Package 'MEDIPS'

September 24, 2012

Type Package

Title MeDIP-Seq data analysis

Version 1.6.0

Date 2010-09-28

Author Lukas Chavez, Joern Dietrich

Maintainer Lukas Chavez <chavez@molgen.mpg.de>

Description MEDIPS was developed for analyzing data derived from methylated DNA immunoprecipitation (MeDIP) experiments followed by sequencing (MeDIP-Seq).
Nevertheless, functionalities like the quality controls may be applied to other types of sequencing data (e.g. ChIP-Seq).
MEDIPS adresses several aspects in the context of MeDIP-Seq data analysis.

License GPL (>=2)

LazyLoad yes

biocViews Sequencing, DNAMethylation, CpGIsland,DifferentialExpression, HighThroughputSequencing, ChIPseq,Preprocessing, QualityControl, Visualization, Methylseq

Depends R (>= 2.12.0), BiocGenerics (>= 0.1.3), BSgenome

Imports methods, BiocGenerics, Biostrings, BSgenome, graphics, gtools,IRanges, stats, utils

Suggests BSgenome.Hsapiens.UCSC.hg19

R topics documented:

`MEDIPS-package` *MeDIP-Seq data analysis*

Description

MEDIPS was developed for analyzing data derived from methylated DNA immunoprecipitation (MeDIP) experiments followed by sequencing (MeDIP-Seq). Nevertheless, functionalities like the quality controls may be applied to other types of sequencing data (e.g. ChIP-Seq). MEDIPS adresses several aspects in the context of MeDIP-Seq data analysis.

Details

Package:	MEDIPS
Type:	Package
Version:	1.0.0
Date:	2010-09-28
License:	GPL (>=2)
LazyLoad:	yes
Depends:	R (>= 2.12.0), BSgenome

Author(s)

Lukas Chavez, Joern Dietrich

Maintainer: Lukas Chavez <chavez@molgen.mpg.de>

References

Chavez et al., 2010

`MEDIPS.annotate` *Funtion to annotate given genomic coordinates.*

Description

The function annotates any matrix containing genomic coordinates (region) by a given annotation file (anno) containing genomic regions of interest. During a typical MEDIPS workflow, this is of interest for anotating identified differentially methylated regions (DMRs) derived after having xcecuted the MEDIPS.selectSignificants() or MEDIPS.mergeFrames() function. For annotating DMRs, you have to provide an annotation file that contains pre-defined ROIs. For each provided region, the function returns all annotations from the provided annotation file. In case there are several overlapping annotations, the region is returned several times in sparated rows, each row associated to one annotation.

Usage

```
MEDIPS.annotate(region, anno)
```

Arguments

region	a matrix that contains row-wise genomic regions, e.g. DMRs. The columns are: chromosome, start, stop.
anno	the annotation data object contains row-wise the genomic coordinates of annotations. The columns are: chromosome, start, stop, ID

Value

The annotation function returns a matrix where the rows contain the regions from the given frames object (here DMRs) and the columns are:

chr	the chromosome name of the DMR
start	the start position of the DMR
stop	the stop position of the DMR
annotation	the name of the annotation

Author(s)

Joern Dietrich

Examples

```
region = list(chr="chr22", start=25170186, stop=25170687)
anno = system.file("extdata", "hg19.chr22.txt", package="MEDIPS")

annotated = MEDIPS.annotate(region=region, anno=anno)

annotated
```

MEDIPS.calibrationCurve

Function that calculates the calibration curve

Description

Based on the calculated genome vector (MEDIPS.genomeVector) and on the coupling vector (MEDIPS.couplingVector) of a MEDIPS SET, the function examins the dependency of local MeDIP-Seq signal intensities and local pattern (e.g. CpG) densities. Calculation of the calibration curve is achieved by first dividing the total range of coupling factors into several levels. Second, all genomic bins are partitioned into these levels by considering their associated coupling factors. Finally, for each level of coupling factors, MEDIPS calculates the mean signal and mean coupling factor of all genomic bins that fall into this level. The calibration curve represents these averaged signals and coupling factors over the full range of coupling factors. It indicates the experiment specific dependency between signal intensity and CpG density. Subsequently, the function performs a linear regression for small coupling factors of the calibration curve and records the slope and intercept of the resulting linear curve.

Usage

```
MEDIPS.calibrationCurve(data = NULL)
```

Arguments

data	has to be a MEDIPS SET object

Value

The slots of the stated MEDIPS SET object associated to the calibration curve will be occupied afterwards. These are the informations about the mean signals and mean coupling factors representing the calibration curve and the estimated normalization parameters intercept and slope.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")

CONTROL.SET = MEDIPS.calibrationCurve(data = CONTROL.SET)
```

MEDIPS.couplingVector *Calculates the sequence pattern densities for genome wide bins.*

Description

Based on the coordinates of the bins of the genome vector included in the stated MEDIPS SET object, the function calculates the local density of a pre-defined sequence pattern (e.g. CpGs).

Usage

```
MEDIPS.couplingVector(data = NULL, distFile = "empty", fragmentLength = 700, func = "count")
```

Arguments

data	has to be a MEDIPS SET object
distFile	only accessed, if the parameter func=custom. By setting the parameter func to custom, it is required to specify any custom distance weights file using the parameter distFile.
fragmentLength	Only sequence pattern (e.G. CpGs) within the range of (bin_position-fragmentLength), bin_position+fragmentLength] will contribute to the final local coupling factor. The optimized value for the fragmentLength will reflect the estimated size of your sonicated DNA fragments.
func	There are several possible weghting function. MEDIPS supports setting the weighting function parameter func to count: simply count the number of CpGs within the predifined maximal distance to the current bin; linear: the weights for CpGs decreases in a linear way and end at 0 at the predifined maximal distance to the current bin; exp: the weights for CpGs decreases in an exponential way and end at 0 at the predifined maximal distance to the current bin; log: the weights for CpGs decreases in a logarithmic way and end at 0 at the predifined maximal distance to the current bin; custom: by setting the parameter to custom, it is required to specify a custom distance weights file using the parameter distFile. You can create any of such a distance file by your own and specify it here. Here, the fragmentLength parameter will be neglected and the maximal distance within your provided distance file will be the limit.

Value

The slots of the stated MEDIPS SET object associated to the coupling vector will be occupied afterwards. These are the informations about the selected distance function, possibly about the provided distance weights file, the fragment length and the calculated coupling factors for the genomic bins.

Author(s)

Lukas Chavez and Joern Dietrich

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
```

```
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")

CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")
```

`MEDIPS.coverageAnalysis`

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 coverage analysis is to test the number of CpGs (or any other predefined sequence pattern) covered by the given short reads and to have a look at the depth of coverage. Before you can start the coverage analysis, it is necessary that you have created a MEDIPS SET and excecuted the MEDIPS.getPositions function. For the coverage analysis, the total set of available regions is divided into random subsets of equal size where the number of subsets is determined by the parameter no_iterations. The coverage analysis iteratively selects an increasing number of subsets and and tests how many CpGs are covered by the available regions. Moreover, it is tested how many CpGs are covered at least 1x, 2x, 3x, 4x, 5x, and 10x. These levels of coverage depths can be adjusted by setting the coverages parameter (see below). As the regions are typically of short length (e.g. 36bp), it is recommended to extend the region length by an extend value.

Usage

```
MEDIPS.coverageAnalysis(data = NULL, coverages = c(1, 2, 3, 4, 5, 10), no_iterations = 10, no_ra
```

Arguments

`data`	has to be a MEDIPS SET object
`coverages`	default is c(1, 2, 3, 4, 5, 10). The coverages define the depth levels for testing how often a CpG was covered by the given regions. Just specify any other vector of coverage depths you would like to test.
`no_iterations`	defines the number of subsets created from the full set of available regions (default=10).
`no_random_iterations`	approaches that randomly select data entries may be processed several times in order to obtain more stable results. By specifying the no_random_iterations parameter (default=1) it is possible to run the coverage analysis several times. The final results returned to the coverage results object are the averaged results of each random iteration step.
`extend`	extends the region lengths before the coverage analysis is performed.

Value

`matrix`	Contains the number of covered CpGs in each iteration (rows) and for different levels of coverages (columns)
`maxPos`	is the total number of sequence patterns (e.g. CpGs) within the refernce genome
`pattern`	is the defined sequence pattern

coveredPos — shows the number of covered sequence pattern (e.g. CpGs) using the total set of available regions for several depths of coverages (columns). The last row shows the percentage of covered sequence pattern relative to the total number of available sequence patterns within the reference genome.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")

cr.control = MEDIPS.coverageAnalysis(data = CONTROL.SET, extend = 400, no_iterations = 10)

cr.control
```

MEDIPS.CpGenrich — *Calculates the enrichment of provided CpG rich regions 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 reference genome of the stated MEDIPS SET. 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(data = data, extend = NULL)
```

Arguments

data — has to be a MEDIPS SET object

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.

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

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET=MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)

enrich=MEDIPS.CpGenrich(data=CONTROL.SET)

enrich
```

MEDIPS.exportWIG *Exports reads per million, relative methylation score or sequence pattern density into a wiggle file.*

Description

The funtion allows for exporting the calculated methylation values (rpm or rms) or sequence pattern denisties from a MEDIPS SET into a wiggle (WIG) file. The wiggle file contains values for all genomic bins of the genome vector and can be used for data visualization using appropiate genome browsers.

Usage

```
MEDIPS.exportWIG(data = NULL, file = NULL, raw = FALSE, descr = "", pattern.density = FALSE)
```

Arguments

data	has to be a MEDIPS SET object
file	degines the name of the exported file
raw	if set to TRUE, the reads per million values will be exported. If set to FALSE, the rms values will be exported (default=FALSE).
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.
pattern.density	if set to TRUE, the wiggle file will contain the sequence pattern densities of the coupling vector instead of the methylation values (default=FALSE).

Value

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

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)

MEDIPS.exportWIG(file = "example.output.WIG", data = CONTROL.SET, raw = TRUE, descr = "example")
```

MEDIPS.genomeVector *Calculates the genome wide short read coverage on a user specified resolution*

Description

Based on the regions included within a previously created MEDIPS SET (see MEDIPS.readAlignedSequiences), the function calculates the genome wide coverage on a user specified resolution. Each chromosome inside the MEDIPS SET will be divided into bins of size bin_size and the short read coverage will be calculated on this resolution. The bin representation of the genome is the 'genome vector'.

Usage

```
MEDIPS.genomeVector(data = NULL, extend = 400, bin_size = 50)
```

Arguments

data	has to be a MEDIPS SET object
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.
bin_size	defines the size of genome wide bins and therefore, the size of the genome vector. Read coverages and coupling factors will be calculated for bins separated by bin_size base pairs.

Value

The slots of the stated MEDIPS SET object associated to the genome vector will be occupied afterwards. These are the informations about the bin_size, the extend value, the chromosome and position of the bins, and the number regions within the MEDIPS SET that overlap with the genomic bin.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)

CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
```

MEDIPS.getPositions *Identifies genomic sequence pattern positions within the reference genome.*

Description

The function identifies the genomic positions of the stated sequence pattern (e.g. CpGs). For sequence pattern that are reverse complementary, only the positions on the plus strand will be returned. Otherwise, all genomic positions of the pattern on the plus and minus strand will be returned. The reference genome is the genome (or only some chromosomes of a genome) that was specified by excecuting the MEDIPS.readAlignedSequences function.

Usage

```
MEDIPS.getPositions(data = NULL, pattern = NULL)
```

Arguments

data	has to be a MEDIPS SET object
pattern	defines the sequence pattern, e.g. CG for CpGs.

Value

The slots of the stated MEDIPS SET object associated to the sequence pattern will be occupied afterwards. These are informations about the pattern itself and their chromosome and genomic positions.

Author(s)

Joern Dietrich

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)

CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
```

MEDIPS.mergeFrames *Merges genomic coordinates of overlapping frames into one supersized frame*

Description

In case, the MEDIPS.diffMethyl function was excecuted by setting the value of the step parameter < the value of frame_size the parameter, one may end up with overlapping significant frames. For these cases it is worthwhile to merge overlapping regions into one supersized frame.

Usage

```
MEDIPS.mergeFrames(frames = NULL)
```

Arguments

frames	is a matrix received by the MEDIPS.selectSignificants() function.

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

The results table does not contain any merged rpm, rms, variance, p.value, etc. values.

Author(s)

Lukas Chavez

Examples

```
regions=as.data.frame(list(chr=c("chr22", "chr22"), start=c(1000, 1250), stop=c(1500,1750)))
regions.merged=MEDIPS.mergeFrames(regions)

regions.merged
```

MEDIPS.methylProfiling

Funtion calculates mean methylation values (rpm, rms) and ams values, ratios, variances, and pvalues comparing two MEDIPS SETs for user supplied regions of interests (ROIs) or genome wide frames.

Description

In order to compare two different conditions, first you have to create and process two MEDIPS SETs. For the identification of DMRs, MEDIPS provides two alternative approaches. First, you can specify pre-defined regions of interest (ROIs). Second, MEDIPS offers the possibility to calculate differential methylation for genome wide frames. The function calculates summarized methylation values for the defined ROIs. Here, these are the mean values for both provided MEDIPS SETs as well as the ratio of means. Moreover, for each ROI, MEDIPS calculates p-values by comparing the set of rpm values (or rms values, respectively) within the ROI of the one MEDIPS SET against the set of rpm values (or rms values, respectively) within the ROI of the second MEDIPS SET using R's wilcox.test and t.test functions. Additionally, it is recommended (but not necessary) to provide background data from an INPUT experiment (that is sequencing of none-enriched DNA fragments). By providing an INPUT data set, MEDIPS additionally returns mean INPUT rpm values for the specified ROIs. Please note, the function takes a long processing time when called for genome wide short windows (up to days).

Usage

```
MEDIPS.methylProfiling(data1 = NULL, data2 = NULL, input = NULL, ROI_file = NULL, frame_size = N
```

Arguments

data1	has to be a MEDIPS SET object (the control data)
data2	has to be a MEDIPS SET object (the treatment data)
input	has to be a MEDIPS SET object (the input data)
ROI_file	instead of processing genome wide frames using the parameters frame_size and step, here you can provide a file containing predefined ROIs.
frame_size	Besides summarizing methylation values for pre-defined ROIs, MEDIPS allows for calculating mean methylation values along the full chromosomes. For this, you have to specify a desired frame size here.
math	default=mean; Here, you can specify other functions available in R for sumarizing values like median or sum.
step	The step parameter defines the number of bases by which the frames are shifted along the chromosome. If you e.g. set the frame_size parameter to 500 and the step parameter to 250, then MEDIPS calculates mean methylation values for overlapping 500bp windows, where the size of the overlap will be 250bp for all neighbouring windows.

select	can be either 1 or 2. If set to 1, the variance, ratio, and p-values will be calculated based on the rpm values; if set to 2, the rms values will be considered instead.
chr	only the specified chromosome will be evaluated (e.g. chr1)
transf	If set to TRUE, MEDIPS transforms the mean rms and ams values into log2 scale and subsequently transforms their resulting data range into the consistent interval $[0,1000]$ before finally stored.

Value

chr	the chromosome of the ROI
start	the start position of the ROI
stop	the stop position of the ROI
length	the number of genomic bins included in the ROI
coupling	the mean coupling factor of the ROI
input	the mean reads per million value of the INPUT MEDIPS SET at input (if provided)
rpm_A	the mean reads per million value for the MEDIPS SET at data1
rpm_B	the mean reads per million value for the MEDIPS SET at data2
rms_A	the mean relative mathylation score for the MEDIPS SET at data1
rms_B	the mean relative methylation score for the MEDIPS SET at data2
ams_A	the mean absolute mathylation score for the MEDIPS SET at data1. The ams scores are derived by dividing the mean rms value of the ROI by the mean coupling factor of the ROI before the log2 and interval transformations are performed.
ams_B	the mean absolute mathylation score for the MEDIPS SET at data2. The ams scores are derived by dividing the mean rms value of the ROI by the mean coupling factor of the ROI before the log2 and interval transformations are performed.
var_A	the variance of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data1
var_B	the variance of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data2
var_co_A	the variance coefficient of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data1
var_co_B	the variance coefficient of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data2
ratio	rpm_A/rpm_B or rms_A/rms_B, respectively (please see the parameter select)
pvalue.wilcox	the p.value returned by R's wilcox.test function for comparing the rpm values (or rms values, respectively; please see the parameter select) of the MEDIPS SET at data1 and of the MEDIPS SET at data2
pvalue.ttest	the p.value returned by R's t.test function for comparing the rpm values (or rms values, respectively; please see the parameter select) of the MEDIPS SET at data1 and of the MEDIPS SET at data2

Author(s)

Joern Dietrich

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")
CONTROL.SET = MEDIPS.calibrationCurve(data = CONTROL.SET)
CONTROL.SET = MEDIPS.normalize(data = CONTROL.SET)
ROI_file = system.file("extdata", "hg19.chr22.txt", package="MEDIPS")

promoter = MEDIPS.methylProfiling(data1 = CONTROL.SET, ROI_file = ROI_file, math = mean, select = 2)
```

MEDIPS.normalize *Function that normalizes raw signals by local sequence pattern (e.g. CpG) densities.*

Description

The normalization function accesses the pre-calculated slope and intercept values derived from the MEDIPS.calibrationCurve function in order to weight the raw signals. The relative methlyation score (rms) for the genomic bins is then defined by rms = x\((y-intercept)/slope), where x is the raw signal and y is the coupling factor of a genomic bin. Based on the total number of regions within the MEDIPS SET, the rms values will be transformed into a reads per million format and afterwards transformed into the log2 scale. In order to make the rms values visualizable by common genome browsers, MEDIPS transforms its resulting data range into the consistent interval [0, 1000] before finally stored.

Usage

```
MEDIPS.normalize(data = NULL)
```

Arguments

data	has to be a MEDIPS SET object

Value

The slot of the stated MEDIPS SET object associated to the rms values will be occupied afterwards.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
```

```
CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")
CONTROL.SET = MEDIPS.calibrationCurve(data = CONTROL.SET)

CONTROL.SET = MEDIPS.normalize(data = CONTROL.SET)
```

`MEDIPS.plotCalibrationPlot` *Plots the results of the MEDIPS.calibrationCurve function.*

Description

Visualizes the dependency of raw MeDIP-Seq signals and CpG densities together with the results of the calcibration curve calculation.

Usage

```
MEDIPS.plotCalibrationPlot(data=NULL, xrange=NULL, linearFit=FALSE, plot_chr="all", rpm=F, main=
```

Arguments

`data`	has to be a MEDIPS SET object
`xrange`	The mean signal range of the calibration curve typically falls into a low signal range. By setting the xrange parameter to e.g. 50, the calibration plot will only plot genomic bins associated with signals <=50. Therefore, the effect of an increased CpG density to an increased signal can be better visualized, especially if the data contains genomic bins with high signals.
`rpm`	can be either TRUE or FALSE. If set to TRUE, the signals will be transformed into reads per million (rpm) before plotted. The coupling values remain untouched.
`linearFit`	When the parameter linearFit is set to TRUE, the plot contains the calculated linear curve that represents the dependency between signals and CpG densities.
`plot_chr`	default="all". Please don't forget to call a e.g. png("file.png") function before calling the plot command using "all" because R might not be able to plot the full amount of data in reasonable time. Alternatively, you can specify a selected chromosome (e.g. chr1). Here, the plot_chr parameter only affects the plot and does not affect the MEDIPS SET.
`main`	The main parameter is the same as the main parameter for the plot() command. If it remains empty, the main header of the plot will be Calibration plot.

Value

The calibration plot will be visualized.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")
CONTROL.SET = MEDIPS.calibrationCurve(data = CONTROL.SET)

MEDIPS.plotCalibrationPlot(data = CONTROL.SET, linearFit = TRUE, plot_chr = "chr22")
```

MEDIPS.plotCoverage *Function plots the results of the MEDIPS.coverageAnalysis function.*

Description

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

Usage

```
MEDIPS.plotCoverage(coverageObj = NULL)
```

Arguments

coverageObj	The coverage results object returned by the MEDIPS.coverageAnalysis function

Value

The coverage plot will be visualized.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
cr.control = MEDIPS.coverageAnalysis(data = CONTROL.SET, extend = 400, no_iterations = 10)

MEDIPS.plotCoverage(cr.control)
```

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)
```

Arguments

saturationObj	The saturation results object returned by the MEDIPS.saturationAnalysis function

Value

The coverage plot will be visualized.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
sr.control = MEDIPS.saturationAnalysis(data = CONTROL.SET, bin_size = 50, extend = 400, no_iterations = 10

MEDIPS.plotSaturation(sr.control)
```

MEDIPS.readAlignedSequences *Creates a MEDIPS SET by reading a suitable input file*

Description

Reads the input file and creates a MEDIPS SET. After reading the input file, the MEDIPS SET contains the information about the input regions, like 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 the start, stop and strand informations of the regions. All further slots, for example for the weighting parameters and normalized data are still empty and will be filled during the workflow.

Usage

```
MEDIPS.readAlignedSequences(file = NULL, BSgenome = NULL, numrows = -1)
```

Arguments

file	Path and file name of the input data
BSgenome	The reference genome name as defined by BSgenome
numrows	The number of short reads (number of rows) within the input file

Value

An object of class MEDIPSset is returned where the region dependent informations are stored in the according slots. These are informations about the input file, the reference genome, the total number of provided regions, the chromosomes which are covered by the regions, the total chromosome lengths, and the start and stop positions and strand informations of the regions.

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")

CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
```

MEDIPS.saturationAnalysis

Function calculates the saturation/reproducibility of the provided MeDIP-Seq data.

Description

The saturation analysis addresses the question, whether the number of input regions is sufficient to generate a saturated and reproducible methylation profile of the reference genome. The main idea is that an insufficent 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 methylation profile will be reproducible by another independent set of a similar number of short reads.

Usage

```
MEDIPS.saturationAnalysis(data = NULL, no_iterations = 10, no_random_iterations = 1, empty_bins
```

Arguments

data	has to be a MEDIPS SET object
no_iterations	defines the number of subsets created from the full sets of available regions (default=10)

`no_random_iterations`
: approaches that randomly select data entries may be processed several times in order to obtain more stable results. By specifying the no_random_iterations 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 bins instead of considering the counts. Setting this parameter to TRUE is a more robust approach that reduces the effect of possible occuring outliers (these are bins 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.

`bin_size`
: defines the size of genome wide bins and therefore, the size of the genome vector. Read coverages will be calculated for bins separated by bin_size base pairs.

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(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
```

```
sr.control = MEDIPS.saturationAnalysis(data = CONTROL.SET, bin_size = 50, extend = 400, no_iterations = 10,

sr.control
```

MEDIPS.selectSignificants

Selects candidate ROIs that show significant differential methylation between two MEDIPS SETs.

Description

Based on the results matrix returned from the MEDIPS.diffMethyl function, the function selects candidate ROIs that show significant differential methylation between the CONTROL.SET and the TREAT.SET in consideration of the background data included in the INPUT.SET. Filtering for significant frames proceeds in the following order: ROIs that do not contain any data either in the CONTROL.SET nor in the TREAT.SET are neglected first; ROIs associated to p-values > p.value are neglected; ROIs with a CONTROL/TREATMENT ratio < up (or > down, respectively) are neglected; From the INPUT mean rpm distribution, a mean rpm threshold was defined by the quant parameter and all ROIs that have a mean rpm value within the CONTROL.SET (or TREAT.SET, respectively) smaller than the estimated background rpm threshold are discarded; The last filter is again based on the INPUT data. While the latter filter estimates a minimum rpm signal for the CONTROL.SET (or TREAT.SET, respectively) from the total background distribution, we now define that the rpm value from the CONTROL SET (or TREAT.SET, respectively) of a ROI exceeds the local background data of the INPUT.SET by the parameter up. This is, because MeDIP-Seq background data varies along the chromosomes due to varying DNA availability.

Usage

```
MEDIPS.selectSignificants(frames = NULL, input = T, control = T, up = 1.333333, down = 0.75, p.v
```

Arguments

frames	specifies the results table derived from the MEDIPS.diffMethyl
input	default=T; Setting the parameter to TRUE requires that the results table includes a column for summarized rpm values of an INPUT SET. In case, there is no INPUT data available, the input parameter has to be set to a rpm value that will be used as threshold during the subsequent analysis. How to estimate such a threshold without background data is not yet solved by MEDIPS.
control	can be either TRUE or FALSE; MEDIPS allows for selecting frames that are higher methylated in the CONTROL SET compared to the TREAT SET and vice versa but both approaches have to be perfomed in two independent runs. By setting control=T, MEDIPS selects genomic regions, where the CONTROL SET is higher methylated. By setting control=F, MEDIPS selects genomic regions, where the TREAT SET is higher methylated.
up	default=1.333333; defines the lower threshold for the ratio CONTROL/TREAT as well as for the lower ratio for CONTROL/INPUT (if control=T) or TREATMENT/INPUT (if control=F), respectively.
down	default=0.75; defines the upper threshold for the ratio: CONTROL/TREATMENT (only if control=F).

p.value	default=0.01; defines the threshold for the p-values. One of the p-values derived from the wilcox.test or t.test function has to be <= p.value.
quant	default=0.9; from the distribution of all summarized INPUT rpm values, MEDIPS calculates the rpm value that represents the quant quantile of the whole INPUT distribution.

Value

chr	the chromosome of the ROI
start	the start position of the ROI
stop	the stop position of the ROI
length	the number of genomic bins included in the ROI
coupling	the mean coupling factor of the ROI
input	the mean reads per million value of the INPUT MEDIPS SET at input (if provided)
rpm_A	the mean reads per million value for the MEDIPS SET at data1
rpm_B	the mean reads per million value for the MEDIPS SET at data2
rms_A	the mean relative mathylation score for the MEDIPS SET at data1
rms_B	the mean relative methylation score for the MEDIPS SET at data2
ams_A	the mean absolute mathylation score for the MEDIPS SET at data1. The ams scores are derived by dividing the mean rms value of the ROI by the mean coupling factor of the ROI before the log2 and interval transformations are performed.
ams_B	the mean absolute mathylation score for the MEDIPS SET at data2. The ams scores are derived by dividing the mean rms value of the ROI by the mean coupling factor of the ROI before the log2 and interval transformations are performed.
var_A	the variance of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data1
var_B	the variance of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data2
var_co_A	the variance coefficient of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data1
var_co_B	the variance coefficient of the rpm or rms values (please see the parameter select) of the MEDIPS SET at data2
ratio	rpm_A/rpm_B or rms_A/rms_B, respectively (please see the parameter select)
pvalue.wilcox	the p.value returned by R's wilcox.test function for comparing the rpm values (or rms values, respectively; please see the parameter select) of the MEDIPS SET at data1 and of the MEDIPS SET at data2
pvalue.ttest	the p.value returned by R's t.test function for comparing the rpm values (or rms values, respectively; please see the parameter select) of the MEDIPS SET at data1 and of the MEDIPS SET at data2

Author(s)

Lukas Chavez

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
file=system.file("extdata", "MeDIP_hESCs_chr22.txt", package="MEDIPS")
CONTROL.SET = MEDIPS.readAlignedSequences(BSgenome="BSgenome.Hsapiens.UCSC.hg19", file=file)
CONTROL.SET = MEDIPS.genomeVector(data = CONTROL.SET, bin_size = 50, extend = 400)
CONTROL.SET = MEDIPS.getPositions(data = CONTROL.SET, pattern = "CG")
CONTROL.SET = MEDIPS.couplingVector(data = CONTROL.SET, fragmentLength = 700, func = "count")
CONTROL.SET = MEDIPS.calibrationCurve(data = CONTROL.SET)
CONTROL.SET = MEDIPS.normalize(data = CONTROL.SET)

file=system.file("extdata", "MeDIP_DE_chr22.txt", package="MEDIPS")
TREAT.SET = MEDIPS.readAlignedSequences(BSgenome = "BSgenome.Hsapiens.UCSC.hg19", file = file)
TREAT.SET = MEDIPS.genomeVector(data = TREAT.SET, bin_size = 50, extend = 400)
TREAT.SET = MEDIPS.getPositions(data = TREAT.SET, pattern = "CG")
TREAT.SET = MEDIPS.couplingVector(data = TREAT.SET, fragmentLength = 700, func = "count")
TREAT.SET = MEDIPS.calibrationCurve(data = TREAT.SET)
TREAT.SET = MEDIPS.normalize(data = TREAT.SET)

file=system.file("extdata", "Input_StemCells_chr22.txt", package="MEDIPS")
INPUT.SET = MEDIPS.readAlignedSequences(BSgenome = "BSgenome.Hsapiens.UCSC.hg19", file = file)
INPUT.SET = MEDIPS.genomeVector(data = INPUT.SET, bin_size = 50, extend = 400)

diff.methyl = MEDIPS.methylProfiling(data1 = CONTROL.SET, data2= TREAT.SET, input=INPUT.SET, chr="chr22", f

diff.methyl.sig=MEDIPS.selectSignificants(diff.methyl)
```

MEDIPSset-class *Class "MEDIPSset"*

Description

This class is used in the MEDIPS library to store and extract all the MEDIPS objects and inormations generated during the workflow

Objects from the Class

Objects can be created by calls of the form `new("MEDIPSset", ...)`. Objects of the class contain the information about the provided regions, their chromosome, start, stop and strand informations, the raw and normalized signals, the sequence pattern positions and informations about all specified parameters. The MEDIPS SET will be created by reading the input regions file using the MEDIPS.readAlignedSequences function and all further slots will be filled during the workflow.

Slots

`regions_chr`: Object of class `"character"` : the chromosomes of the regions

`regions_start`: Object of class `"numeric"` : the start positions of the regions

`regions_stop`: Object of class `"numeric"` : the stop positions of the regions

`regions_strand`: Object of class `"character"` : the strand informations of the regions

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

`pattern_chr`: Object of class `"character"` : the chromosomes of the sequence pattern

`pattern_pos`: Object of class `"numeric"` : the positions of the sequence pattern

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

`genome_chr`: Object of class `"character"` : the chromosome of the genomic bins

`genome_pos`: Object of class `"numeric"` : the positions of the genomic bins

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

`genome_raw`: Object of class `"numeric"` : the raw MeDIP-Seq signals at the genomic bins

`genome_norm`: Object of class `"numeric"` : the rms values at the genomic bins

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

`bin_size`: Object of class `"numeric"` : the bin size for the genome vector

`extend`: Object of class `"numeric"` : the number of bases by which the regions are extended

`fragmentLength`: Object of class `"numeric"` : the estimated fragment length of the DNA fragments

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

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

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

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

`distFunction`: Object of class `"character"` : the distance function specified for calculating the coupling factors

`distFile`: Object of class `"character"` : the file containing the custom distance weights, if provided

`calcurve_mean_signals`: Object of class `"numeric"` : the mean signals of the calibration curve

`calcurve_mean_coupling`: Object of class `"numeric"` : the mean coupling factors of the calibration curve

`calcurve_var`: Object of class `"numeric"` : the signal variance of the levels of the calibration curve

`intercept`: Object of class `"numeric"` : the intercept value calculated from the calibration curve

`slope`: Object of class `"numeric"` : the slope value calculated from the calibration curve

`cali_chr`: Object of class `"character"` : the chromosomes used for calibration

Methods

bin_size `signature(object = "MEDIPSset")`: extracts the bins size from the bin_size slot

calcurve_mean_coupling `signature(object = "MEDIPSset")`: extracts the mean coupling factors of the calibration curve

calcurve_mean_signals `signature(object = "MEDIPSset")`: extracts the mean signals of the calibration curve

calcurve_var `signature(object = "MEDIPSset")`: extracts the variances of the calibration curve

cali_chr `signature(object = "MEDIPSset")`: extracts the chromosome used for calculating the calibration curve

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

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

distFile `signature(object = "MEDIPSset")`: extracts the name of file containing the custom distance weights, if provided

distFunction `signature(object = "MEDIPSset")`: extracts the distance function specified for calculating the coupling factors

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

fragmentLength `signature(object = "MEDIPSset")`: extracts the estimated fragment length of the DNA fragments

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

genome_chr `signature(object = "MEDIPSset")`: extracts the chromosome of the genomic bins

genome_name `signature(object = "MEDIPSset")`: extracts the refernce genome

genome_norm `signature(object = "MEDIPSset")`: extracts the rms values at the genomic bins

genome_pos `signature(object = "MEDIPSset")`: extracts the positions of the genomic bins

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

intercept `signature(object = "MEDIPSset")`: extracts the intercept value calculated from the calibration curve

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

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

pattern_chr `signature(object = "MEDIPSset")`: extracts the chromosomes of the sequence pattern

pattern_pos `signature(object = "MEDIPSset")`: extracts the positions of the sequence pattern

regions_chr `signature(object = "MEDIPSset")`: extracts the chromosomes of the regions

regions_start `signature(object = "MEDIPSset")`: extracts the start positions of the regions

regions_stop `signature(object = "MEDIPSset")`: extracts the stop positions of the regions

regions_strand `signature(object = "MEDIPSset")`: extracts the strand informations of the regions

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

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

show `signature(object = "MEDIPSset")`: prints a summary of the object content

slope `signature(object = "MEDIPSset")`: extracts the slope value calculated from the calibration curve

MEDIPS.distributeReads `signature(object = "MEDIPSset")`: help function for distributing the reads over the genome vector

MEDIPS.transform `signature(object = "MEDIPSset")`: help function for transforming ams values

Author(s)

Lukas Chavez, Joern Dietrich

Examples

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

Index