

# Package ‘DMRcaller’

April 7, 2026

**Type** Package

**Title** Differentially Methylated Regions Caller

**Version** 1.43.1

**Date** 2019-01-18

**Encoding** UTF-8

**Author** Nicolae Radu Zabet <r.zabet@qmul.ac.uk>, Jonathan Michael Foonlan Tsang <jmft2@cam.ac.uk>, Alessandro Pio Greco <apgrec@essex.ac.uk>, Ryan Merritt <rmerrri@essex.ac.uk> and Young Jun Kim <qc25039@qmul.ac.uk>

**Maintainer** Nicolae Radu Zabet <r.zabet@qmul.ac.uk>

**Description** Uses Bisulfite sequencing data in two conditions and identifies differentially methylated regions between the conditions in CG and non-CG context. The input is the CX report files produced by Bismark and the output is a list of DMRs stored as GRanges objects.

**License** GPL-3

**LazyLoad** yes

**Imports** parallel, Rcpp, RcppRoll, betareg, grDevices, graphics, methods, stats, utils, Rsamtools, GenomicRanges, GenomicAlignments, Biostrings, BSgenome, BiocManager, S4Vectors, IRanges, InteractionSet, stringr, inflection, BiocParallel, Seqinfo, GenomeInfoDb

**Depends** R (>= 3.5), GenomicRanges, IRanges, S4Vectors

**Suggests** knitr, RUnit, BiocGenerics, rmarkdown, bookdown, BiocStyle, betareg, rtracklayer, BSgenome.Hsapiens.UCSC.hg38

**biocViews** DifferentialMethylation, DNAMethylation, Software, Sequencing, Coverage

**VignetteBuilder** knitr

**NeedsCompilation** no

**RoxygenNote** 7.3.3

**git\_url** <https://git.bioconductor.org/packages/DMRcaller>

**git\_branch** devel  
**git\_last\_commit** 67d4000  
**git\_last\_commit\_date** 2025-11-12  
**Repository** Bioconductor 3.23  
**Date/Publication** 2026-04-07

## Contents

analyseReadsInsideRegionsForCondition	3
analyseReadsInsideRegionsForConditionPMD	4
computeCoMethylatedPositions	6
computeCoMethylatedRegions	8
computeDMRs	11
computeDMRsReplicates	14
computeMethylationDataCoverage	17
computeMethylationDataSpatialCorrelation	18
computeMethylationProfile	19
computeOverlapProfile	20
computePMDs	22
computeVMDs	25
DMRcaller	27
DMRsNoiseFilterCG	36
extractGC	36
filterDMRs	37
filterPMDs	39
filterVMDs	41
filterVMRsONT	43
GEs	45
GEs_hg38	46
getWholeChromosomes	46
joinReplicates	47
mergeDMRsIteratively	48
mergePMDsIteratively	50
methylationDataList	52
ontSampleGRangesList	53
ont_gr_GM18870_chr1_PMD_bins_1k	54
ont_gr_GM18870_chr1_sorted_bins_1k	54
plotLocalMethylationProfile	55
plotMethylationDataCoverage	57
plotMethylationDataSpatialCorrelation	59
plotMethylationProfile	61
plotMethylationProfileFromData	63
plotOverlapProfile	65
PMDsBinsCG	67
PMDsNoiseFilterCG	68
poolMethylationDatasets	68

poolTwoMethylationDatasets . . . . .	69
readBismark . . . . .	70
readBismarkPool . . . . .	71
readONTbam . . . . .	72
saveBismark . . . . .	74
scanBamChr1Random5 . . . . .	75
selectCytosine . . . . .	75
syntheticDataReplicates . . . . .	77

<b>Index</b>	<b>78</b>
--------------	-----------

---

analyseReadsInsideRegionsForCondition  
*Analyse reads inside regions for condition*

---

## Description

This function extracts from the methylation data the total number of reads, the number of methylated reads and the number of cytosines in the specific context from a region (e.g. DMRs)

## Usage

```
analyseReadsInsideRegionsForCondition(
  regions,
  methylationData,
  context,
  label = "",
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

## Arguments

regions	a <a href="#">GRanges</a> object with a list of regions on the genome; e.g. could be a list of DMRs
methylationData	the methylation data in one condition (see <a href="#">methylationDataList</a> ).
context	the context in which to extract the reads ("CG", "CHG" or "CHH").
label	a string to be added to the columns to identify the condition
parallel	Logical; run in parallel if TRUE.
BPPARAM	A <a href="#">BiocParallelParam</a> object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed <code>BPPARAM\$workers</code> ). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

a [GRanges](#) object with additional four metadata columns

**sumReadsM** the number of methylated reads

**sumReadsN** the total number of reads

**proportion** the proportion methylated reads

**cytosinesCount** the number of cytosines in the regions

**Author(s)**

Nicolae Radu Zabet

**See Also**

[readONTbam](#), [filterDMRs](#), [computedDMRs](#), [DMRsNoiseFilterCG](#), and [mergeDMRsIteratively](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

#load the DMRs in CG context. These DMRs were computed with minGap = 200.
data(DMRsNoiseFilterCG)

#retrive the number of reads in CHH context in WT
DMRsNoiseFilterCGreadsCHH <- analyseReadsInsideRegionsForCondition(
  DMRsNoiseFilterCG[1:10],
  methylationDataList[["WT"]], context = "CHH",
  label = "WT")
```

---

analyseReadsInsideRegionsForConditionPMD

*Analyse reads inside regions for condition*

---

**Description**

This function extracts from the methylation data the total number of reads, the number of methylated reads and the number of cytosines in the specific context from a region (e.g. PMDs)

**Usage**

```
analyseReadsInsideRegionsForConditionPMD(
  regions,
  methylationData,
  context,
  label = "",
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

**Arguments**

regions	a <a href="#">GRanges</a> object with a list of regions on the genome; e.g. could be a list of PMDs
methylationData	the methylation data in one condition (see <a href="#">ontSampleGRangesList</a> ).
context	the context in which to extract the reads ("CG", "CHG" or "CHH").
label	a string to be added to the columns to identify the condition
parallel	Logical; run in parallel if TRUE.
BPPARAM	A <a href="#">BiocParallelParam</a> object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed <code>BPPARAM\$workers</code> ). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

a [GRanges](#) object with additional four metadata columns

**sumReadsM** the number of methylated reads

**sumReadsN** the total number of reads

**proportion** the proportion methylated reads

**cytosinesCount** the number of cytosines in the regions

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[filterPMDs](#), [computePMDs](#), [PMDsNoiseFilterCG](#), and [mergePMDsIteratively](#)

**Examples**

```
# load the ONT methylation data
data(ontSampleGRangesList)

#load the PMDs in CG context. These PMDs were computed with minGap = 200.
data(PMDsNoiseFilterCG)

#retrive the number of reads in CG context in GM18501
PMDsNoiseFilterCGreadsCG <- analyseReadsInsideRegionsForConditionPMD(
  PMDsNoiseFilterCG[1:10],
  ontSampleGRangesList[["GM18501"]], context = "CG",
  label = "GM18501")
```

---

```
computeCoMethylatedPositions
```

*Compute pairwise co-methylation statistics for cytosine sites within regions*

---

**Description**

computeCoMethylatedPositions() calculates pairwise co-methylation between all cytosine sites within each given region, using ONT methylation calls annotated to each site. For each pair of cytosines within the same strand and PMD, it builds a 2x2 contingency table reflecting the overlap state of reads (both methylated, only one methylated, or neither), performs a statistical test (Fisher's exact by default), and reports FDR-adjusted p-values.

**Usage**

```
computeCoMethylatedPositions(
  methylationData,
  regions,
  minDistance = 150,
  maxDistance = 1000,
  minCoverage = 4,
  pValueThreshold = 0.01,
  alternative = "two.sided",
  test = "fisher",
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

**Arguments**

```
methylationData
```

A GRanges object containing cytosine sites, annotated with per-site ONT methylation calls (columns ONT\_Cm, ONT\_C, readsN, etc).

<code>regions</code>	A GRanges object with list including genomic context such as gene and/or transposable elements coordinates which possibly have DMRs, VMRs or PMDs.
<code>minDistance</code>	Minimum distance (in bp) between two cytosines to consider for co-methylation (default: 150).
<code>maxDistance</code>	Maximum distance (in bp) between two cytosines to consider (default: 1000).
<code>minCoverage</code>	Minimum read coverage required for both cytosines in a pair (default: 4).
<code>pValueThreshold</code>	FDR-adjusted p-value threshold for reporting significant co-methylation (default: 0.01).
<code>alternative</code>	indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2 by 2 case. This is used only for Fisher's test.
<code>test</code>	Statistical test to use for co-methylation ("fisher" for Fisher's exact [default], or "permutation" for chi-squared). NOTE: highly recommended to do parallel when use permutation test.
<code>parallel</code>	Logical; run in parallel if TRUE.
<code>BPPARAM</code>	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
<code>cores</code>	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

## Details

Compute Co-Methylation Positions within Regions (CMPs)

Pairwise tests are performed separately for each strand (+ and -) within each region. FDR correction is performed for all pairs within each region and strand.

## Value

A list of length equal to `regions`, where each entry is a `GInteractions` object of significant cytosine pairs (by strand), annotated with:

**C1\_C2** number of reads methylated at both cytosines

**C1\_only** number methylated at only first cytosine

**C2\_only** number methylated at only second cytosine

**neither** number methylated at neither cytosines

**strand** The DNA strand ("+" or "-") on which the two CpGs reside.

**genomic\_position** The original region (from `regions`) containing this cytosines pair, formatted in UCSC or IGV style, e.g. "chr1:1522971-1523970".

**p.value** FDR-adjusted p-value for co-methylation association

## Author(s)

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[readONTbam](#), [computePMDs](#), [ontSampleGRangesList](#)

**Examples**

```
## Not run:
# load the ONT methylation data and PMD data
data(ont_gr_GM18870_chr1_PMD_bins_1k)
data(ont_gr_GM18870_chr1_sorted_bins_1k)

# compute the co-methylations with Fisher's exact test
coMetylationFisher <- computeCoMethylatedPositions(
  ont_gr_GM18870_chr1_sorted_bins_1k,
  regions = ont_gr_GM18870_chr1_PMD_bins_1k[1:4],
  minDistance = 150,
  maxDistance = 1000,
  minCoverage = 4,
  pValueThreshold = 0.01,
  test = "fisher",
  parallel = FALSE)

# compute the co-methylations with Permutation test
coMetylationPermutation <- computeCoMethylatedPositions(
  ont_gr_GM18870_chr1_sorted_bins_1k,
  regions = ont_gr_GM18870_chr1_PMD_bins_1k[1:4],
  minDistance = 150,
  maxDistance = 1000,
  minCoverage = 4,
  pValueThreshold = 0.01,
  test = "permutation",
  parallel = FALSE) # highly recommended to set as TRUE

## End(Not run)
```

---

computeCoMethylatedRegions

*Compute pairwise co-methylation statistics between regions*

---

**Description**

computeCoMethylatedRegions() calculates pairwise correlation statistics for methylation levels across defined genomic regions (e.g., PMDs, Enhancer binding sites). For each region pair within the specified distance range, the function computes per-read methylation proportions and performs correlation testing (Pearson, Spearman, or Kendall). Pairs with strong correlations (beyond user-defined thresholds) and significant p-values (FDR-adjusted) are returned.

**Usage**

```
computeCoMethylatedRegions(
  methylationData,
  regions,
  minDistance = 500,
  maxDistance = 50000,
  minCoverage = 4,
  pValueThreshold = 0.05,
  correlation_test = "pearson",
  minCorrelation = -0.5,
  maxCorrelation = 0.5,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

**Arguments**

methylationData	A GRanges object containing cytosine sites, annotated with ONT methylation metadata (columns ONT_Cm, ONT_C, etc.).
regions	A GRanges object defining genomic regions (e.g., PMDs, Enhancer binding sites) to evaluate for CMRs.
minDistance	Minimum genomic distance (in base pairs) between two regions to be considered (default: 500).
maxDistance	Maximum genomic distance (in base pairs) between two regions (default: 50,000).
minCoverage	Minimum number of shared reads (per region pair) required to compute correlation (default: 4).
pValueThreshold	Significance threshold for FDR-adjusted p-values (default: 0.05).
correlation_test	Statistical method to compute correlation; must be one of "pearson", "spearman", or "kendall" (default: "pearson").
minCorrelation	Minimum allowed correlation value for a significant result (must be in between -1 and 0; default: -0.5).
maxCorrelation	Maximum allowed correlation value for a significant result (must be in between 0 and 1; default: 0.5).
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Details****Compute Co-Methylated Regions (CMRs)**

The function first identifies all region pairs within the user-defined distance range. For each pair, it calculates methylation proportions per read across both regions, extracts common reads, and tests correlation using the selected method. FDR correction is applied globally across all region pairs.

**Value**

A GInteractions object, where each row represents a significantly correlated pair of genomic regions from the input regions. The anchors of each interaction correspond to original regions, and their genomic coordinates are retained in the anchor1 and anchor2 slots.

Additionally, a genomic\_position meta-column is included to indicate the original coordinate ranges (in UCSC/IGV format) for each interaction, aiding downstream interpretation or visualisation.

Each interaction is annotated with:

**correlation** Correlation coefficient between the two regions

**coverage** Number of shared reads used for correlation

**p.value** FDR-adjusted p-value for the correlation test

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[readONTbam](#), [computePMDs](#), [ontSampleGRangesList](#)

**Examples**

```
# Load methylation data and PMD regions
data("ont_gr_GM18870_chr1_sorted_bins_1k")
data("ont_gr_GM18870_chr1_PMD_bins_1k")

# Compute highly correlated regions (Pearson)
coMethylationRegion_pearson <- computeCoMethylatedRegions(ont_gr_GM18870_chr1_sorted_bins_1k,
  ont_gr_GM18870_chr1_PMD_bins_1k[1:5],
  minDistance = 500,
  maxDistance = 50000,
  minCoverage = 4,
  pValueThreshold = 0.05,
  correlation_test = "pearson",
  minCorrelation = -0.5,
  maxCorrelation = 0.5,
  parallel = FALSE,
  BPPARAM = NULL)
```

---

`computeDMRs`*Compute DMRs*

---

## Description

This function computes the differentially methylated regions between two conditions.

## Usage

```
computeDMRs(  
  methylationData1,  
  methylationData2,  
  regions = NULL,  
  context = "CG",  
  method = "noise_filter",  
  windowSize = 100,  
  kernelFunction = "triangular",  
  lambda = 0.5,  
  binSize = 100,  
  test = "fisher",  
  pValueThreshold = 0.01,  
  minCytosinesCount = 4,  
  minProportionDifference = 0.4,  
  minGap = 200,  
  minSize = 50,  
  minReadsPerCytosine = 4,  
  parallel = FALSE,  
  BPPARAM = NULL,  
  cores = NULL  
)
```

## Arguments

<code>methylationData1</code>	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
<code>methylationData2</code>	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ).
<code>regions</code>	a <a href="#">GRanges</a> object with the regions where to compute the DMRs. If NULL, the DMRs are computed genome-wide.
<code>context</code>	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
<code>method</code>	the method used to compute the DMRs ("noise_filter", "neighbourhood" or "bins"). The "noise_filter" method uses a triangular kernel to smooth the number of reads and then performs a statistical test to determine which regions display different levels of methylation in the two conditions. The "neighbourhood" method computes differentially methylated cytosines. Finally, the "bins"

method partitions the genome into equal sized tiling bins and performs the statistical test between the two conditions in each bin. For all three methods, the cytosines or bins are then merged into DMRs without affecting the initial parameters used when calling the differentially methylated cytosines/bins (p-value, difference in methylation levels, minimum number of reads per cytosine).

windowSize	the size of the tiling bins measured in nucleotides. This parameter is required only if the selected method is "noise_filter".
kernelFunction	a character indicating which kernel function to be used. Can be one of "uniform", "triangular", "gaussian" or "epanechnikov". This is required only if the selected method is "noise_filter".
lambda	numeric value required for the Gaussian filter ( $K(x) = \exp(-\lambda x^2)$ ). This is required only if the selected method is "noise_filter" and the selected kernel function is "gaussian".
binSize	the size of the tiling bins in nucleotides. This parameter is required only if the selected method is "bins".
test	the statistical test used to call DMRs ("fisher" for Fisher's exact test or "score" for Score test).
pValueThreshold	DMRs with p-values (when performing the statistical test; see test) higher or equal than pValueThreshold are discarded. Note that we adjust the p-values using the Benjamini and Hochberg's method to control the false discovery rate.
minCytosinesCount	DMRs with less cytosines in the specified context than minCytosinesCount will be discarded.
minProportionDifference	DMRs where the difference in methylation proportion between the two conditions is lower than minProportionDifference are discarded.
minGap	DMRs separated by a gap of at least minGap are not merged. Note that only DMRs where the change in methylation is in the same direction are joined.
minSize	DMRs with a size smaller than minSize are discarded.
minReadsPerCytosine	DMRs with the average number of reads lower than minReadsPerCytosine are discarded.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Value

the DMRs stored as a [GRanges](#) object with the following metadata columns:

**direction** a number indicating whether the region lost (-1) or gain (+1) methylation in condition 2 compared to condition 1.

**context** the context in which the DMRs was computed ("CG", "CHG" or "CHH").

**sumReadsM1** the number of methylated reads in condition 1.

**sumReadsN1** the total number of reads in condition 1.

**proportion1** the proportion methylated reads in condition 1.

**sumReadsM2** the number of methylated reads in condition 2.

**sumReadsN2** the total number reads in condition 2.

**proportion2** the proportion methylated reads in condition 2.

**cytosinesCount** the number of cytosines in the DMR.

**regionType** a string indicating whether the region lost ("loss") or gained ("gain") methylation in condition 2 compared to condition 1.

**pValue** the p-value (adjusted to control the false discovery rate with the Benjamini and Hochberg's method) of the statistical test when the DMR was called.

### Author(s)

Nicolae Radu Zabet and Jonathan Michael Foonlan Tsang

### See Also

[filterDMRs](#), [mergedDMRsIteratively](#), [analyseReadsInsideRegionsForCondition](#) and [DMRsNoiseFilterCG](#)

### Examples

```
# load the methylation data
data(methylationDataList)

# the regions where to compute the DMRs
regions <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E5))

# compute the DMRs in CG context with noise_filter method
DMRsNoiseFilterCG <- computeDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], regions = regions,
  context = "CG", method = "noise_filter",
  windowSize = 100, kernelFunction = "triangular",
  test = "score", pValueThreshold = 0.01,
  minCytosinesCount = 4, minProportionDifference = 0.4,
  minGap = 200, minSize = 50, minReadsPerCytosine = 4,
  cores = 1)

## Not run:
# compute the DMRs in CG context with neighbourhood method
DMRsNeighbourhoodCG <- computeDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], regions = regions,
  context = "CG", method = "neighbourhood",
  test = "score", pValueThreshold = 0.01,
  minCytosinesCount = 4, minProportionDifference = 0.4,
  minGap = 200, minSize = 50, minReadsPerCytosine = 4,
  cores = 1)
```

```
# compute the DMRs in CG context with bins method
DMRsBinsCG <- computeDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], regions = regions,
  context = "CG", method = "bins", binSize = 100,
  test = "score", pValueThreshold = 0.01, minCytosinesCount = 4,
  minProportionDifference = 0.4, minGap = 200, minSize = 50,
  minReadsPerCytosine = 4, cores = 1)

## End(Not run)
```

---

computeDMRsReplicates *Compute DMRs*

---

### Description

This function computes the differentially methylated regions between replicates with two conditions.

### Usage

```
computeDMRsReplicates(
  methylationData,
  condition = NULL,
  regions = NULL,
  context = "CG",
  method = "neighbourhood",
  binSize = 100,
  test = "betareg",
  pseudocountM = 1,
  pseudocountN = 2,
  pValueThreshold = 0.01,
  minCytosinesCount = 4,
  minProportionDifference = 0.4,
  minGap = 200,
  minSize = 50,
  minReadsPerCytosine = 4,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

### Arguments

`methylationData` the methylation data containing all the conditions for all the replicates.

`condition` a vector of strings indicating the conditions for each sample in `methylationData`. Two different values are allowed (for the two conditions).

regions	a <a href="#">GRanges</a> object with the regions where to compute the DMRs. If NULL, the DMRs are computed genome-wide.
context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
method	the method used to compute the DMRs "neighbourhood" or "bins"). The "neighbourhood" method computates differentially methylated cytosines. Finally, the "bins" method partiones the genome into equal sized tiling bins and performs the statistical test between the two conditions in each bin. For all three methods, the cytosines or bins are then merged into DMRs without affecting the inital parameters used when calling the differentially methylated cytosines/bins (p-value, difference in methylation levels, minimum number of reads per cytosine).
binSize	the size of the tiling bins in nucleotides. This parameter is required only if the selected method is "bins".
test	the statistical test used to call DMRs ("betareg" for Beta regression).
pseudocountM	numerical value to be added to the methylated reads before modelling beta regression.
pseudocountN	numerical value to be added to the total reads before modelling beta regression.
pValueThreshold	DMRs with p-values (when performing the statistical test; see test) higher or equal than pValueThreshold are discarded. Note that we adjust the p-values using the Benjamini and Hochberg's method to control the false discovery rate.
minCytosinesCount	DMRs with less cytosines in the specified context than minCytosinesCount will be discarded.
minProportionDifference	DMRs where the difference in methylation proportion between the two conditions is lower than minProportionDifference are discarded.
minGap	DMRs separated by a gap of at least minGap are not merged. Note that only DMRs where the change in methylation is in the same direction are joined.
minSize	DMRs with a size smaller than minSize are discarded.
minReadsPerCytosine	DMRs with the average number of reads lower than minReadsPerCytosine are discarded.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Value

the DMRs stored as a [GRanges](#) object with the following metadata columns:

**direction** a number indicating whether the region lost (-1) or gain (+1) methylation in condition 2 compared to condition 1.



---

computeMethylationDataCoverage  
*Compute methylation data coverage*

---

### Description

This function computes the coverage for bisulfite sequencing data. It returns a vector with the proportion (or raw count) of cytosines that have the number of reads higher or equal than a vector of specified thresholds.

### Usage

```
computeMethylationDataCoverage(  
  methylationData,  
  regions = NULL,  
  context = "CG",  
  breaks = NULL,  
  proportion = TRUE  
)
```

### Arguments

methylationData	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
regions	a <a href="#">GRanges</a> object with the regions where to compute the coverage. If NULL, the coverage is computed genome-wide.
context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
breaks	a numeric vector specifying the different values for the thresholds when computing the coverage.
proportion	a logical value indicating whether to compute the proportion (TRUE) or raw counts (FALSE).

### Value

a vector with the proportion (or raw count) of cytosines that have the number of reads higher or equal than the threshold values specified in the breaks vector.

### Author(s)

Nicolae Radu Zabet and Jonathan Michael Foonlan Tsang

### See Also

[plotMethylationDataCoverage](#), [methylationDataList](#)

## Examples

```
# load the methylation data
data(methylationDataList)

# compute coverage in CG context
breaks <- c(1,5,10,15)
coverage_CG_wt <- computeMethylationDataCoverage(methylationDataList[["WT"]],
                                                  context="CG", breaks=breaks)
```

---

```
computeMethylationDataSpatialCorrelation
      Compute methylation data spatial correlation
```

---

## Description

This function computes the correlation of the methylation levels as a function of the distances between the Cytosines. The function returns a vector with the correlation of methylation levels at distance equal to a vector of specified thresholds.

## Usage

```
computeMethylationDataSpatialCorrelation(
  methylationData,
  regions = NULL,
  context = "CG",
  distances = NULL
)
```

## Arguments

methylationData	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
regions	a <a href="#">GRanges</a> object with the regions where to compute the correlation. If NULL, the correlation is computed genome-wide.
context	the context in which the correlation is computed ("CG", "CHG" or "CHH").
distances	a numeric vector specifying the different values for the distances when computing the correlation.

## Value

a vector with the correlation of the methylation levels for Cytosines located at distances specified in the distances vector.

**Author(s)**

Nicolae Radu Zabet

**See Also**[plotMethylationDataSpatialCorrelation](#), [methylationDataList](#)**Examples**

```
# load the methylation data
data(methylationDataList)

# compute spatial correlation in CG context
distances <- c(1,5,10,15)
correlation_CG_wt <- computeMethylationDataSpatialCorrelation(methylationDataList[["WT"]],
  context="CG", distances=distances)
```

---

```
computeMethylationProfile
  Compute methylation profile
```

---

**Description**

This function computes the low resolution profiles for the bisulfite sequencing data.

**Usage**

```
computeMethylationProfile(
  methylationData,
  region,
  windowSize = floor(width(region)/500),
  context = "CG"
)
```

**Arguments**

methylationData	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
region	a <a href="#">GRanges</a> object with the regions where to compute the DMRs.
windowSize	a numeric value indicating the size of the window in which methylation is averaged.
context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").

**Value**

a [GRanges](#) object with equal sized tiles of the region. The object consists of the following metadata

**sumReadsM** the number of methylated reads.

**sumReadsN** the total number of reads.

**Proportion** the proportion of methylated reads.

**cytosinesCount** the number of cytosines in the regions

**context** the context ("CG", "CHG" or "CHH").

**Author(s)**

Nicolae Radu Zabet and Jonathan Michael Foonlan Tsang

**See Also**

[plotMethylationProfileFromData](#), [plotMethylationProfile](#), [methylationDataList](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

# the region where to compute the profile
region <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E6))

# compute low resolution profile in 20 Kb windows
lowResProfileWTCHH <- computeMethylationProfile(methylationDataList[["WT"]],
        region, windowSize = 20000, context = "CHH")

## Not run:
# compute low resolution profile in 10 Kb windows
lowResProfileWTTCG <- computeMethylationProfile(methylationDataList[["WT"]],
        region, windowSize = 10000, context = "CG")

lowResProfileMet13CG <- computeMethylationProfile(
        methylationDataList[["met1-3"]], region,
        windowSize = 10000, context = "CG")

## End(Not run)
```

---

computeOverlapProfile *Compute Overlaps Profile*

---

**Description**

This function computes the distribution of a subset of regions ([GRanges](#) object) over a large region ([GRanges](#) object)

**Usage**

```
computeOverlapProfile(  
  subRegions,  
  largeRegion,  
  windowSize = floor(width(largeRegion)/500),  
  binary = TRUE,  
  cores = 1  
)
```

**Arguments**

subRegions	a <a href="#">GRanges</a> object with the sub regions; e.g. can be the DMRs.
largeRegion	a <a href="#">GRanges</a> object with the region where to compute the overlaps; e.g. a chromosome
windowSize	The largeRegion is partitioned into equal sized tiles of width windowSize.
binary	a value indicating whether to count 1 for each overlap or to compute the width of the overlap
cores	the number of cores used to compute the DMRs.

**Value**

a [GRanges](#) object with equal sized tiles of the regions. The object has one metadata file score which represents: the number of subRegions overlapping with the tile, in the case of binary = TRUE, and the width of the subRegions overlapping with the tile , in the case of binary = FALSE.

**Author(s)**

Nicolae Radu Zabet

**See Also**

[plotOverlapProfile](#), [filterDMRs](#), [computeDMRs](#) and [mergeDMRsIteratively](#)

**Examples**

```
# load the methylation data  
data(methylationDataList)  
  
# load the DMRs in CG context  
data(DMRsNoiseFilterCG)  
  
# the coordinates of the area to be plotted  
largeRegion <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E5))  
  
# compute overlaps distribution  
hotspots <- computeOverlapProfile(DMRsNoiseFilterCG, largeRegion,  
  windowSize = 10000, binary = FALSE)
```

---

 computePMDs

*Compute PMDs*


---

### Description

This function computes the partially methylated domains between pre-set min and max proportion values.

### Usage

```
computePMDs(
  methylationData,
  regions = NULL,
  context = "CG",
  method = "noise_filter",
  windowSize = 100,
  kernelFunction = "triangular",
  lambda = 0.5,
  binSize = 100,
  minCytosinesCount = 4,
  minMethylation = 0.4,
  maxMethylation = 0.6,
  minGap = 200,
  minSize = 50,
  minReadsPerCytosine = 4,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

### Arguments

methylationData	the methylation data in condition (see <a href="#">ontSampleGRangesList</a> ).
regions	a <a href="#">GRanges</a> object with the regions where to compute the PMDs. If NULL, the PMDs are computed genome-wide.
context	the context in which the PMDs are computed ("CG", "CHG" or "CHH").
method	Character string specifying the algorithm for PMD detection. If "noise_filter", a sliding window of size windowSize is applied with the specified kernelFunction (and lambda for a Gaussian kernel) to smooth methylation counts before calling and merging PMDs. If "neighbourhood", individual partially methylated cytosines are identified first and then merged into PMDs. If "bins", the genome is partitioned into fixed bins of size binSize, partially methylation is sorted per bin, and significant bins are merged.
windowSize	the size of the triangle base measured in nucleotides. This parameter is required only if the selected method is "noise_filter".

kernelFunction	a character indicating which kernel function to be used. Can be one of "uniform", "triangular", "gaussian" or "epanechnikov". This is required only if the selected method is "noise_filter".
lambda	numeric value required for the Gaussian filter ( $K(x) = \exp(-\lambda x^2)$ ). This is required only if the selected method is "noise_filter" and the selected kernel function is "gaussian".
binSize	the size of the tiling bins in nucleotides. This parameter is required only if the selected method is "bins".
minCytosinesCount	PMDs with less cytosines in the specified context than minCytosinesCount will be discarded.
minMethylation	Numeric [0,1]; minimum mean methylation proportion.
maxMethylation	Numeric [0,1]; maximum mean methylation proportion.
minGap	PMDs separated by a gap of at least minGap are not merged. Note that only PMDs where the change in methylation is in the same direction are joined.
minSize	PMDs with a size smaller than minSize are discarded.
minReadsPerCytosine	PMDs with the average number of reads lower than minReadsPerCytosine are discarded.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Value

the PMDs stored as a [GRanges](#) object with the following metadata columns:

**context** the context in which the PMDs was computed ("CG", "CHG" or "CHH").

**sumReadsM** the number of methylated reads.

**sumReadsN** the total number of reads.

**proportion** the proportion methylated reads filtered between minMethylation and maxMethylation.

**cytosinesCount** the number of cytosines in the PMDs.

### Author(s)

Nicolae Radu Zabet, Jonathan Michael Foonlan Tsang and Young Jun Kim

### See Also

[readONTbam](#), [filterPMDs](#), [mergePMDsIteratively](#), [analyseReadsInsideRegionsForConditionPMD](#) and [PMDsNoiseFilterCG](#)

**Examples**

```
# load the ONT methylation data
data(ontSampleGRangesList)

# the regions where to compute the PMDs
chr1_ranges <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))

# compute the PMDs in CG context with noise_filter method
PMDsNoiseFilterCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                                regions = chr1_ranges,
                                context = "CG",
                                windowSize = 100,
                                method = "noise_filter",
                                kernelFunction = "triangular",
                                lambda = 0.5,
                                minCytosinesCount = 4,
                                minMethylation = 0.4,
                                maxMethylation = 0.6,
                                minGap = 200,
                                minSize = 50,
                                minReadsPerCytosine = 4,
                                cores = 1,
                                parallel = FALSE)

## Not run:
# compute the PMDs in CG context with neighbourhood method
PMDsNeighbourhoodCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                                    regions = chr1_ranges,
                                    context = "CG",
                                    method = "neighbourhood"
                                    minCytosinesCount = 4,
                                    minMethylation = 0.4,
                                    maxMethylation = 0.6,
                                    minGap = 200,
                                    minSize = 50,
                                    minReadsPerCytosine = 4,
                                    cores = 1,
                                    parallel = FALSE)

# compute the PMDs in CG context with bins method
PMDsBinsCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                          regions = chr1_ranges,
                          context = "CG",
                          method = "bins",
                          binSize = 100,
                          minCytosinesCount = 4,
                          minMethylation = 0.4,
                          maxMethylation = 0.6,
                          minGap = 200,
                          minSize = 50,
                          minReadsPerCytosine = 4,
                          cores = 1,
```

```

parallel = FALSE)

## End(Not run)

```

---

computeVMDs

*Compute VMDs*


---

## Description

This function computes the variance methylated domains between pre-set min and max proportion values.

## Usage

```

computeVMDs(
  methylationData,
  regions = NULL,
  context = "CG",
  binSize = 100,
  minCytosinesCount = 4,
  sdCutoffMethod = "per.high",
  percentage = 0.05,
  minGap = 200,
  minSize = 50,
  minReadsPerCytosine = 4,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)

```

## Arguments

methylationData	the methylation data in condition (see <a href="#">ontSampleGRangesList</a> ).
regions	a <a href="#">GRanges</a> object with the regions where to compute the VMDs. If NULL, the VMDs are computed genome-wide.
context	the context in which the VMDs are computed ("CG", "CHG" or "CHH").
binSize	the size of the tiling bins in nucleotides. This parameter is required only if the selected method is "bins".
minCytosinesCount	VMDs with less cytosines in the specified context than minCytosinesCount will be discarded.
sdCutoffMethod	Character string specifying how to determine the cutoff for filtering VMDs based on their methylation variance (standard deviation). Available options are: "per.high" Selects the top percentage of regions with the highest variance (standard deviation).

	"per .low" Selects the bottom percentage of regions with the lowest variance.
	"EDE .high" Uses the elbow point (inflection/knee) from the descendingly sorted variance values to determine a data-driven high-variance cutoff. Retains regions with SD above this elbow point.
	"EDE .low" Uses the elbow point from the ascendingly sorted variance values to define a low-variance cutoff. Retains regions with SD below this point.
	This allows either quantile-based filtering or automatic detection of variance thresholds based on distribution shape.
percentage	Numeric cutoff used when sdCutoffMethod is set to "per .high" or "per .low". Represents the quantile threshold: for example, percentage = 0.05 keeps the top 5% or bottom 5% of bins based on standard deviation, depending on the selected method.
minGap	VMDs separated by a gap of at least minGap are not merged. Note that only VMDs where the change in methylation is in the same direction are joined.
minSize	VMDs with a size smaller than minSize are discarded.
minReadsPerCytosine	VMDs with the average number of reads lower than minReadsPerCytosine are discarded.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Value

the VMDs stored as a [GRanges](#) object with the following metadata columns:

**context** the context in which the VMDs was computed ("CG", "CHG" or "CHH").

**sumReadsM** the number of methylated reads.

**sumReadsN** the total number of reads.

**proportion** the proportion from total methylated reads.

**cytosinesCount** the number of cytosines in the VMDs.

**mean** mean value comparing per-read proportions

**sd** standard deviation comparing per-read proportions

**w\_mean** weighted mean value comparing per-read proportions

**w\_sd** weighted standard deviation comparing per-read proportions

### Author(s)

Nicolae Radu Zabet, Jonathan Michael Foonlan Tsang and Young Jun Kim

### See Also

[readONTbam](#), [filterVMDs](#) and [analyseReadsInsideRegionsForCondition](#)

**Examples**

```
## Not run:
# load the ONT methylation data
data(ontSampleGRangesList)

# the regions where to compute the VMDs
chr1_ranges <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,1E6+6E5))

# compute the VMDs in CG context with bins method
VMDsBinsCG <- computeVMDs(ontSampleGRangesList[["GM18501"]],
                          regions = NULL,
                          context = "CG",
                          binSize = 100,
                          minCytosinesCount = 4,
                          sdCutoffMethod = "EDE.high",
                          percentage = 0.05,
                          minGap = 200,
                          minSize = 50,
                          minReadsPerCytosine = 4,
                          parallel = FALSE,
                          BPPARAM = NULL,
                          cores = 1)

## End(Not run)
```

---

DMRcaller

---

*Call Differentially Methylated Regions (DMRs) between two samples*


---

**Description**

Supports both Bisulfite Sequencing (Bismark CX reports) and Oxford Nanopore Sequencing (MM/ML tags) for per-site methylation calling. Identifies differentially methylated regions between two samples in CG and non-CG contexts.

**Details**

For bisulfite data, the input is Bismark CX report files and the output is a list of DMRs stored as a GRanges object.

- readsM — count of modified reads per site
- readsN — total same-strand coverage per site

For Nanopore data, the input is an indexed ONT BAM with calling [readONTbam](#) function in this package and the output is a GRanges augmented with metadata columns:

- ONT\_Cm — comma-delimited read-indices called “modified”
- ONT\_C — comma-delimited read-indices covering but not modified
- readsM — count of modified reads per site

- readsN — total same-strand coverage per site

The most important functions in the **DMRcaller** are:

`readBismark` reads the Bismark CX report files in a `GRanges` object.

`readBismarkPool` Reads multiple CX report files and pools them together.

`saveBismark` saves the methylation data stored in a `GRanges` object into a Bismark CX report file.

`selectCytosine` Enumerates cytosine positions in a `BSgenome` reference, optionally filtering by methylation context (CG/CHG/CHH), chromosome and genomic region.

`readONTbam` Loads an Oxford Nanopore BAM (with MM/ML tags), decodes per-C modification probabilities and counts modified vs. unmodified reads per site.

`poolMethylationDatasets` pools together multiple methylation datasets.

`poolTwoMethylationDatasets` pools together two methylation datasets.

`computeMethylationDataCoverage` Computes the coverage for the bisulfite sequencing data.

`plotMethylationDataCoverage` Plots the coverage for the bisulfite sequencing data.

`computeMethylationDataSpatialCorrelation` Computes the correlation between methylation levels as a function of the distances between the Cytosines.

`plotMethylationDataSpatialCorrelation` Plots the correlation of methylation levels for Cytosines located at a certain distance apart.

`computeMethylationProfile` Computes the low resolution profiles for the bisulfite sequencing data at certain locations.

`plotMethylationProfile` Plots the low resolution profiles for the bisulfite sequencing data at certain locations.

`plotMethylationProfileFromData` Plots the low resolution profiles for the loaded bisulfite sequencing data.

`computeDMRs` Computes the differentially methylated regions between two conditions.

`filterDMRs` Filters a list of (potential) differentially methylated regions.

`mergeDMRsIteratively` Merge DMRs iteratively.

`analyseReadsInsideRegionsForCondition` Analyse reads inside regions for condition.

`plotLocalMethylationProfile` Plots the methylation profile at one locus for the bisulfite sequencing data.

`computeOverlapProfile` Computes the distribution of a set of subregions on a large region.

`plotOverlapProfile` Plots the distribution of a set of subregions on a large region.

`getWholeChromosomes` Computes the `GRanges` objects with each chromosome as an element from the `methylationData`.

`joinReplicates` Merges two `GRanges` objects with single reads columns. It is necessary to start the analysis of DMRs with biological replicates.

`computeDMRsReplicates` Computes the differentially methylated regions between two conditions with multiple biological replicates.

`selectCytosine` Enumerates cytosine positions in a `BSgenome` reference.

`readONTbam` Loads an ONT BAM (MM/ML tags), decodes per-C modification probabilities, and counts modified vs. unmodified reads per site.

**computePMDs** Partitions the genome into PMDs via three methods ("noise\_filter", "neighbourhood", "bins").

**filterPMDs** Filters a set of PMDs by methylation level and read depth.

**mergePMDsIteratively** Merge PMDs while preserving statistical significance.

**analyseReadsInsideRegionsForConditionPMD** Counts reads in each PMD for one condition.

**computeVMDs** Computes the variance methylated domains between pre-set min and max proportion values

**filterVMRsONT** Filters VMRs with ONT-specific variance tests and CI filters

**computeCoMethylatedPositions** Computes pairwise co-methylation between Cytosine sites within regions.

**computeCoMethylatedRegions** Computes pairwise co-methylation statistics between regions.

### Author(s)

Nicolae Radu Zabet <r.zabet@qmul.ac.uk>, Jonathan Michael Foonlan Tsang <jmft2@cam.ac.uk>, Alessandro Pio Greco <apgrec@essex.ac.uk>, Young Jun Kim <qc25039@qmul.ac.uk>  
 Maintainer: Nicolae Radu Zabet <r.zabet@qmul.ac.uk>

### See Also

See vignette("rd", package = "DMRcaller") for an overview of the package.

### Examples

```
## Not run:
# load the methylation data
data(methylationDataList)
library(BSgenome.Hsapiens.UCSC.hg38)

# All cytosines in hg38:
gr_all <- selectCytosine()

# Only CpG sites on chr1 and chr2:
gr_chr1_2 <- selectCytosine(context="CG", chr=c("chr1","chr2"))

# CHH sites in a specific region on chr3:
my_region <- GRanges("chr3", IRanges(1e6, 1e6 + 1e5))
gr_region <- selectCytosine(context="CHH", chr="chr3", region=my_region)

# set the bam file directory
bam_path <- system.file("extdata", "scanBamChr1Random5.bam", package="DMRcaller")

# read ONTbam file (chromosome 1 only) in CG context with BSgenome.Hsapiens.UCSC.hg38
ONTSampleGRanges <- readONTbam(bamfile = bam_path, ref_gr = NULL, modif = "C+m?",
  prob_thresh = 0.50, genome = BSgenome.Hsapiens.UCSC.hg38,
  context = "CG", chr = "chr1", region = NULL,
  synonymous = FALSE, parallel = FALSE, BPPARAM = NULL)

# plot the low resolution profile at 5 Kb resolution
```

```

par(mar=c(4, 4, 3, 1)+0.1)
plotMethylationProfileFromData(methylationDataList[["WT"]],
                               methylationDataList[["met1-3"]],
                               conditionsNames=c("WT", "met1-3"),
                               windowSize = 5000, autoscale = TRUE,
                               context = c("CG", "CHG", "CHH"),
                               labels = LETTERS)

# compute low resolution profile in 10 Kb windows in CG context
lowResProfileWTCHG <- computeMethylationProfile(methylationDataList[["WT"]],
                                               region, windowSize = 10000, context = "CG")

lowResProfileMet13CG <- computeMethylationProfile(
  methylationDataList[["met1-3"]], region,
  windowSize = 10000, context = "CG")

lowResProfileCG <- GRangesList("WT" = lowResProfileWTCHG,
                              "met1-3" = lowResProfileMet13CG)

# compute low resolution profile in 10 Kb windows in CHG context
lowResProfileWTCHG <- computeMethylationProfile(methylationDataList[["WT"]],
                                               region, windowSize = 10000, context = "CHG")

lowResProfileMet13CHG <- computeMethylationProfile(
  methylationDataList[["met1-3"]], region,
  windowSize = 10000, context = "CHG")

lowResProfileCHG <- GRangesList("WT" = lowResProfileWTCHG,
                              "met1-3" = lowResProfileMet13CHG)

# plot the low resolution profile
par(mar=c(4, 4, 3, 1)+0.1)
par(mfrow=c(2,1))
plotMethylationProfile(lowResProfileCG, autoscale = FALSE,
  labels = LETTERS[1],
  title="CG methylation on Chromosome 3",
  col=c("#D55E00", "#E69F00"), pch = c(1,0),
  lty = c(4,1))
plotMethylationProfile(lowResProfileCHG, autoscale = FALSE,
  labels = LETTERS[2],
  title="CHG methylation on Chromosome 3",
  col=c("#0072B2", "#56B4E9"), pch = c(16,2),
  lty = c(3,2))

# plot the coverage in all three contexts
plotMethylationDataCoverage(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]],
  breaks = 1:15, regions = NULL,
  conditionsNames = c("WT", "met1-3"),
  context = c("CG", "CHG", "CHH"),
  proportion = TRUE, labels = LETTERS, col = NULL,
  pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
  contextPerRow = FALSE)

```

```
# plot the correlation of methylation levels as a function of distance
plotMethylationDataSpatialCorrelation(methylationDataList[["WT"]],
                                     distances = c(1,5,10,15), regions = NULL,
                                     conditionsNames = c("WT", "met1-3"),
                                     context = c("CG"),
                                     labels = LETTERS, col = NULL,
                                     pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
                                     contextPerRow = FALSE)

# the regions where to compute the DMRs
regions <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E6))

# compute the DMRs in CG context with noise_filter method
DMRsNoiseFilterCG <- computeDMRs(methylationDataList[["WT"]],
                                 methylationDataList[["met1-3"]], regions = regions,
                                 context = "CG", method = "noise_filter",
                                 windowSize = 100, kernelFunction = "triangular",
                                 test = "score", pValueThreshold = 0.01,
                                 minCytosinesCount = 4, minProportionDifference = 0.4,
                                 minGap = 200, minSize = 50, minReadsPerCytosine = 4,
                                 cores = 1)

# compute the DMRs in CG context with neighbourhood method
DMRsNeighbourhoodCG <- computeDMRs(methylationDataList[["WT"]],
                                   methylationDataList[["met1-3"]], regions = regions,
                                   context = "CG", method = "neighbourhood",
                                   test = "score", pValueThreshold = 0.01,
                                   minCytosinesCount = 4, minProportionDifference = 0.4,
                                   minGap = 200, minSize = 50, minReadsPerCytosine = 4,
                                   cores = 1)

# compute the DMRs in CG context with bins method
DMRsBinsCG <- computeDMRs(methylationDataList[["WT"]],
                          methylationDataList[["met1-3"]], regions = regions,
                          context = "CG", method = "bins", binSize = 100,
                          test = "score", pValueThreshold = 0.01, minCytosinesCount = 4,
                          minProportionDifference = 0.4, minGap = 200, minSize = 50,
                          minReadsPerCytosine = 4, cores = 1)

# load the gene annotation data
data(GEs)

# select the genes
genes <- GEs[which(GEs$type == "gene")]

# the regions where to compute the DMRs
genes <- genes[overlapsAny(genes, regions)]

# filter genes that are differentially methylated in the two conditions
DMRsGenesCG <- filterDMRs(methylationDataList[["WT"]],
                           methylationDataList[["met1-3"]], potentialDMRs = genes,
                           context = "CG", test = "score", pValueThreshold = 0.01,
```

```

        minCytosinesCount = 4, minProportionDifference = 0.4,
        minReadsPerCytosine = 3, cores = 1)

# merge the DMRs
DMRsNoiseFilterCGLarger <- mergeDMRsIteratively(DMRsNoiseFilterCG,
        minGap = 500, respectSigns = TRUE,
        methylationDataList[["WT"]],
        methylationDataList[["met1-3"]],
        context = "CG", minProportionDifference=0.4,
        minReadsPerCytosine = 1, pValueThreshold=0.01,
        test="score", alternative = "two.sided")

# select the genes
genes <- GEs[which(GEs$type == "gene")]

# the coordinates of the area to be plotted
chr3Reg <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(510000,530000))

# load the DMRs in CG context
data(DMRsNoiseFilterCG)

DMRsCGList <- list("noise filter"=DMRsNoiseFilterCG,
        "neighbourhood"=DMRsNeighbourhoodCG,
        "bins"=DMRsBinsCG,
        "genes"=DMRsGenesCG)

# plot the CG methylation
par(mar=c(4, 4, 3, 1)+0.1)
par(mfrow=c(1,1))
plotLocalMethylationProfile(methylationDataList[["WT"]],
        methylationDataList[["met1-3"]], chr3Reg,
        DMRsCGList, c("WT", "met1-3"), GEs,
        windowSize=100, main="CG methylation")

hotspotsHypo <- computeOverlapProfile(
        DMRsNoiseFilterCG[DMRsNoiseFilterCG$regionType == "loss"],
        region, windowSize=2000, binary=TRUE, cores=1)

hotspotsHyper <- computeOverlapProfile(
        DMRsNoiseFilterCG[DMRsNoiseFilterCG$regionType == "gain"],
        region, windowSize=2000, binary=TRUE, cores=1)

plotOverlapProfile(GRangesList("Chr3"=hotspotsHypo),
        GRangesList("Chr3"=hotspotsHyper),
        names=c("loss", "gain"), title="CG methylation")

# loading synthetic data
data("syntheticDataReplicates")

# creating condition vector
condition <- c("a", "a", "b", "b")

```

```
# computing DMRs using the neighbourhood method
DMRsReplicatesNeighbourhood <- computeDMRsReplicates(methylationData = methylationData,
                                                    condition = condition,
                                                    regions = NULL,
                                                    context = "CHH",
                                                    method = "neighbourhood",
                                                    test = "betareg",
                                                    pseudocountM = 1,
                                                    pseudocountN = 2,
                                                    pValueThreshold = 0.01,
                                                    minCytosinesCount = 4,
                                                    minProportionDifference = 0.4,
                                                    minGap = 200,
                                                    minSize = 50,
                                                    minReadsPerCytosine = 4,
                                                    cores = 1)

# load the ONT methylation data
data(ontSampleGRangesList)

# the regions where to compute the PMDs
chr1_ranges <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))

# compute the PMDs in CG context with noise_filter method
PMDsNoiseFilterCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                                regions = chr1_ranges,
                                context = "CG",
                                windowSize = 100,
                                method = "noise_filter",
                                kernelFunction = "triangular",
                                lambda = 0.5,
                                minCytosinesCount = 4,
                                minMethylation = 0.4,
                                maxMethylation = 0.6,
                                minGap = 200,
                                minSize = 50,
                                minReadsPerCytosine = 4,
                                cores = 1,
                                parallel = FALSE)

# compute the PMDs in CG context with neighbourhood method
PMDsNeighbourhoodCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                                   regions = chr1_ranges,
                                   context = "CG",
                                   method = "neighbourhood"
                                   minCytosinesCount = 4,
                                   minMethylation = 0.4,
                                   maxMethylation = 0.6,
                                   minGap = 200,
                                   minSize = 50,
                                   minReadsPerCytosine = 4,
                                   cores = 1,
                                   parallel = FALSE)
```

```

# compute the PMDs in CG context with bins method
PMDsBinsCG <- computePMDs(ontSampleGRangesList[["GM18501"]],
                          regions = chr1_ranges,
                          context = "CG",
                          method = "bins",
                          binSize = 100,
                          minCytosinesCount = 4,
                          minMethylation = 0.4,
                          maxMethylation = 0.6,
                          minGap = 200,
                          minSize = 50,
                          minReadsPerCytosine = 4,
                          cores = 1,
                          parallel = FALSE)

# load the gene annotation data
data(GEs_hg38)

# select the transcript
transcript <- GEs_hg38[which(GEs_hg38$type == "transcript")]

# the regions where to compute the PMDs
regions <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))
transcript <- transcript[overlapsAny(transcript, regions)]

# filter genes that are partially methylated in the two conditions
PMDsGenesCG <- filterPMDs(ontSampleGRangesList[["GM18501"]],
                           potentialPMDs = transcript,
                           context = "CG", minMethylation = 0.4, maxMethylation = 0.6,
                           minCytosinesCount = 4, minReadsPerCytosine = 3, cores = 1)

# load the PMDs in CG context they were computed with minGap = 200
data(PMDsNoiseFilterCG)

# merge the PMDs
PMDsNoiseFilterCGLarger <- mergePMDsIteratively(PMDsNoiseFilterCG[1:100],
                                                minGap = 500, respectSigns = TRUE,
                                                ontSampleGRangesList[["GM18501"]], context = "CG",
                                                minReadsPerCytosine = 4, minMethylation = 0.4,
                                                maxMethylation = 0.6, cores = 1)

# set genomic coordinates where to compute PMDs
chr1_ranges <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))

# compute PMDs and remove gaps smaller than 200 bp
PMDsNoiseFilterCG200 <- computePMDs(ontSampleGRangesList[["GM18501"]],
                                    regions = chr1_ranges, context = "CG", method = "noise_filter",
                                    windowSize = 100, kernelFunction = "triangular",
                                    minCytosinesCount = 1, minMethylation = 0.4,
                                    maxMethylation = 0.6, minGap = 0, minSize = 200,
                                    minReadsPerCytosine = 1, cores = 1)
PMDsNoiseFilterCG0 <- computePMDs(ontSampleGRangesList[["GM18501"]],

```

```

regions = chr1_ranges, context = "CG", method = "noise_filter",
windowSize = 100, kernelFunction = "triangular",
minCytosinesCount = 1, minMethylation = 0.4,
maxMethylation = 0.6, minGap = 0, minSize = 0,
minReadsPerCytosine = 1, cores = 1)
PMDsNoiseFilterCG0Merged200 <- mergePMDsIteratively(PMDsNoiseFilterCG0,
minGap = 200, respectSigns = TRUE,
ontSampleGRangesList[["GM18501"]], context = "CG",
minReadsPerCytosine = 4, minMethylation = 0.4,
maxMethylation = 0.6, cores = 1)

#check that all newley computed PMDs are identical
print(all(PMDsNoiseFilterCG200 == PMDsNoiseFilterCG0Merged200))

#retrive the number of reads in CG context in GM18501
PMDsNoiseFilterCGreadsCG <- analyseReadsInsideRegionsForConditionPMD(
PMDsNoiseFilterCG[1:10],
ontSampleGRangesList[["GM18501"]], context = "CG",
label = "GM18501")

# load the PMD data
data(PMDsBinsCG)

# compute the co-methylations with Fisher's exact test
coMetylationFisher <- computeCoMethylatedPositions(
ontSampleGRangesList[[1]],
regions = PMDsBinsCG,
minDistance = 150,
maxDistance = 1000,
minCoverage = 4,
pValueThreshold = 0.01,
test = "fisher",
parallel = FALSE)

# compute the co-methylations with Permutation test
coMetylationPermutation <- computeCoMethylatedPositions(
ontSampleGRangesList[[1]],
regions = PMDsBinsCG,
minDistance = 150,
maxDistance = 1000,
minCoverage = 4,
pValueThreshold = 0.01,
test = "permutation",
parallel = FALSE) # highly recommended to set as TRUE

# select the transcript
transcript <- GEs_hg38[which(GEs_hg38$type == "transcript")]

# the regions where to compute the PMDs
regions <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))
transcript <- transcript[overlapsAny(transcript, regions)]

# filter genes that are differentially methylated in the two conditions

```

```
VMRsGenesCG <- filterVMsONT(ontSampleGRangesList[["GM18501"]],
  ontSampleGRangesList[["GM18876"]], potentialVMs = transcript,
  context = "CG", pValueThreshold = 0.01,
  minCytosinesCount = 4, minProportionDifference = 0.01,
  minReadsPerCytosine = 3, ciExcludesOne = TRUE,
  varRatioFc = NULL, parallel = TRUE) # parallel recommended
```

```
## End(Not run)
```

---

DMRsNoiseFilterCG      *The DMRs between WT and met1-3 in CG context*

---

### Description

A `GRangesList` object containing the DMRs between Wild Type (WT) and met1-3 mutant (met1-3) in *Arabidopsis thaliana* (see [methylationDataList](#)). The DMRs were computed on the first 1 Mbp from Chromosome 3 with noise filter method using a triangular kernel and a windowSize of 100 bp

### Format

The `GRanges` element contain 11 metadata columns; see [computeDMRs](#)

### See Also

[filterDMRs](#), [computeDMRs](#), [analyseReadsInsideRegionsForCondition](#) and [mergeDMRsIteratively](#)

---

extractGC      *Extract GC*

---

### Description

This function extracts GC sites in the genome

### Usage

```
extractGC(methylationData, genome, contexts = c("ALL", "CG", "CHG", "CHH"))
```

### Arguments

`methylationData`      the methylation data stored as a [GRanges](#) object with four metadata columns (see [methylationDataList](#)).

`genome`                a `BSgenome` with the DNA sequence of the organism

`contexts`              the context in which the DMRs are computed ("ALL", "CG", "CHG" or "CHH").

**Value**

the a subset of methylationData consisting of all GC sites.

**Author(s)**

Ryan Merritt

**Examples**

```
## Not run:
# load the genome sequence
if(!require("BSgenome.Athaliana.TAIR.TAIR9", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Athaliana.TAIR.TAIR9")
}
library(BSgenome.Athaliana.TAIR.TAIR9)

# load the methylation data
data(methylationDataList)

methylationDataWTGpCpG <- extractGC(methylationDataList[["WT"]],
                                   BSgenome.Athaliana.TAIR.TAIR9,
                                   "CG")

## End(Not run)
```

---

filterDMRs

*Filter DMRs*

---

**Description**

This function verifies whether a set of potential DMRs (e.g. genes, transposons, CpG islands) are differentially methylated or not.

**Usage**

```
filterDMRs(
  methylationData1,
  methylationData2,
  potentialDMRs,
  context = "CG",
  test = "fisher",
  pValueThreshold = 0.01,
  minCytosinesCount = 4,
  minProportionDifference = 0.4,
```

```

    minReadsPerCytosine = 3,
    parallel = FALSE,
    BPPARAM = NULL,
    cores = NULL
  )

```

### Arguments

**methylationData1**  
the methylation data in condition 1 (see [methylationDataList](#)).

**methylationData2**  
the methylation data in condition 2 (see [methylationDataList](#)).

**potentialDMRs** a [GRanges](#) object with potential DMRs where to compute the DMRs. This can be a list of gene and/or transposable elements coordinates.

**context** the context in which the DMRs are computed ("CG", "CHG" or "CHH").

**test** the statistical test used to call DMRs ("fisher" for Fisher's exact test or "score" for Score test).

**pValueThreshold**  
DMRs with p-values (when performing the statistical test; see **test**) higher or equal than **pValueThreshold** are discarded. Note that we adjust the p-values using the Benjamini and Hochberg's method to control the false discovery rate.

**minCytosinesCount**  
DMRs with less cytosines in the specified context than **minCytosinesCount** will be discarded.

**minProportionDifference**  
DMRs where the difference in methylation proportion between the two conditions is lower than **minProportionDifference** are discarded.

**minReadsPerCytosine**  
DMRs with the average number of reads lower than **minReadsPerCytosine** are discarded.

**parallel** Logical; run in parallel if TRUE.

**BPPARAM** A [BiocParallelParam](#) object controlling parallel execution. This value will automatically set when **parallel** is TRUE, also able to set as manually.

**cores** Integer number of workers (must not exceed **BPPARAM\$workers**). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Value

a [GRanges](#) object with 11 metadata columns that contain the DMRs; see [computeDMRs](#).

### Author(s)

Nicolae Radu Zabet

### See Also

[DMRsNoiseFilterCG](#), [computeDMRs](#), [analyseReadsInsideRegionsForCondition](#) and [mergeDMRsIteratively](#)

**Examples**

```

# load the methylation data
data(methylationDataList)
# load the gene annotation data
data(GEs)

#select the genes
genes <- GEs[which(GEs$type == "gene")]

# the regions where to compute the DMRs
regions <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E5))
genes <- genes[overlapsAny(genes, regions)]

# filter genes that are differentially methylated in the two conditions
DMRsGenesCG <- filterDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], potentialDMRs = genes,
  context = "CG", test = "score", pValueThreshold = 0.01,
  minCytosinesCount = 4, minProportionDifference = 0.4,
  minReadsPerCytosine = 3, cores = 1)

```

---

filterPMDs

*Filter PMDs*


---

**Description**

This function verifies whether a set of potential PMDs (e.g. genes, transposons, CpG islands) are partially methylated or not.

**Usage**

```

filterPMDs(
  methylationData,
  potentialPMDs,
  context = "CG",
  minCytosinesCount = 4,
  minMethylation = 0.4,
  maxMethylation = 0.6,
  minReadsPerCytosine = 3,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)

```

**Arguments**

methylationData

the methylation data in condition (see [ontSampleGRangesList](#)).

potentialPMDs a [GRanges](#) object with potential PMDs where to compute the PMDs. This can be a list of gene and/or transposable elements coordinates.

context the context in which the PMDs are computed ("CG", "CHG" or "CHH").

minCytosinesCount PMDs with less cytosines in the specified context than minCytosinesCount will be discarded.

minMethylation Numeric [0,1]; minimum mean methylation proportion.

maxMethylation Numeric [0,1]; maximum mean methylation proportion.

minReadsPerCytosine PMDs with the average number of reads lower than minReadsPerCytosine are discarded.

parallel Logical; run in parallel if TRUE.

BPPARAM A [BiocParallelParam](#) object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.

cores Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

a [GRanges](#) object with 5 metadata columns that contain the PMDs; see [computePMDs](#).

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[PMDsNoiseFilterCG](#), [computePMDs](#), [analyseReadsInsideRegionsForCondition](#) and [mergePMDsIteratively](#)

**Examples**

```
# load the ONT methylation data
data(ontSampleGRangesList)
# load the gene annotation data
data(GEs_hg38)

# select the transcript
transcript <- GEs_hg38[which(GEs_hg38$type == "transcript")]

# the regions where to compute the PMDs
regions <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))
transcript <- transcript[overlapsAny(transcript, regions)]

# filter genes that are partially methylated in the two conditions
PMDsGenesCG <- filterPMDs(ontSampleGRangesList[["GM18501"]],
  potentialPMDs = transcript,
  context = "CG", minMethylation = 0.4, maxMethylation = 0.6,
  minCytosinesCount = 4, minReadsPerCytosine = 3, cores = 1)
```

---

 filterVMDs

*Filter VMDs*


---

### Description

This function verifies whether a set of potential VMDs (e.g. genes, transposons, CpG islands) are variance methylated or not.

### Usage

```
filterVMDs(
  methylationData,
  potentialVMDs,
  context = "CG",
  minCytosinesCount = 4,
  minReadsPerCytosine = 3,
  sdCutoffMethod = "per.high",
  percentage = 0.05,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

### Arguments

**methylationData** the methylation data in condition (see [ontSampleGRangesList](#)).

**potentialVMDs** a [GRanges](#) object with potential VMDs where to compute the VMDs. This can be a list of gene and/or transposable elements coordinates.

**context** the context in which the VMDs are computed ("CG", "CHG" or "CHH").

**minCytosinesCount** VMDs with less cytosines in the specified context than `minCytosinesCount` will be discarded.

**minReadsPerCytosine** VMDs with the average number of reads lower than `minReadsPerCytosine` are discarded.

**sdCutoffMethod** Character string specifying how to determine the cutoff for filtering VMDs based on their methylation variance (weighted standard deviation). Available options are:

- "per.high" Selects the top percentage of regions with the highest variance (standard deviation).
- "per.low" Selects the bottom percentage of regions with the lowest variance.
- "EDE.high" Uses the elbow point (inflection/knee) from the descendingly sorted variance values to determine a data-driven high-variance cutoff. Retains regions with SD above this elbow point.

	"EDE.low" Uses the elbow point from the ascendingly sorted variance values to define a low-variance cutoff. Retains regions with SD below this point. This allows either quantile-based filtering or automatic detection of variance thresholds based on distribution shape.
percentage	Numeric cutoff used when sdCutoffMethod is set to "per.high" or "per.low". Represents the quantile threshold: for example, percentage = 0.05 keeps the top 5% or bottom 5% of bins based on weighted standard deviation, depending on the selected method.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

a [GRanges](#) object with 9 metadata columns that contain the VMDs; see [computeVMDs](#).

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[computeVMDs](#) and [analyseReadsInsideRegionsForCondition](#)

**Examples**

```
# load the ONT methylation data
data(ontSampleGRangesList)
# load the gene annotation data
data(GEs_hg38)

# select the transcript
transcript <- GEs_hg38[which(GEs_hg38$type == "transcript")]

# the regions where to compute the VMDs
regions <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5, 2E6))
transcript <- transcript[overlapsAny(transcript, regions)]

# filter genes that are variance methylated in the two conditions
VMDsGenesCG <- filterVMDs(ontSampleGRangesList[["GM18501"]],
  potentialVMDs = transcript,
  context = "CG", sdCutoffMethod = "per.high", percentage = 0.05,
  minCytosinesCount = 4, minReadsPerCytosine = 3, cores = 1)
```

---

 filterVMRsONT

*Filter VMRs for ONT Data*


---

## Description

Filter VMRs with ONT-specific variance tests and CI filters

## Usage

```
filterVMRsONT(
  methylationData1,
  methylationData2,
  potentialVMRs,
  context = "CG",
  pValueThreshold = 0.01,
  minCytosinesCount = 4,
  minProportionDifference = 0.4,
  minReadsPerCytosine = 3,
  ciExcludesOne = TRUE,
  varRatioFc = NULL,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

## Arguments

**methylationData1** A GRanges of methylation calls for condition 1 (see [ontSampleGRangesList](#)).

**methylationData2** A GRanges of methylation calls for condition 2.

**potentialVMRs** A GRanges of candidate VMR regions (genes, TEs, CpG islands, etc.).

**context** Character string specifying cytosine context ("CG", "CHG", or "CHH").

**pValueThreshold** Numeric p-value threshold ( $0 < \text{value} < 1$ ) for both Wilcoxon and F-tests after FDR adjustment.

**minCytosinesCount** Integer minimum number of cytosines per region.

**minProportionDifference** Numeric minimum methylation difference between conditions ( $0 < \text{value} < 1$ ).

**minReadsPerCytosine** Integer minimum average coverage per cytosine.

**ciExcludesOne** Logical; if TRUE, filter out regions whose F-test 95% confidence interval spans 1 (i.e., no significant variance change).

<code>varRatioFc</code>	Optional; numeric fold-change cutoff on variance ratio (e.g., 2 for twofold variance difference). Regions with variance ratio outside $[1/\text{varRatioFc}, \text{varRatioFc}]$ are kept when set.
<code>parallel</code>	Logical; run in parallel if TRUE.
<code>BPPARAM</code>	A <code>BiocParallelParam</code> object controlling parallel execution. This value will automatically set when <code>parallel</code> is TRUE, also able to set as manually.
<code>cores</code>	Integer number of workers (must not exceed <code>BPPARAM\$workers</code> ). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Details

This function verifies whether a set of potential VMRs (e.g., genes, transposons, CpG islands) are differentially methylated or not in ONT data, adding per-read Wilcoxon and F-tests on per-site proportions, confidence interval filtering, and optional variance-fold change cutoffs.

For each potential VMR, per-site methylation proportions are aggregated per read, then a two-sample Wilcoxon rank-sum test compares means (`wilcox_pvalue`), and an F-test compares variances (`f_pvalue`). You may further filter by requiring the 95 apply a fold-change cutoff on the variance ratio (`varRatioFc`).

### Value

A `GRanges` with the same ranges as `regions`, plus these metadata:

**sumReadsM1** total methylated reads in condition 1

**sumReadsN1** total reads in condition 1

**proportion1** methylation proportion (`sumReadsM1/sumReadsN1`)

**variance1** variance of per-read methylation proportions in condition 1

**sumReadsM2** total methylated reads in condition 2

**sumReadsN2** total reads in condition 2

**proportion2** methylation proportion (`sumReadsM2/sumReadsN2`)

**variance2** variance of per-read methylation proportions in condition 2

**cytosinesCount** number of cytosines observed in each region

**wilcox\_pvalue** FDR adjusted p-value from Wilcoxon rank-sum test comparing per-read proportions

**f\_pvalue** FDR adjusted p-value from F-test comparing variances of per-read proportions

**var\_ratio** Ratio of variances (`variance1 / variance2`)

**wilcox\_result** Full `htest` object returned by `wilcox.test`

**F\_test\_result** Full `htest` object returned by `var.test`

**direction** a number indicating whether the region lost (-1) or gain (+1) methylation in condition 2 compared to condition 1.

**regionType** a string indicating whether the region lost ("loss") or gained ("gain") methylation in condition 2 compared to condition 1.

**is\_DMR** logical; TRUE if region passed the `wilcox.test`

**is\_VMR** logical; TRUE if region passed the `var.test`

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[readONTbam](#), [computePMDs](#), [computeCoMethylatedPositions](#), [ontSampleGRangesList](#), [GEs\\_hg38](#)

**Examples**

```
## Not run:
# load the ONT methylation data
data(ontSampleGRangesList)
# load the gene annotation data
data(GEs_hg38)

# select the transcript
transcript <- GEs_hg38[which(GEs_hg38$type == "transcript")]

# the regions where to compute the PMDs
regions <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5, 2E6))
transcript <- transcript[overlapsAny(transcript, regions)]

# filter genes that are differentially methylated in the two conditions
VMRsGenesCG <- filterVMRsONT(ontSampleGRangesList[["GM18501"]],
  ontSampleGRangesList[["GM18876"]], potentialVMRs = transcript,
  context = "CG", pValueThreshold = 0.01,
  minCytosinesCount = 4, minProportionDifference = 0.01,
  minReadsPerCytosine = 3, ciExcludesOne = TRUE,
  varRatioFc = NULL, parallel = TRUE) # parallel recommended

## End(Not run)
```

---

GEs

*The genetic elements data*

---

**Description**

A GRanges object containing the annotation of the Arabidopsis thaliana

**Format**

A GRanges object

**Source**

The object was created by calling `import.gff3` function from `rtracklayer` package for [ftp://ftp.arabidopsis.org/Maps/gbrowse\\_data/TAIR10/TAIR10\\_GFF3\\_genes\\_transposons.gff](ftp://ftp.arabidopsis.org/Maps/gbrowse_data/TAIR10/TAIR10_GFF3_genes_transposons.gff)

---

GEs\_hg38

*The genetic elements data of GRCh38 Genome Reference*

---

### Description

A GRanges object containing the annotation of the Homo sapiens (hg38)

### Format

A GRanges object

### Source

The object was created by loading the gtf file from <https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/genes/hg38.refGene.gtf.gz>. and concatenated by range in 1.5 ~ 2 Mbp from Chromosome 1

---

getWholeChromosomes

*Get whole chromosomes from methylation data*

---

### Description

Returns a [GRanges](#) object spanning from the first cytosine until the last one on each chromosome

### Usage

```
getWholeChromosomes(methylationData)
```

### Arguments

methylationData

the methylation data stored as a [GRanges](#) object with four metadata columns (see [methylationDataList](#)).

### Value

a [GRanges](#) object with all chromosomes.

### Author(s)

Nicolae Radu Zabet

## Examples

```
# load the methylation data
data(methylationDataList)

# get all chromosomes
chromosomes <- getWholeChromosomes(methylationDataList[["WT"]])
```

---

joinReplicates	<i>Joins together two GRanges objects in a single containing all the replicates</i>
----------------	---

---

## Description

This function joins together data that come from biological replicates to perform analysis

## Usage

```
joinReplicates(methylationData1, methylationData2, usecomplete = FALSE)
```

## Arguments

methylationData1	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
methylationData2	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
usecomplete	Boolean, determine wheter, when the two dataset differ for number of cytosines, if the smaller dataset should be added with zero reads to match the bigger dataset.

## Value

returns a [GRanges](#) object containing multiple metadata columns with the reads from each object passed as parameter

## Author(s)

Alessandro Pio Greco and Nicolae Radu Zabet

**Examples**

```
# load the methylation data
data(methylationDataList)

# Joins the wildtype and the mutant in a single object
joined_data <- joinReplicates(methylationDataList[["WT"]],
                             methylationDataList[["met1-3"]], FALSE)
```

---

mergeDMRsIteratively *Merge DMRs iteratively*

---

**Description**

This function takes a list of DMRs and attempts to merge DMRs while keeping the new DMRs statistically significant.

**Usage**

```
mergeDMRsIteratively(
  DMRs,
  minGap,
  respectSigns = TRUE,
  methylationData1,
  methylationData2,
  context = "CG",
  minProportionDifference = 0.4,
  minReadsPerCytosine = 4,
  pValueThreshold = 0.01,
  test = "fisher",
  alternative = "two.sided",
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

**Arguments**

DMRs	the list of DMRs as a <a href="#">GRanges</a> object; e.g. see <a href="#">computeDMRs</a>
minGap	DMRs separated by a gap of at least minGap are not merged.
respectSigns	logical value indicating whether to respect the sign when joining DMRs.
methylationData1	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
methylationData2	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ).

context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
minProportionDifference	two adjacent DMRs are merged only if the difference in methylation proportion of the new DMR is higher than minProportionDifference.
minReadsPerCytosine	two adjacent DMRs are merged only if the number of reads per cytosine of the new DMR is higher than minReadsPerCytosine.
pValueThreshold	two adjacent DMRs are merged only if the p-value of the new DMR (see test below) is lower than pValueThreshold. Note that we adjust the p-values using the Benjamini and Hochberg's method to control the false discovery rate.
test	the statistical test used to call DMRs ("fisher" for Fisher's exact test or "score" for Score test).
alternative	indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less".
parallel	Logical; run in parallel if TRUE.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

the reduced list of DMRs as a [GRanges](#) object; e.g. see [computeDMRs](#)

**Author(s)**

Nicolae Radu Zabet

**See Also**

[filterDMRs](#), [computeDMRs](#), [analyseReadsInsideRegionsForCondition](#) and [DMRsNoiseFilterCG](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

#load the DMRs in CG context they were computed with minGap = 200
data(DMRsNoiseFilterCG)

#merge the DMRs
DMRsNoiseFilterCGLarger <- mergeDMRsIteratively(DMRsNoiseFilterCG[1:100],
  minGap = 500, respectSigns = TRUE,
  methylationDataList[["WT"]],
  methylationDataList[["met1-3"]],
```

```

context = "CG", minProportionDifference=0.4,
minReadsPerCytosine = 1, pValueThreshold=0.01,
test="score",alternative = "two.sided")

## Not run:
#set genomic coordinates where to compute DMRs
regions <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E5))

# compute DMRs and remove gaps smaller than 200 bp
DMRsNoiseFilterCG200 <- computeDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], regions = regions,
  context = "CG", method = "noise_filter",
  windowSize = 100, kernelFunction = "triangular",
  test = "score", pValueThreshold = 0.01,
  minCytosinesCount = 1, minProportionDifference = 0.4,
  minGap = 200, minSize = 0, minReadsPerCytosine = 1,
  cores = 1)

DMRsNoiseFilterCG0 <- computeDMRs(methylationDataList[["WT"]],
  methylationDataList[["met1-3"]], regions = regions,
  context = "CG", method = "noise_filter",
  windowSize = 100, kernelFunction = "triangular",
  test = "score", pValueThreshold = 0.01,
  minCytosinesCount = 1, minProportionDifference = 0.4,
  minGap = 0, minSize = 0, minReadsPerCytosine = 1,
  cores = 1)
DMRsNoiseFilterCG0Merged200 <- mergeDMRsIteratively(DMRsNoiseFilterCG0,
  minGap = 200, respectSigns = TRUE,
  methylationDataList[["WT"]],
  methylationDataList[["met1-3"]],
  context = "CG", minProportionDifference=0.4,
  minReadsPerCytosine = 1, pValueThreshold=0.01,
  test="score",alternative = "two.sided")

#check that all newly computed DMRs are identical
print(all(DMRsNoiseFilterCG200 == DMRsNoiseFilterCG0Merged200))

## End(Not run)

```

---

mergePMDsIteratively *Merge PMDs iteratively*

---

### Description

This function takes a list of PMDs and attempts to merge PMDs while keeping the new PMDs statistically significant.

**Usage**

```
mergePMDsIteratively(
  PMDs,
  minGap = 200,
  respectSigns = TRUE,
  methylationData,
  context = "CG",
  minReadsPerCytosine = 4,
  minMethylation = 0.4,
  maxMethylation = 0.6,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

**Arguments**

PMDs	the list of PMDs as a <a href="#">GRanges</a> object; e.g. see <a href="#">computePMDs</a>
minGap	PMDs separated by a gap of at least minGap are not merged.
respectSigns	logical value indicating whether to respect the sign when joining PMDs.
methylationData	the methylation data in <a href="#">GRanges</a> (see <a href="#">ontSampleGRangesList</a> ).
context	the context in which the PMDs are computed ("CG", "CHG" or "CHH").
minReadsPerCytosine	two adjacent PMDs are merged only if the number of reads per cytosine of the new DMR is higher than minReadsPerCytosine.
minMethylation	Numeric [0,1]; minimum mean methylation proportion.
maxMethylation	Numeric [0,1]; maximum mean methylation proportion.
parallel	Logical; run in parallel if TRUE.
BPPARAM	A <a href="#">BiocParallelParam</a> object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed <code>BPPARAM\$workers</code> ). This value will automatically set as the maximum number of system workers, also able to set as manually.

**Value**

the reduced list of PMDs as a [GRanges](#) object; e.g. see [computePMDs](#)

**Author(s)**

Nicolae Radu Zabet and Young Jun Kim

**See Also**

[filterPMDs](#), [computePMDs](#), [analyseReadsInsideRegionsForCondition](#) and [PMDsNoiseFilterCG](#)

**Examples**

```

# load the ONT methylation data
data(ontSampleGRangesList)

# load the PMDs in CG context they were computed with minGap = 200
data(PMDsNoiseFilterCG)

# merge the PMDs
PMDsNoiseFilterCGLarger <- mergePMDsIteratively(PMDsNoiseFilterCG[1:100],
  minGap = 500, respectSigns = TRUE,
  ontSampleGRangesList[["GM18501"]], context = "CG",
  minReadsPerCytosine = 4, minMethylation = 0.4,
  maxMethylation = 0.6, cores = 1)

## Not run:
# set genomic coordinates where to compute PMDs
chr1_ranges <- GRanges(seqnames = Rle("chr1"), ranges = IRanges(1E6+5E5,2E6))

# compute PMDs and remove gaps smaller than 200 bp
PMDsNoiseFilterCG200 <- computePMDs(ontSampleGRangesList[["GM18501"]],
  regions = chr1_ranges, context = "CG", method = "noise_filter",
  windowSize = 100, kernelFunction = "triangular",
  minCytosinesCount = 1, minMethylation = 0.4,
  maxMethylation = 0.6, minGap = 0, minSize = 200,
  minReadsPerCytosine = 1, cores = 1)
PMDsNoiseFilterCG0 <- computePMDs(ontSampleGRangesList[["GM18501"]],
  regions = chr1_ranges, context = "CG", method = "noise_filter",
  windowSize = 100, kernelFunction = "triangular",
  minCytosinesCount = 1, minMethylation = 0.4,
  maxMethylation = 0.6, minGap = 0, minSize = 0,
  minReadsPerCytosine = 1, cores = 1)
PMDsNoiseFilterCG0Merged200 <- mergePMDsIteratively(PMDsNoiseFilterCG0,
  minGap = 200, respectSigns = TRUE,
  ontSampleGRangesList[["GM18501"]], context = "CG",
  minReadsPerCytosine = 4, minMethylation = 0.4,
  maxMethylation = 0.6, cores = 1)

#check that all newly computed PMDs are identical
print(all(PMDsNoiseFilterCG200 == PMDsNoiseFilterCG0Merged200))

## End(Not run)

```

**Description**

A GRangesList object containing the methylation data at each cytosine location in the genome in Wild Type (WT) and met1-3 mutant (met1-3) in *Arabidopsis thaliana*. The data only contains the first 1 Mbp from Chromosome 3.

**Format**

The GRanges elements contain four metadata columns

**context** the context in which the DMRs are computed ("CG", "CHG" or "CHH").

**readsM** the number of methylated reads.

**readsN** the total number of reads.

**trinucleotide\_context** the specific context of the cytosine (H is replaced by the actual nucleotide).

**Source**

Each element was created by by calling `readBismark` function on the CX report files generated by Bismark <http://www.bioinformatics.babraham.ac.uk/projects/bismark/> for <http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM980986> dataset in the case of Wild Type (WT) and <http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM981032> dataset in the case of met1-3 mutant (met1-3).

---

ontSampleGRangesList *The ONT methylation data list*

---

**Description**

A GRangesList object containing the methylation data at each cytosine location in the genome in GM18501 and GM18876 B-Lymphocyte cell lines in *Homo sapiens*. The data only contains the 1.5 ~ 2 Mbp from Chromosome 1.

**Format**

The GRanges elements contain six metadata columns

**context** the context in which the DMRs are computed "CG".

**trinucleotide\_context** the specific context of the cytosine (H is replaced by the actual nucleotide).

**ONT\_Cm** comma-delimited read-indices called modified

**ONT\_C** comma-delimited read-indices covering but unmodified

**readsM** the number of methylated reads.

**readsN** the total number of reads.

## Source

Each element was created by calling bam files with `readONTbam` function which in `DMRcaller` package. The sample pod5 files were from the nanopore dataset from 1000 genome project <https://pmc.ncbi.nlm.nih.gov/articles/PMC10942501/>. [https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5\\_data/GM18501\\_R9/](https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5_data/GM18501_R9/) .pod5 files in the case of GM18501 and [https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5\\_data/GM18876\\_R9/](https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5_data/GM18876_R9/) .pod5 files in the case of GM18876 cell line. For base-calling and alignment, run dorado, ver.0.9.6 <https://github.com/nanoporetech/dorado?tab=readme-ov-file#dna-models> to generate the bam files with `dna_r10.4.1_e8.2_400bps_hac@v5.2.0` as basecalling model.

---

ont\_gr\_GM18870\_chr1\_PMD\_bins\_1k

*Partially Methylated Domains example*

---

## Description

Partially methylated domains called on chr1 in GM18870 cells called in 1Kb bins with `computePMDs` function.

## Format

The `GRanges` elements contain seven metadata columns:

**context** the context in which the PMDs was computed ("CG", "CHG" or "CHH").

**sumReadsM** the number of methylated reads.

**sumReadsN** the total number of reads.

**proportion** the proportion methylated reads filtered between `minMethylation` and `maxMethylation`.

**cytosinesCount** the number of cytosines in the PMDs.

## Source

data from <https://genome.cshlp.org/content/34/11/2061>.

---

ont\_gr\_GM18870\_chr1\_sorted\_bins\_1k

*The ONT methylation data example*

---

## Description

A `GRanges` object containing cytosine sites, annotated with per-site ONT methylation calls

**Format**

The GRanges elements contain four additional metadata columns:

**ONT\_Cm** comma-delimited read-indices called modified

**ONT\_C** comma-delimited read-indices covering but unmodified

**readsM** integer count of modified reads per site

**readsN** integer count of same-strand reads covering each site

**Source**

data from <https://genome.cshlp.org/content/34/11/2061>.

---

plotLocalMethylationProfile

*Plot local methylation profile*

---

**Description**

This function plots the methylation profile at one locus for the bisulfite sequencing data. The points on the graph represent methylation proportion of individual cytosines, their colour which sample they belong to and the intensity of the the colour how many reads that particular cytosine had. This means that darker colors indicate stronger evidence that the corresponding cytosine has the corresponding methylation proportion, while lighter colors indicate a weaker evidence. The solid lines represent the smoothed profiles and the intensity of the line the coverage at the corresponding position (darker colors indicate more reads while lighter ones less reads). The boxes on top represent the DMRs, where a filled box will represent a DMR which gained methylation while a box with a pattern represent a DMR that lost methylation. The DMRs need to have a metadatafield "regionType" which can be either "gain" (where there is more methylation in condition 2 compared to condition 1) or "loss" (where there is less methylation in condition 2 compared to condition 1). In case this metadatafield is missing all DMRs are drawn using a filled box. Finally, we also allow annotation of the DNA sequence. We represent by a black boxes all the exons, which are joined by a horizontal black line, thus, marking the full body of the gene. With grey boxes we mark the transposable elements. Both for genes and transposable elements we plot them over a mid line if they are on the positive strand and under the mid line if they are on the negative strand.

**Usage**

```
plotLocalMethylationProfile(  
  methylationData1,  
  methylationData2,  
  region,  
  DMRs = NULL,  
  conditionsNames = NULL,  
  gff = NULL,  
  windowSize = 150,  
  context = "CG",
```

```

    labels = NULL,
    col = NULL,
    main = "",
    plotMeanLines = TRUE,
    plotPoints = TRUE
  )

```

### Arguments

<code>methylationData1</code>	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
<code>methylationData2</code>	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ).
<code>region</code>	a <a href="#">GRanges</a> object with the region where to plot the high resolution profile.
<code>DMRs</code>	a <a href="#">GRangesList</a> object or a list with the list of DMRs (see <a href="#">computedDMRs</a> or <a href="#">filterDMRs</a> ).
<code>conditionsNames</code>	the names of the two conditions. This will be used to plot the legend.
<code>gff</code>	a <a href="#">GRanges</a> object with all elements usually imported from a GFF3 file. The gff file needs to have an metafield "type". Only the elements of type "gene", "exon" and "transposable_element" are plotted. Genes are represented as horizontal black lines, exons as a black rectangle and transposable elements as a grey rectangle. The elements are plotted on the corresponding strand (+ or -).
<code>windowSize</code>	the size of the triangle base used to smooth the average methylation profile.
<code>context</code>	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
<code>labels</code>	a vector of character used to add a subfigure characters to the plot. If NULL nothing is added.
<code>col</code>	a character vector with the colors. It needs to contain a minimum of 4 length(DMRs) colors. If not or if NULL, the default colors will be used.
<code>main</code>	a character with the title of the plot
<code>plotMeanLines</code>	a logical value indicating whether to plot the mean lines or not.
<code>plotPoints</code>	a logical value indicating whether to plot the points or not.

### Value

Invisibly returns NULL

### Author(s)

Nicolae Radu Zabet

### Examples

```

# load the methylation data
data(methylationDataList)
# load the gene annotation data

```

```

data(GEs)

#select the genes
genes <- GEs[which(GEs$type == "gene")]

# the coordinates of the area to be plotted
chr3Reg <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(510000,530000))

# load the DMRs in CG context
data(DMRsNoiseFilterCG)

DMRsCGlist <- list("noise filter"=DMRsNoiseFilterCG)

# plot the CG methylation
par(mar=c(4, 4, 3, 1)+0.1)
par(mfrow=c(1,1))
plotLocalMethylationProfile(methylationDataList[["WT"]],
                           methylationDataList[["met1-3"]], chr3Reg,
                           DMRsCGlist, c("WT", "met1-3"), GEs,
                           windowSize=100, main="CG methylation")

```

---

```

plotMethylationDataCoverage
      Plot methylation data coverage

```

---

### Description

This function plots the coverage for the bisulfite sequencing data.

### Usage

```

plotMethylationDataCoverage(
  methylationData1,
  methylationData2 = NULL,
  breaks,
  regions = NULL,
  conditionsNames = NULL,
  context = "CG",
  proportion = TRUE,
  labels = NULL,
  col = NULL,
  pch = c(1, 0, 16, 2, 15, 17),
  lty = c(4, 1, 3, 2, 6, 5),
  contextPerRow = FALSE
)

```

**Arguments**

methylationData1	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
methylationData2	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ). This is optional.
breaks	a numeric vector specifying the different values for the thresholds when computing the coverage.
regions	a <a href="#">GRanges</a> object with the regions where to compute the coverage. If NULL, the coverage is computed genome-wide.
conditionsNames	a vector of character with the names of the conditions for methylationData1 and methylationData2.
context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
proportion	a logical value indicating whether proportion or counts will be plotted.
labels	a vector of character used to add a subfigure character to the plot. If NULL nothing is added.
col	a character vector with the colors. It needs to contain a minimum of 2 colors per condition. If not or if NULL, the default colors will be used.
pch	the R symbols used to plot the data. It needs to contain a minimum of 2 symbols per condition. If not or if NULL, the default symbols will be used.
lty	the line types used to plot the data. It needs to contain a minimum of 2 line types per condition. If not or if NULL, the default line types will be used.
contextPerRow	a logical value indicating if the each row represents an individual context. If FALSE, each column will represent an individual context.

**Details**

This function plots the proportion of cytosines in a specific context that have at least a certain number of reads (x-axis)

**Value**

Invisibly returns NULL

**Author(s)**

Nicolae Radu Zabet

**See Also**

[computeMethylationDataCoverage](#), [methylationDataList](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

# plot the coverage in CG context
par(mar=c(4, 4, 3, 1)+0.1)
plotMethylationDataCoverage(methylationDataList[["WT"]],
                             methylationDataList[["met1-3"]],
                             breaks = c(1,5,10,15), regions = NULL,
                             conditionsNames = c("WT", "met1-3"),
                             context = c("CG"), proportion = TRUE,
                             labels = LETTERS, col = NULL,
                             pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
                             contextPerRow = FALSE)

## Not run:
# plot the coverage in all three contexts
plotMethylationDataCoverage(methylationDataList[["WT"]],
                             methylationDataList[["met1-3"]],
                             breaks = 1:15, regions = NULL,
                             conditionsNames = c("WT", "met1-3"),
                             context = c("CG", "CHG", "CHH"),
                             proportion = TRUE, labels = LETTERS, col = NULL,
                             pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
                             contextPerRow = FALSE)

## End(Not run)
```

---

plotMethylationDataSpatialCorrelation

*Plot methylation data spatial correlation*

---

**Description**

This function plots the correlation of methylation levels for Cytosines located at a certain distance apart.

**Usage**

```
plotMethylationDataSpatialCorrelation(
  methylationData1,
  methylationData2 = NULL,
  distances,
  regions = NULL,
  conditionsNames = NULL,
  context = "CG",
  labels = NULL,
  col = NULL,
```

```

pch = c(1, 0, 16, 2, 15, 17),
lty = c(4, 1, 3, 2, 6, 5),
contextPerRow = FALSE,
log = ""
)

```

### Arguments

methylationData1	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
methylationData2	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ). This is optional.
distances	a numeric vector specifying the different values for the distances when computing the correlation.
regions	a <a href="#">GRanges</a> object with the regions where to compute the correlation. If NULL, the coverage is computed genome-wide.
conditionsNames	a vector of character with the names of the conditions for methylationData1 and methylationData2.
context	the context in which the DMRs are computed ("CG", "CHG" or "CHH").
labels	a vector of character used to add a subfigure character to the plot. If NULL nothing is added.
col	a character vector with the colors. It needs to contain a minimum of 2 colors per condition. If not or if NULL, the default colors will be used.
pch	the R symbols used to plot the data. It needs to contain a minimum of 2 symbols per condition. If not or if NULL, the default symbols will be used.
lty	the line types used to plot the data. It needs to contain a minimum of 2 line types per condition. If not or if NULL, the default line types will be used.
contextPerRow	a logical value indicating if the each row represents an individual context. If FALSE, each column will represent an individual context.
log	a character indicating if any of the axes will be displayed on log scale. This argument will be passed to <a href="#">plot</a> function.

### Details

This function plots the proportion of cytosines in a specific context that have at least a certain number of reads (x-axis)

### Value

Invisibly returns NULL

### Author(s)

Nicolae Radu Zabet

**See Also**

[computeMethylationDataSpatialCorrelation](#), [methylationDataList](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

# plot the spatial correlation in CG context
par(mar=c(4, 4, 3, 1)+0.1)
plotMethylationDataSpatialCorrelation(methylationDataList[["WT"]],
                                     methylationDataList[["met1-3"]],
                                     distances = c(1,5,10,15), regions = NULL,
                                     conditionsNames = c("WT", "met1-3"),
                                     context = c("CG"),
                                     labels = LETTERS, col = NULL,
                                     pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
                                     contextPerRow = FALSE)

## Not run:
# plot the spatial correlation in all three contexts
plotMethylationDataSpatialCorrelation(methylationDataList[["WT"]],
                                     methylationDataList[["met1-3"]],
                                     distances = c(1,5,10,15,20,50,100,150,200,500,1000),
                                     regions = NULL, conditionsNames = c("WT", "met1-3"),
                                     context = c("CG", "CHG", "CHH"),
                                     labels = LETTERS, col = NULL,
                                     pch = c(1,0,16,2,15,17), lty = c(4,1,3,2,6,5),
                                     contextPerRow = FALSE, log="x")

## End(Not run)
```

---

plotMethylationProfile

*Plot Methylation Profile*

---

**Description**

This function plots the low resolution profiles for the bisulfite sequencing data.

**Usage**

```
plotMethylationProfile(
  methylationProfiles,
  autoscale = FALSE,
  labels = NULL,
  title = "",
  col = NULL,
```

```

    pch = c(1, 0, 16, 2, 15, 17),
    lty = c(4, 1, 3, 2, 6, 5)
)

```

### Arguments

methylationProfiles	a GRangesList object. Each <a href="#">GRanges</a> object in the list is generated by calling the function <a href="#">computeMethylationProfile</a> .
autoscale	a logical value indicating whether the values are autoscaled for each context or not.
labels	a vector of character used to add a subfigure characters to the plot. If NULL nothing is added.
title	the plot title.
col	a character vector with the colours. It needs to contain a minimum of 2 colours per context. If not or if NULL, the default colours will be used.
pch	the R symbols used to plot the data.
lty	the line types used to plot the data.

### Value

Invisibly returns NULL

### Author(s)

Nicolae Radu Zabet

### See Also

[plotMethylationProfileFromData](#), [computeMethylationProfile](#) and [methylationDataList](#)

### Examples

```

# load the methylation data
data(methylationDataList)

# the region where to compute the profile
region <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E6))

# compute low resolution profile in 20 Kb windows
lowResProfileWTCG <- computeMethylationProfile(methylationDataList[["WT"]],
      region, windowSize = 20000, context = "CG")

lowResProfilsCG <- GRangesList("WT" = lowResProfileWTCG)

#plot the low resolution profile
par(mar=c(4, 4, 3, 1)+0.1)
par(mfrow=c(1,1))

```

```

plotMethylationProfile(lowResProfilsCG, autoscale = FALSE,
                      title="CG methylation on Chromosome 3",
                      col=c("#D55E00","#E69F00"), pch = c(1,0),
                      lty = c(4,1))

## Not run:
# compute low resolution profile in 10 Kb windows in CG context
lowResProfileWT CG <- computeMethylationProfile(methylationDataList[["WT"]],
                                             region, windowSize = 10000, context = "CG")

lowResProfileMet13CG <- computeMethylationProfile(
  methylationDataList[["met1-3"]], region,
  windowSize = 10000, context = "CG")

lowResProfileCG <- GRangesList("WT" = lowResProfileWT CG,
                              "met1-3" = lowResProfileMet13CG)

# compute low resolution profile in 10 Kb windows in CHG context
lowResProfileWT CHG <- computeMethylationProfile(methylationDataList[["WT"]],
                                             region, windowSize = 10000, context = "CHG")

lowResProfileMet13CHG <- computeMethylationProfile(
  methylationDataList[["met1-3"]], region,
  windowSize = 10000, context = "CHG")

lowResProfileCHG <- GRangesList("WT" = lowResProfileWT CHG,
                              "met1-3" = lowResProfileMet13CHG)

# plot the low resolution profile
par(mar=c(4, 4, 3, 1)+0.1)
par(mfrow=c(2,1))
plotMethylationProfile(lowResProfileCG, autoscale = FALSE,
                      labels = LETTERS[1],
                      title="CG methylation on Chromosome 3",
                      col=c("#D55E00","#E69F00"), pch = c(1,0),
                      lty = c(4,1))
plotMethylationProfile(lowResProfileCHG, autoscale = FALSE,
                      labels = LETTERS[2],
                      title="CHG methylation on Chromosome 3",
                      col=c("#0072B2", "#56B4E9"), pch = c(16,2),
                      lty = c(3,2))

## End(Not run)

```

---

plotMethylationProfileFromData

*Plot methylation profile from data*


---

**Description**

This function plots the low resolution profiles for all bisulfite sequencing data.

**Usage**

```
plotMethylationProfileFromData(
  methylationData1,
  methylationData2 = NULL,
  regions = NULL,
  conditionsNames = NULL,
  context = "CG",
  windowSize = NULL,
  autoscale = FALSE,
  labels = NULL,
  col = NULL,
  pch = c(1, 0, 16, 2, 15, 17),
  lty = c(4, 1, 3, 2, 6, 5),
  contextPerRow = TRUE
)
```

**Arguments**

methylationData1	the methylation data in condition 1 (see <a href="#">methylationDataList</a> ).
methylationData2	the methylation data in condition 2 (see <a href="#">methylationDataList</a> ). This is optional.
regions	a <a href="#">GRanges</a> object with the regions where to plot the profiles.
conditionsNames	the names of the two conditions. This will be used to plot the legend.
context	a vector with all contexts in which the DMRs are computed ("CG", "CHG" or "CHH").
windowSize	a numeric value indicating the size of the window in which methylation is averaged.
autoscale	a logical value indicating whether the values are autoscaled for each context or not.
labels	a vector of character used to add a subfigure character to the plot. If NULL nothing is added.
col	a character vector with the colours. It needs to contain a minimum of 2 colours per condition. If not or if NULL, the default colours will be used.
pch	the R symbols used to plot the data. It needs to contain a minimum of 2 symbols per condition. If not or if NULL, the default symbols will be used.
lty	the line types used to plot the data. It needs to contain a minimum of 2 line types per condition. If not or if NULL, the default line types will be used.
contextPerRow	a logical value indicating if each row represents an individual context. If FALSE, each column will represent an individual context.

**Value**

Invisibly returns NULL

**Author(s)**

Nicolae Radu Zabet

**See Also**

[plotMethylationProfile](#), [computeMethylationProfile](#) and [methylationDataList](#)

**Examples**

```
# load the methylation data
data(methylationDataList)

#plot the low resolution profile at 10 Kb resolution
par(mar=c(4, 4, 3, 1)+0.1)
plotMethylationProfileFromData(methylationDataList[["WT"]],
                               methylationDataList[["met1-3"]],
                               conditionsNames=c("WT", "met1-3"),
                               windowSize = 20000, autoscale = TRUE,
                               context = c("CHG"))

## Not run:
#plot the low resolution profile at 5 Kb resolution
par(mar=c(4, 4, 3, 1)+0.1)
plotMethylationProfileFromData(methylationDataList[["WT"]],
                               methylationDataList[["met1-3"]],
                               conditionsNames=c("WT", "met1-3"),
                               windowSize = 5000, autoscale = TRUE,
                               context = c("CG", "CHG", "CHH"),
                               labels = LETTERS)

## End(Not run)
```

---

plotOverlapProfile      *Plot overlap profile*

---

**Description**

This function plots the distribution of a set of subregions on a large region.

**Usage**

```
plotOverlapProfile(  
  overlapsProfiles1,  
  overlapsProfiles2 = NULL,  
  names = NULL,  
  labels = NULL,  
  col = NULL,  
  title = "",  
  logscale = FALSE,  
  maxValue = NULL  
)
```

**Arguments**

**overlapsProfiles1** a [GRanges](#) object with the overlaps profile; see [computeOverlapProfile](#).

**overlapsProfiles2** a [GRanges](#) object with the overlaps profile; see [computeOverlapProfile](#). This is optional. For example, one can use `overlapsProfiles1` to display hypomethylated regions and `overlapsProfiles2` the hypermethylated regions.

**names** a vector of character to add labels for the two overlapsProfiles. This is an optional parameter.

**labels** a vector of character used to add a subfigure character to the plot. If NULL nothing is added.

**col** a character vector with the colours. It needs to contain 2 colours. If not or if NULL, the default colours will be used.

**title** the title of the plot.

**logscale** a logical value indicating if the colours are on logscale or not.

**maxValue** a maximum value in a region. Used for the colour scheme.

**Value**

Invisibly returns NULL.

**Author(s)**

Nicolae Radu Zabet

**See Also**

[computeOverlapProfile](#), [filterDMRs](#), [computeDMRs](#) and [mergeDMRsIteratively](#)

**Examples**

```
# load the methylation data  
data(methylationDataList)
```

```

# load the DMRs in CG context
data(DMRsNoiseFilterCG)

# the coordinates of the area to be plotted
largeRegion <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E5))

# compute overlaps distribution
hotspotsHypo <- computeOverlapProfile(DMRsNoiseFilterCG, largeRegion,
                                     windowSize = 10000, binary = FALSE)

plotOverlapProfile(GRangesList("Chr3"=hotspotsHypo),
                  names = c("hypomethylated"), title = "CG methylation")

## Not run:

largeRegion <- GRanges(seqnames = Rle("Chr3"), ranges = IRanges(1,1E6))

hotspotsHypo <- computeOverlapProfile(
  DMRsNoiseFilterCG[DMRsNoiseFilterCG$regionType == "loss"],
  largeRegion, windowSize=2000, binary=TRUE, cores=1)

hotspotsHyper <- computeOverlapProfile(
  DMRsNoiseFilterCG[DMRsNoiseFilterCG$regionType == "gain"],
  largeRegion, windowSize=2000, binary=TRUE, cores=1)

plotOverlapProfile(GRangesList("Chr3"=hotspotsHypo),
                  GRangesList("Chr3"=hotspotsHyper),
                  names=c("loss", "gain"), title="CG methylation")

## End(Not run)

```

---

PMDsBinsCG

*The PMDs between GM18501 and GM18876 using Bins method*


---

### Description

A GRangesList object containing the PMDs between GM18501 and GM18876 B-Lymphocyte cell lines in Homo sapiens (see [ontSampleGRangesList](#)). The PMDs were computed on the 1.5 ~ 2 Mbp from Chromosome 1 with bins method using binSize of 1 kbp

### Format

The GRanges element contain 5 metadata columns; see [computePMDs](#)

### See Also

[filterPMDs](#), [computePMDs](#), [analyseReadsInsideRegionsForConditionPMD](#) and [mergePMDsIteratively](#)

---

PMDsNoiseFilterCG      *The PMDs between GM18501 and GM18876 using Noise\_filter method*

---

### Description

A GRangesList object containing the PMDs between GM18501 and GM18876 B-Lymphocyte cell lines in Homo sapiens (see [ontSampleGRangesList](#)). The PMDs were computed on the 1.5 ~ 2 Mbp from Chromosome 1 with noise filter method using a triangular kernel and a windowSize of 100 bp

### Format

The GRanges element contain 5 metadata columns; see [computePMDs](#)

### See Also

[filterPMDs](#), [computePMDs](#), [analyseReadsInsideRegionsForConditionPMD](#) and [mergePMDsIteratively](#)

---

poolMethylationDatasets  
*Pool methylation data*

---

### Description

This function pools together multiple methylation datasets.

### Usage

```
poolMethylationDatasets(methylationDataList)
```

### Arguments

methylationDataList  
a GRangesList object where each element of the list is a GRanges object with the methylation data in the corresponding condition (see [methylationDataList](#)).

### Value

the methylation data stored as a GRanges object with four metadata columns (see [methylationDataList](#)). If the Granges are from ONT datasets, its have six metedata columns include as ONT\_Cm and ONT\_C (see [readONTbam](#)).

### Author(s)

Nicolae Radu Zabet updated by Young Jun Kim

## Examples

```
# load methylation data object
data(methylationDataList)

# pools the two datasets together
pooledMethylationData <- poolMethylationDatasets(methylationDataList)
```

---

```
poolTwoMethylationDatasets
  Pool two methylation datasets
```

---

## Description

This function pools together two methylation datasets.

## Usage

```
poolTwoMethylationDatasets(
  methylationData1,
  methylationData2,
  sample1_name = NULL,
  sample2_name = NULL
)
```

## Arguments

`methylationData1` a [GRanges](#) object with the methylation data (see [methylationDataList](#)).

`methylationData2` a [GRanges](#) object with the methylation data (see [methylationDataList](#)).

`sample1_name` the label used for sample 1.

`sample2_name` the label used for sample 2.

## Value

the methylation data stored as a [GRanges](#) object with four metadata columns (see [methylationDataList](#)). If the Granges are from ONT datasets, its have six metadata columns include as `ONT_Cm` and `ONT_C` (see [readONTbam](#)).

## Author(s)

Nicolae Radu Zabet updated by Young Jun Kim

### Examples

```
# load methylation data object
data(methylationDataList)

# save the two datasets together
pooledMethylationData <- poolTwoMethylationDatasets(methylationDataList[[1]],
  methylationDataList[[2]])
```

---

readBismark

*Read Bismark*

---

### Description

This function takes as input a CX report file produced by Bismark and returns a [GRanges](#) object with four metadata columns. The file represents the bisulfite sequencing methylation data.

### Usage

```
readBismark(file)
```

### Arguments

**file**            The filename (including path) of the methylation (CX report generated by Bismark) to be read.

### Value

the methylation data stored as a [GRanges](#) object with four metadata columns (see [methylationDataList](#)).

### Author(s)

Nicolae Radu Zabet and Jonathan Michael Foonlan Tsang

### Examples

```
# load methylation data object
data(methylationDataList)

# save the one datasets into a file
saveBismark(methylationDataList[["WT"]], "chr3test_a_thaliana_wt.CX_report")

# load the data
methylationDataWT <- readBismark("chr3test_a_thaliana_wt.CX_report")

#check that the loading worked
all(methylationDataWT == methylationDataList[["WT"]])
```

---

readBismarkPool	<i>Read Bismark pool</i>
-----------------	--------------------------

---

## Description

This function takes as input a vector of CX report file produced by Bismark and returns a [GRanges](#) object with four metadata columns (see [methylationDataList](#)). The file represents the pooled bisulfite sequencing data.

## Usage

```
readBismarkPool(files)
```

## Arguments

files	The filenames (including path) of the methylation (CX report generated with Bismark) to be read
-------	---

## Value

the methylation data stored as a [GRanges](#) object with four metadata columns (see [methylationDataList](#)).

## Author(s)

Nicolae Radu Zabet and Jonathan Michael Foonlan Tsang

## Examples

```
# load methylation data object
data(methylationDataList)

# save the two datasets
saveBismark(methylationDataList[["WT"]],
            "chr3test_a_thaliana_wt.CX_report")
saveBismark(methylationDataList[["met1-3"]],
            "chr3test_a_thaliana_met13.CX_report")

# reload the two datasets and pool them
filenames <- c("chr3test_a_thaliana_wt.CX_report",
              "chr3test_a_thaliana_met13.CX_report")
methylationDataPool <- readBismarkPool(filenames)
```

---

readONTbam	<i>Load ONT BAM, decode MM/ML, and count modified vs. unmodified reads</i>
------------	--

---

### Description

readONTbam() takes an indexed Nanopore BAM file with MM/ML tags, decodes each read's per-C modification probabilities, and overlays them on a GRanges of candidate cytosine sites. It returns a copy of ref\_gr augmented with:

- ONT\_Cm — comma-delimited read-indices called “modified”
- ONT\_C — comma-delimited read-indices covering but `_not_` modified
- readsM — count of modified reads at each site
- readsN — total same-strand coverage at each site

You can either supply your own ref\_gr (e.g. from selectCytosine()) or leave it NULL and pass context, chr, region to build ref\_gr on the fly.

### Usage

```
readONTbam(
  bamfile,
  ref_gr = NULL,
  modif = "C+m?",
  prob_thresh = 0.5,
  genome = BSgenome.Hsapiens.UCSC.hg38,
  context = "CG",
  chr = NULL,
  region = NULL,
  synonymous = FALSE,
  parallel = FALSE,
  BPPARAM = NULL,
  cores = NULL
)
```

### Arguments

bamfile	Path to an indexed ONT BAM with MM/ML tags.
ref_gr	A GRanges of genomic cytosine positions to annotate. If NULL, will be created via selectCytosine() using context, chr, region.
modif	Character vector of MM codes to treat as “modified” (e.g. "C+m?", "C+h?", "C+m. ").
prob_thresh	Numeric in $[0,1]$ — minimum ML probability to call a read “modified.”
genome	A BSgenome object such as BSgenome.Hsapiens.UCSC.hg38. This is used to extract sequence context and must be loaded in advance. <b>Note:</b> When running on an HPC system, please ensure that the required BSgenome package is already installed and loaded in advance.

context	Sequence context for selectCytosine() (e.g. "CG", "CHG", "CHH").
chr	Chromosome names to restrict selectCytosine().
region	A GRanges to further subset selectCytosine().
synonymous	Logical (default: FALSE). If TRUE, include modified calls that match the specified context sequence (e.g. CGG), even if the site was previously excluded due to deletion or mismatch. For example, if a deletion occurs at position 234523, but the surrounding context still forms CGG, then the modified C at 234523 will be retained (Nmod=1).
parallel	Logical. If TRUE, automatically detect your system condition and decoding will use parallel threads via BiocParallel::. If FALSE (default), decoding is done serially.
BPPARAM	A BiocParallelParam object controlling parallel execution. This value will automatically set when parallel is TRUE, also able to set as manually.
cores	Integer number of workers (must not exceed BPPARAM\$workers). This value will automatically set as the maximum number of system workers, also able to set as manually.

### Details

This function read and annotate ONT MM/ML tags against a cytosine reference

### Value

A GRanges of the same length as ref\_gr, with four additional metadata columns:

**ONT\_Cm** comma-delimited read-indices called modified

**ONT\_C** comma-delimited read-indices covering but unmodified

**readsM** integer count of modified reads per site

**readsN** integer count of same-strand reads covering each site

### Author(s)

Nicolae Radu Zabet and Young Jun Kim

### See Also

[selectCytosine](#), [computeDMRs](#), [computePMDs](#), [computeCoMethylatedPositions](#), [computeCoMethylatedRegions](#), [filterVMRsONT](#), [ontSampleGRangesList](#), [scanBamChr1Random5](#)

### Examples

```
## Not run:
library(DMRcaller)
library(BSgenome.Hsapiens.UCSC.hg38)

# set the bam file directory
bam_path <- system.file("extdata", "scanBamChr1Random5.bam", package="DMRcaller")
```

```
# read ONTbam file (chromosome 1 only) in CG context with BSgenome.Hsapiens.UCSC.hg38
ONTSampleGRanges <- readONTbam(bamfile = bam_path, ref_gr = NULL, modif = "C+m?",
                               prob_thresh = 0.50, genome = BSgenome.Hsapiens.UCSC.hg38,
                               context = "CG", chr = "chr1", region = NULL,
                               synonymous = FALSE, parallel = FALSE, BPPARAM = NULL)

## End(Not run)
```

---

saveBismark

*Save Bismark*


---

## Description

This function takes as input a [GRanges](#) object generated with [readBismark](#) and saves the output to a file using Bismark CX report format.

## Usage

```
saveBismark(methylationData, filename)
```

## Arguments

methylationData	the methylation data stored as a <a href="#">GRanges</a> object with four metadata columns (see <a href="#">methylationDataList</a> ).
filename	the filename where the data will be saved.

## Value

Invisibly returns NULL

## Author(s)

Nicolae Radu Zabet

## Examples

```
# load methylation data object
data(methylationDataList)

# save one dataset to a file
saveBismark(methylationDataList[["WT"]], "chr3test_a_thaliana_wt.CX_report")
```

---

scanBamChr1Random5      *The bam file from ONT nanopore .pod5 files*

---

### Description

A .bam file containing the basecalling result of 5 sequences containing with the MM, ML tag from dorado.

### Format

A .bam object

### Source

The object was created by base-calling and alignment by dorado, ver.0.9.6 <https://github.com/nanoporetech/dorado?tab=readme-ov-file#dna-models> to generate the bam files with dna\_r10.4.1\_e8.2\_400bps\_hac@v5.2.0 as basecalling model from GM18501 cell line [https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5\\_data/GM18501\\_R9/](https://s3.amazonaws.com/1000g-ont/index.html?prefix=pod5_data/GM18501_R9/). To subset, call the bam file by `scanBam` function from `Rsamtools` package and randomly select the 5 sequences. and repackaging the subsetted list to the bam file for test running.

---

selectCytosine      *Select Cytosine Positions*

---

### Description

Constructs a `GRanges` of all cytosine positions in the specified `BSgenome` (or `BSgenome` package name), optionally filtering by methylation context ("CG", "CHG", "CHH"), by chromosome, and by genomic region.

### Usage

```
selectCytosine(
  genome = BSgenome.Hsapiens.UCSC.hg38,
  context = c("CG", "CHG", "CHH"),
  chr = NULL,
  region = NULL
)
```

### Arguments

`genome`      A `BSgenome` object or the name (character) of a `BSgenome` package to use as the reference genome. If a package name is given, it will be loaded automatically (default: `BSgenome.Hsapiens.UCSC.hg38`). **Note:** When running on an HPC system, please ensure that the required `BSgenome` package is already installed and loaded in advance.

context	A character vector of one or more methylation contexts to include: "CG", "CHG", and/or "CHH". Defaults to all three.
chr	An optional character vector of chromosome names to restrict the enumeration. If NULL, all sequences in genome are used.
region	An optional <a href="#">GRanges</a> object specifying subregions to keep. Requires chr to be non-NULL.

### Details

This function enumerates cytosine positions in a BSgenome reference

### Value

A [GRanges](#) object with one 1-bp range per cytosine, and two metadata columns:

**context** A factor indicating the context ("CG", "CHG", or "CHH").

**trinucleotide\_context** Character or factor giving the surrounding trinucleotide sequence (or NA for "CG").

### Author(s)

Nicolae Radu Zabet and Young Jun Kim

### See Also

[readONTbam](#), [ontSampleGRangesList](#)

### Examples

```
## Not run:
library(BSgenome.Hsapiens.UCSC.hg38)

# Only CpG sites on chr1 and chr2:
gr_chr1_2 <- selectCytosine(context="CG", chr=c("chr1","chr2"))

# CHH sites in a specific region on chr3:
my_region <- GRanges("chr3", IRanges(1e6, 1e6 + 1e5))
gr_region <- selectCytosine(context="CHH", chr="chr3", region=my_region)

## End(Not run)
```

---

`syntheticDataReplicates`*Simulated data for biological replicates*

---

**Description**

A `GRanges` object containing simulated data for methylation in four samples. The conditions associated with each sample are a, a, b and b.

**Format**

A `GRanges` object containing multiple metadata columns with the reads from each object passed as parameter

**Source**

The object was created by calling `joinReplicates` function.

# Index

- analyseReadsInsideRegionsForCondition, [3](#), [13](#), [26](#), [28](#), [36](#), [38](#), [40](#), [42](#), [49](#), [51](#)
- analyseReadsInsideRegionsForConditionPMD, [4](#), [23](#), [29](#), [67](#), [68](#)
- computeCoMethylatedPositions, [6](#), [29](#), [45](#), [73](#)
- computeCoMethylatedRegions, [8](#), [29](#), [73](#)
- computeDMRs, [4](#), [11](#), [21](#), [28](#), [36](#), [38](#), [48](#), [49](#), [56](#), [66](#), [73](#)
- computeDMRsReplicates, [14](#), [28](#)
- computeMethylationDataCoverage, [17](#), [28](#), [58](#)
- computeMethylationDataSpatialCorrelation, [18](#), [28](#), [61](#)
- computeMethylationProfile, [19](#), [28](#), [62](#), [65](#)
- computeOverlapProfile, [20](#), [28](#), [66](#)
- computePMDs, [5](#), [8](#), [10](#), [22](#), [29](#), [40](#), [45](#), [51](#), [67](#), [68](#), [73](#)
- computeVMDs, [25](#), [29](#), [42](#)
- DMRcaller, [27](#), [54](#)
- DMRcaller-package (DMRcaller), [27](#)
- DMRsNoiseFilterCG, [4](#), [13](#), [36](#), [38](#), [49](#)
- extractGC, [36](#)
- filterDMRs, [4](#), [13](#), [21](#), [28](#), [36](#), [37](#), [49](#), [56](#), [66](#)
- filterPMDs, [5](#), [23](#), [29](#), [39](#), [51](#), [67](#), [68](#)
- filterVMDs, [26](#), [41](#)
- filterVMRsONT, [29](#), [43](#), [73](#)
- GEs, [45](#)
- GEs\_hg38, [45](#), [46](#)
- getWholeChromosomes, [28](#), [46](#)
- GRanges, [3–5](#), [11](#), [12](#), [15](#), [17–23](#), [25](#), [26](#), [28](#), [36](#), [38](#), [40–42](#), [46–49](#), [51](#), [56](#), [58](#), [60](#), [62](#), [64](#), [66–71](#), [74–77](#)
- GRangesList, [56](#), [68](#)
- joinReplicates, [28](#), [47](#), [77](#)
- mergeDMRsIteratively, [4](#), [13](#), [21](#), [28](#), [36](#), [38](#), [48](#), [66](#)
- mergePMDsIteratively, [5](#), [23](#), [29](#), [40](#), [50](#), [67](#), [68](#)
- methylationDataList, [3](#), [11](#), [17–20](#), [36](#), [38](#), [46–48](#), [52](#), [56](#), [58](#), [60–62](#), [64](#), [65](#), [68–71](#), [74](#)
- ont\_gr\_GM18870\_chr1\_PMD\_bins\_1k, [54](#)
- ont\_gr\_GM18870\_chr1\_sorted\_bins\_1k, [54](#)
- ontSampleGRangesList, [5](#), [8](#), [10](#), [22](#), [25](#), [39](#), [41](#), [43](#), [45](#), [51](#), [53](#), [67](#), [68](#), [73](#), [76](#)
- plot, [60](#)
- plotLocalMethylationProfile, [28](#), [55](#)
- plotMethylationDataCoverage, [17](#), [28](#), [57](#)
- plotMethylationDataSpatialCorrelation, [19](#), [28](#), [59](#)
- plotMethylationProfile, [20](#), [28](#), [61](#), [65](#)
- plotMethylationProfileFromData, [20](#), [28](#), [62](#), [63](#)
- plotOverlapProfile, [21](#), [28](#), [65](#)
- PMDsBinsCG, [67](#)
- PMDsNoiseFilterCG, [5](#), [23](#), [40](#), [51](#), [68](#)
- poolMethylationDatasets, [28](#), [68](#)
- poolTwoMethylationDatasets, [28](#), [69](#)
- readBismark, [28](#), [53](#), [70](#), [74](#)
- readBismarkPool, [28](#), [71](#)
- readONTbam, [4](#), [8](#), [10](#), [23](#), [26–28](#), [45](#), [54](#), [68](#), [69](#), [72](#), [76](#)
- Rsamtools, [75](#)
- saveBismark, [28](#), [74](#)
- scanBam, [75](#)
- scanBamChr1Random5, [73](#), [75](#)
- selectCytosine, [28](#), [73](#), [75](#)
- syntheticDataReplicates, [77](#)