signature 'character,character,character'
methRead(
  location,
  sample.id,
  assembly,
  dbtype,
  pipeline,
  header,
  skip,
  sep,
  context,
  resolution,
  dbdir,
  mincov
)

## S4 method for signature 'character,ANY,ANY'
methRead(
  location,
  sample.id,
  assembly,
  dbtype,
  pipeline,
  header,
  skip,
  sep,
  context,
  resolution,
  dbdir,
  mincov
)

## S4 method for signature 'list,list,character'
methRead(
  location,
  sample.id,
  assembly,
  dbtype = NA,
  pipeline = "amp",
  header = TRUE,
```

```
  skip = 0,
  sep = "\t",
  context = "CpG",
  resolution = "base",
  treatment = NA,
  dbdir = getwd(),
  mincov = 10
)

## S4 method for signature 'list,ANY,ANY'
methRead(
  location,
  sample.id,
  assembly,
  dbtype = NA,
  pipeline = "amp",
  header = TRUE,
  skip = 0,
  sep = "\t",
  context = "CpG",
  resolution = "base",
  treatment = NA,
  dbdir = getwd(),
  mincov = 10
)
```

Arguments

`location`	file location(s), either a list of locations (each a character string) or one location string
`sample.id`	sample.id(s)
`assembly`	a string that defines the genome assembly such as hg18, mm9. this is just a string for book keeping. It can be any string. Although, when using multiple files from the same assembly, this string should be consistent in each object.
`dbtype`	type of the flat file database, currently only option other than NA is "tabix". When "tabix" is given the objects are stored in tabix files, which are compressed and indexed. The default value is NA, in which case the objects are stored in memory.
`pipeline`	name of the alignment pipeline, it can be either "amp", "bismark","bismarkCoverage", "bismarkCytosineReport" or a list (default:'amp'). The methylation text files generated from other pipelines can be read as generic methylation text files by supplying a named `list` argument as "pipeline" argument. The named `list` should containt column numbers which denotes which column of the text file corresponds to values and genomic location of the methylation events. See Details for more on possible values for this argument.
`header`	if the input file has a header or not (default: TRUE)
`skip`	number of lines to skip when reading. Can be set to 1 for bed files with track line (default: 0)
`sep`	seperator between fields, same as `read.table` argument (default: "\t")
`context`	methylation context string, ex: CpG,CHG,CHH, etc. (default:CpG)

`resolution`	designates whether methylation information is base-pair resolution or regional resolution. allowed values ’base’ or ’region’. Default ’base’
`treatment`	a vector contatining 0 and 1 denoting which samples are control which samples are test
`dbdir`	directory where flat file database(s) should be stored, defaults to getwd(), working directory.
`mincov`	minimum read coverage to be included in the methylKit objects. defaults to 10. Any methylated base/region in the text files below the mincov value will be ignored.

Value

returns methylRaw, methylRawList, methylRawDB, methylRawListDB object

Details

The output of `methRead` is determined by specific input arguments,as there are `location`, `sample.id`, `assembly` and `dbtype`. The first three are obligatory, while if the last argument is given database features are enabled. If then `location` refers to an uncompressed file the function will create a flat file database and the associated methylRawDB object will link to this database. If then `location` refers to an earlier created database file then the object will directly link to this database, skipping the preprocessing steps.

When `pipeline` argument is a list, it is exptected to provide a named list with following names. ’fraction’ is a logical value, denoting if the column frequency of Cs has a range from [0-1] or [0-100]. If true it assumes range is [0-1]. ’chr.col" is the number of the column that has chrosome string. ’start.col’ is the number of the column that has start coordinate of the base/region of the methylation event. ’end.col’ is the number of the column that has end coordinate of the base/region of the methylation event. ’coverage.col’ is the number of the column that has read coverage values. ’strand.col’ is the number of the column that has strand information, the strand information in the file has to be in the form of ’+’ or ’-’, ’freqC.col’ is the number of the column that has the frequency of Cs. See examples to see how to read a generic methylation text file.

Other possible values for `pipeline` argument are ’amp’,’bismark’, ’bismarkCoverage’ and ’bismarkCytosineReport’. For ’amp’ and ’bismark’ the function expects a tabular format shown in the webpage (http://github.com/al2na/methylKit). "amp" and "bismark" expect identical input and are kept for historical reasons. ’amp’ was a pipeline used in Akalin et al. 2012 Plos Genetics paper, publicly available in googlecode.

Bismark aligner can output methylation information per base in multiple formats. With `pipeline='bismarkCoverage'`, the function reads bismark coverage files, which have chr,start,end, number of cytosines (methylated bases) and number of thymines (unmethylated bases) format. If bismark coverage files are used the function will not have the strand information,so beware of that fact. With `pipeline='bismarkCytosineReport'`, the function expects cytosine report files from Bismark, which have chr,start, strand, number of cytosines (methylated bases) , number of thymines (unmethylated bases),context and trinucletide context format.

The function can also read gzipped files. On unix systems, this is achieved by using `zcat filename` and feeding that into `data.table::fread` . On Windows, the file is first uncompressed then read into R using `data.table::fread`.

Examples

```
# this is a list of example files, ships with the package
# for your own analysis you will just need to provide set of paths to files
```

```
# you will not need the "system.file(..."  part
file.list=list(
         system.file("extdata", "test1.myCpG.txt", package = "methylKit"),
         system.file("extdata", "test2.myCpG.txt", package = "methylKit"),
         system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
         system.file("extdata", "control2.myCpG.txt", package = "methylKit")
         )

# read the files to a methylRawList object: myobj
myobj=methRead(file.list,
            sample.id=list("test1","test2","ctrl1","ctrl2"),
            assembly="hg18",treatment=c(1,1,0,0))

# read one file as methylRaw object
myobj=methRead( file.list[[1]],
            sample.id="test1",assembly="hg18")

# read a generic text file containing CpG methylation values
# let's first look at the content of the file
generic.file=system.file("extdata", "generic1.CpG.txt",package = "methylKit")
read.table(generic.file,header=TRUE)

# And this is how you can read that generic file as a methylKit object
 myobj=methRead( generic.file,
             pipeline=list(fraction=FALSE,chr.col=1,start.col=2,end.col=2,
                           coverage.col=4,strand.col=3,freqC.col=5),
             sample.id="test1",assembly="hg18")

# This creates tabix files that save methylation data
# Without specified dbdir first creates a folder named the following
# in working directory:
# paste("methylDB",Sys.Date(),paste(sample(c(0:9, letters, LETTERS),3,
# replace=TRUE),collapse=""))
#
# Then, saves tabix files from methylKit objects there
 myobj=methRead( file.list,
               sample.id=list("test1","test2","ctrl1","ctrl2"),
               assembly="hg18",treatment=c(1,1,0,0),
               dbtype="tabix")

# This creates a single tabix files that saves methylation data
# first creates a "methylDB_objects" directory
# Then, saves tabix file from methylKit objects there
 myobj=methRead(file.list[[1]],
               sample.id="test1",
               assembly="hg18",
               dbtype="tabix",dbdir="methylDB_objects")

 # tidy up
 rm(myobj)
 unlink(list.files(pattern = "methylDB",full.names = TRUE),recursive = TRUE)
```

`methSeg` *Segment methylation or differential methylation profile*

Description

The function uses a segmentation algorithm (`fastseg`) to segment the methylation profiles. Following that, it uses gaussian mixture modelling to cluster the segments into k components. This process uses mean methylation value of each segment in the modeling phase. Each component ideally indicates quantitative classification of segments, such as high or low methylated regions.

Usage

```
methSeg(
  obj,
  diagnostic.plot = TRUE,
  join.neighbours = FALSE,
  initialize.on.subset = 1,
  ...
)
```

Arguments

`obj`	`GRanges`, `methylDiff`, `methylDiffDB`, `methylRaw` or `methylRawDB` . If the object is a `GRanges` it should have one meta column with methylation scores and has to be sorted by position, i.e. ignoring the strand information.
`diagnostic.plot`	if TRUE a diagnostic plot is plotted. The plot shows methylation and length statistics per segment group. In addition, it shows diagnostics from mixture modeling: the density function estimated and BIC criterion used to decide the optimum number of components in mixture modeling.
`join.neighbours`	if TRUE neighbouring segments that cluster to the same seg.group are joined by extending the ranges, summing up num.marks and averaging over seg.means.
`initialize.on.subset`	a numeric value indicating either percentage or absolute value of regions to subsample from segments before performing the mixture modeling. The value can be either between 0 and 1, e.g. 0.1 means that 10 integer higher than 1 to define the number of regions to sample. By default uses the whole dataset, which can be time consuming on large datasets. (Default: 1)
`...`	arguments to `fastseg` function in fastseg package, or to `densityMclust` in Mclust package, could be used to fine tune the segmentation algorithm. E.g. Increasing "alpha" will give more segments. Increasing "cyberWeight" will give also more segments."maxInt" controls the segment extension around a breakpoint. "minSeg" controls the minimum segment length. "G" argument denotes number of components used in BIC selection in mixture modeling. For more details see fastseg and Mclust documentation.

Details

To be sure that the algorithm will work on your data, the object should have at least 5000 records

After initial segmentation with fastseg(), segments are clustered into self-similar groups based on their mean methylation profile using mixture modeling. Mixture modeling estimates the initial parameters of the distributions by using the whole dataset. If "initialize.on.subset" argument set as described, the function will use a subset of the data to initialize the parameters of the mixture modeling prior to the Expectation-maximization (EM) algorithm used by `Mclust` package.

Value

A `GRanges` object with segment classification and information. 'seg.mean' column shows the mean methylation per segment. 'seg.group' column shows the segment groups obtained by mixture modeling

Author(s)

Altuna Akalin, contributions by Arsene Wabo and Katarzyna Wreczycka

See Also

`methSeg2bed`, `joinSegmentNeighbours`

Examples

```
 download.file(
 "https://raw.githubusercontent.com/BIMSBbioinfo/compgen2018/master/day3_diffMeth/data/H1.chr21.chr22.rds",
 destfile="H1.chr21.chr22.rds",method="curl")

 mbw=readRDS("H1.chr21.chr22.rds")

 # it finds the optimal number of componets as 6
 res=methSeg(mbw,diagnostic.plot=TRUE,maxInt=100,minSeg=10)

 # however the BIC stabilizes after 4, we can also try 4 componets
 res=methSeg(mbw,diagnostic.plot=TRUE,maxInt=100,minSeg=10,G=1:4)

 # get segments to BED file
 methSeg2bed(res,filename="H1.chr21.chr22.trial.seg.bed")


unlink(list.files(pattern="H1.chr21.chr22",full.names=TRUE))
```

`methSeg2bed` *Export segments to BED files*

Description

The segments are color coded based on their score (methylation or differential methylation value). They are named by segment group (components in mixture modeling) and the score in the BED file is obtained from 'seg.mean' column of segments object.

Usage

```
methSeg2bed(
  segments,
  filename,
 trackLine = "track name='meth segments' description='meth segments' itemRgb=On",
  colramp = colorRamp(c("gray", "green", "darkgreen"))
)
```

Arguments

`segments`	GRanges object with segment classification and information. This should be the result of methSeg function
`filename`	name of the output data
`trackLine`	UCSC browser trackline
`colramp`	color scale to be used in the BED display defaults to gray,green, darkgreen scale.

Value

A BED files with the segmented data which can be visualized in the UCSC browser

See Also

`methSeg`

`methylBase-class` *An S4 class for methylation events sampled in multiple experiments*

Description

This class is designed to contain methylation information such as coverage, number of methylated bases, etc.. The methylation events contained in the class must be sampled in multiple experiments (ex: only CpG bases covered in multiple experiments are stored in the object of this class). The class extends `data.frame` and creates an object that holds methylation information and genomic location. The object belonging to this class is produced by `unite` function.

Slots

`sample.ids`: character vector for ids of samples in the object

`assembly`: name of the genome assembly

`context`: context of methylation. Ex: CpG,CpH,CHH, etc

`treatment`: treatment vector denoting which samples are test and control

`coverage.index`: vector denoting which columns in the data correspons to coverage values

`numCs.index`: vector denoting which columns in the data correspons to number of methylatedCs values

`numTs.index`: vector denoting which columns in the data correspons to number of unmethylated Cs values

`destranded`: logical value. If `TRUE` object is destranded, if `FALSE` it is not.

`resolution`: resolution of methylation information, allowed values: ’base’ or ’region’

Details

`methylBase` class extends `data.frame` class therefore providing novice and experienced R users with a data structure that is well known and ubiquitous in many R packages.

Subsetting

In the following code snippets, `x` is a `methylBase`. Subsetting by `x[i,]` will produce a new object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[.

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Coercion

`methylBase` object can be coerced to `GRanges` object via `as` function.

Examples

```
data(methylKit)
library(GenomicRanges)
my.gr=as(methylBase.obj,"GRanges")
```

`methylBase.obj` *Example methylBase object.*

Description

`methylBase`, `methylDiff` and `methylRawList`. You can load the data using `data(methylKit)`

Format

methylBase.obj object stores the location and methylation information for bases that are covered in all samples. `methylBase` partially extends `data.frame` S3 class.

`methylBaseDB-class` *An S4 class for storing methylation events sampled in multiple experiments as flat file database*

Description

This class is designed to contain methylation information such as coverage, number of methylated bases, etc... The class creates an object that holds methylation information and genomic location as flat file database. The object belonging to this class is produced by `unite` function.

Slots

`dbpath`: path to flat file database(s)

`num.records`: number of records (lines) in the object

`sample.ids`: character vector for ids of samples in the object

`assembly`: name of the genome assembly

`context`: context of methylation. Ex: CpG,CpH,CHH, etc

`treatment`: treatment vector denoting which samples are test and control

`coverage.index`: vector denoting which columns in the data correspond to coverage values

`numCs.index`: vector denoting which columns in the data correspond to number of methylatedCs values

`numTs.index`: vector denoting which columns in the data correspond to number of unmethylated Cs values

`destranded`: logical value. If `TRUE` object is destranded, if `FALSE` it is not.

`resolution`: resolution of methylation information, allowed values: 'base' or 'region'

`dbtype`: string for type of the flat file database, ex: tabix

Details

`methylBaseDB` class has the same functionality as `methylBase` class, but the data is saved in a flat database file and therefore allocates less space in memory.

Subsetting

In the following code snippets, `x` is a `methylBaseDB`. Subsetting by `x[i,]` will produce a new `methylBase` object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[.

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Coercion

`methylBaseDB` object can be coerced to: `GRanges` object via `as` function. `methylBase` object via `as` function.

Examples

```
data(methylKit)
methylBaseDB.obj <- unite(methylRawList.obj,save.db=TRUE,dbdir="methylDB")
library(GenomicRanges)
my.gr=as(methylBaseDB.obj,"GRanges")


# remove Database again
rm(methylBaseDB.obj)
unlink("methylDB",recursive=TRUE)
```

`methylDiff-class` *An S4 class that holds differential methylation information*

Description

This class is designed to hold statistics and locations for differentially methylated regions/bases. It extends `data.frame` class. `calculateDiffMeth` function returns an object with `methylDiff` class.

Slots

`sample.ids` ids/names of samples in a vector

`assembly` a name of genome assembly, such as :hg18,mm9, etc

`context` numeric vector identifying which samples are which group

`treatment` numeric vector identifying which samples are which group

`destranded` logical denoting if methylation inormation is destranded or not

`resolution` string either 'base' or 'region' defining the resolution of methylation information

`.Data` data.frame holding the locations and statistics

Details

`methylDiff` class extends `data.frame` class therefore providing novice and experienced R users with a data structure that is well known and ubiquitous in many R packages.

Subsetting

In the following code snippets, `x` is a `methylDiff` object. Subsetting by `x[i,]` will produce a new object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[.

Coercion

`methylDiff` object can be coerced to `GRanges` object via `as` function.

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Examples

```
data(methylKit)
library(GenomicRanges)
my.gr=as(methylDiff.obj,"GRanges")
```

`methylDiff.obj` *Example methylKit objects.*

Description

`methylBase`, `methylDiff` and `methylRawList`. You can load the data using `data(methylKit)`

Format

The Differential methylation information is stored in methylDiff.obj object. `methylBase` partially extends `data.frame` S3 class.

`methylDiffDB-class` *An S4 class that holds differential methylation information as flat file database*

Description

This class is designed to hold statistics and locations for differentially methylated regions/bases as flat file database. `calculateDiffMeth` function returns an object with `methylDiffDB` class.

Slots

`dbpath`: path to flat file database(s)

`num.records`: number of records (lines) in the object

`sample.ids` ids/names of samples in a vector

`assembly` a name of genome assembly, such as :hg18,mm9, etc

`context` numeric vector identifying which samples are which group

`treatment` numeric vector identifying which samples are which group

`destranded` logical denoting if methylation inormation is destranded or not

`resolution` string either 'base' or 'region' defining the resolution of methylation information

`dbtype`: string for type of the flat file database, ex: tabix

Details

`methylDiffDB` class has the same functionality as `methylDiff` class, but the data is saved in a flat database file and therefore allocates less space in memory.

Subsetting

In the following code snippets, `x` is a `methylDiffDB`. Subsetting by `x[i,]` will produce a new object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[.

Coercion

`methylDiffDB` object can be coerced to: `GRanges` object via `as` function. `methylDiff` object via `as` function.

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Examples

```
data(methylKit)

methylDiffDB.obj <- calculateDiffMeth(methylBase.obj,save.db=TRUE,dbdir="methylDB")

library(GenomicRanges)
my.gr=as(methylDiffDB.obj,"GRanges")

# remove Database again
rm(methylDiffDB.obj)
unlink("methylDB",recursive=TRUE)
```

`methylKit-defunct` *Deprecated/Defunct functions*

Description

These are deprecated or defunct functions. Most of them are replaced by genomation functions. See the vignette for examples on how to use genomation functions for annotation purposes.

Usage

```
annotate.WithFeature()

annotate.WithFeature.Flank()

annotate.WithGenicParts()

read.bed()

read.feature.flank()

read.transcript.features()

getFeatsWithTargetsStats()

getFlanks()

getMembers()
```

`getTargetAnnotationStats()`

`plotTargetAnnotation()`

`read()`

`read.bismark()`

`adjust.methylC()`

`get.methylDiff()`

`methylRaw-class` *An S4 class for holding raw methylation data from an alignment pipeline.*

Description

This object stores the raw mehylation data that is read in through read function and extends `data.frame`.The raw methylation data is basically percent methylation values and read coverage values per genomic base/region.

Slots

`sample.id`: string for an identifier of the sample

`assembly`: string for genome assembly, ex: hg18,hg19,mm9

`context`: methylation context string, ex: CpG,CpH,CHH, etc.

`resolution`: resolution of methylation information, ’base’ or ’region’

Details

`methylRaw` class extends `data.frame` class therefore providing novice and experienced R users with a data structure that is well known and ubiquitous in many R packages.

Subsetting

In the following code snippets, `x` is a `methylRaw`. Subsetting by `x[i,]` will produce a new object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[.

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Coercion

`methylRaw` object can be coerced to `GRanges` object via `as` function.

Examples

```
# example of a raw methylation data as a text file
read.table(system.file("extdata", "control1.myCpG.txt",
                                package = "methylKit"),
                                    header=TRUE,nrows=5)

data(methylKit)

# example of a methylRaw object
head(methylRawList.obj[[1]])
str(head(methylRawList.obj[[1]]))

library(GenomicRanges)

#coercing methylRaw object to GRanges object
my.gr=as(methylRawList.obj[[1]],"GRanges")
```

`methylRawDB-class` *An S4 class for storing raw methylation data as flat file database.*

Description

This object stores the raw mehylation data that is read in through read function as flat file database.The raw methylation data is basically percent methylation values and read coverage values per genomic base/region.

Slots

`dbpath`: path to flat file database

`num.records`: number of records (lines) in the object

`sample.id`: string for an identifier of the sample

`assembly`: string for genome assembly, ex: hg18,hg19,mm9

`context`: methylation context string, ex: CpG,CpH,CHH, etc.

`resolution`: resolution of methylation information, 'base' or 'region'

`dbtype`: string for type of the flat file database, ex: tabix

Details

`methylRawDB` is created via `read` function and has the same functionality as `methylRaw` class, but the data is saved in a flat database file and therefore allocates less space in memory.

Subsetting

In the following code snippets, `x` is a `methylRawDB`. Subsetting by `x[i,]` will produce a new `methylRaw` object if subsetting is done on rows. Column subsetting is not directly allowed to prevent errors in the downstream analysis. see ?methylKit[. `x[]` will return the `methylRawDB` object as new `methylRaw` object

Accessors

The following functions provides access to data slots of methylDiffDB: - `getData`: get the data slot from the methylKit objects, - `getAssembly`: get assembly of the genome, - `getContext`: get the context of methylation

Coercion

`methylRawDB` object can be coerced to: `GRanges` object via `as` function. `methylRaw` object via `as` function.

Examples

```
# example of a raw methylation data contained as a text file
read.table(system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
header=TRUE,nrows=5)


methylRawDB.obj <- methRead(
system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
                        sample.id = "ctrl1", assembly = "hg18",
                        dbtype = "tabix", dbdir = "methylDB")

# example of a methylRawDB object
methylRawDB.obj
str(methylRawDB.obj)

library(GenomicRanges)

#coercing methylRawDB object to GRanges object
my.gr=as(methylRawDB.obj,"GRanges")

#coercing methylRawDB object to methylRaw object
myRaw=as(methylRawDB.obj,"methylRaw")

# remove Database again
rm(methylRawDB.obj)
unlink("methylDB",recursive=TRUE)
```

`methylRawList-class` *An S4 class for holding a list of methylRaw objects.*

Description

This class stores the list of `methylRaw` objects. Functions such as `lapply` can be used on this list. It extends `list` class. This object is primarily produced by `methRead` function.

Usage

```
methylRawList(..., treatment)
```

Arguments

`...`	vector of methylRaw objects
`treatment`	vector of treatment values

Slots

`treatment` numeric vector denoting control and test samples

`.Data` a list of `methylRaw` objects

Constructor

`methylRawList(...)` combine multiple methylRaw objects supplied in ... into a methylRawList object.

Examples

```
data(methylKit)

#applying functions designed for methylRaw on methylRawList object
lapply(methylRawList.obj,"getAssembly")
```

`methylRawList.obj` *Example methylRawList object.*

Description

`methylBase`, `methylDiff` and `methylRawList`. You can load the data using `data(methylKit)`

Format

Methylation data from multiple the samples regardless of common coverage are stored in methylRawList.obj object. `methylRawList` extends `list` S3 class

---

`methylRawListDB-class` *An S4 class for holding a list of methylRawDB objects.*

Description

This class stores the list of `methylRawDB` objects. Functions such as `lapply` can be used on this list. It extends `list` class. This object is primarily produced by `methRead` function.

Usage

```
methylRawListDB(..., treatment)
```

Arguments

`...`	vector of methylRawDB files
`treatment`	vector of treatment values

Slots

`treatment` numeric vector denoting control and test samples

`.Data` a list of `methylRawDB` objects

Constructor

`methylRawListDB(...)` combine multiple methylRawDB objects supplied in ... into a methylRawListDB object.

Examples

```
file.list=list( system.file("extdata", "test1.myCpG.txt",
package = "methylKit"),
                system.file("extdata", "test2.myCpG.txt",
                package = "methylKit"),
                system.file("extdata", "control1.myCpG.txt",
                package = "methylKit"),
                system.file("extdata", "control2.myCpG.txt",
                package = "methylKit") )

methylRawListDB.obj <- methRead(file.list,
sample.id = list("test1","test2","ctrl1","ctrl2"),
                            assembly = "hg18",treatment = c(1,1,0,0),
                            dbtype = "tabix",dbdir = "methylDB")

#applying functions designed for methylRawDB on methylRawListDB object
lapply(methylRawListDB.obj,"getAssembly")


# remove Database again
rm(methylRawListDB.obj)
unlink("methylDB",recursive=TRUE)
```

---

normalizeCoverage *normalize read coverage between samples*

Description

The function normalizes coverage values between samples using a scaling factor derived from differences between mean or median of coverage distributions

Usage

```
normalizeCoverage(obj,method="median",chunk.size,save.db,...)

## S4 method for signature 'methylRawList'
normalizeCoverage(
  obj,
  method = "median",
  chunk.size = 1e+06,
  save.db = FALSE,
```

```
  ...
)

## S4 method for signature 'methylRawListDB'
normalizeCoverage(
  obj,
  method = "median",
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)
```

Arguments

`obj`	`methylRawList` or `methylRawListDB` object
`method`	a string "mean" or "median" which denotes median or mean should be used to calculate scaling factor. (Default:median)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawListDB` objects. (Default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_filtered" to current filename if database already exists or generate new file with filename "sampleID_filtered" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database

Value

a `methylRawList` or `methylRawList` object depending on class of input object

Details

The parameter `chunk.size` is only used when working with `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawListDB`, while being per default FALSE for `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Author(s)

Altuna Akalin

Examples

```
data(methylKit)
# normalize by the median coverage
newObj = normalizeCoverage(methylRawList.obj,method="median")

# normalize by mean coverage and save to database in folder methylDB
newDBObj = normalizeCoverage(methylRawList.obj,method="mean",
save.db=TRUE,dbdir="methylDB")
```

`PCASamples` *Principal Components Analysis of Methylation data*

Description

The function does a PCA analysis using `prcomp` function using percent methylation matrix as an input.

Usage

```
PCASamples(.Object, screeplot=FALSE, adj.lim=c(0.0004,0.1), scale=TRUE,
center=TRUE,comp=c(1,2),transpose=TRUE,sd.filter=TRUE,
           sd.threshold=0.5,filterByQuantile=TRUE,obj.return=FALSE,chunk.size)

## S4 method for signature 'methylBase'
PCASamples(
  .Object,
  screeplot,
  adj.lim,
  scale,
  center,
  comp,
  transpose,
  sd.filter,
  sd.threshold,
  filterByQuantile,
  obj.return
)

## S4 method for signature 'methylBaseDB'
PCASamples(
  .Object,
  screeplot = FALSE,
  adj.lim = c(4e-04, 0.1),
  scale = TRUE,
  center = TRUE,
  comp = c(1, 2),
  transpose = TRUE,
  sd.filter = TRUE,
  sd.threshold = 0.5,
  filterByQuantile = TRUE,
```

```
  obj.return = FALSE,
  chunk.size = 1e+06
)
```

Arguments

`.Object`	a `methylBase` or `methylBaseDB` object
`screeplot`	a logical value indicating whether to plot the variances against the number of the principal component. (default: FALSE)
`adj.lim`	a vector indicating the propotional adjustment of xlim (adj.lim[1]) and ylim (adj.lim[2]). This is primarily used for adjusting the visibility of sample labels on the on the PCA plot. (default: c(0.0004,0.1))
`scale`	logical indicating if `prcomp` should scale the data to have unit variance or not (default: TRUE)
`center`	logical indicating if `prcomp` should center the data or not (default: TRUE)
`comp`	vector of integers with 2 elements specifying which components to be plotted.
`transpose`	if TRUE (default) percent methylation matrix will be transposed, this is equivalent to doing PCA on variables that are regions/bases. The resulting plot will location of samples in the new coordinate system if FALSE the variables for the matrix will be samples and the resulting plot whill show how each sample (variable) contributes to the principle component.the samples that are highly correlated should have similar contributions to the principal components.
`sd.filter`	If `TRUE`, the bases/regions with low variation will be discarded prior to PCA (default:TRUE)
`sd.threshold`	A numeric value. If `filterByQuantile` is `TRUE`, the value should be between 0 and 1 and the features whose standard deviations is less than the quantile denoted by `sd.threshold` will be removed. If `filterByQuantile` is `FALSE`, then features whose standard deviations is less than the value of `sd.threshold` will be removed.(default:0.5)
`filterByQuantile`	A logical determining if `sd.threshold` is to be interpreted as a quantile of all standard deviation values from bases/regions (the default), or as an absolute value
`obj.return`	if the result of `prcomp` function should be returned or not. (Default:FALSE)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawListDB` objects, default: 1e6

Value

The form of the value returned by `PCASamples` is the summary of principal component analysis by `prcomp`.

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Note

cor option is not in use anymore, since `prcomp` is used for PCA analysis instead of `princomp`

Examples

```
data(methylKit)

# do PCA with filtering rows with low variation, filter rows with standard
# deviation lower than the 50th percentile of Standard deviation distribution
PCASamples(methylBase.obj,screeplot=FALSE, adj.lim=c(0.0004,0.1),
           scale=TRUE,center=TRUE,comp=c(1,2),transpose=TRUE,sd.filter=TRUE,
           sd.threshold=0.5,filterByQuantile=TRUE,obj.return=FALSE)
```

`percMethylation` *get percent methylation scores from methylBase or methylBaseDB object*

Description

get percent methylation scores from methylBase or methylBaseDB object

Usage

```
percMethylation(
  methylBase.obj,
  rowids = FALSE,
  save.txt = FALSE,
  chunk.size = 1e+06
)

## S4 method for signature 'methylBase'
percMethylation(methylBase.obj, rowids = FALSE)

## S4 method for signature 'methylBaseDB'
percMethylation(
  methylBase.obj,
  rowids = FALSE,
  save.txt = FALSE,
  chunk.size = 1e+06
)
```

Arguments

`methylBase.obj`	a methylBase or methylBaseDB object
`rowids`	if TRUE, matrix rownames have identifiers as base/region location (default:FALSE)
`save.txt`	if TRUE, the matrix will be written to a text file, but only for methylBaseDB objects (default: FALSE)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` objects (default: 1e6)

Value

matrix with percent methylation values per base/region across all samples, row names would be base/region identifiers

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

Examples

```
data(methylKit)
mat=percMethylation(methylBase.obj)
head(mat)
```

`pool` *Pool replicates within groups to a single sample per group*

Description

The function sums up coverage, numCs and numTs values within each group so one representative sample for each group will be created in a new methylBase object

Usage

```
pool(obj, sample.ids, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'methylBase'
pool(obj, sample.ids, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'methylBaseDB'
pool(obj, sample.ids, chunk.size = 1e+06, save.db = TRUE, ...)
```

Arguments

`obj`	methylBase or methylBaseDB object with two groups or more and each group should have multiple samples
`sample.ids`	a character vector of new sample.ids ex:c("test","control"), should follow the same order as unique treatment vector, and should be equal to the length of the unique treatment vector
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawListDB` objects, default: 1e6
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections

...	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: The default suffix is a 13-character random string appended to the fixed prefix “methylBase”, e.g. “methylBase_16d3047c1a254.txt.bgz” `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

a methylBase or methylBaseDB object depending on class of input object

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylBaseDB`, while being per default FALSE for `methylBase`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Author(s)

Altuna Akalin

Examples

```
data(methylKit)

# methylBase.obj has two groups, each group has two samples,
# the following function will pool the samples in each group
# so that each group will be represented by one pooled sample
pooled.methylBase=pool(methylBase.obj,sample.ids=c("test","control"))
```

`processBismarkAln` *Get methylation percentage from sorted Bismark alignments*

Description

The function calls methylation percentage per base from sorted Bismark SAM or BAM files and reads methylation information as methylKit objects. Bismark is a popular aligner for high-throughput bisulfite sequencing experiments and it outputs its results in SAM format by default, and can be converted to BAM. Bismark SAM/BAM format contains aligner specific tags which are absolutely necessary for methylation percentage calling using `processBismarkAln`. SAM/BAM files from other aligners will not work with this function.

Usage

```
processBismarkAln(
  location,
  sample.id,
  assembly,
  save.folder = NULL,
  save.context = c("CpG"),
  read.context = "CpG",
  nolap = FALSE,
  mincov = 10,
  minqual = 20,
  phred64 = FALSE,
  treatment = NULL,
  save.db = FALSE,
  verbose = 1
)

## S4 method for signature 'character,character,character'
processBismarkAln(
  location,
  sample.id,
  assembly,
  save.folder = NULL,
  save.context = c("CpG"),
  read.context = "CpG",
  nolap = FALSE,
  mincov = 10,
  minqual = 20,
  phred64 = FALSE,
  treatment = NULL,
  save.db = FALSE,
  verbose = 1
)

## S4 method for signature 'list,list,character'
processBismarkAln(
  location,
  sample.id,
  assembly,
  save.folder = NULL,
  save.context = c("CpG"),
  read.context = "CpG",
  nolap = FALSE,
  mincov = 10,
  minqual = 20,
  phred64 = FALSE,
  treatment = NULL,
  save.db = FALSE,
  verbose = 1
)
```

Arguments

`location`	location of sam or bam file(s). If multiple files are given this argument must be a list.
`sample.id`	the id(s) of samples in the same order as file. If multiple sam files are given this arugment must be a list.
`assembly`	string that determines the genome assembly. Ex: mm9,hg18 etc. This is just a string for book keeping. It can be any string. Although, when using multiple files from the same assembly, this string should be consistent in each object.
`save.folder`	The folder which will be used to save methylation call files, if set to NULL no methylation call file will be saved as a text file. The files saved can be read into R in less time using `methRead` function in `methylKit`
`save.context`	A character vector consisting following strings: "CpG","CHG","CHH". The methylation percentages for these methylation contexts will be saved to save.folder
`read.context`	One of the 'CpG','CHG','CHH' or 'none' strings. Determines what type of methylation context will be read-in to the memory which can be immediately used for analysis. If given as 'none', processBismarkAln will not return any object, but if a save.folder argument given it will save the methylation percentage call files.
`nolap`	if set to TRUE and the SAM/BAM file has paired-end reads, the one read of the overlapping paired-end read pair will be ignored for methylation calling.
`mincov`	minimum read coverage to call a methylation status for a base.
`minqual`	minimum phred quality score to call a methylation status for a base.
`phred64`	logical (default: FALSE) you will not need to set this TRUE, Currently bismark gives only phred33 scale
`treatment`	treatment vector only to be used when location and sample.id parameters are `lists` and you are trying to read-in multiple samples that are related to eachother in down-stream analysis.
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not (default: FALSE). With the default value, a text file containing methylation values will be saved. If TRUE, database will either be saved to location `save.folder` or if this is NULL, to a new folder in the current working directory named after this scheme: "methylDB <Date> <3randomlettersornumbers>"
`verbose`	logical set verbosity of methCall (default: TRUE)

Value

`methylRaw` or `methylRawList` object

Note

SAM files should be sorted with samtools sort or unix sort. Other sorting methods can alter the order of fields(columns) in the SAM file and that will result in an error when using `processBismarkAln()`.

Examples

```
# reading one bismark file:
my.file=system.file("extdata", "test.fastq_bismark.sorted.min.sam",
                                package = "methylKit")
```

```
obj=processBismarkAln(my.file,"test",assembly="hg18",save.folder=NULL,
                 save.context="CpG",read.context="CpG")

# reading multiple files
file.list2=list(system.file("extdata", "test.fastq_bismark.sorted.min.sam",
package = "methylKit"),
               system.file("extdata", "test.fastq_bismark.sorted.min.sam",
               package = "methylKit"),
              system.file("extdata", "test.fastq_bismark.sorted.min.sam",
              package = "methylKit"),
               system.file("extdata", "test.fastq_bismark.sorted.min.sam",
                package = "methylKit"))

 objs=processBismarkAln(location=file.list2
             ,sample.id=list("test1","test2","ctrl1","ctrl1"),assembly="hg18",
             save.folder=NULL,save.context=NULL,read.context="CpG",
             nolap=FALSE,mincov=10,minqual=20,phred64=FALSE,
             treatment=c(1,1,0,0))
```

`readMethylDB` *load tabix file with header to methylDB*

Description

The function reads the header from a given tabix file and loads it into corresponding methylDB object.

Usage

```
readMethylDB(dbpath)
```

Arguments

`dbpath`	path to a tabix file with header

Value

an `methylBaseDB`,`methylRawDB`, `methylRawListDB` or an `methylDiffDB` object

Examples

```
## Not run:
data(methylKit)

baseDB.obj <- makeMethylDB(methylBase.obj,"my/path")
mydbpath <- getDBPath(baseDB.obj)
rm(baseDB.obj)
readMethylDB(mydbpath)


## End(Not run)
```

`reconstruct` *Reconstruct methylBase or methylBaseDB object based on a new methylation percentage matrix*

Description

The function reconstructs a new methylBase object from an input methylBase object and percent methylation matrix. Basically, it uses the read coverages in the input methylBase object and deduces new number of methylated Cs and unmethylated Cs based on the input percent methylation matrix. It is ideally to be used to reconstruct methylBase objects after batch correction on percent methylation values. The percent methylation matrix rows must match methylBase object rows in order ,and in addition column order (the order of samples) in input methylBase must match the order in percent methylation matrix.

Usage

```
reconstruct(methMat, mBase, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'ANY,methylBase'
reconstruct(methMat, mBase, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'ANY,methylBaseDB'
reconstruct(methMat, mBase, chunk.size = 1e+06, save.db = TRUE, ...)
```

Arguments

`methMat`	percent methylation matrix, row order and order of the samples same as the methylBase object
`mBase`	`methylBase` or `methylBaseDB` object to be reconstructed
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` objects (default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_reconstructed" to current filename if database already exists or generate new file with filename "methylBase_reconstructed" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

new `methylBase` or `methylBase` object where methylation percentage matches input `methMat` and coverages matches input `mBase`

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylBaseDB`, while being per default FALSE for `methylBase`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Note

Batch effect correction (if any batch effect exists) is a tricky issue. We provide some simple ways to deal with it (see `assocComp` and `removeComp`), But if you can find other ways to correct for batch effects and want to create a methylBase object with the corrected percent methylation values, you can use this function.

Author(s)

Altuna Akalin

Examples

```
data(methylKit)

# get percent methylation
mat=percMethylation(methylBase.obj)

# do some changes in the matrix
# this is just a toy example
# ideally you want to correct the matrix
# for batch effects
mat[mat==100]=80

# reconstruct the methylBase from the corrected matrix
newobj=reconstruct(mat,methylBase.obj)
```

`regionCounts` *Get regional counts for given GRanges or GRangesList object*

Description

Convert `methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object into regional counts for a given `GRanges` or `GRangesList` object.

Usage

```
regionCounts(object,regions,cov.bases=0,strand.aware=FALSE,chunk.size,save.db,...)

## S4 method for signature 'methylRaw,GRanges'
regionCounts(
```

```
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylBase,GRanges'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRaw,GRangesList'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylBase,GRangesList'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawList,GRanges'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)
```

```
## S4 method for signature 'methylRawList,GRangesList'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawDB,GRanges'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylRawDB,GRangesList'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylRawListDB,GRanges'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylRawListDB,GRangesList'
regionCounts(
  object,
  regions,
  cov.bases = 0,
  strand.aware = FALSE,
  chunk.size = 1e+06,
```

```
    save.db = TRUE,
    ...
  )

  ## S4 method for signature 'methylBaseDB,GRanges'
  regionCounts(
    object,
    regions,
    cov.bases = 0,
    strand.aware = FALSE,
    chunk.size = 1e+06,
    save.db = TRUE,
    ...
  )

  ## S4 method for signature 'methylBaseDB,GRangesList'
  regionCounts(
    object,
    regions,
    cov.bases = 0,
    strand.aware = FALSE,
    chunk.size = 1e+06,
    save.db = TRUE,
    ...
  )
```

Arguments

`object`	a `methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object NOTE: The given regions (Granges/GrangesList object) will be orderd based on chromosome and position before searching for overlaps, so the resulting methylKit object might have a different ording than expected. See details section for the reasoning of this choice and ways to still get custom ordering of regions.
`regions`	a GRanges or GRangesList object. Make sure that the GRanges objects are unique in chr,start,end and strand columns.You can make them unique by using unique() function.
`cov.bases`	number minimum bases covered per region (Default:0). Only regions with base coverage above this threshold are returned.
`strand.aware`	if set to TRUE only CpGs that match the strand of the region will be summarized. (default:FALSE)
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylDB` objects (default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_regions" to current filename if database already exists or generate new file with filename "sampleID_regions" or "methylBase_filtered" dependent on input object

`dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database

`dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

a new methylRaw,methylBase or methylRawList object. If `strand.aware` is set to FALSE (default). Even though the resulting object will have the strand information of `regions` it will still contain methylation information from both strands.

Details

The given regions (Granges/GrangesList object) will be orderd based on chromosome and position before searching for overlaps, so the resulting methylKit object might have a different ording than expected. We are doing this is to ensure that resulting output is consistent for in-memory and database based objects, as database based objects always have to be sorted to enable tabix indexing and providing fast random access.

If you to still want get a custom ordering of the output regions you can access the single regions in any object providing your indices to the `select` or `extract` functions.

The parameter `chunk.size` is only used when working with `methylRawDB`, `methylBaseDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawDB`, `methylBaseDB` or `methylRawListDB`, while being per default FALSE for `methylRaw`, `methylBase` or `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Examples

```
data(methylKit)

# get the windows of interest as a GRanges object, this can be any set
# of genomic locations
library(GenomicRanges)
my.win=GRanges(seqnames="chr21",
ranges=IRanges(start=seq(from=9764513,by=10000,length.out=20),width=5000) )

# getting counts per region
regional.methylRaw=regionCounts(object=methylRawList.obj, regions=my.win,
cov.bases=0,strand.aware=FALSE)
```

`removeComp` *Remove principal components from a methylBase object*

Description

This function can remove a given principal componet from a given methylBase object. First, it calculates principal components from percent methylation matrix and removes the given component(s), reconstructs the methylation matrix then reconstructs number of methylated and unmethylated Cs per position based on the reconstructed percent methylation matrix, and finally returns a new `methylBase` object.

Usage

```
removeComp(mBase, comp = NULL, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'methylBase'
removeComp(mBase, comp = NULL, chunk.size = 1e+06, save.db = FALSE, ...)

## S4 method for signature 'methylBaseDB'
removeComp(mBase, comp = NULL, chunk.size = 1e+06, save.db = TRUE, ...)
```

Arguments

`mBase`	`methylBase` or `methylBaseDB` object with no NA values, that means all bases should be covered in all samples.
`comp`	vector of component numbers to be removed
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` objects (default: 1e6)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_reconstructed" to current filename if database already exists or generate new file with filename "methylBase_reconstructed" `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option "tabix"

Value

new `methylBase` or `methylBaseDB` object

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylBaseDB`, while being per default FALSE for `methylBase`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Examples

```
data(methylKit)

# remove 1st principal component
newObj=removeComp(methylBase.obj,comp=1)

# remove 3rd and 4th  principal components
newObj=removeComp(methylBase.obj,comp=c(3,4))
```

`reorganize` *Reorganize methylKit objects by creating new objects from subset of samples*

Description

The function creates a new `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object by selecting a subset of samples from the input object, which is a `methylRawList` or `methylBase` object. You can use the function to partition a large methylRawList or methylBase object to smaller object based on sample ids or when you want to reorder samples and/or give a new treatmet vector.

Usage

```
reorganize(
  methylObj,
  sample.ids,
  treatment,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylBase'
reorganize(
  methylObj,
  sample.ids,
  treatment,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawList'
reorganize(
  methylObj,
  sample.ids,
```

```
  treatment,
  chunk.size = 1e+06,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawListDB'
reorganize(
  methylObj,
  sample.ids,
  treatment,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylBaseDB'
reorganize(
  methylObj,
  sample.ids,
  treatment,
  chunk.size = 1e+06,
  save.db = TRUE,
  ...
)
```

Arguments

`methylObj`	a `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object
`sample.ids`	a vector for sample.ids to be subset. Order is important and the order should be similar to treatment. sample.ids should be a subset or reordered version of sample ids in the input object.
`treatment`	treatment vector, should be same length as sample.ids vector
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylBaseDB` or `methylRawListDB` objects, default: 1e6
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: For “methylBase”: The default suffix is a 13-character random string appended to the fixed prefix “methylBase”, e.g. “methylBase_16d3047c1a254.txt.bgz”. For “methylRawList”: ignored. `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option "tabix"

Value

returns a `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object depending on the input object

Details

The parameter `chunk.size` is only used when working with `methylBaseDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylBaseDB` and `methylRawListDB`, while being per default FALSE for `methylBase` and `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Examples

```
# this is a list of example files, ships with the package
file.list=list( system.file("extdata", "test1.myCpG.txt", package = "methylKit"),
               system.file("extdata", "test2.myCpG.txt", package = "methylKit"),
               system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
               system.file("extdata", "control2.myCpG.txt", package = "methylKit") )


# read the files to a methylRawList object: myobj
myobj=methRead( file.list,
          sample.id=list("test1","test2","ctrl1","ctrl2"),
          assembly="hg18",pipeline="amp",treatment=c(1,1,0,0))
meth=unite(myobj,destrand=TRUE)

# get samples named "test1" and "ctrl2" from myobj and create a new methylRawList object
myobj2=reorganize(myobj,sample.ids=c("test1","ctrl2"),treatment=c(1,0) )

# # get samples named "test1" and "ctrl2" from meth and create a new methylBase object
meth2 =reorganize(meth,sample.ids=c("test1","ctrl2"),treatment=c(1,0) )
```

`select` *selects rows from of methylKit objects*

Description

The function returns a subset of data contained in the `methylKit` objects.

Usage

```
select(x,i)

## S4 method for signature 'methylBase'
select(x, i)

## S4 method for signature 'methylRaw'
select(x, i)

## S4 method for signature 'methylDiff'
```

```
select(x, i)

## S4 method for signature 'methylRawDB'
select(x, i)

## S4 method for signature 'methylBaseDB'
select(x, i)

## S4 method for signature 'methylDiffDB'
select(x, i)
```

Arguments

`x`	an `methylBase`,`methylBaseDB`, `methylRaw`,`methylRawDB` or `methylDiff` object
`i`	a numeric or logical vector. This vector corresponds to bases or regions contained in `methylKit` objects.The vector is used to subset the data.

Value

a `methylBase`,`methylRaw` or `methylDiff` object depending on the input object.

Examples

```
data(methylKit)


methylRawDB.obj=methRead( system.file("extdata","test1.txt.bgz",package="methylKit"),
                          sample.id="test1", assembly="hg18",
                          dbtype = "tabix",dbdir = "methylDB")

methylBaseDB.obj=unite(methylRawList.obj,save.db=TRUE,dbdir="methylDB")


 # selects first hundred rows, returns a methylRaw object
subset1=select(methylRawList.obj[[1]],1:100)
subset1=select(methylRawDB.obj,1:100)

# selects first hundred rows, returns a methylBase object
subset2=select(methylBase.obj,1:100)
subset2=select(methylBaseDB.obj,1:100)

# selects first hundred rows, returns a methylDiff object
subset3=select(methylDiff.obj,1:100)




# remove Database again
rm(methylBaseDB.obj)
rm(methylRawDB.obj)
unlink("methylDB",recursive=TRUE)
```

`selectByOverlap` *selects records of methylDB objects lying inside a GRanges range*

Description

The function selects records from any `methylKit` object that lie inside the regions given by `ranges` of class `GRanges` and returns an in-memory equivalent of this object

Usage

```
selectByOverlap(object,ranges)

## S4 method for signature 'methylRaw,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylRawList,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylBase,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylDiff,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylRawDB,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylRawListDB,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylBaseDB,GRanges'
selectByOverlap(object, ranges)

## S4 method for signature 'methylDiffDB,GRanges'
selectByOverlap(object, ranges)
```

Arguments

`object`	an `methylRaw`,`methylRawDB`, `methylRawList`, `methylRawListDB`, `methylBase`, `methylBaseDB`, `methylDiff` or `methylDiffDB` object
`ranges`	a GRanges object specifying the regions of interest

Value

a `methylBase`,`methylRaw`, `methylRawList` or `methylDiff` object depending on the input object.

Author(s)

Alexander Gosdschan

Examples

```
data(methylKit)

file.list=list( system.file("extdata", "test1.myCpG.txt", package = "methylKit"),
                system.file("extdata", "test2.myCpG.txt", package = "methylKit"),
                system.file("extdata", "control1.myCpG.txt", package = "methylKit"),
                system.file("extdata", "control2.myCpG.txt", package = "methylKit") )

methylRawListDB.obj=methRead(file.list,
                         sample.id=list("test1","test2","ctrl1","ctrl2"),
                         assembly="hg18",treatment=c(1,1,0,0),
                         dbtype = "tabix",dbdir = "methylDB")

methylBaseDB.obj=unite(methylRawListDB.obj)

methylDiffDB.obj = calculateDiffMeth(methylBaseDB.obj)

# define the windows of interest as a GRanges object, this can be any set
# of genomic locations
library(GenomicRanges)
my.win=GRanges(seqnames="chr21",
ranges=IRanges(start=seq(from=9764513,by=10000,length.out=20),width=5000) )

# selects the records that lie inside the regions
myRaw <- selectByOverlap(methylRawListDB.obj[[1]],my.win)

# selects the records that lie inside the regions
myBase <- selectByOverlap(methylBaseDB.obj,my.win)

# selects the records that lie inside the regions
myDiff <- selectByOverlap(methylDiffDB.obj,my.win)

# selects the records that lie inside the regions
myRaw2 <- selectByOverlap(methylRawList.obj[[1]],my.win)

# selects the records that lie inside the regions
myRawList2 <- selectByOverlap(methylRawList.obj,my.win)

# selects the records that lie inside the regions
myBase2 <- selectByOverlap(methylBase.obj,my.win)

# selects the records that lie inside the regions
myDiff2 <- selectByOverlap(methylDiff.obj,my.win)


rm(methylRawListDB.obj)
rm(methylBaseDB.obj)
rm(methylDiffDB.obj)
unlink("methylDB",recursive=TRUE)
```

---

show,methylBase-method

show method for methylKit classes

Description

The show method works for `methylRaw`,`methylRawDB`, `methylRawList`,`methylRawListDB`, `methylBase`,`methylBaseDB` and `methylDiff` objects

Usage

```
## S4 method for signature 'methylBase'
show(object)

## S4 method for signature 'methylRaw'
show(object)

## S4 method for signature 'methylRawList'
show(object)

## S4 method for signature 'methylDiff'
show(object)

## S4 method for signature 'methylRawDB'
show(object)

## S4 method for signature 'methylRawListDB'
show(object)

## S4 method for signature 'methylBaseDB'
show(object)

## S4 method for signature 'methylDiffDB'
show(object)
```

Arguments

`object`	any methylKit object

Examples

```
data(methylKit)
methylDiff.obj
show(methylDiff.obj)
```

`tileMethylCounts` *Get methylated/unmethylated base counts for tilling windows*

Description

The function summarizes methylated/unmethylated base counts over tilling windows accross genome. This function can be used when differential methylation analysis is preferable to tilling windows instead of base pairs.

Usage

```
tileMethylCounts(object,win.size=1000,step.size=1000,cov.bases=0,mc.cores=1,save.db,...)

## S4 method for signature 'methylRaw'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
  cov.bases = 0,
  mc.cores = 1,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawList'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
  cov.bases = 0,
  mc.cores = 1,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylBase'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
  cov.bases = 0,
  mc.cores = 1,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawDB'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
  cov.bases = 0,
  mc.cores = 1,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylRawListDB'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
```

```
  cov.bases = 0,
  mc.cores = 1,
  save.db = TRUE,
  ...
)

## S4 method for signature 'methylBaseDB'
tileMethylCounts(
  object,
  win.size = 1000,
  step.size = 1000,
  cov.bases = 0,
  mc.cores = 1,
  save.db = TRUE,
  ...
)
```

Arguments

`object`	`methylRaw`, `methylRawDB`, `methylRawList`, `methylRawListDB`, `methylBase` or `methylBaseDB` object containing base pair resolution methylation information
`win.size`	an integer for the size of the tiling windows
`step.size`	an integer for the step size of tiling windows
`cov.bases`	minimum number of bases to be covered in a given window
`mc.cores`	number of cores to use when processing `methylDB` objects, default: 1, but always 1 for Windows)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: append "_tiled" to current filename if database already exists or generate new file with filename "sampleID_tiled" or "methylBase_tiled" dependent on input object `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database

Value

`methylRaw`,`methylBase` or `methylRawList` object

Details

The parameter `chunk.size` is only used when working with `methylRawDB`, `methylBaseDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawDB`, `methylBaseDB` or `methylRawListDB`, while being per default FALSE for `methylRaw`, `methylBase` or `methylRawList`.

If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might want to change the value of this parameter.

Examples

```
data(methylKit)

tiled.methylRaw=tileMethylCounts(object=methylRawList.obj,win.size=1000,
                                 step.size=1000,cov.bases=0)
```

`unite` *unite methylRawList to a single table*

Description

This functions unites `methylRawList` and `methylRawListDB` objects that only bases with coverage from all samples are retained. The resulting object is either of class `methylBase` or `methylBaseDB` depending on input.

Usage

```
unite(
  object,
  destrand = FALSE,
  min.per.group = NULL,
  chunk.size = 1e+06,
  mc.cores = 1,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawList'
unite(
  object,
  destrand = FALSE,
  min.per.group = NULL,
  chunk.size = 1e+06,
  mc.cores = 1,
  save.db = FALSE,
  ...
)

## S4 method for signature 'methylRawListDB'
unite(
  object,
  destrand = FALSE,
  min.per.group = NULL,
  chunk.size = 1e+06,
  mc.cores = 1,
```

```
    save.db = TRUE,
    ...
)
```

Arguments

`object`	a methylRawList or methylRawListDB object to be merged by common locations covered by reads
`destrand`	if TRUE, reads covering both strands of a CpG dinucleotide will be merged, do not set to TRUE if not only interested in CpGs (default: FALSE). If the methylRawList object contains regions rather than bases setting destrand to TRUE will have no effect.
`min.per.group`	an integer denoting minimum number of samples per replicate needed to cover a region/base. By default only regions/bases that are covered in all samples are united as methylBase object, however by supplying an integer for this argument users can control how many samples needed to cover region/base to be united as methylBase object. For example, if min.per.group set to 2 and there are 3 replicates per condition, the bases/regions that are covered in at least 2 replicates will be united and missing data for uncovered bases/regions will appear as NAs.
`chunk.size`	Number of rows to be taken as a chunk for processing the `methylRawListDB` objects, default: 1e6
`mc.cores`	number of cores to use when processing `methylRawListDB` objects, default: 1, but always 1 for Windows)
`save.db`	A Logical to decide whether the resulting object should be saved as flat file database or not, default: explained in Details sections
`...`	optional Arguments used when save.db is TRUE `suffix` A character string to append to the name of the output flat file database, only used if save.db is true, default actions: The default suffix is a 13-character random string appended to the fixed prefix “methylBase”, e.g. “methylBase_16d3047c1a254.txt.bgz” `dbdir` The directory where flat file database(s) should be stored, defaults to getwd(), working directory for newly stored databases and to same directory for already existing database `dbtype` The type of the flat file database, currently only option is "tabix" (only used for newly stored databases)

Value

a methylBase or methylBaseDB object depending on input

Details

The parameter `chunk.size` is only used when working with `methylRawDB` or `methylRawListDB` objects, as they are read in chunk by chunk to enable processing large-sized objects which are stored as flat file database. Per default the chunk.size is set to 1M rows, which should work for most systems. If you encounter memory problems or have a high amount of memory available feel free to adjust the `chunk.size`.

The parameter `save.db` is per default TRUE for methylDB objects as `methylRawListDB`, while being per default FALSE for `methylRawList`. If you wish to save the result of an in-memory-calculation as flat file database or if the size of the database allows the calculation in-memory, then you might change the value of this parameter.

Examples

```
data(methylKit)
## Following
my.methylBase=unite(methylRawList.obj)
my.methylBase=unite(methylRawList.obj,destrand=TRUE)
```

`updateMethObject` *update methylKit objects The method updates object from earlier versions (<v0.9.1) to latest object.*

Description

update methylKit objects

The method updates object from earlier versions (<v0.9.1) to latest object.

Usage

```
updateMethObject(object)
```

Arguments

`object`	a methylKit object: methylRaw, methylRawList, methylBase or methylDiff

Value

`methylRaw`,`methylDiff`,`methylBase` or `methylRawList` object

@export @docType methods @rdname updateMethObject

Index