Package ‘MEDIPS’

January 20, 2026

Type Package

Title DNA IP-seq data analysis

Version 1.62.0

Date 2020-02-15

Author Lukas Chavez, Matthias Lienhard, Joern Dietrich, Isaac Lopez Moyado

Maintainer Lukas Chavez <lukaschavez@ucsd.edu>

Description MEDIPS was developed for analyzing data derived from
methylated DNA immunoprecipitation (MeDIP) experiments followed
by sequencing (MeDIP-
seq). However, MEDIPS provides functionalities for the analysis of any kind of quantitative
sequencing data (e.g. ChIP-seq, MBD-seq, CMS-seq and others)
including calculation of differential coverage between groups
of samples and saturation and correlation analysis.

License GPL (>=2)

LazyLoad yes

Depends R (>= 3.0), BSgenome, Rsamtools

Imports GenomicRanges, Biostrings, graphics, gtools, IRanges, methods,
stats, utils, edgeR, DNAcopy, biomaRt, rtracklayer,
preprocessCore

Suggests BSgenome.Hsapiens.UCSC.hg19, MEDIPSData, BiocStyle

biocViews DNAMethylation, CpGIsland, DifferentialExpression,
Sequencing, ChIPSeq, Preprocessing, QualityControl,
Visualization, Microarray, Genetics, Coverage,
GenomeAnnotation, CopyNumberVariation, SequenceMatching

NeedsCompilation no

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

git_branch RELEASE_3_22

git_last_commit 8748557

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-01-19

Contents

MEDIPS-package 2
COUPLINGset-class 3
MEDIPS.addCNV 4
MEDIPS.correlation 5
MEDIPS.couplingVector 6
MEDIPS.CpGenrich 6
MEDIPS.createROIset 8
MEDIPS.createSet 9
MEDIPS.exportWIG 10
MEDIPS.getAnnotation 11
MEDIPS.mergeFrames 12
MEDIPS.mergeSets 13
MEDIPS.meth 14
MEDIPS.plotCalibrationPlot 16
MEDIPS.plotSaturation 17
MEDIPS.plotSeqCoverage 18
MEDIPS.saturation 19
MEDIPS.selectROIs 21
MEDIPS.selectSig 22
MEDIPS.seqCoverage 23
MEDIPS.setAnnotation 24
MEDIPSroiSet-class 25
MEDIPSset-class 26

Index **29**

`MEDIPS-package` *(MeD)IP-seq data analysis*

Description

MEDIPS was developed for analyzing data derived from methylated DNA immunoprecipitation (MeDIP) experiments followed by sequencing (MeDIP-seq). Nevertheless, several functionalities may be applied to other types of sequencing data (e.g. differential coverage or testing the saturation of ChIP-seq data). MEDIPS addresses several aspects in the context of MeDIP-seq data analysis including basic data processing, several quality controls, normalization, and identification of differential coverage.

Details

Package:	MEDIPS
Type:	Package
License:	GPL (>=2)
LazyLoad:	yes

Author(s)

Lukas Chavez, Matthias Lienhard, Joern Dietrich

Maintainer: Lukas Chavez <lchavez@liai.org>

References

Chavez L, Jozefczuk J, Grimm C, Dietrich J, Timmermann B, Lehrach H, Herwig R, Adjaye J., Computational analysis of genome-wide DNA methylation during the differentiation of human embryonic stem cells along the endodermal lineage, Genome Res. 2010 Oct;20(10):1441-50. Epub 2010 Aug 27.

`COUPLINGset-class` *COUPLINGset class and internal functions*

Description

COUPLINGset class is used in the MEDIPS library to store and extract information generated during the creation of a coupling vector.

Objects from the Class

Objects of the classes contain information about sequence pattern information, included chromosomes, and further parameter settings. A COUPLING SET object is created by the MEDIPS.couplingVector() function. According slots will be filled during the workflow.

Slots

`genome_name`: Object of class `"character"` : the refernce genome

`window_size`: Object of class `"numeric"` : the window size for the genome vector

`chr_names`: Object of class `"character"` : the names of the chromosomes included within the MEDIPS/COUPLING SET

`chr_lengths`: Object of class `"numeric"` : the lengths of the chromosomes included within the MEDIPS/COUPLING SET

`seq_pattern`: Object of class `"character"` : the sequence pattern (e.g. CG)

`genome_CF`: Object of class `"numeric"` : the coupling factor at the genomic bins

`number_pattern`: Object of class `"numeric"` : the total number of sequence pattern

Methods

genome_name `signature(object = "COUPLINGset")`: extracts the reference genome of the COUPLING SET

window_size `signature(object = "COUPLINGset")`: extracts the window size from the window_size slot COUPLING SET

chr_names `signature(object = "COUPLINGset")`: extracts the names of the chromosomes included within the COUPLING SET

chr_lengths `signature(object = "COUPLINGset")`: extracts the length of the chromosomes included within the COUPLING SET

seq_pattern `signature(object = "COUPLINGset")`: extracts the sequence pattern (e.g. CpG)

genome_CF `signature(object = "COUPLINGset")`: extracts the coupling factor at the genomic bins

number_pattern `signature(object = "COUPLINGset")`: extracts the total number of sequence pattern

show `signature(object = "COUPLINGset")`: prints a summary of the COUPLING SET object content

Author(s)

Lukas Chavez, Matthias Lienhard, Joern Dietrich

Examples

```
showClass("COUPLINGset")
```

`MEDIPS.addCNV` *Function to run a copy number variation analysis.*

Description

Function calculates a CNV analysis based on two INPUT SETs by employing the DNAcopy package. The results are attached to a provided result table.

Usage

```
MEDIPS.addCNV(ISet1, ISet2, results, cnv.Frame=1000)
```

Arguments

`ISet1`	First group of INPUT SETs
`ISet2`	Second group of INPUT SETs
`results`	result table as returned by the MEDIPS.meth function
`cnv.Frame`	window size used for calculating CNVs. Can be of different size than the result table.

Value

The result table with an additional column containing DNAcopy's log-ratio.

Author(s)

Joern Dietrich

Examples

```
library(MEDIPSData)
library("BSgenome.Hsapiens.UCSC.hg19")

bam.file.hESCs.Input = system.file("extdata", "hESCs.Input.chr22.bam", package="MEDIPSData")
bam.file.DE.Input = system.file("extdata", "DE.Input.chr22.bam", package="MEDIPSData")

hESCs.Input = MEDIPS.createSet(file=bam.file.hESCs.Input, BSgenome="BSgenome.Hsapiens.UCSC.hg19", extend=250
DE.Input = MEDIPS.createSet(file=bam.file.DE.Input, BSgenome="BSgenome.Hsapiens.UCSC.hg19", extend=250, shif

data(resultTable)

resultTable = MEDIPS.addCNV(cnv.Frame=10000, ISet1=hESCs.Input, ISet2=DE.Input, results=resultTable)
```

MEDIPS.correlation *Calculates pairwise Pearson correlations between provided MEDIPS SETs*

Description

The function calculates genome wide Pearson correlations between all pairs of provided MEDIPS SETs.

Usage

```
MEDIPS.correlation(MSets=NULL, plot = T, method="spearman")
```

Arguments

`MSets`	a concatenated set of MEDIPS SETs
`plot`	if specified, the correlation will be depicted as a scatter plot
`method`	default: spearman; alternatives: kendall, spearman

Value

a correlation matrix

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
data(hESCs_MeDIP)
data(DE_MeDIP)

correlation = MEDIPS.correlation(MSets=c(hESCs_MeDIP[[1]], DE_MeDIP[[1]]), plot = FALSE)
```

`MEDIPS.couplingVector` *Calculates the sequence pattern densities at genome wide windows.*

Description

The function calculates the local densities of a defined sequence pattern (e.g. CpGs) and returns a COUPLING SET object which is necessary for normalizing MeDIP data.

Usage

```
MEDIPS.couplingVector(pattern="CG", refObj=NULL)
```

Arguments

`pattern`	defines the sequence pattern, e.g. CG for CpGs.
`refObj`	a MEDIPS Set or MEDIPS ROI Set that serves as reference for the genome and window parameters.

Value

A COUPLING SET object.

Author(s)

Lukas Chavez

Examples

```
library("MEDIPSData")
library("BSgenome.Hsapiens.UCSC.hg19")

data(hESCs_MeDIP)
CS = MEDIPS.couplingVector(pattern="CG", refObj=hESCs_MeDIP)
```

`MEDIPS.CpGenrich` *Calculates CpG enrichment of provided short reads compared to the reference genome.*

Description

As a quality check for the enrichment of CpG rich DNA fragments obtained by the immunoprecipitation step of a MeDIP experiment, this function provides the functionality to calculate CpG enrichment values. The main idea is to check, how strong the regions are enriched for CpGs compared to the reference genome. For this, the function counts the number of Cs, the number of Gs, the number CpGs, and the total number of bases within the stated reference genome. Subsequently, the function calculates the relative frequency of CpGs and the observed/expected ratio of CpGs present in the reference genome. Additionally, the function calculates the same for the DNA sequences underlying the given regions. The final enrichment values result by dividing the relative frequency of CpGs (or the observed/expected value, respectively) of the regions by the relative frequency of CpGs (or the observed/expected value, respectively) of the reference genome.

Usage

```
MEDIPS.CpGenrich(file=NULL, BSgenome=NULL, extend=0, shift=0, uniq=1e-3, chr.select=NULL, paired=F
```

Arguments

`file`	Path and file name of the input data
`BSgenome`	The reference genome name as defined by BSgenome
`extend`	defines the number of bases by which the region will be extended before the genome vector is calculated. Regions will be extended along the plus or the minus strand as defined by their provided strand information.
`shift`	As an alternative to the extend parameter, the shift parameter can be specified. Here, the reads are not extended but shifted by the specified number of nucleotides with respect to the given strand infomation. One of the two parameters extend or shift has to be 0.
`uniq`	The uniq parameter determines, if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0) (deafult: 1e-3).
`chr.select`	only data at the specified chromosomes will be processed.
`paired`	option for paired end reads

Value

`regions.CG`	the numbe of CpGs within the regions
`regions.C`	the number of Cs within the regions
`regions.G`	the number of Gs within the regions
`regions.relH`	the relative frequency of CpGs within the regions
`regions.GoGe`	the observed/expected ratio of CpGs within the regions
`genome.CG`	the numbe of CpGs within the reference genome
`genome.C`	the number of Cs within the reference genome
`genome.G`	the number of Gs within the reference genome
`genome.relH`	the relative frequency of CpGs within the reference genome
`genome.GoGe`	the observed/expected ratio of CpGs within the reference genome
`enrichment.score.relH`	regions.relH/genome.relH
`enrichment.score.GoGe`	regions.GoGe/genome.GoGe

Author(s)

Joern Dietrich and Matthias Lienhard

Examples

```
library(MEDIPSData)
library("BSgenome.Hsapiens.UCSC.hg19")
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

#er=MEDIPS.CpGenrich(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", chr.select="ch
```

MEDIPS.createROIset *Creates a MEDIPS ROI SET by reading a suitable input file*

Description

Reads the input file and calculates the short read coverage (counts) for the specified regions of interest(ROI). After reading of the input file, the MEDIPS ROI SET contains information about the input file name, the dependent organism, the chromosomes included in the input file, the length of the included chromosomes (automatically loaded), the number of regions, and a GRange object of the ROIs.

Usage

```
MEDIPS.createROIset(file=NULL, ROI=NULL, extend=0, shift=0, bn=1, BSgenome=NULL, uniq=1e-3, chr.se
```

Arguments

`file`	Path and file name of the input data
`ROI`	Data.frame with columns "chr", "start", "end" and "name" of regions of interest
`extend`	defines the number of bases by which the region will be extended before the genome vector is calculated. Regions will be extended along the plus or the minus strand as defined by their provided strand information.
`shift`	As an alternative to the extend parameter, the shift parameter can be specified. Here, the reads are not extended but shifted by the specified number of nucleotides with respect to the given strand infomation. One of the two parameters extend or shift has to be 0.
`bn`	Number of bins per ROI
`BSgenome`	The reference genome name as defined by BSgenome
`uniq`	The uniq parameter determines, if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0) (deafult: 1e-3). The smaller the p-value, the more reads at the same genomic position are potentially allowed.
`chr.select`	only data at the specified chromosomes will be processed.
`paired`	option for paired end reads
`sample_name`	name of the sample to be stored with the MEDIPSROI SET.
`isSecondaryAlignment`	option to import only primary alignments.
`simpleCigar`	option to import only alignments with simple Cigar string.

Value

An object of class MEDIPSroiSet.

Author(s)

Lukas Chavez and Matthias Lienhard

Examples

```
library("BSgenome.Hsapiens.UCSC.hg19")
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")
rois=data.frame(chr=c("chr22","chr22"), start=c(19136001, 19753401), stop=c(19136200, 19753500), ID=c("ID_1"
MSet=MEDIPS.createROIset(file=bam.file.hESCs.Rep1.MeDIP,ROI=rois, BSgenome="BSgenome.Hsapiens.UCSC.hg19", e
```

`MEDIPS.createSet` *Creates a MEDIPS SET by reading a suitable input file*

Description

Reads the input file and calculates genome wide short read coverage (counts) at the specified window size. After reading of the input file, the MEDIPS SET contains information about the input file name, the dependent organism, the chromosomes included in the input file, the length of the included chromosomes (automatically loaded), and the number of regions.

Usage

```
MEDIPS.createSet(file=NULL, extend=0, shift=0, window_size=300, BSgenome=NULL, uniq=1e-3, chr.sele
```

Arguments

`file`	Path and file name of the input data
`BSgenome`	The reference genome name as defined by BSgenome
`extend`	defines the number of bases by which the region will be extended before the genome vector is calculated. Regions will be extended along the plus or the minus strand as defined by their provided strand information.
`shift`	As an alternative to the extend parameter, the shift parameter can be specified. Here, the reads are not extended but shifted by the specified number of nucleotides with respect to the given strand infomation. One of the two parameters extend or shift has to be 0.
`uniq`	The uniq parameter determines, if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0) (deafult: 1e-3). The smaller the p-value, the more reads at the same genomic position are potentially allowed.
`chr.select`	only data at the specified chromosomes will be processed.
`window_size`	defines the genomic resolution by which short read coverage is calculated.
`paired`	option for paired end reads

`sample_name`	name of the sample to be stored with the MEDIPS SET.
`isSecondaryAlignment`	option to import only primary alignments.
`simpleCigar`	option to import only alignments with simple Cigar string.

Value

An object of class MEDIPSset.

Author(s)

Lukas Chavez, Mathias Lienhard, Isaac Lopez Moyado

Examples

```
library("BSgenome.Hsapiens.UCSC.hg19")
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

MSet=MEDIPS.createSet(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", chr.select="c
```

`MEDIPS.exportWIG` *Exports count, rpkm, or sequence pattern densities into a wiggle file.*

Description

The function allows for exporting the calculated methylation values (counts or rpkm) or sequence pattern densities from a MEDIPS or COUPLING SET into a wiggle (WIG) file. The wiggle file will contain values for all genomic windows of the genome/coupling vector and can be used for data visualization using appropriate genome browsers. Either a MEDIPS SET (parameter MSet) or a COUPLING SET (parameter CSet) has to be given.

Usage

```
MEDIPS.exportWIG(Set=NULL, CSet=NULL, file=NULL, format="rpkm", descr="")
```

Arguments

`Set`	has to be a MEDIPS SET object. Required when the parameter 'format' is 'count', 'rpkm', or 'rms'.
`CSet`	has to be a COUPLING SET object. Required when the parameter 'format' is 'pdensity' or 'rms'.
`file`	defines the name of the exported file
`format`	if set to "count", there must be a MEDIPS SET at 'Set'. The number of overlapping (extended) short reads per window will be exported. if set to "rpkm", there must be a MEDIPS SET at 'Set'. The rpkm values will be exported (default). If set to "pdensity", there must be a COUPLING SET at 'CSet'. The pattern densities (counts per window) will be exported (parameter Set will be ignored). If set to 'rms', there must be a MEDIPS SET at 'Set' and a corresponding COUPLING SET at 'CSet'. The CpG density normalized methylatin estimates will be exported.

`descr`	the exported wiggle file will include a track name and description that will be visualized by the utilized genome browser. Both, track name and description will be the same as defined here.

Value

the funtion exports the specified data from the MEDIPS or COUPLING SET into the stated file

Author(s)

Lukas Chavez

Examples

```
library("BSgenome.Hsapiens.UCSC.hg19")

bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")
MSet=MEDIPS.createSet(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", chr.select="c

MEDIPS.exportWIG(Set=MSet, file="hESCs.Rep1.wig", format="rpkm", descr="hESCs.Rep1")
```

`MEDIPS.getAnnotation` *Funtion to fetch annotations from biomaRt.*

Description

The function receives predifined annotations from ensembl biomaRt for subsequent annotation of a result table.

Usage

```
MEDIPS.getAnnotation(host="www.ensembl.org",dataset=c("hsapiens_gene_ensembl","mmusculus_gene_en
```

Arguments

`host`	BioMart database host you want to connect to. For current ensembl version, use "www.ensembl.org". For other versions, set to the respective archive host, e.g. "may2012.archive.ensembl.org" for Ensembl 67 Please ensure that the ensembl version is compatible to the used genome version.
`dataset`	The dataset you want to use. To see the different datasets available within a biomaRt you can e.g. do: mart = useMart('ensembl'), followed by listDatasets(mart).
`annotation`	Type of annotation you want to retrieve. You can select "EXON" for exonic regions, "GENE" for gene regions including introns and "TSS" for regions at the transcirption start site.
`tssSz`	Defines the TSS region: start and end position relative to the strand and position of the transcript.
`chr`	Chromosome names for which the annotations should be filtered.

Value

The MEDIPS.getAnnotation function returns a list of annotation tables where each table consists of

`id`	the id of the annotation
`chr`	the chromosome of the annotation
`start`	the start position (5') of the annotation
`end`	the end position (3') of the annotation

Author(s)

Joern Dietrich and Matthias Lienhard

Examples

```
#homo sapiens, current ensembl version
#annotation_hs = MEDIPS.getAnnotation(dataset="hsapiens_gene_ensembl", annotation="TSS", chr=c("chr22"),tssS

#mus musculus, ensembl version 58 (mm9)
#annotation_mm9 = MEDIPS.getAnnotation(host="may2010.archive.ensembl.org",dataset="mmusculus_gene_ensembl",
```

`MEDIPS.mergeFrames` *Merges genomic coordinates of neighboring windows into one supersized window*

Description

After having filtered the result table returned by the MEDIPS.meth function using the MEDIPS.selectSig function, there might be neighboring significant frames. For these cases it is worthwhile to merge neighbouring regions into one supersized frame.

Usage

```
MEDIPS.mergeFrames(frames=NULL, distance=1)
```

Arguments

`frames`	is a filtered result table received by the MEDIPS.selectSig function.
`distance`	allows an according number of bases as a gap between neighboring significant windows to be merged. The default value is 1 in order to merge adjacent windows.

Value

The remaining distinct frames are represented only by their genomic coordinates within the returned results table

`chromosome`	the chromosome of the merged frame
`start`	the start position of the merged frame
`stop`	the stop position of the merged frame
`ID`	a numbered ID of the merged frame

The result table does not contain any merged significant values.

Author(s)

Lukas Chavez

Examples

```
regions=as.data.frame(list(chr=c("chr22", "chr22"), start=c(1001, 1501), stop=c(1500,1750)))
regions.merged=MEDIPS.mergeFrames(regions)

regions.merged
```

`MEDIPS.mergeSets` *Creates one merged MEDIPS SET out of two.*

Description

A MEDIPS SET contains a genome vector which is the count coverage at genome wide windows. Moreover, the MEDIPS SET stores the total number of reads given for calculating the genome vector. Two MEDIPS SETs can be merged whenever they have been construcetd based on the same reference genome, the same set of chromosomes and for the same window size. The returned MEDIPS SET will contain a genome vector where at each window the counts of both given MEDIPS SETs are added. In addtion, the total number of reads will be the sum of both given MEDIPS SETs. Please note, several other attributes like the extend or shift value can be different in both of the given MEDIPS SETs and will be empty in the merged MEDIPS SET. The merged MEDIPS SET will not contain any path to a concrete input file anymore and therefore, cannot be used for the MEDIPS.addCNV function anymore.

Usage

```
MEDIPS.mergeSets(MSet1=NULL, MSet2=NULL, name="Merged Set")
```

Arguments

`MSet1`	A MEDIPS SET object as created by the MEDIPS.createSet function
`MSet2`	A MEDIPS SET object as created by the MEDIPS.createSet function
`name`	The new sample name of the merged MEDIPS SET

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
data(hESCs_Input)
data(DE_Input)

merged_Set = MEDIPS.mergeSets(hESCs_Input, DE_Input, name="Merged_input")

merged_Set
```

`MEDIPS.meth`	*Funtion summarizes coverage profiles for given MEDIPS SETs and allows to calculate differental coverage and copy number vartiation, if applicable.*

Description

The function summarizes coverage profiles (counts, rpkm) for given MEDIPS SETs at the slots MSet1, MSet2, ISet1, and ISet1. In case the parameter MeDIP is set to TRUE and a COUPLING SET was provided at the slot CS, the function will calculate CpG density normalized methylation profiles (relative methylation score, rms) for the MEDIPS SETs at the slots MSet1 and MSet2. In case two groups of MEDIPS SETs have been provided at MSet1 and MSet2, the function will calculate differential coverage. In case two groups of MEDIPS SETs have been provided at ISet1 and ISet2 and the parameter CNV was set to TRUE, the function will calculate copy number variation. The function allows for processing a variable number of provided MEDIPS SETs and therefore, the returned matrix is of variable length.

Usage

```
MEDIPS.meth(MSet1 = NULL, MSet2 = NULL, CSet = NULL, ISet1 = NULL, ISet2 = NULL, chr = NULL, p.adj="bo
```

Arguments

`MSet1`	has to be one or a concatenated list of MEDIPS SET objects (the control replicates)
`MSet2`	has to be one or a concatenated list of MEDIPS SET objects (the treatment data) or empty
`CSet`	has to be a COUPLING SET object (must fit the given MEDIPS SET objects with respect to reference genome and represented chromosomes)
`ISet1`	has to be one or a concatenated list of Input derived MEDIPS SET objects (general Input data or Inputs from the control replicates) or empty
`ISet2`	has to be one or a concatenated list of Input derived MEDIPS SET objects (Inputs from the treatment replicates) or empty
`chr`	specify one or several chromosomes (e.g. c("chr1", "chr2")), if only a subset of available chromosomes have to be processed.
`p.adj`	in order to correct p.values derived from the differential coverage analysis for multiple testing, MEDIPS uses Rs' p.adjust function. Therefore, the following methods are available: holm, hochberg, hommel, bonferroni (default) , BH, BY, fdr, none.
`diff.method`	method for calculating differential coverage. Available methods: ttest (default) and edgeR.
`CNV`	In case there are INPUT SETs provided at both Input slots (i.e. ISet1 and ISet2), copy number variation will be tested by applying the package DNAcopy to the window-wise log-ratios calculated based on the the means per group. By setting CNV=F this function will be disabled (default: CNV=TRUE). Please note, there is the function MEDIPS.addCNV which allows to run the CNV analysis on two groups of INPUT SETs using another (typically increased) window size.

`MeDIP`	This parameter determines, whether for the MEDIPS SETs given at the slots MSet1 and MSet2, CpG density dependent normalization values (rms and prob) will be calculated (default: MeDIP=TRUE).
`minRowSum`	threshold for the sum of counts in a window for the staistical test (default=10).
`diffnorm`	This parameter defines which normalisation method is applied prior to testing for differential enrichment between conditions (default='tmm'). tmm, quantile and none are possible when diff.method=edgeR while rpkm and rms are possible when diff.method=ttest.

Value

`Chr`	the chromosome of the ROI
`Start`	the start position of the ROI
`Stop`	the stop position of the ROI
`CF`	the number of CpGs in the window
`*counts`	a variable number of columns (according to the number of provided MEDIPS SETs) containing for each set the number of (extended/shifted) reads that overlap with the window.
`*rpkm`	a variable number of columns (according to the number of provided MEDIPS SETs) containing for each set the rpkm value of the window.
`*rms`	optional (if MeDIP=TRUE): a variable number of columns (according to the number of provided MEDIPS SETs) containing for each set the rms value of the window.
`*counts`	optional (if INPUT SETs given): a variable number of columns (according to the number of provided INPUT SETs) containing for each set the number of (extended/shifted) reads that overlap with the window.
`*rpkm`	optional (if INPUT SETs given): a variable number of columns (according to the number of provided INPUT SETs) containing for each set the rpkm value of the window.
`MSets1.counts.mean`	optional (if more than one MEDIPS SET given): the mean count over all MEDIPS SETs at MSet1.
`MSets1.rpkm.mean`	optional (if more than one MEDIPS SET given): the mean rpkm value over all MEDIPS SETs at MSet1.
`MSets1.rms.mean`	optional (if more than one MEDIPS SET given): the mean rms value over all MEDIPS SETs at MSet1.
`MSets2.counts.mean`	optional (if more than one MEDIPS SET given): the mean count over all MEDIPS SETs at MSet2.
`MSets2.rpkm.mean`	optional (if more than one MEDIPS SET given): the mean rpkm value over all MEDIPS SETs at MSet2.
`MSets2.rms.mean`	optional (if more than one MEDIPS SET given): the mean rms value over all MEDIPS SETs at MSet2.
`ISets1.counts.mean`	optional (if more than one INPUT SET given): the mean count over all INPUT SETs at ISet1.

`ISets1.rpkm.mean`	optional (if more than one INPUT SET given): the mean rpkm value over all INPUT SETs at ISet1.
`ISets2.counts.mean`	optional (if more than one INPUT SET given): the mean count over all INPUT SETs at ISet2.
`ISets2.rpkm.mean`	optional (if more than one INPUT SET given): the mean rpkm value over all INPUT SETs at ISet2.
`edgeR.logFC`	optional (if diff.method=edgeR): log fold change between MSet1 and MSet2 as returned by edgeR.
`edgeR.logCPM`	optional (if diff.method=edgeR): logCPM between MSet1 and MSet2 as returned by edgeR.
`edgeR.p.value`	optional (if diff.method=edgeR): p.value as returned by edgeR.
`edgeR.adj.p.value`	optional (if diff.method=edgeR): adjusted p.value as calculated by the p.adjust function using edgeR's p.values as input.
`score.log2.ratio`	optional (if diff.method=ttest): log2 fold change between the means of the groups MSet1 and MSet2.
`score.p.value`	optional (if diff.method=ttest): p.value as returned by the t.test function.
`score.adj.p.value`	optional (if diff.method=ttest): adjusted p.value as calculated by the p.adjust function using the ttest p.values as input.
`score`	optional (if diff.method=ttest): score = (-log10(score.p.value)*10)*log(score.log2.ratio)
`CNV.log2.ratio`	optional (if two INPUT SETs given and CNV=TRUE): the log2 ratio for segments as calculated by the DNAcopy package.

Author(s)

Lukas Chavez, Matthias Lienhard, Joern Dietrich

Examples

```
library(MEDIPSData)
data(hESCs_MeDIP)
data(DE_MeDIP)
data(hESCs_Input)
data(DE_Input)
data(CS)

resultTable = MEDIPS.meth(MSet1 = hESCs_MeDIP, MSet2 = DE_MeDIP, CSet = CS, ISet1 = hESCs_Input, ISet2 = DE_Inpu
```

`MEDIPS.plotCalibrationPlot` *Creates the calibration plot.*

Description

Visualizes the dependency between MeDIP-seq signals and CpG densities together with the calcibration curve and the results of the linear modelling.

Usage

```
MEDIPS.plotCalibrationPlot(MSet=NULL, ISet=NULL, CSet=NULL, plot_chr="all", rpkm=T, main="Calibra
```

Arguments

`MSet`	a MEDIPS SET object
`ISet`	an INPUT SET (i.e. a MEDIPS SET created from Input sequening data)
`CSet`	an according COUPLING SET object
`plot_chr`	default="all". It is recommended to call a graphics device (e.g. png("calibrationPlot.png")) before calling the plot command, because R might not be able to plot the full amount of data in reasonable time.
`rpkm`	can be either TRUE or FALSE. If set to TRUE, the methylation signals will be transformed into rpkm before plotted. The coupling values remain untouched. It is necessary to set rpkm=T, if both, a MSet and an ISet are given and plotted at the same time.
`main`	The title of the calibration plot.
`xrange`	The signal range of the calibration curve typically falls into a low signal range. By setting the xrange parameter to TRUE, the calibration plot will visualize the low signal range only.

Value

The calibration plot will be visualized.

Author(s)

Lukas Chavez, Matthias Lienhard

Examples

```
library(MEDIPSData)
data(hESCs_MeDIP)
data(CS)

MEDIPS.plotCalibrationPlot(CSet=CS, main="Calibration Plot", MSet=hESCs_MeDIP[[1]], plot_chr="chr22",  rpkm=
```

`MEDIPS.plotSaturation` *Function plots the results of the MEDIPS.saturationAnalysis function.*

Description

The results of the saturation analysis will be visualized by the function.

Usage

```
MEDIPS.plotSaturation(saturationObj = NULL, main="Saturation analysis")
```

Arguments

`saturationObj`	The saturation results object returned by the MEDIPS.saturationAnalysis function
`main`	The title of the coverage plot.

Value

The coverage plot will be visualized.

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
library(BSgenome.Hsapiens.UCSC.hg19)
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

sr=MEDIPS.saturation(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", uniq=1e-3, exte
MEDIPS.plotSaturation(saturationObj = sr, main="Saturation analysis")
```

MEDIPS.plotSeqCoverage

Function plots the results of the MEDIPS.seqCoverage function.

Description

The results of the sequence pattern coverage analysis will be visualized in two possible ways.

Usage

```
MEDIPS.plotSeqCoverage(seqCoverageObj=NULL, main=NULL, type="pie", cov.level = c(0,1,2,3,4,5), t="
```

Arguments

`seqCoverageObj`	The coverage results object returned by the MEDIPS.seqCoverage function.
`main`	The title of the coverage plot.
`type`	there are two types of visualization. The pie chart (default) illustrates the fraction of CpGs covered by the given reads at different coverage level (see also the parameter cov.level). As an alternative, a histogram over all coverage level can be ploted ("hist").
`cov.level`	The pie chart illustrates the fraction of CpGs covered by the given reads according to their coverage level. The visualized coverage levels can be adjusted by the cov.level parameter.
`t`	specifies the maximal coverage depth to be plotted, if type="hist"

Value

The sequence pattern coverage plot will be visualized.

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
library(BSgenome.Hsapiens.UCSC.hg19)
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

cr=MEDIPS.seqCoverage(file=bam.file.hESCs.Rep1.MeDIP, pattern="CG", BSgenome="BSgenome.Hsapiens.UCSC.hg19",

MEDIPS.plotSeqCoverage(seqCoverageObj=cr, main="Sequence pattern coverage", type="pie", cov.level = c(0,1,2,3
```

`MEDIPS.saturation` *Function calculates the saturation/reproducibility of the provided IP-Seq data.*

Description

The saturation analysis addresses the question, whether the number of short reads is sufficient to generate a saturated and reproducible coverage profile of the reference genome. The main idea is that an insuffcent number of short reads will not result in a saturated methylation profile. Only if there is a sufficient number of short reads, the resulting genome wide coverage profile will be reproducible by another independent set of a similar number of short reads.

Usage

```
MEDIPS.saturation(file=NULL, BSgenome=NULL, nit=10, nrit=1, empty_bins=TRUE, rank=FALSE, extend=0,
```

Arguments

`file`	Path and file name of the IP data
`BSgenome`	The reference genome name as defined by BSgenome
`nit`	defines the number of subsets created from the full sets of available regions (default=10)
`nrit`	methods which randomly select data entries may be processed several times in order to obtain more stable results. By specifying the nrit parameter (default=1) it is possible to run the saturation analysis several times. The final results returned to the saturation results object are the averaged results of each random iteration step.
`empty_bins`	can be either TRUE or FALSE (default TRUE). This parameter effects the way of calculating correlations between the resulting genome vectors. A genome vector consists of concatenated vectors for each included chromosome. The size of the vectors is defined by the bin_size parameter. If there occur genomic bins which contain no overlapping regions, neither from the subsets of A nor from the subsets of B, these bins will be neglected when the paramter is set to FALSE.

`rank`	can be either TRUE or FALSE (default FALSE). This parameter also effects the way of calculating correlations between the resulting genome vectors. If rank is set to TRUE, the correlation will be calculated for the ranks of the windows instead of considering the counts (Spearman correlation). Setting this parameter to TRUE is a more robust approach that reduces the effect of possible occuring outliers (these are windows with a very high number of overlapping regions) to the correlation.
`extend`	defines the number of bases by which the region will be extended before the genome vector is calculated. Regions will be extended along the plus or the minus strand as defined by their provided strand information. Please note, the extend and shift parameter are mutual exclusive.
`shift`	defines the number of bases by which the region will be shifted before the genome vector is calculated. Regions will be shifted along the plus or the minus strand as defined by their provided strand information. Please note, the extend and shift parameter are mutual exclusive.
`window_size`	defines the size of genome wide windows and therefore, the size of the genome vector.
`uniq`	The uniq parameter determines, if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0) (deafult: 1e-3). The smaller the p-value, the more reads at the same genomic position are potentially allowed.
`chr.select`	specify a subset of chromosomes for which the saturation analysis is performed.
`paired`	option for paired end reads
`isSecondaryAlignment`	option to import only primary alignments.
`simpleCigar`	option to import only alignments with simple Cigar string.

Value

`distinctSets`	Contains the results of each iteration step (row-wise) of the saturation analysis. The first column is the number of considered regions in each set, the second column is the resulting pearson correlation coefficient when comparing the two independent genome vectors.
`estimation`	Contains the results of each iteration step (row-wise) of the estimated saturation analysis. The first column is the number of considered regions in each set, the second column is the resulting pearson correlation coefficient when comparing the two independent genome vectors.
`distinctSets`	the total number of available regions
`maxEstCor`	contains the best pearson correlation (second column) obtained by considering the artifically doubled set of reads (first column)
`distinctSets`	contains the best pearson correlation (second column) obtained by considering the total set of reads (first column)

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
library(BSgenome.Hsapiens.UCSC.hg19)
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

sr=MEDIPS.saturation(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", uniq=1e-3, ext
```

`MEDIPS.selectROIs` *Selects row-wise subsets of a result table as returned by the MEDIPS.meth function.*

Description

MEDIPS provides the functionality to select subsets of the result matrix returned by the MEDIPS.meth function according to any given set of regions of interest (ROIs).

Usage

```
MEDIPS.selectROIs(results=NULL, rois=NULL, columns=NULL, summarize=NULL)
```

Arguments

`results`	a result table as returned by the function MEDIPS.meth
`rois`	A matrix containing genomic coordinates of any regions of interest.
`columns`	Only selected columns will be returned as determined by the columns parameter. It is possible to specify one or more concrete column names, please see an example below.
`summarize`	By setting summarize=NULL (default) all windows included in the genomic ranges of the given ROIs will be returned. As an alternative, it is possible to calculate mean (summarize = "avg") or sum (summarize = "sum") values over the individual windows included in each ROI, or to select only the most significant window within each given ROI (summarize="minP").

Author(s)

Lukas Chavez, Matthias Lienhard

Examples

```
library(MEDIPSData)
data(resultTable)

rois=data.frame(chr=c("chr22","chr22"), start=c(19136001, 19753401), stop=c(19136200, 19753500), ID=c("ID_1"
columns=names(resultTable)[grep("counts|rpkm|logFC",names(resultTable))]
s = MEDIPS.selectROIs(results=resultTable, rois=rois, columns=columns, summarize=NULL)
```

MEDIPS.selectSig — *Selects windows which show significant differential coverage between two MEDIPS SETs from a resultTable (as returned by the function MEDIPS.meth).*

Description

Based on the results table returned by the MEDIPS.meth function, the function selects windows which show significant differential coverage between the two groups of MEDIPS SETs. Selection of significant windows follows according to the specification of the available parameters.

Usage

```
MEDIPS.selectSig(results=NULL, p.value=0.01, adj=T, ratio=NULL, bg.counts=NULL, CNV=F)
```

Arguments

`results`	specifies the result table derived from the MEDIPS.meth function.
`p.value`	this is the p.value threshold as calculated either by the ttest or edgeR method
`adj`	this parameter specifies whether the p.value or the adjusted p.values is considered
`ratio`	this parameter sets an additional thresold for the ratio where the ratio is either score.log2.ratio or edgeR.logFC depending on the previously selected method. Please note, the specified value will be transformed into log2 internally.
`bg.counts`	as an additional filter parameter, it is possible to require a minimal number of reads per window in at least one of the MEDIPS SET groups. For this, the mean of counts per group is considered. The parameter bg.counts can either be a concrete integer or an appropriate column name of the result table. By specifying a column name, the 0.95 quantile of the according genome wide count distribution is determined and used as a minimal background threshold (please note, only count columns are reasonable).
`CNV`	The information on CNVs present in the samples of interest can be used for correcting differential coverage observed in the corresponding IP data (e.g. MeDIP or ChIP data). In case Input data has been provided for both conditions, MEDIPS is capable of calculating genome wide CNV ratios by employing the package DNAcopy. In case the parameter CNV is set to TRUE, MEDIPS will subtract the CNV ratio from the IP ratio. Subsequently, only genomic windows having a CNV corrected IP ratio higher than the specified ratio threshold (specification of the ratio parameter is required in this case) will be considered as windows with sufficient differential IP coverage.

Author(s)

Lukas Chavez, Matthias Lienhard

Examples

```
library(MEDIPSData)
data(resultTable)
```

```
sig = MEDIPS.selectSig(results=resultTable, p.value=0.05, adj=TRUE, ratio=NULL, bg.counts=NULL, CNV=FALSE)
```

MEDIPS.seqCoverage *The function identifies the number of CpGs (or any other predefined sequence pattern) covered by the given short reads.*

Description

The main idea of the sequence pattern coverage analysis is to test the number of CpGs (or any other predefined sequence pattern) covered by the given short reads and to test the depth of coverage.

Usage

```
MEDIPS.seqCoverage(file = NULL, BSgenome = NULL, pattern = "CG", extend = 0, shift = 0, uniq = 1e-3, c
```

Arguments

`file`	Path and file name of the input data
`BSgenome`	The reference genome name as defined by BSgenome
`pattern`	defines the sequence pattern, e.g. CG for CpGs.
`extend`	defines the number of bases by which the region will be extended before the genome vector is calculated. Regions will be extended along the plus or the minus strand as defined by their provided strand information. Please note, the extend and shift parameter are mutual exclusive.
`shift`	defines the number of bases by which the region will be shifted before the genome vector is calculated. Regions will be shifted along the plus or the minus strand as defined by their provided strand information. Please note, the extend and shift parameter are mutual exclusive.
`uniq`	The uniq parameter determines, if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0) (deafult: 1e-3). The smaller the p-value, the more reads at the same genomic position are potentially allowed.
`chr.select`	specify a subset of chromosomes for which the saturation analysis is performed.
`paired`	option for paired end reads
`isSecondaryAlignment`	option to import only primary alignments.
`simpleCigar`	option to import only alignments with simple Cigar string.

Author(s)

Lukas Chavez

Examples

```
library(MEDIPSData)
library(BSgenome.Hsapiens.UCSC.hg19)
bam.file.hESCs.Rep1.MeDIP = system.file("extdata", "hESCs.MeDIP.Rep1.chr22.bam", package="MEDIPSData")

cr = MEDIPS.seqCoverage(file=bam.file.hESCs.Rep1.MeDIP, BSgenome="BSgenome.Hsapiens.UCSC.hg19", pattern="CG"
```

MEDIPS.setAnnotation *Funtion to annotate a matrix of genomic coordinates (i.e. a result table) by a given annotation object.*

Description

The function appends any annotation IDs included in the given annotation object to the given regions object. An annotation object can be retrived by the MEDIPS.getAnnotation function and the regions object is typically a (filtered) result table as returned by the MEDIPS.meth function. An annotation ID is appended to a genomic region if their genomic coordinates overlap by at least one base. There will be as many columns added to the regions object as overlapping annotations exist in the annotation object.

Usage

```
MEDIPS.setAnnotation(regions, annotation, cnv=F)
```

Arguments

regions	a matrix that contains row-wise genomic regions, e.g. as a result of the MEDIPS.meth function.
annotation	the annotation data object contains the genomic coordinates of annotations. An annotation object can be e.g. retrived by the MEDIPS.getAnnotation function.
cnv	the MEDIPS.setAnnotation function is also internally used by the MEDIPS.addCNV function which automatically sets this parameter to TRUE. Otherwise cnv should be set to FALSE.

Value

The provided result object with added columns containing overlapping annotations.

Author(s)

Joern Dietrich, Matthias Lienhard

Examples

```
library(MEDIPSData)
data(resultTable)

sig = MEDIPS.selectSig(results=resultTable, p.value=0.05, adj=TRUE, ratio=NULL, bg.counts=NULL, CNV=FALSE)
sig = MEDIPS.mergeFrames(frames=sig, distance=1)
#ens_gene = MEDIPS.getAnnotation( annotation="GENE",chr="chr22")
#sig = MEDIPS.setAnnotation(regions=sig, annotation=ens_gene)
```

MEDIPSroiSet-class *MEDIPSroiSet class and internal functions*

Description

MEDIPSroiSet class is used in the MEDIPS library in order to store and extract objects and information of the specified regions of interest (ROI) from the input file as well as parameter settings specified during the workflow.

Objects from the Class

Objects of the classes contain information about the provided short reads, MeDIP raw/count signals, and further parameter settings. A MEDIPS ROI SET object is created by the MEDIPS.createROIset() function. According slots will be filled during the workflow.

Slots

`genome_name`**:** Object of class `"character"` : the refernce genome

`chr_names`**:** Object of class `"character"` : the names of the chromosomes included within the MEDIPS ROI SET

`chr_lengths`**:** Object of class `"numeric"` : the lengths of the chromosomes included within the MEDIPS ROI SET

`sample_name`**:** Object of class `"character"` : the name of the input file

`path_name`**:** Object of class `"character"` : the path to the input file

`number_regions`**:** Object of class `"numeric"` : the total number of included regions

`genome_count`**:** Object of class `"numeric"` : the raw MeDIP-seq signals at the bins

`extend`**:** Object of class `"numeric"` : the length of the reads after extension

`shifted`**:** Object of class `"numeric"` : the number of bases by which the reads are shifted along the sequencing direction

`uniq`**:** Object of class `"logical"` : determines if reads mapping to exactly the same genomic position should be replaced by only on representative

`ROI`**:** Object of class `"GRanges"`: the genomic positions of the regions of interest

`bin_number`**:** Object of class `"numeric"`: the number of bins per region

Methods

genome_name `signature(object = "MEDIPSroiSet")`: extracts the reference genome of the MEDIPS ROI SET

bin_number `signature(object = "MEDIPSroiSet")`: extracts the number of bins per ROI the bin_number slot of the MEDIPS ROI SET

chr_names `signature(object = "MEDIPSroiSet")`: extracts the names of the chromosomes included within the MEDIPS ROI SET

chr_lengths `signature(object = "MEDIPSroiSet")`: extracts the length of the chromosomes included within the MEDIPS ROI SET

sample_name `signature(object = "MEDIPSroiSet")`: extracts the name of the input file

path_name `signature(object = "MEDIPSroiSet")`: extracts the path to the input file

number_regions `signature(object = "MEDIPSroiSet")`: extracts the total number of included regions

genome_count `signature(object = "MEDIPSroiSet")`: extracts the raw MeDIP-Seq signals at the genomic bins

extend `signature(object = "MEDIPSroiSet")`: extracts the number of bases by which the regions are extended

show `signature(object = "MEDIPSroiSet")`: prints a summary of the MEDIPS SET object content

shifted `signature(object = "MEDIPSroiSet")`: extracts the number of bases by which the regions are shifted

uniq `signature(object = "MEDIPSroiSet")`: extracts the specified value for the uniq parameter

rois `signature(object = "MEDIPSroiSet")`: extracts the GRange object containing the regions of interest

MEDIPS.calibrationCurve `signature(MSet = "MEDIPSroiSet", CSet="COUPLINGset")`: internal function for calculating the calibration curve

MEDIPS.negBin `signature(MSet="MEDIPSroiSet", CSet="COUPLINGset")`: internal function for calculating methylatiopn probabilities with respect to CpG density dependent negative binomial distributions

MEDIPS.pois `signature(MSet="MEDIPSroiSet", CSet="COUPLINGset")`: internal function for calculating methylatiopn probabilities with respect to CpG density dependent poisson distributions

MEDIPS.rms `signature(MSet="MEDIPSroiSet", CSet="COUPLINGset")`: internal function for calculating relative methylation scores

Author(s)

Lukas Chavez, Joern Dietrich

Examples

```
showClass("MEDIPSroiSet")
```

`MEDIPSset-class` *MEDIPSset class and internal functions*

Description

MEDIPSset class is used in the MEDIPS library in order to store and extract objects and information from the input file as well as parameter settings specified during the workflow.

Objects from the Class

Objects of the classes contain information about the provided short reads, MeDIP raw/count signals, and further parameter settings. A MEDIPS SET object is created by the MEDIPS.genomeVector() function. According slots will be filled during the workflow.

Slots

- `genome_name`: Object of class `"character"` : the refernce genome
- `window_size`: Object of class `"numeric"` : the window size for the genome vector
- `chr_names`: Object of class `"character"` : the names of the chromosomes included within the MEDIPS/COUPLING SET
- `chr_lengths`: Object of class `"numeric"` : the lengths of the chromosomes included within the MEDIPS/COUPLING SET
- `sample_name`: Object of class `"character"` : the name of the input file
- `path_name`: Object of class `"character"` : the path to the input file
- `number_regions`: Object of class `"numeric"` : the total number of included regions
- `genome_count`: Object of class `"numeric"` : the raw MeDIP-seq signals at the genomic bins
- `extend`: Object of class `"numeric"` : the length of the regions after extension
- `shifted`: Object of class `"numeric"` : the number of bases by which the reads are shifted along the sequencing direction
- `uniq`: Object of class `"logical"` : determines if all reads mapping to exactly the same genomic position should be kept (uniq = 0), replaced by only one representative (uniq = 1), or if the number of stacked reads should be capped by a maximal number of stacked reads per genomic position determined by a poisson distribution of stacked reads genome wide and by a given p-value (1 > uniq > 0).

Methods

- **genome_name** `signature(object = "MEDIPSset")`: extracts the reference genome of the MEDIPS SET
- **window_size** `signature(object = "MEDIPSset")`: extracts the window size from the bin_size slot of the MEDIPS SET
- **chr_names** `signature(object = "MEDIPSset")`: extracts the names of the chromosomes included within the MEDIPS SET
- **chr_lengths** `signature(object = "MEDIPSset")`: extracts the length of the chromosomes included within the MEDIPS SET
- **fragmentLength** `signature(object = "MEDIPSset")`: extracts the estimated fragment length of the DNA fragments
- **sample_name** `signature(object = "MEDIPSset")`: extracts the name of the input file
- **path_name** `signature(object = "MEDIPSset")`: extracts the path to the input file
- **number_regions** `signature(object = "MEDIPSset")`: extracts the total number of included regions
- **genome_count** `signature(object = "MEDIPSset")`: extracts the raw MeDIP-Seq signals at the genomic bins
- **extend** `signature(object = "MEDIPSset")`: extracts the number of bases by which the regions are extended
- **show** `signature(object = "MEDIPSset")`: prints a summary of the MEDIPS SET object content
- **shifted** `signature(object = "MEDIPSset")`: extracts the number of bases by which the regions are shifted
- **uniq** `signature(object = "MEDIPSset")`: extracts the specified value for the uniq parameter

MEDIPS.GenomicCoordinates `signature(object = "MEDIPSset")`: internal function for calculating coordinates for the genomic bins

MEDIPS.readRegionsFile `signature(object = "MEDIPSset")`: internal function for reading short read information

MEDIPS.calibrationCurve `signature(object = "MEDIPSset")`: internal function for calculating the calibration curve

MEDIPS.cnv `signature(object = "MEDIPSset")`: internal function for calculating CNVs in case two groups of INPUT SETs have been provided to the MEDIPS.meth function

MEDIPS.diffMeth `signature(object = "MEDIPSset")`: internal function for calculating differential coverage in case two groups of MEDIPS SETs have been provided to the MEDIPS.meth function

MEDIPS.getPositions `signature(object = "MEDIPSset")`: internal function for receiving genomic coordinates of a given sequence pattern (e.g. CG)

MEDIPS.rms `signature(object = "MEDIPSset")`: internal function for calculating relative methylation scores

matNnotNA `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matMin `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matDiff `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matMax `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matMean `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matSd `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

matTtest `signature(object = "MEDIPSset")`: internal function for vectorized calculation of the t.test

Author(s)

Lukas Chavez, Joern Dietrich

Examples

```
showClass("MEDIPSset")
```

Index

∗ **classes**
  COUPLINGset-class, 3
  MEDIPSroiSet-class, 25
  MEDIPSset-class, 26
∗ **package**
  MEDIPS-package, 2

adjustReads *(MEDIPSset-class)*, 26

bin.ROIs *(MEDIPS.createROIset)*, 8
bin_number *(MEDIPSroiSet-class)*, 25
bin_number,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
bin_size *(MEDIPSset-class)*, 26
bin_size,MEDIPSset-method *(MEDIPSset-class)*, 26

chr_lengths *(MEDIPSset-class)*, 26
chr_lengths,COUPLINGset-method *(COUPLINGset-class)*, 3
chr_lengths,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
chr_lengths,MEDIPSset-method *(MEDIPSset-class)*, 26
chr_names *(MEDIPSset-class)*, 26
chr_names,COUPLINGset-method *(COUPLINGset-class)*, 3
chr_names,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
chr_names,MEDIPSset-method *(MEDIPSset-class)*, 26
COUPLINGset *(COUPLINGset-class)*, 3
COUPLINGset-class, 3

extend *(MEDIPSset-class)*, 26
extend,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
extend,MEDIPSset-method *(MEDIPSset-class)*, 26

genome_CF *(COUPLINGset-class)*, 3
genome_CF,COUPLINGset-method *(COUPLINGset-class)*, 3
genome_count *(MEDIPSset-class)*, 26
genome_count,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
genome_count,MEDIPSset-method *(MEDIPSset-class)*, 26
genome_name *(MEDIPSset-class)*, 26
genome_name,COUPLINGset-method *(COUPLINGset-class)*, 3
genome_name,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
genome_name,MEDIPSset-method *(MEDIPSset-class)*, 26
getGRange *(MEDIPSset-class)*, 26
getMObjectFromWIG *(MEDIPS.createSet)*, 9
getPairedGRange *(MEDIPS.createSet)*, 9
getTypes *(MEDIPSset-class)*, 26

matDiff *(MEDIPSset-class)*, 26
matMax *(MEDIPSset-class)*, 26
matMean *(MEDIPSset-class)*, 26
matMin *(MEDIPSset-class)*, 26
matNnotNA *(MEDIPSset-class)*, 26
matSd *(MEDIPSset-class)*, 26
matTtest *(MEDIPSset-class)*, 26
MEDIPS *(MEDIPS-package)*, 2
MEDIPS-package, 2
MEDIPS.addCNV, 4
MEDIPS.Bam2GRanges *(MEDIPSset-class)*, 26
MEDIPS.Bed2Granges *(MEDIPSset-class)*, 26
MEDIPS.calibrationCurve *(MEDIPSset-class)*, 26
MEDIPS.cnv *(MEDIPSset-class)*, 26
MEDIPS.correlation, 5
MEDIPS.couplingVector, 6
MEDIPS.CpGenrich, 6
MEDIPS.createROIset, 8
MEDIPS.createSet, 9
MEDIPS.diffMeth *(MEDIPS.meth)*, 14
MEDIPS.exportWIG, 10
MEDIPS.GenomicCoordinates *(MEDIPSset-class)*, 26
MEDIPS.getAnnotation, 11
MEDIPS.getPositions *(MEDIPSset-class)*, 26
MEDIPS.mergeFrames, 12

MEDIPS.mergeSets, 13
MEDIPS.meth, 14
MEDIPS.plotCalibrationPlot, 16
MEDIPS.plotSaturation, 17
MEDIPS.plotSeqCoverage, 18
MEDIPS.rms *(MEDIPSset-class)*, 26
MEDIPS.saturation, 19
MEDIPS.selectROIs, 21
MEDIPS.selectSig, 22
MEDIPS.seqCoverage, 23
MEDIPS.setAnnotation, 24
MEDIPSroiSet *(MEDIPSroiSet-class)*, 25
MEDIPSroiSet-class, 25
MEDIPSset *(MEDIPSset-class)*, 26
MEDIPSset-class, 26

number_pattern *(COUPLINGset-class)*, 3
number_pattern,COUPLINGset-method *(COUPLINGset-class)*, 3
number_regions *(MEDIPSset-class)*, 26
number_regions,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
number_regions,MEDIPSset-method *(MEDIPSset-class)*, 26

path_name *(MEDIPSset-class)*, 26
path_name,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
path_name,MEDIPSset-method *(MEDIPSset-class)*, 26

readRegionsFile *(MEDIPS.createSet)*, 9
ROI,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
rois *(MEDIPSroiSet-class)*, 25
rois,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25

sample_name *(MEDIPSset-class)*, 26
sample_name,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
sample_name,MEDIPSset-method *(MEDIPSset-class)*, 26
scanBamToGRanges *(MEDIPS.createSet)*, 9
seq_pattern *(COUPLINGset-class)*, 3
seq_pattern,COUPLINGset-method *(COUPLINGset-class)*, 3
setTypes *(MEDIPSset-class)*, 26
shifted *(MEDIPSset-class)*, 26
shifted,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
shifted,MEDIPSset-method *(MEDIPSset-class)*, 26
show *(MEDIPSset-class)*, 26
show,COUPLINGset-method *(COUPLINGset-class)*, 3
show,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
show,MEDIPSset-method *(MEDIPSset-class)*, 26

uniq *(MEDIPSset-class)*, 26
uniq,MEDIPSroiSet-method *(MEDIPSroiSet-class)*, 25
uniq,MEDIPSset-method *(MEDIPSset-class)*, 26

window_size *(MEDIPSset-class)*, 26
window_size,COUPLINGset-method *(COUPLINGset-class)*, 3
window_size,MEDIPSset-method *(MEDIPSset-class)*, 26