Package 'DSS'

November 11, 2026

Title Dispersion shrinkage for sequencing data

Version 2.60.0

Date 2026-4-9

Author Hao Wu<hao.wu@emory.edu>, Hao Feng<hxf155@case.edu>

Depends R (>= 3.5.0), methods, Biobase, BiocParallel, bsseq, parallel

Imports utils, graphics, stats, splines

Maintainer Hao Wu <hao.wu@emory.edu>, Hao Feng <hxf155@case.edu>

Description DSS is an R library performing differntial analysis for count-based sequencing data. It detectes differentially expressed genes (DEGs) from RNA-seq, and differentially methylated loci or regions (DML/DMRs) from bisulfite sequencing (BS-seq). The core of DSS is a new dispersion shrinkage method for estimating the dispersion parameter from Gamma-Poisson or Beta-Binomial distributions.

License GPL

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown, edgeR

biocViews Sequencing, RNASeq, DNAMethylation,GeneExpression, DifferentialExpression,DifferentialMethylation

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

git_branch RELEASE_3_23

git_last_commit a954918

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

DSS-package 2
callDML 2

callDMR . . . 4
design . . . 7
dispersion . . . 7
DMLfit.multiFactor . . . 8
DMLtest . . . 10
DMLtest.multiFactor . . . 12
DSS.DE . . . 15
estDispersion . . . 16
estNormFactors . . . 17
makeBSseqData . . . 18
normalizationFactor . . . 19
RRBS . . . 20
SeqCountSet-class . . . 21
seqData . . . 22
showOneDMR . . . 23
waldTest . . . 24

Index **26**

`DSS-package` *Dispersion shrinkage for sequencing data*

Description

DSS is an R library performing the differential expression analysis for RNA-seq count data. Compared with other similar packages (DESeq, edgeR), DSS implements a new dispersion shrinkage method to estimate the gene-specific biological variance. Extensive simulation results showed that DSS performs favorably compared to DESeq and edgeR when the variation of biological variances is large.

DSS only works for two group comparison at this time. We plan to extend the functionalities and make it work for more general experimental designs in the near future.

Author(s)

Hao Wu <hao.wu@emory.edu>

`callDML` *Function to detect differntially methylated loci (DML) from bisulfite sequencing (BS-seq) data.*

Description

This function takes the results from DML testing procedure ('DMLtest' function) and calls DMLs. Regions will CpG sites being statistically significant are deemed as DMLs.

Usage

```
callDML(DMLresult, delta=0.1, p.threshold=1e-5)
```

Arguments

DMLresult	A data frame representing the results for DML detection. This should be the result returned from 'DMLtest' function.
delta	A threshold for defining DML. In DML testing procedure, hypothesis test that the two groups means are equalis is conducted at each CpG site. Here if 'delta' is specified, the function will compute the posterior probability that the difference of the means are greater than delta,and then call DML based on that. This only works when the test results are from 'DMLtest', which is for two-group comparison. For general design, this has to be set to 0.
p.threshold	When delta is not specified, this is the threshold of p-values for defining DML, e.g. Loci with p-values less than this threshold will be deemed DMLs. When delta is specified, CpG sites with posterior probability greater than 1-p.threshold are deemed DML.

Value

A data frame for DMLs. Each row is for a DML. DMLs are sorted by statistical significance. The columns are

chr	Chromosome number.
pos	Genomic coordinates.
mu1, mu2	Mean methylations of two groups.
diff	Difference of mean methylations of two groups.
diff.se	Standard error of the methylation difference.
stat	Wald statistics.
phi1, phi2	Estimated dispersions in two groups.
pval	P-values. This is obtained from normal distribution.
fdr	False discovery rate.
postprob.overThreshold	The posterior probability of the difference in methylation greater than delta. This columns is only available when delta>0.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

DMLtest, callDMR

Examples

```
## Not run:
require(bsseq)

## first read in methylation data.
path <- file.path(system.file(package="DSS"), "extdata")
dat1.1 <- read.table(file.path(path, "cond1_1.txt"), header=TRUE)
dat1.2 <- read.table(file.path(path, "cond1_2.txt"), header=TRUE)
dat2.1 <- read.table(file.path(path, "cond2_1.txt"), header=TRUE)
dat2.2 <- read.table(file.path(path, "cond2_2.txt"), header=TRUE)

## make BSseq objects
BSobj <- makeBSseqData( list(dat1.1, dat1.2, dat2.1, dat2.2),
  c("C1","C2", "N1", "N2") )

##  DML test
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2"), group2=c("N1","N2"))

## call DML
dmls <- callDML(dmlTest)
head(dmls)

## call DML with a threshold
dmls2 <- callDML(dmlTest, delta=0.1)
head(dmls2)

## For whole-genome BS-seq data, perform DML test with smoothing
require(bsseqData)
data(BS.cancer.ex)
## takea smallportionof data and test
BSobj <- BS.cancer.ex[10000:15000,]
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2", "C3"), group2=c("N1","N2","N3"),
   smoothing=TRUE, smoothing.span=500)
dmls <- callDML(dmlTest)
head(dmls)

## from multifactor design
data(RRBS)
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell)
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef="cellrN")
dml = callDML(DMLtest.cell, p.threshold=0.05) ## this produce a warning
dml = callDML(DMLtest.cell, delta=0, p.threshold=0.05) ## no warning

head(dml)


## End(Not run)
```

`callDMR` *Function to detect differntially methylated regions (DMR) from bisulfite sequencing (BS-seq) data.*

Description

This function takes the results from DML testing procedure ('callDML' function) and calls DMRs. Regions will CpG sites being statistically significant are detected as DMRs. Nearby DMRs are merged into longer ones. Some restrictions including the minimum length, minimum number of CpG sites, etc. are applied.

Usage

```
callDMR(DMLresult, delta=0, p.threshold=1e-5,
        minlen=50, minCG=3, dis.merge=100, pct.sig=0.5)
```

Arguments

DMLresult	A data frame representing the results for DML detection. This should be the result returned from 'DMLtest' or 'DMLtest.multiFactor' function.
delta	A threshold for defining DMR. In DML detection procedure, a hypothesis test that the two groups means are equal is conducted at each CpG site. Here if 'delta' is specified, the function will compute the posterior probability that the difference of the means are greater than delta, and then construct DMR based on that. This only works when the test results are from 'DMLtest', which is for two-group comparison.
p.threshold	A threshold of p-values for calling DMR. Loci with p-values less than this threshold will be picked and joint to form the DMRs. See 'details' for more information.
minlen	Minimum length (in basepairs) required for DMR. Default is 50 bps.
minCG	Minimum number of CpG sites required for DMR. Default is 3.
dis.merge	When two DMRs are very close to each other and the distance (in bps) is less than this number, they will be merged into one. Default is 50 bps.
pct.sig	In all DMRs, the percentage of CG sites with significant p-values (less than p.threshold) must be greater than this threshold. Default is 0.5. This is mainly used for correcting the effects of merging of nearby DMRs.

Details

The choices of 'delta' and 'p.threshold' are somewhat arbitrary. The default value for p-value threshold for calling DMR is 1e-5. The statistical test on loci level is less powerful when smoothing is NOT applied, so users can consider to use a less stringent criteria, such as 0.001, in order to get satisfactory number of DMRs. This function is reasonably fast since the computationally intesnsive part is in 'DMLtest'. Users can try different p.threshold values to obtain satisfactory results.

'delta' is only supported when the experiment is for two-group comparison. This is because in multifactor design, the estimated coefficients in the regression are based on a GLM framework (loosely speaking), thus they don't have clear meaning of methylation level differences. So when the input DMLresult is from DMLtest.multiFactor, 'delta' cannot be specified.

When specifying a 'delta' value, the posterior probability (pp) of each CpG site being DML is computed. Then the p.threshold is applied on 1-pp, e.g., sites with 1-pp<p.threshold is deemed significant. In this case, the criteria for DMR calling is more stringent and users might consider to use a more liberal p.threshold in order to get more regions.

Value

A data frame for DMRs. Each row is for a DMR. Rows are sorted by "areaStat", which is the sum of test statistics of all CpG sites in the region. The columns are:

`chr`	Chromosome number.
`start`, `end`	Genomic coordinates.
`length`	Length of the DMR, in bps.
`nCG`	Number of CpG sites contained in the DMR.
`meanMethy1`, `meanMethy2`	Average methylation levels in two conditions.
`diff.Methy`	The difference in the methylation levels between two conditions. diff.Methy=meanMethy1-meanMethy2.
`areaStat`	The sum of the test statistics of all CpG sites within the DMR.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

DMLtest, callDML

Examples

```
## Not run:
require(bsseq)
require(bsseqData)
data(BS.cancer.ex)

## take a small portion of data and test
BSobj <- BS.cancer.ex[140000:150000,]
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2", "C3"), group2=c("N1","N2","N3"),
   smoothing=TRUE, smoothing.span=500)

## call DMR based on test results
dmrs <- callDMR(dmlTest)
head(dmrs)

## or one can specify a threshold for difference in methylation level
dmrs2 <- callDMR(dmlTest, delta=0.1)
head(dmrs2)

## visualize one DMR
showOneDMR(dmrs[1,], BSobj)

## from multifactor design - using a loose threshold to demonstrate

data(RRBS)
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell)
```

```
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef="cellrN")
dmr = callDMR(DMLtest.cell, p.threshold=0.05)
dmr

## End(Not run)
```

---

design *Experimental design for the example RRBS dataset*

Description

The RRBS dataset is from 16 samples with two factors (case and cell), each has two levels (so it's a 2x2 design).

Usage

```
data(RRBS)
```

Examples

```
data(RRBS)
RRBS
design
```

---

dispersion *Acessor functions for the 'dispersion' slot in a SeqCountData object.*

Description

Dispersion parameter for a gene represents its coefficient of variation of expressions. It characterizes the biological variations.

Usage

```
## S4 method for signature 'SeqCountSet'
dispersion(object)
## S4 replacement method for signature 'SeqCountSet,numeric'
dispersion(object) <- value
```

Arguments

object	A SeqCountData object.
value	A numeric vector with the same length as number of genes.

Details

If the counts from biological replicates are modeled as negative binomial distribution, the variance (v) and mean (m) should hold following relationship: v=m+m^2*phi, where phi is the dispersion. Another interpretation is that phi represents the biological variations among replicates when underlying expressions are modeled as a Gamma distribution.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

normalizationFactor

Examples

```
data(seqData)
## obtain
seqData=estNormFactors(seqData, "quantile")
seqData=estDispersion(seqData)
dispersion(seqData)

## assign
dispersion(seqData)=rep(0.1, nrow(exprs(seqData)))
```

DMLfit.multiFactor *Fit a linear model for BS-seq data from general experimental design*

Description

This function takes a BSseq object, a data frame for experimental design and a model formula and then fit a linear model.

Usage

```
DMLfit.multiFactor(BSobj, design, formula, smoothing=FALSE, smoothing.span=500)
```

Arguments

BSobj	An object of BSseq class for the BS-seq data.
design	A data frame for experimental design. Number of rows must match the number of columns of the counts in BSobj.
formula	A formula for the linear model.
smoothing	A flag to indicate whether to apply smoothing. When true, the counts will be smoothed by a simple moving average method.
smoothing.span	The size of smoothing window, in basepairs. Default is 500.

Details

The lineear model fitting is done through ordinary least square on the arscine transformed methylation percentages. The estimated standard errors are computed with consideration of the data (count) distribution and transformation. This function is extremely efficient. The computation takes around 20 minutes for 4 million CpG sites.

Value

A list with following components

`gr`	An object of 'GRanges' for locations of the CpG sites.
`design`	The input data frame for experimental design.
`formula`	The input formula for the model.
`X`	The design matrix used in regression. It is created based on design and formula.
`fit`	The model fitting results. This is a list itself, with three components: 'beta' - the estimated coefficients; 'var.beta' - estimated variance/covariance matrices for beta. 'phi' - estimated beta-binomial dispersion parameters. Note that var.beta for a CpG site should be a ncol(X) x ncol(X) matrix, but is flattend to a vector so that the matrices for all CpG sites can be saved as a matrix.

Author(s)

Hao Wu<hao.wu@emory.edu>

See Also

DMLtest.multiFactor, DMLtest

Examples

```
## Not run:
data(RRBS)
## model fitting
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell)

## with smoothing:
DMLfit.sm = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell, smoothing=TRUE)

## hypothesis testing
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef=3)

## look at distributions of test statistics and p-values
par(mfrow=c(1,2))
hist(DMLtest.cell$stat, 100, main="test statistics")
hist(DMLtest.cell$pvals, 100, main="P values")

## End(Not run)
```

DMLtest — *Function to perform statistical test of differntially methylated loci (DML) for two group comparisons of bisulfite sequencing (BS-seq) data.*

Description

This function takes a BSseq object and two group labels, then perform statistical tests for differntial methylation at each CpG site.

Usage

```
DMLtest(BSobj, group1, group2, equal.disp = FALSE, smoothing = FALSE,
        smoothing.span = 500, ncores)
```

Arguments

BSobj	An object of BSseq class for the BS-seq data.
group1, group2	Vectors of sample names or indexes for the two groups to be tested. See more description in details.
equal.disp	A flag to indicate whether the dispersion in two groups are deemed equal. Default is FALSE, and the dispersion shrinkages are performed on two conditions independently.
smoothing	A flag to indicate whether to apply smoothing in estimating mean methylation levels.
smoothing.span	The size of smoothing window, in basepairs. Default is 500.
ncores	Number of CPU cores used in parallel computing. See sections 'Parallelization' for details.

Details

This is the core function for DML/DMR detection. Tests are performed at each CpG site under the null hypothesis that two groups means are equal. There is an option for applying smoothing or not in estimating mean methylation levels. We recommend to use smoothing=TRUE for whole-genome BS-seq data, and smoothing=FALSE for sparser data such like from RRBS or hydroxyl-methylation data (TAB-seq). If there is not biological replicate, smoothing=TRUE is required. See "Single replicate" section for details.

The BS-seq count data are modeled as Beta-Binomial distribution, where the biological variations are captured by the dispersion parameter. The dispersion parameters are estimated through a shrinakge estimator based on a Bayesian hierarchical model. Then a Wald test is performed at each CpG site.

Due to the differences in coverages, some CpG sites are not covered in both groups, and the test cannot be performed. Those loci will be ignored in test and results will be "NA".

Value

A data frame with each row corresponding to a CpG site. Rows are sorted by chromosome number and genomic coordinates. The columns include:

`chr`	Chromosome number.
`pos`	Genomic coordinates.
`mu1`, `mu2`	Mean methylations of two groups.
`diff`	Difference of mean methylations of two groups. diff=mu1-mu2.
`diff.se`	Standard error of the methylation difference.
`stat`	Wald statistics.
`pval`	P-values. This is obtained from normal distribution.
`fdr`	False discovery rate.

Single replicate

When there is no biological replicate in one or both treatment groups, users can either (1) specify equal.disp=TRUE, which assumes both groups have the same dispersion, then the data from two groups are combined and used as replicates to estimate dispersion; or (2) specify smoothing=TRUE, which uses the smoothed means (methylation levels) to estimate dispersions via a shrinkage estimator. This smoothing procedure uses data from neighboring CpG sites as "pseudo-replicate" for estimating biological variance.

Parallelization

The shrinkage estimation for dispersion is the most computational component in DML testing. We use the 'mcapply' function in 'parallel' package to implement parallelization. Note that older version of DSS (<2.4x) used the 'bplapply' function in 'BiocParallel' package. However, that function somehow has significantly reduced performance in the new release (>1.25), so we switched to mcapply. A drawback is that the progress bar cannot be displayed under the paralelle computing setting.

Users might experience problems on Windows, since the mcapply function relies on forking but Windows does not support forking. Thus, we suggest to use ncores=1 on Windows. For more details, please read the 'parallel' documentation.

Estimating mean methylation levels

When smoothing=FALSE, the mean methylation levels are estimated based on the ratios of methylated and total read counts, and the spatial correlations among nearby CpG sites are ignored. When smoothing=TRUE, smoothing based on moving average or the BSmooth method is used to estimate the mean methylaion level at each site. Moving average is recommended because it is much faster than BSmooth, and the results are reasonable similar in terms of mean estimation, dispersion estimation, and DMR calling results.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

makeBSseqData, callDML, callDMR

Examples

```
## Not run:
require(bsseq)

## first read in methylation data.
path <- file.path(system.file(package="DSS"), "extdata")
dat1.1 <- read.table(file.path(path, "cond1_1.txt"), header=TRUE)
dat1.2 <- read.table(file.path(path, "cond1_2.txt"), header=TRUE)
dat2.1 <- read.table(file.path(path, "cond2_1.txt"), header=TRUE)
dat2.2 <- read.table(file.path(path, "cond2_2.txt"), header=TRUE)

## make BSseq objects
BSobj <- makeBSseqData( list(dat1.1, dat1.2, dat2.1, dat2.2),
  c("C1","C2", "N1", "N2") )

##  DML test without smoothing
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2"), group2=c("N1", "N2"))
head(dmlTest)

## For whole-genome BS-seq data, perform DML test with smoothing
require(bsseqData)
data(BS.cancer.ex)
## take a small portion of data and test
BSobj <- BS.cancer.ex[10000:15000,]
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2", "C3"), group2=c("N1","N2","N3"),
                   smoothing=TRUE, smoothing.span=500)
head(dmlTest)

## Examples for Parallelization
## use single core - this has not parallelization
system.time(dmlTest <- DMLtest(BSobj, group1=c("C1", "C2"), group2=c("N1", "N2"), ncores=1))

## use 4 cores - it's about twice as fast
system.time(dmlTest <- DMLtest(BSobj, group1=c("C1", "C2"), group2=c("N1", "N2"), ncores=4))


## End(Not run)
```

`DMLtest.multiFactor` *Perform statistical test for BS-seq data from general experimental design*

Description

This function takes the linar model fitting results and performs Wald test at each CpG site, then return test statistics, p-values and FDR.

Usage

```
DMLtest.multiFactor(DMLfit, coef = 2, term, Contrast)
```

Arguments

DMLfit	Result object returned from 'DMLfit.multiFactor' function.
coef	It can be an integer to indicate which coefficient in the linear model is be tested for being zero. Be careful of intercept. If the model contains intercept, coef=2 indicate testing the first factor in the formula. If the model has no intercept, testing first factor should use coef=1. It can also be a character for the terms to be tested. In that case it must match one of the column names in the design matrix. One can look at colnames(DMLfit$X) to obtain the column names.
term	The term(s) to be tested, as one or a vector of characters. Can be multiple terms. See 'Hypothesis test' section for details.
Contrast	A contrast matrix for hypothesis testing. The number of rows must equal to the number of columns in the design matrix ncol(DMLfit$X). See 'Hypothesis test' section for details.

Value

A data frame with following columns: chr, pos, stat, pvals, fdr. Each row is for a CpG site. Note that the CpG sites are sorted by chromosome and position.

Hypothesis test

User can specify one of the following parameter for testing: 'coef', 'term', or 'Contrast'.

When specifying 'coef', it tests *one* parameter in the model, which corresponds to one column in the design matrix. In this case, A Wald test is performed using the estimated coefficient and standard error from 'DMLfit.multiFactor'. P-values are derived from test statistics based on normal distribution.

When specifying 'term', it tests the whole term in the model. If the term is continuous or a categorical variable with only two levels (one degree of freedom), it is equivalent to specifying 'coef' because it tests only one parameter in the model. However, when the term is a categorical variable with more than two levels, it will test multiple parameters at the same time so it's a compound hypothesis test, and a F-test will be performed.

Specifying 'Contrast' matrix provides the most flexible test procedure. It can test any linear combination of the parameter, and F-test will be performed. Let L be the contrast matrix. The hypothesis test performed is H0: L^T * beta = 0. Thus the number of rows in L must equal to the number of items of beta (which is the number of columns in the design matrix).

Using 'term' or 'Contrast' will be slower especially when there are a lot of CpG sites, because the computation cannot be vectorized (each CpG site has a different variance/covariance matrix for the estimated coefficients).

FDR is computed using cannonical Benjamini-Hochberg procedure.

Author(s)

Hao Wu<hao.wu@emory.edu>

See Also

DMLfit.multiFactor, DMLtest

Examples

```
## Not run:
data(RRBS)
## model fitting
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell)

## hypothesis testing - following two lines do the same thing
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef=3)
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef="cellrN")

## this doesn't work
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef="cell")

## look at distributions of test statistics and p-values
par(mfrow=c(1,2))
hist(DMLtest.cell$stat, 100, main="test statistics")
hist(DMLtest.cell$pvals, 100, main="P values")

## Using term or Contrast
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell)

## following 4 tests should produce the same results,
## since 'case' only has two levels.
## However the p-values from F-tests (using term or Contrast) are
## slightly different, due to normal approximation in Wald test.
test1 = DMLtest.multiFactor(DMLfit, coef=2)
test2 = DMLtest.multiFactor(DMLfit, coef="caseSLE")
test3 = DMLtest.multiFactor(DMLfit, term="case")
Contrast = matrix(c(0,1,0), ncol=1)
test4 = DMLtest.multiFactor(DMLfit, Contrast=Contrast)
cor(cbind(test1$pval, test2$pval, test3$pval, test4$pval))


## note the different usage of term and coef.
## 'term' has to be in the formula, whereas 'coef' has to be in colnames
## of the design matrix.
DMLfit = DMLfit.multiFactor(RRBS, design, ~case+cell+case:cell)
DMLtest.cell = DMLtest.multiFactor(DMLfit, coef="cellrN")
DMLtest.cell = DMLtest.multiFactor(DMLfit, term="cell")
DMLtest.int = DMLtest.multiFactor(DMLfit, coef="caseSLE:cellrN")
DMLtest.int = DMLtest.multiFactor(DMLfit, term="case:cell")


## End(Not run)
```

DSS.DE *Perform RNA-seq differential expression analysis in two-group comparison*

Description

This is the top level wrapper function for RNA-seq differential expression analysis in a two-group comparison. Users only need to provide the count matrix and a vector for design, and obtain DE test results.

Usage

```
DSS.DE(counts, design)
```

Arguments

counts	A matrix of integers with rows corresponding to genes and columns for samples.
design	A vector representing the treatment groups. It must be a vector of 0 and 1. The length of the vector must match the number of columns of input count matrix.

Value

A data frame with each row corresponding to a gene. Rows are sorted according to wald test statistics. The columns are:

gene Index	index for input gene orders, integers from 1 to the number of genes.
muA	sample mean (after normalization) for sample A.
muB	sample mean (after normalization) for sample B.
lfc	log fold change of expressions between two groups.
difExpr	differences in expressions between two groups.
stats	Wald test statistics.
pval	p-values.
others	input gene annotations supplied as AnnotatedDataFrame when constructed the SeqCountData object.

Author(s)

Hao Wu <hao.wu@emory.edu>

Examples

```
counts = matrix(rpois(600, 10), ncol=6)
design = c(0,0,0,1,1,1)
result = DSS.DE(counts, design)
head(result)
```

`estDispersion` *Estimate and shrink tag-specific dipsersions*

Description

This function first estimate tag-specific dipsersions using a method of moment estimator. Then the dipsersions are shrunk based a penalized likelihood approach. The function works for general experimental designs.

Usage

```
## S4 method for signature 'SeqCountSet'
estDispersion(seqData, trend=FALSE)
```

Arguments

`seqData`	An object of SeqCountSet class.
`trend`	A binary indicator for modeling the dispersion~expression trend.

Details

The function takes and object of seqCountData class and return the same oject with "dispersion" field filled.

With "trend=TRUE" the dependence of dispersion on mean expressions will be modeled. In that case the shrinkage will be performed conditional on mean expressions.

The function works for multiple factor designs. But option "trend=TRUE" only applicable for single factor experiment.

Author(s)

Hao Wu <hao.wu@emory.edu>

Examples

```
data(seqData)
seqData=estNormFactors(seqData)
seqData=estDispersion(seqData)
head(dispersion(seqData))

## For multiple factor design
data(seqData)
Y=exprs(seqData)
design=data.frame(gender=c(rep("M",4), rep("F",4)), strain=rep(c("WT", "Mutant"),4))
X=as.data.frame(model.matrix(~gender+strain, data=design))
seqData=newSeqCountSet(Y, X)
seqData=estDispersion(seqData)
head(dispersion(seqData))
```

```
## the hypothesis testing for multifactor experiments can be performed
## using edgeR function, with DSS estimated dispersions
## Not run:
library(edgeR)
fit.edgeR <- glmFit(Y, X, lib.size=normalizationFactor(seqData), dispersion=dispersion(seqData))
lrt.edgeR <- glmLRT(fit.edgeR, coef=2)
head(lrt.edgeR$table)

## End(Not run)
```

`estNormFactors` *Estimate normalization factors*

Description

This function estimates normalization factors for the input 'seqCountSet' object and return the same object with normalizationFactor field filled or replaced.

Usage

```
## S4 method for signature 'SeqCountSet'
estNormFactors(seqData, method=c("lr", "quantile", "total", "median"))
```

Arguments

`seqData`	An object of "SeqCountSet" class.
`method`	Methods to be used in computing normalization factors. Currently available options only include methods to compute normalization factor to adjust for sequencing depths. Available options use (1) "lr": using median of logratio of counts. Similar to the TMM method. (2) "quantile" (default): 75th quantile, (3) "total": total counts, or (4) "median": median counts to constuct the normalization factors. From all methods the normalization factor will be a vector with same length as number of columns for input counts.

Value

The same "SeqCountSet" object with normalizationFactor field filled or replaced.

Author(s)

Hao Wu <hao.wu@emory.edu>

Examples

```
data(seqData)
## compare different methods
seqData=estNormFactors(seqData, "lr")
k1=normalizationFactor(seqData)
seqData=estNormFactors(seqData, "quantile")
k2=normalizationFactor(seqData)
seqData=estNormFactors(seqData, "total")
k3=normalizationFactor(seqData)
cor(cbind(k1,k2,k3))

## assign size factor
normalizationFactor(seqData)=k1

## or normalization factor can be a matrix
dd=exprs(seqData)
f=matrix(runif(length(dd), 1,10), nrow=nrow(dd), ncol=ncol(dd))
normalizationFactor(seqData)=f
head(normalizationFactor(seqData))
```

`makeBSseqData` *Create an object of BSseq class from several data frames.*

Description

This is an utility function to merge BS-seq data from replicated experiment and create an object of BSseq class.

After sequence alignment and proper processing, the BS-seq data can be summarized by following information at each C position (mostly CpG sites, with some CH): chromosome number, genomic coordinate, total number of reads covering the position, and number of reads showing methylation at this position. For replicated samples, the data need to be merged based on the chromosome number and genomic coordinates. This function provide such functionality. It takes replicated data as a list of data frames, merged them, and create a BSseq object.

Usage

```
makeBSseqData(dat, sampleNames)
```

Arguments

`dat`	A list of multiple data frames from biological replicates. Each element represents data from one replicate. The data frame MUST contain following columns in correct order: (1) Chromosome number; (2) Genomic coordinates; (3) Read coverage of the position from BS-seq data; (4) Number of reads showing methylation of the position. The colnames MUST BE "chr", "pos", "N", "X".
`sampleNames`	A vector of characters for the sample names. The length of the vector should match the length of the input list.

Value

An object of 'BSseq' class.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

callDML

Examples

```
require(bsseq)

## first read in methylation data.
path <- file.path(system.file(package="DSS"), "extdata")
dat1.1 <- read.table(file.path(path, "cond1_1.txt"), header=TRUE)
dat1.2 <- read.table(file.path(path, "cond1_2.txt"), header=TRUE)
dat2.1 <- read.table(file.path(path, "cond2_1.txt"), header=TRUE)
dat2.2 <- read.table(file.path(path, "cond2_2.txt"), header=TRUE)

## make BSseq objects
BSobj <- makeBSseqData( list(dat1.1, dat1.2, dat2.1, dat2.2),
  c("C1","C2", "N1", "N2") )

BSobj
sampleNames(BSobj)
```

---

normalizationFactor *Accessor functions for the 'normalizationFactor' slot in a SeqCountData object.*

Description

The normalization factors are used to adjust for technical or biological biases in the sequencing experiments. The factors can either be (1) a vector with length equals to the number of columns of the count data; or (2) a matrix with the same dimension of the count data.

Usage

```
## S4 method for signature 'SeqCountSet'
normalizationFactor(object)
## S4 replacement method for signature 'SeqCountSet,numeric'
normalizationFactor(object) <- value
## S4 replacement method for signature 'SeqCountSet,matrix'
normalizationFactor(object) <- value
```

Arguments

object	A SeqCountData object.
value	A numeric vector or matrix. If it is a vector it must have length equals to the number of columns of the count data. For matrix it must have the same dimension of the count data.

Details

The vector normalization factors are used mostly to correct for sequencing depth from different datasets. The matrix factor applies a different normalizing constant for each gene at each sample to adjust for a broader range of artifacts such as GC content.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

dispersion

Examples

```
data(seqData)
## obtain nomalization factor
seqData=estNormFactors(seqData, "quantile")
normalizationFactor(seqData)

## assign as vector
normalizationFactor(seqData)=rep(1, ncol(exprs(seqData))) ## getan error here

## or assign as a matrix
f=matrix(1, nrow=nrow(exprs(seqData)), ncol=ncol(exprs(seqData)))
normalizationFactor(seqData)=f
```

RRBS *An example dataset for multiple factor design*

Description

The dataset contains RRBS data for 5000 CpG sites from 16 samples. The experimental design is provided in the 'design' data frame.

Usage

```
data("RRBS")
```

Examples

```
data(RRBS)
RRBS
design
```

SeqCountSet-class *Class* "SeqCountSet" - *container for count data from sequencing experiment*

Description

This class is the main container for storing *RNA-seq* data. It is directly inherited fro 'ExpressionSet' class, with two more fields 'normalizationFactor' for normalization factors and 'dispersion' for gene-wise dispersions.

The class for BS-seq data is *BSseq*, which is imported from bsseq package.

Slots

`normalizationFactor`: Normalization factor for counts.

`dispersion`: Gene-wise dispersions.

`experimentData`: See 'ExpressionSet'.

`assayData`: See 'ExpressionSet'.

`phenoData`: See 'ExpressionSet'.

`featureData`: See 'ExpressionSet'.

`annotation`: See 'ExpressionSet'.

`protocolData`: See 'ExpressionSet'.

Extends

Class "ExpressionSet", directly. Class "eSet", by class "ExpressionSet", distance 2. Class "VersionedBiobase", by class "ExpressionSet", distance 3. Class "Versioned", by class "ExpressionSet", distance 4.

Constructor

`newSeqCountSet(counts,designs,normalizationFactor,featureData)`: Creates a 'SeqCountSet' object.

`counts` A matrix of integers with rows corresponding to genes and columns for samples.

`designs` A vector or data frame representing experimental design. The length of the vector or number of rows of the data frame must match the number of columns of input counts. This field can be accessed using 'pData' function.

`normalizationFactor` A vector or matrix of normalization factors for the counts.

`featureData` Additional information for genes as an 'AnnotatedDataFrame' object. This field can be access by using 'featureData' function.

Methods

dispersion, dispersion<- : Access and set gene-wise dispersions.

normalizationFactor, normalizationFactor<- : Access and set normalization factors.

Note

This is similar to 'CountDataSet' in DESeq or 'DGEList' in edgeR.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

dispersion, normalizationFactor

Examples

```
## simulate data from RNA-seq
counts=matrix(rpois(600, 10), ncol=6)
designs=c(0,0,0,1,1,1)
seqData=newSeqCountSet(counts, designs)
seqData
pData(seqData)
head(exprs(seqData))

## multiple factor designs
design=data.frame(gender=c(rep("M",4), rep("F",4)), strain=rep(c("WT", "Mutant"),4))
X=model.matrix(~gender+strain, data=design)
counts=matrix(rpois(800, 10), ncol=8)
seqData=newSeqCountSet(counts, as.data.frame(X))
seqData
pData(seqData)
```

---

seqData *A simulated 'SeqCountData' object.*

Description

The object is created based on simulation for 1000 genes and two treatment groups with 4 replicates in each group.

Usage

```
data(seqData)
```

Examples

```
data(seqData)
seqData
```

---

showOneDMR *Visualze the count data for one DMR*

Description

Given one DMR and an BSseq object, this function generate a multiple panel figure, each for a sample, to visualze the counts. There is a bar at each CpG, the gray bar shows the total coverage, and the black bar shows the methylated count.

Usage

```
showOneDMR(OneDMR, BSobj, ext = 500, ylim)
```

Arguments

OneDMR	A data frame with one row representing one DMR. It must have chr, start, and end columns. This is typically a row from the result generated from callDMR.
BSobj	An object of class BSseq.
ext	The amount (in bps) the plotting region should be extended in both directions.
ylim	Y-axis limit.

Value

This function only generates a figure and has no return values.

Author(s)

Hao Wu <hao.wu@emory.edu>

See Also

callDMR

Examples

```
## Not run:
require(bsseq)
require(bsseqData)
data(BS.cancer.ex)

## takea small portion of data and test
BSobj <- BS.cancer.ex[140000:150000,]
dmlTest <- DMLtest(BSobj, group1=c("C1", "C2", "C3"), group2=c("N1","N2","N3"),
```

```
   smoothing=TRUE, smoothing.span=500)

## call DMR based on test results
dmrs <- callDMR(dmlTest)

## visualize one DMR
showOneDMR(dmrs[1,], BSobj)

## End(Not run)
```

`waldTest` *Perform gene-wise Wald test for two group comparisons for sequencing count data.*

Description

The counts from two groups are modeled as negative binomial random variables with means and dispersions estimated. Wald statistics will be constructed. P-values will be obtained based on Gaussian assumption.

Usage

```
## S4 method for signature 'SeqCountSet'
waldTest(seqData, sampleA, sampleB, equal.var, fdr.method=c("BH", "locfdr"))
```

Arguments

`seqData`	An object of SeqCountSet class.
`sampleA`	The sample labels for the first sample to be compared in two-group comparison.
`sampleB`	The sample labels for the second sample to be compared in two-group comparison.
`equal.var`	A boolean to indicate whether to use the same or different means in two groups for computing variances in Wald test. Default is FALSE.
`fdr.method`	Method to compute FDR. Availabile options are "BH" for Benjamini-Hochberg FDR method, or local FDR from "locfdr" package.

Details

The input seqCountData object Must have normalizationFactor and dispersion fields filled, e.g., estNormFactors and estDispersion need to be called prior to this. With group means and shrunk dispersions ready, the variances for difference in group means will be constructed based on Negative Binomial distribution. P-values will be obtained under the assumption that the Wald test statistics are normally distributed. Genes with 0 counts in both groups will be assigned 0 for test statistics and 1 for p-values.

Value

A data frame with each row corresponding to a gene. Rows are sorted according to wald test statistics. The columns are:

gene Index	index for input gene orders, integers from 1 to the number of genes.
muA	sample mean (after normalization) for sample A.
muB	sample mean (after normalization) for sample B.
lfc	log fold change of expressions between two groups.
difExpr	differences in expressions between two groups.
stats	Wald test statistics.
pval	p-values.
fdr	FDR.
local.fdr	Local FDR if the FDR method is "locfdr".
others	input gene annotations supplied as AnnotatedDataFrame when constructed the SeqCountData object.

Author(s)

Hao Wu <hao.wu@emory.edu>

Examples

```
data(seqData)
seqData=estNormFactors(seqData)
seqData=estDispersion(seqData)
result=waldTest(seqData, 0, 1)
head(result)
```

Index

∗ **RNA-seq**
estDispersion, 16
∗ **classes**
SeqCountSet-class, 21
∗ **datasets**
design, 7
RRBS, 20
seqData, 22
∗ **normalization**
estNormFactors, 17
∗ **package**
DSS-package, 2

callDML, 2
callDMR, 4

design, 7
dispersion, 7
dispersion,SeqCountSet-method
*(*dispersion*)*, 7
dispersion<- *(*dispersion*)*, 7
dispersion<-,SeqCountSet,numeric-method
*(*dispersion*)*, 7
DMLfit.multiFactor, 8
DMLtest, 10
DMLtest.multiFactor, 12
DSS *(*DSS-package*)*, 2
DSS-package, 2
DSS.DE, 15

eSet, *21*
estDispersion, 16
estDispersion,SeqCountSet-method
*(*estDispersion*)*, 16
estNormFactors, 17
estNormFactors,SeqCountSet-method
*(*estNormFactors*)*, 17
ExpressionSet, *21*

makeBSseqData, 18

newSeqCountSet *(*SeqCountSet-class*)*, 21
normalizationFactor, 19
normalizationFactor,SeqCountSet-method
*(*normalizationFactor*)*, 19
normalizationFactor<-
*(*normalizationFactor*)*, 19
normalizationFactor<-,SeqCountSet,matrix-method
*(*normalizationFactor*)*, 19
normalizationFactor<-,SeqCountSet,numeric-method
*(*normalizationFactor*)*, 19

RRBS, 20

SeqCountSet *(*SeqCountSet-class*)*, 21
SeqCountSet-class, 21
seqData, 22
showOneDMR, 23

Versioned, *21*
VersionedBiobase, *21*

waldTest, 24
waldTest,SeqCountSet-method *(*waldTest*)*,
24