Package 'MAnorm2'

September 13, 2021

Type Package

Title Tools for Normalizing and Comparing ChIP-seq Samples

Version 1.2.0

Description Chromatin immunoprecipitation followed by high-throughput sequencing (ChIP-seq) is the premier technology for profiling genome-wide localization of chromatin-binding proteins, including transcription factors and histones with various modifications.
This package provides a robust method for normalizing ChIP-seq signals across individual samples or groups of samples. It also designs a self-contained system of statistical models for calling differential ChIP-seq signals between two or more biological conditions as well as for calling hypervariable ChIP-seq signals across samples. Refer to Tu et al. (2021) <doi:10.1101/gr.262675.120> and Chen et al. (2021) <doi:10.1101/2021.07.27.453915> for associated statistical details.

URL https://github.com/tushiqi/MAnorm2

BugReports https://github.com/tushiqi/MAnorm2/issues

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports stats, graphics, methods, locfit (>= 1.5.9), scales (>= 0.3.0), statmod (>= 1.4.34)

Suggests gplots (>= 3.0.1), DescTools (>= 0.99.24), knitr, rmarkdown

Date 2021-09-10

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Shiqi Tu [aut, cre] (<https://orcid.org/0000-0003-3534-5714>)

Maintainer Shiqi Tu <tushiqi@picb.ac.cn>

Repository CRAN

Date/Publication 2021-09-13 10:30:05 UTC

R topics documented:

`aovBioCond` *Perform a Moderated Analysis of Variance on* `bioCond` *Objects*

Description

Given a set of `bioCond` objects with which a mean-variance curve is associated, `aovBioCond` performs a one-way ANOVA-like analysis on them. More specifically, it separately tests for each genomic interval the null hypothesis that mean signal intensity in the interval remains invariant across all the biological conditions.

Usage

```
aovBioCond(conds, min.var = 0, df.prior = NULL)
```

Arguments

`conds`	A list of `bioCond` objects on which the analysis of variance is to be performed. They must be associated with the same mean-variance curve (i.e., they must have the same `"mvcID"`; see also `fitMeanVarCurve`).
`min.var`	Lower bound of variances read from the mean-variance curve. Any variance read from the curve less than `min.var` will be adjusted to this value. It's primarily used for safely getting the prior variances and taking into account the practical significance of a signal variation.
`df.prior`	Number of prior degrees of freedom associated with the mean-variance curve. Must be non-negative. Can be set to `Inf` (see "Details"). By default, `aovBioCond` checks if all the `bioConds` in `conds` have the same `"df.prior"` component, and uses it as the number of prior degrees of freedom if yes (an error is raised otherwise).

Details

aovBioCond adopts the modeling strategy implemented in limma (see "References"), except that each interval has its own prior variance, which is read from the mean-variance curve associated with the bioCond objects. Technically, this function calculates a moderated *F* statistic for each genomic interval to test the null hypothesis. The moderated *F* statistic is simply the *F* statistic from an ordinary one-way ANOVA with its denominator (i.e., sample variance) replaced by posterior variance, which is defined to be a weighted average of sample and prior variances, with the weights being proportional to their respective numbers of degrees of freedom. This method of incorporating the prior information increases the statistical power for the tests.

Two extreme values can be specified for the argument df.prior (number of degrees of freedom associated with the prior variances), representing two distinct cases: when it's set to 0, the prior information won't be used at all, and the tests reduce to ordinary F tests in one-way ANOVA; when it's set to Inf, the denominators of moderated F statistics are simply the prior variances, and these F statistics reduce to following a scaled chi-squared distribution. Other values of df.prior represent intermediate cases. To be noted, the number of prior degrees of freedom is automatically estimated for each mean-variance curve by a specifically designed statistical method (see also fitMeanVarCurve and setMeanVarCurve) and, by default, aovBioCond uses the estimation result to perform the tests. It's highly not recommended to specify df.prior explicitly when calling aovBioCond, unless you know what you are really doing. Besides, aovBioCond won't adjust variance ratio factors of the provided bioConds based on the specified number of prior degrees of freedom (see estimatePriorDf for a description of variance ratio factor).

Note also that, if df.prior is set to 0, of the bioCond objects in conds there must be at least one that contains two or more ChIP-seq samples. Otherwise, there is no way to measure the variance associated with each interval, and an error is raised.

Considering the practical significance of this analysis, which is to select genomic intervals with differential ChIP-seq signals between at least one pair of the biological conditions, those intervals not occupied by any of the bioCond objects in conds may be filtered out before making the selections. Thus, the statistical power of the tests could potentially be improved by re-adjusting *p*-values of the remaining intervals.

Value

aovBioCond returns an object of class c("aovBioCond","data.frame"), recording the test results for each genomic interval by each row. The data frame consists of the following variables:

conds.mean Mean signal intensity at the interval across biological conditions.

between.ms Between-condition mean of squares as from an ordinary one-way ANOVA.

within.ms Within-condition mean of squares as from an ordinary one-way ANOVA.

prior.var Prior variance deduced by reading from the mean-variance curve associated with the bioCond objects in conds.

posterior.var A weighted average of within.ms and prior.var, with the weights being proportional to their respective numbers of degrees of freedom.

mod.f Moderated *F* statistic, which is the ratio of between.ms to posterior.var.

pval *P*-value for the statistical significance of this moderated F statistic.

padj *P*-value adjusted for multiple testing with the "BH" method (see p.adjust), which controls false discovery rate.

Row names of the returned data frame inherit from those of `conds[[1]]$norm.signal`. Besides, several attributes are added to the returned object:

`bioCond.names` Names of the `bioCond` objects in `conds`.

`mean.var.curve` A function representing the mean-variance curve. It accepts a vector of mean signal intensities and returns the corresponding prior variances. Note that this function has incorporated the `min.var` argument.

`df` A length-4 vector giving the numbers of degrees of freedom of `between.ms`, `within.ms`, `prior.var` and `posterior.var`.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve for a set of `bioCond` objects; `setMeanVarCurve` for setting the mean-variance curve of a set of `bioConds`; `estimatePriorDf` for estimating number of prior degrees of freedom as well as adjusting variance ratio factors accordingly.

`plot.aovBioCond` for creating a plot to demonstrate an `aovBioCond` object; `diffTest` for calling differential intervals between two `bioCond` objects; `varTestBioCond` for calling hypervariable and invariant intervals across ChIP-seq samples contained in a `bioCond`.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Call differential genomic intervals among GM12890, GM12891 and GM12892
## cell lines.

# Perform MA normalization and construct bioConds to represent the cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Variations in ChIP-seq signals across biological replicates of a cell line
# are generally of a low level, and their relationship with the mean signal
# intensities is expected to be well modeled by the presumed parametric
# form.
```

```
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
plotMeanVarCurve(conds, subset = "occupied")

# Perform a moderated ANOVA on these cell lines.
res <- aovBioCond(conds)
head(res)
plot(res, padj = 1e-6)
```

---

bioCond *Create a* bioCond *Object to Group ChIP-seq Samples*

Description

bioCond creates an object which represents a biological condition, given a set of ChIP-seq samples belonging to the condition. Such objects, once created, can be supplied to fitMeanVarCurve to fit the mean-variance trend, and subsequently to diffTest for calling differential ChIP-seq signals between two conditions.

Usage

```
bioCond(
  norm.signal,
  occupancy = NULL,
  occupy.num = 1,
  name = "NA",
  weight = NULL,
  strMatrix = NULL,
  meta.info = NULL
)
```

Arguments

norm.signal	A matrix or data frame of normalized signal intensities, where each row should represent a genomic interval and each column a sample.
occupancy	A matrix or data frame of logical values with the same dimension as of norm.signal, marking the occupancy status of each interval in each sample. This argument is only used to derive the occupancy status of each interval in the biological condition. By default, each interval is considered to be occupied by each sample.
occupy.num	For each interval, the minimum number of samples occupying it required for the interval to be considered as occupied by the biological condition (see also "Details").
name	A character scalar specifying the name of the biological condition. Used only for demonstration.

`weight` A matrix or data frame specifying the relative precisions of signal intensities in `norm.signal`. Must have the same number of columns as `norm.signal`. A vector is interpreted as a matrix having a single row. Note that rows of `weight` are recycled if necessary. By default, the same weight is assigned to each measurement in `norm.signal`.

`strMatrix` An optional list of symmetric matrices specifying directly the structure matrix of each genomic interval. Elements of it are recycled if necessary. This argument, if set, overrides the `weight` argument. See "Details" and `setWeight` for information about structure matrix.

`meta.info` Optional extra information (e.g., genomic coordinates of intervals). If set, the supplied argument is stored in the `meta.info` field of returned `bioCond`, and shall never be used by other tools in `MAnorm2`.

Details

To call this function, one typically needs to first perform an MA normalization on raw read counts of ChIP-seq samples by using `normalize`.

The function will assign an indicator to each genomic interval (stored in the `occupancy` field of the returned object; see also "Value"), marking if the interval is occupied by this biological condition. The argument `occupy.num` controls the minimum number of samples that occupy an interval required for the interval to be determined as occupied by the condition. Notably, the occupancy states of genomic intervals may matter when fitting a mean-variance curve, as one may choose to use only the occupied intervals to fit the curve (see also `fitMeanVarCurve`).

For signal intensities of each genomic interval, `weight` specifies their relative precisions corresponding to different ChIP-seq samples in `norm.signal`. Intrinsically, the weights will be used to construct the structure matrices of the created `bioCond`. Alternatively, one can specify `strMatrix` directly when calling the function. To be noted, MAnorm2 uses a structure matrix to model the relative variances of signal intensities of a genomic interval as well as the correlations among them, by considering them to be associated with a covariance matrix proportional to the structure matrix. See `setWeight` for a detailed description of structure matrix.

Value

`bioCond` returns an object of `class` `"bioCond"`, representing the biological condition to which the supplied ChIP-seq samples belong.

In detail, an object of class `"bioCond"` is a list containing at least the following fields:

`name` Name of the biological condition.

`norm.signal` A matrix of normalized signal intensities of ChIP-seq samples belonging to the condition.

`occupancy` A logical vector marking the occupancy status of each genomic interval.

`meta.info` The `meta.info` argument (only present when it is supplied).

`strMatrix` Structure matrices associated with the genomic intervals.

`sample.mean` A vector of observed mean signal intensities of genomic intervals.

`sample.var` A vector recording the observed variance of signal intensities of each genomic interval.

Note that the sample.mean and sample.var fields are calculated by applying the GLS (generalized least squares) estimation to the signal intensities of each genomic interval, considering them as having a common mean and a covariance matrix proportional to the corresponding structure matrix. Specifically, the sample.var field times the corresponding structure matrices gives an unbiased estimate of the covariance matrix associated with each interval (see setWeight for details).

Besides, a fit.info field will be added to bioCond objects once you have fitted a mean-variance curve for them (see fitMeanVarCurve for details).

There are also other fields used internally for fitting the mean-variance trend and calling differential intervals between conditions. These fields should never be modified directly.

Warning

Among all the fields contained in a bioCond object, only name and meta.info are subject to free modifications; The strMatrix field must be modified through setWeight.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

normalize for performing an MA normalization on ChIP-seq samples; normalizeBySizeFactors for normalizing ChIP-seq samples based on their size factors; setWeight for modifying the structure matrices of a bioCond object.

normBioCond for performing an MA normalization on bioCond objects; normBioCondBySizeFactors for normalizing bioCond objects based on their size factors; cmbBioCond for combining a set of bioCond objects into a single one; MAplot.bioCond for creating an MA plot on two bioCond objects; summary.bioCond for summarizing a bioCond.

fitMeanVarCurve for modeling the mean-variance dependence across intervals in bioCond objects; diffTest for comparing two bioCond objects; aovBioCond for comparing multiple bioCond objects; varTestBioCond for calling hypervariable and invariant intervals across ChIP-seq samples contained in a bioCond.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Construct a bioCond object for the GM12891 cell line.

# Apply MA normalization to the ChIP-seq samples of GM12891.
norm <- normalize(H3K27Ac, 5:6, 10:11)

# Call the constructor and optionally attach some meta information to the
# resulting bioCond, such as the coordinates of genomic intervals.
GM12891 <- bioCond(norm[5:6], norm[10:11], name = "GM12891",
                   meta.info = norm[1:3])
```

```
# Alternatively, you may assign different weights to the replicate samples
# for estimating the mean signal intensities of genomic intervals in this
# cell line. Here the weight of the 2nd replicate is reduced to half the
# weight of the 1st one.
GM12891_2 <- bioCond(norm[5:6], norm[10:11], name = "GM12891",
                     weight = c(1, 0.5))

# Equivalently, you can achieve the same effect by setting the strMatrix
# parameter.
GM12891_3 <- bioCond(norm[5:6], norm[10:11], name = "GM12891",
                     strMatrix = list(diag(c(1, 2))))
```

`checkCountTable` *Check the Regularity of a Count Table*

Description

Check the Regularity of a Count Table

Usage

```
checkCountTable(counts)
```

Arguments

`counts`	A matrix consisting of read counts. Objects of other types are coerced to a matrix.

Value

The function raises an error once the regularity check process fails. It returns `NULL` otherwise.

`checkIndex` *Check the Validity of an Index Vector*

Description

Check the Validity of an Index Vector

Usage

```
checkIndex(index, ns, var.name)
```

Arguments

index	An index vector for subsetting columns of a data frame.
ns	A character vector of variable names in the data frame.
var.name	The index variable name. Simply used to generate potential error messages.

Value

The regularized index vector if it's valid. Otherwise, an error is raised.

---

cmbBioCond *Combine a Set of* bioCond *Objects into a Single* bioCond

---

Description

Given a list of bioCond objects, cmbBioCond combines them into a single bioCond, by treating each bioCond as an individual ChIP-seq sample. This function is primarily used to handle ChIP-seq samples associated with a hierarchical structure (see "Details" for an example).

Usage

```
cmbBioCond(
  conds,
  occupy.num = 1,
  name = "NA",
  weight = NULL,
  strMatrix = NULL,
  meta.info = NULL
)
```

Arguments

conds	A list of bioCond objects to be combined.
occupy.num	For each interval, the minimum number of bioConds occupying it required for the interval to be considered as occupied by the newly constructed bioCond.
name	Name of the constructed biological condition, used only for demonstrating a bioCond object.
weight	A matrix or data frame specifying the relative precisions of signal intensities of the constructed bioCond. Must have the same number of columns as the number of bioConds in conds. A vector is interpreted as a matrix having a single row. Note that rows of weight are recycled if necessary. By default, the same weight is assigned to each measurement in the constructed bioCond.
strMatrix	An optional list of symmetric matrices specifying directly the structure matrix of each genomic interval in the constructed bioCond. Elements of it are recycled if necessary. This argument, if set, overrides the weight argument. See bioCond and setWeight for a detailed description of structure matrix.

`meta.info` Optional extra information about the `bioCond` to be created. If set, the supplied argument is stored in the `meta.info` field of returned `bioCond`, and shall never be used by other tools in `MAnorm2`.

Details

Technically, `cmbBioCond` treats each `bioCond` object in `conds` as a ChIP-seq sample, taking the `sample.mean` and `occupancy` fields stored in each `bioCond` to represent its signal intensities and occupancy indicators, respectively. Then, by grouping these "samples", a new `bioCond` object is constructed following the exact routine as described in `bioCond`. See `bioCond` also for a description of the structure of a `bioCond` object.

Notably, ChIP-seq samples contained in these `bioCond` objects to be combined are supposed to have been normalized to the same level, so that these `bioConds` are comparable to each other. For this purpose, you may choose to normalize the ChIP-seq samples involved all together via `normalize`, or to normalize the `bioCond` objects to be combined via `normBioCond`.

`cmbBioCond` is primarily used to deal with ChIP-seq samples sorted into a hierarchical structure. For example, suppose ChIP-seq samples are available for multiple male and female individuals, where each individual is associated with several replicates. To call differential ChIP-seq signals between males and females, two `bioCond` objects representing these two conditions need to be created. One way to do that is to select one ChIP-seq sample as representative for each individual, and group male and female samples, respectively. Alternatively, to leverage all available ChIP-seq samples, a `bioCond` object could be constructed for each individual, consisting of the samples of him (her). Then, the `bioConds` of male and female can be separately created by grouping the corresponding individuals. See also "Examples" below.

Value

A `bioCond` object, created by combining all the supplied `bioCond` objects.

See Also

`bioCond` for creating a `bioCond` object from a set of ChIP-seq samples; `normalize` for performing an MA normalization on ChIP-seq samples; `normBioCond` for normalizing a set of `bioConds`; `setWeight` for modifying the structure matrices of a `bioCond` object.

`MAplot.bioCond` for creating an MA plot on two `bioCond` objects; `summary.bioCond` for summarizing a `bioCond`.

`fitMeanVarCurve` for modeling the mean-variance dependence across intervals in `bioCond` objects; `diffTest` for comparing two `bioCond` objects; `aovBioCond` for comparing multiple `bioCond` objects; `varTestBioCond` for calling hypervariable and invariant intervals across ChIP-seq samples contained in a `bioCond`.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Construct two bioConds comprised of the male and female individuals,
## respectively.
```

```
# First, normalize ChIP-seq samples separately for each individual (i.e.,
# cell line).
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)

# Then, construct separately a bioCond for each individual, and perform MA
# normalization on the resulting bioConds. Genomic intervals in sex
# chromosomes are not allowed to be common peak regions, since the
# individuals are of different genders.
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Finally, group individuals into bioConds based on their genders.
female <- cmbBioCond(conds[c(1, 3)], name = "female")
male <- cmbBioCond(conds[2], name = "male")
summary(female)
summary(male)
```

`diffTest` *Generic Differential Test*

Description

`diffTest` is a generic function used to perform a differential test (or multiple differential tests) between two R objects, usually of the same type. Described in this page is the method designed for comparing two `bioCond` objects. This method is typically used to call genomic intervals with differentially represented ChIP-seq signals between two biological conditions.

Usage

```
diffTest(x, y, ...)

## S3 method for class 'bioCond'
diffTest(x, y, min.var = 0, df.prior = NULL, ...)
```

Arguments

`x, y`	`x` is any R object for which a `diffTest` method has been defined. For the method for class `"bioCond"`, `x` and `y` are two `bioCond` objects to be compared. They must be associated with the same mean-variance curve (i.e., they must have the same `"mvcID"`; see also `fitMeanVarCurve`).
`...`	Arguments passed to specific methods or from other methods.

`min.var` Lower bound of variances read from the mean-variance curve. Any variance read from the curve less than `min.var` will be adjusted to this value. It's primarily used for safely getting the prior variances and taking into account the practical significance of a signal difference.

`df.prior` Number of prior degrees of freedom associated with the mean-variance curve. Must be non-negative. Can be set to `Inf` (see "Details"). By default, `diffTest` checks if `x` and `y` have the same `"df.prior"` component, and uses it as the number of prior degrees of freedom if yes (an error is raised otherwise).

Details

This method for calling differential genomic intervals between two `bioCond` objects adopts the modeling strategy implemented in `limma` (see "References"), except that each interval has its own prior variance, which is read from the mean-variance curve associated with the `bioConds`. Technically, the final estimate of variance for an individual interval is a weighted average between its prior and observed variances, with the weights being proportional to their respective numbers of degrees of freedom.

Two extreme values can be specified for the argument `df.prior` (number of degrees of freedom associated with the prior variances), representing two distinct cases: when it is set to `0`, the final variance estimate for an individual interval is simply deduced from the signal intensities observed in it, and the statistical test reduces to the ordinary two-sample t-test; when it is set to `Inf`, the final variance estimate is simply read from the mean-variance curve. Other values of `df.prior` represent intermediate cases. To be noted, the number of prior degrees of freedom is automatically estimated for each mean-variance curve by a specifically designed statistical method (see also `fitMeanVarCurve` and `setMeanVarCurve`) and, by default, `diffTest` uses the estimation result to perform the differential tests. It's highly not recommended to specify `df.prior` explicitly when calling `diffTest`, unless you know what you are really doing. Besides, `diffTest` won't adjust variance ratio factors of the two `bioConds` being compared based on the specified number of prior degrees of freedom (see `estimatePriorDf` for a description of variance ratio factor).

Note also that, if `df.prior` is set to `0`, of the two `bioCond` objects being compared there must be at least one that contains two or more samples. Otherwise, there is no way to measure the variance associated with each interval, and an error is raised.

Considering the practical significance of differential ChIP-seq signals, those genomic intervals not occupied by either of the conditions may be filtered out before selecting differential ones. Thus, the statistical power for detecting differential intervals could potentially be increased by re-adjusting *p*-values of the remaining intervals (see "Examples" below).

Value

This method returns an object of `class` `c("diffBioCond","data.frame")`, recording the test results for each genomic interval by each row. The data frame consists of the following variables:

`x.mean, y.mean` Mean signal intensities of the two conditions, respectively. `"x"` and `"y"` in the variable names are replaced by the corresponding actual condition names.

`Mval` Difference in mean signal intensity between the two conditions. An `Mval` of `1` indicates a twofold change in normalized read count.

`Mval.se` Standard error associated with the `Mval`.

`Mval.t` The ratio of `Mval` to `Mval.se`.

`pval` Two sided *p*-value for the statistical significance of this signal difference.

`padj` *P*-value adjusted for multiple testing with the `"BH"` method (see `p.adjust`), which controls false discovery rate.

Row names of the returned data frame inherit from those of `x$norm.signal`. Besides, an attribute named `"Mval.se.df"` is added to the returned object, which is a positive numeric giving the total number of degrees of freedom associated with the standard errors.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve for a set of `bioCond` objects; `setMeanVarCurve` for setting the mean-variance curve of a set of `bioConds`; `estimatePriorDf` for estimating number of prior degrees of freedom as well as adjusting variance ratio factors accordingly.

`MAplot.diffBioCond` for creating an MA plot on results of comparing two `bioCond` objects; `aovBioCond` for comparing multiple `bioCond` objects; `varTestBioCond` for calling hypervariable and invariant intervals across ChIP-seq samples contained in a `bioCond`.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Make a comparison between GM12891 and GM12892 cell lines.

# Perform MA normalization and construct bioConds to represent the two cell
# lines.
norm <- normalize(H3K27Ac, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Variations in ChIP-seq signals across biological replicates of a cell line
# are generally of a low level, and their relationship with the mean signal
# intensities is expected to be well modeled by the presumed parametric
# form.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
plotMeanVarCurve(conds, subset = "occupied")
```

```
# Perform differential tests between the two cell lines.
res1 <- diffTest(conds[[1]], conds[[2]])
head(res1)
MAplot(res1, padj = 0.001)
abline(h = 0, lwd = 2, lty = 5, col = "green3")

## Make a comparison between GM12891 and GM12892 cell lines using only their
## first replicates.

# Perform MA normalization and construct bioConds to represent the two cell
# lines.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
norm <- normalize(H3K27Ac, c(5, 7), c(10, 12),
                  common.peak.regions = autosome)
conds <- list(GM12891 = bioCond(norm[5], norm[10], name = "GM12891"),
              GM12892 = bioCond(norm[7], norm[12], name = "GM12892"))

# Construct a "blind" bioCond that treats the two samples as replicates and
# fit a mean-variance curve accordingly. Only common peak regions of the two
# samples are considered to be occupied by the "blind" bioCond, and only
# these regions are used for fitting the mean-variance curve. This setting
# is for capturing underlying non-differential intervals as accurately as
# possible and avoiding over-estimation of prior variances (i.e., variances
# read from a mean-variance curve).
conds$blind <- bioCond(norm[c(5, 7)], norm[c(10, 12)], occupy.num = 2,
                       name = "blind")
conds <- fitMeanVarCurve(conds, method = "parametric",
                         occupy.only = TRUE, init.coef = c(0.1, 10))
summary(conds[[1]])
summary(conds[[2]])
summary(conds[[3]])
plotMeanVarCurve(conds, subset = "occupied")

# Perform differential tests between the two cell lines.
res2 <- diffTest(conds[[1]], conds[[2]])
head(res2)
MAplot(res2, pval = 0.01)
abline(h = 0, lwd = 2, lty = 5, col = "green3")

# Inspect only the test results of the genomic intervals that are occupied
# by at least one of the two bioConds having been compared. Note the
# globally increased statistical power.
res3 <- res2[conds[[1]]$occupancy | conds[[2]]$occupancy, ]
res3$padj <- p.adjust(res3$pval, method = "BH")
boxplot(list(all = res2$padj, occupied = res3$padj), ylab = "Adj. p-value")

## Examine the consistency of results between the two differential analyses.

# Theoretically, t-statistics resulting from the two differential analyses
# are not directly comparable to each other, since they have different
# numbers of degrees of freedom. Here we map these t-statistics to the
# standard normal distribution in such a manner that the resulting
# z-statistics correspond to the same p-values as do the original
```

```
# t-statistics.
z1 <- qnorm(res1$pval / 2)
z1[res1$Mval > 0] <- -z1[res1$Mval > 0]
z2 <- qnorm(res2$pval / 2)
z2[res2$Mval > 0] <- -z2[res2$Mval > 0]

# Check the correlation between z-statistics from the two differential
# analyses.
cor(z1, z2)
cor(z1, z2, method = "sp")

## Make a comparison between the male and female genders by treating each
## individual (i.e., cell line) as a replicate.

# First perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Group individuals into bioConds based on their genders.
female <- cmbBioCond(conds[c(1, 3)], name = "female")
male <- cmbBioCond(conds[2], name = "male")

# The dependence of variance of ChIP-seq signal intensity across individuals
# on the mean signal intensity is not as regular as in the case for modeling
# biological replicates of cell lines. Better use the local regression to
# adaptively capture the mean-variance trend.
genders <- list(female = female, male = male)
genders <- fitMeanVarCurve(genders, method = "local", occupy.only = FALSE)
genders <- estimatePriorDf(genders, occupy.only = TRUE)
summary(genders$female)
summary(genders$male)
plotMeanVarCurve(genders, subset = "all")

# Perform differential tests between the two genders.
res <- diffTest(genders[[1]], genders[[2]])
head(res)
MAplot(res, pval = 0.01)
abline(h = 0, lwd = 2, lty = 5, col = "green3")

# Examine the distribution of p-values in Y chromosome.
hist(res$pval[H3K27Ac$chrom == "chrY"], col = "red",
     main = "P-values in Y chromosome")
```

distBioCond *Quantify the Distance between Each Pair of Samples in a* bioCond

Description

Given a bioCond object, distBioCond deduces, for each pair of samples contained in it, the average absolute difference in signal intensities of genomic intervals between them. Specifically, the function calculates a weighted minkowski (i.e., p-norm) distance between each pair of vectors of signal intensities, with the weights being inversely proportional to variances of individual intervals (see also "Details"). distBioCond returns a dist object recording the deduced average $|M|$ values. The object effectively quantifies the distance between each pair of samples and can be passed to hclust to perform a clustering analysis (see "Examples" below).

Usage

```
distBioCond(
  x,
  subset = NULL,
  method = c("prior", "posterior", "none"),
  min.var = 0,
  p = 2,
  diag = FALSE,
  upper = FALSE
)
```

Arguments

x	A bioCond object.
subset	An optional vector specifying a subset of genomic intervals to be used for deducing the distances between samples of x. In practice, you may want to use only the intervals associated with large variations across the samples to calculate the distances, as such intervals are most helpful for distinguishing between the samples (see varTestBioCond and "Examples" below).
method	A character string indicating the method to be used for calculating the variances of individual intervals. Must be one of "prior" (default), "posterior" and "none". Can be abbreviated. Note that the "none" method does not consider the mean-variance trend associated with x (see "Details").
min.var	Lower bound of variances read from the mean-variance curve associated with x. Any variance read from the curve less than min.var will be adjusted to this value. It's primarily used for safely reading positive values from the curve and taking into account the practical significance of a signal variation. Ignored if method is set to "none".
p	The power used to calculate the p-norm distance between each pair of samples (see "Details" for the specific formula). Any positive real could be specified, though setting p to a value other than 1 and 2 makes little sense. The default corresponds to the Euclidean distance.
diag, upper	Two arguments to be passed to as.dist.

Details

Variance of signal intensity varies considerably across genomic intervals, due to the heteroscedasticity inherent to count data as well as most of their transformations. On this account, separately scaling the signal intensities of each interval in a bioCond should lead to a more reasonable measure of distances between its samples. Suppose that X and Y are two vectors of signal intensities representing two samples of a bioCond and that xi, yi are their ith elements corresponding to the ith interval. distBioCond calculates the distance between X and Y as follows:

$$d(X,Y) = (sum(wi * |yi - xi|^p)/sum(wi))^(1/p)$$

where wi is the reciprocal of the scaled variance (see below) of interval i, and p defaults to 2. Since the weights of intervals are normalized to have a sum of 1, the resulting distance could be interpreted as an average absolute difference in signal intensities of intervals between the two samples.

Since there typically exists a clear mean-variance dependence across genomic intervals, distBioCond takes advantage of the mean-variance curve associated with the bioCond to improve estimates of variances of individual intervals. By default, prior variances, which are the ones read from the curve, are used to deduce the weights of intervals for calculating the distances. Alternatively, one can choose to use posterior variances of intervals by setting method to "posterior", which are weighted averages of prior and observed variances, with the weights being proportional to their respective numbers of degrees of freedom (see fitMeanVarCurve for details). Since the observed variances of intervals are associated with large uncertainty when the total number of samples is small, it is not recommended to use posterior variances in such cases. To be noted, if method is set to "none", distBioCond will consider all genomic intervals to be associated with a constant variance. In this case, neither the prior variance nor the observed variance of each interval is used to deduce its weight for calculating the distances. This method is particularly suited to bioCond objects that have gone through a variance-stabilizing transformation (see vstBioCond for details and "Examples" below) as well as bioConds whose structure matrices have been specifically designed (see below and "References" also).

Another point deserving special attention is that distBioCond has considered the possibility that genomic intervals in the supplied bioCond are associated with different structure matrices. In order to objectively compare signal variation levels between genomic intervals, distBioCond further scales the variance of each interval (deduced by using whichever method is selected) by multiplying it with the geometric mean of diagonal elements of the interval's structure matrix. See bioCond and setWeight for a detailed description of structure matrix.

Given a set of bioCond objects, distBioCond could also be used to quantify the distance between each pair of them, by first combining the bioConds into a single bioCond and fitting a mean-variance curve for it (see cmbBioCond and "Examples" below).

Value

A dist object quantifying the distance between each pair of samples of x.

References

Law, C.W., et al., *voom: Precision weights unlock linear model analysis tools for RNA-seq read counts.* Genome Biol, 2014. **15**(2): p. R29.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; cmbBioCond for combining a set of bioCond objects into a single one; hclust for performing a hierarchical clustering on a dist object; vstBioCond for applying a variance-stabilizing transformation to signal intensities of samples of a bioCond.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Cluster a set of ChIP-seq samples from different cell lines (i.e.,
## individuals).

# Perform MA normalization and construct a bioCond.
norm <- normalize(H3K27Ac, 4:8, 9:13)
cond <- bioCond(norm[4:8], norm[9:13], name = "all")

# Fit a mean-variance curve.
cond <- fitMeanVarCurve(list(cond), method = "local",
                        occupy.only = FALSE)[[1]]
plotMeanVarCurve(list(cond), subset = "all")

# Measure the distance between each pair of samples and accordingly perform
# a hierarchical clustering. Note that biological replicates of each cell
# line are clustered together.
d1 <- distBioCond(cond, method = "prior")
plot(hclust(d1, method = "average"), hang = -1)

# Measure the distances using only hypervariable genomic intervals. Note the
# change of scale of the distances.
res <- varTestBioCond(cond)
f <- res$fold.change > 1 & res$pval < 0.05
d2 <- distBioCond(cond, subset = f, method = "prior")
plot(hclust(d2, method = "average"), hang = -1)

# Apply a variance-stabilizing transformation and associate a constant
# function with the resulting bioCond as its mean-variance curve.
vst_cond <- vstBioCond(cond)
vst_cond <- setMeanVarCurve(list(vst_cond), function(x)
                            rep_len(1, length(x)), occupy.only = FALSE,
                            method = "constant prior")[[1]]
plotMeanVarCurve(list(vst_cond), subset = "all")

# Repeat the clustering analyses on the VSTed bioCond.
d3 <- distBioCond(vst_cond, method = "none")
plot(hclust(d3, method = "average"), hang = -1)
res <- varTestBioCond(vst_cond)
f <- res$fold.change > 1 & res$pval < 0.05
d4 <- distBioCond(vst_cond, subset = f, method = "none")
plot(hclust(d4, method = "average"), hang = -1)
```

```
## Cluster a set of individuals.

# Perform MA normalization and construct bioConds to represent individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
conds <- normBioCond(conds)

# Group the individuals into a single bioCond and fit a mean-variance curve
# for it.
cond <- cmbBioCond(conds, name = "all")
cond <- fitMeanVarCurve(list(cond), method = "local",
                        occupy.only = FALSE)[[1]]
plotMeanVarCurve(list(cond), subset = "all")

# Measure the distance between each pair of individuals and accordingly
# perform a hierarchical clustering. Note that GM12891 and GM12892 are
# actually a couple and they are clustered together.
d1 <- distBioCond(cond, method = "prior")
plot(hclust(d1, method = "average"), hang = -1)

# Measure the distances using only hypervariable genomic intervals. Note the
# change of scale of the distances.
res <- varTestBioCond(cond)
f <- res$fold.change > 1 & res$pval < 0.05
d2 <- distBioCond(cond, subset = f, method = "prior")
plot(hclust(d2, method = "average"), hang = -1)
```

`estimateD0` *Workhorse Function for Estimating Number of Prior Degrees of Freedom*

Description

`estimateD0` underlies other interface functions for assessing the goodness of fit of an unadjusted mean-variance curve (or a set of unadjusted mean-variance curves).

Usage

```
estimateD0(z, m)
```

Arguments

`z`	A list of which each element is a vector of FZ statistics corresponding to a `bioCond` object (see also "Details").
`m`	A vector of numbers of replicates in `bioCond` objects. Must correspond to `z` one by one in the same order.

Details

For each bioCond object with replicate samples, a vector of FZ statistics can be deduced from the unadjusted mean-variance curve associated with it. More specifically, for each genomic interval in a bioCond with replicate samples, its FZ statistic is defined to be $log(t_hat/v0)$, where t_hat is the observed variance of signal intensities of the interval, and $v0$ is the interval's prior variance read from the corresponding mean-variance curve.

Theoretically, each FZ statistic follows a scaled Fisher's Z distribution plus a constant (since the mean-variance curve is not adjusted yet), and we can use the sample variance (plus a constant) of the FZ statistics of each single bioCond to get an estimate of $[base]trigamma(d0/2)$, where $d0$ is the number of prior degrees of freedom.

The final estimate of $trigamma(d0/2)$ is a weighted mean of estimates across bioCond objects, with the weights being their respective numbers of genomic intervals minus 1 that are used to deduce the FZ statistics. This should be appropriate, as Fisher's Z distribution is roughly normal (see also "References"). The weighted mean is similar to the pooled sample variance in an ANOVA analysis.

Finally, an estimate of $d0$ can be obtained by taking the inverse of $trigamma$ function, which is achieved by applying Newton iteration to it. Note that $d0$ is considered to be infinite if the estimated $trigamma(d0/2)$ is less than or equal to 0.

Value

The estimated number of prior degrees of freedom. Note that the function returns NA if there are not sufficient genomic intervals for estimating it.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; estimatePriorDf for an interface to estimating the number of prior degrees of freedom on bioCond objects; varRatio for a description of variance ratio factor; scaleMeanVarCurve for estimating the variance ratio factor for adjusting a mean-variance curve (or a set of curves).

estimateD0Robust and scaleMeanVarCurveRobust for estimating number of prior degrees of freedom and variance ratio factor *in a robust manner*, respectively.

---

estimateD0Robust *Estimate Number of Prior Degrees of Freedom in a Robust Manner*

Description

estimateD0Robust underlies other interface functions for estimating the number of prior degrees of freedom associated with an unadjusted mean-variance curve (or a set of unadjusted mean-variance curves) *in a robust manner*.

Usage

```
estimateD0Robust(
  z,
  m,
  p_low = 0.01,
  p_up = 0.1,
  d0_low = 0.001,
  d0_up = 1e+06,
  eps = d0_low,
  nw = gauss.quad(128, kind = "legendre")
)
```

Arguments

z	A list of which each element is a vector of FZ statistics corresponding to a bioCond object (see also "Details").
m	A vector of numbers of replicates in bioCond objects. Must correspond to z one by one in the same order.
p_low, p_up	Lower- and upper-tail probabilities for Winsorizing the FZ statistics associated with each bioCond.
d0_low, d0_up	Positive reals specifying the lower and upper bounds of estimated $d0$ (i.e., number of prior degrees of freedom). Inf is *not* allowed. During the estimation process, if $d0$ is sure to be less than or equal to d0_low, it will be considered as 0, and if it is sure to be larger than or equal to d0_up, it will be considered as positive infinity.
eps	The required numeric precision for estimating $d0$.
nw	A list containing nodes and weights variables for calculating the definite integral of a function f over the interval [-1,1], which is approximated by sum(nw$weights * f(nw$nodes)). By default, a set of Gauss-Legendre nodes along with the corresponding weights calculated by gauss.quad is used.

Details

For each bioCond object with replicate samples, a vector of FZ statistics can be deduced from the unadjusted mean-variance curve associated with it. More specifically, for each genomic interval in a bioCond with replicate samples, its FZ statistic is defined to be $log(t_hat/v0)$, where t_hat is the observed variance of signal intensities of the interval, and $v0$ is the interval's prior variance read from the corresponding mean-variance curve.

Theoretically, each FZ statistic follows a scaled Fisher's Z distribution plus a constant (since the mean-variance curve is not adjusted yet), and we derive a robust estimation of $d0$ (i.e., number of prior degrees of freedom) by Winsorizing the FZ statistics of each bioCond and matching the resulting sample variance with the theoretical variance of the Winsorized distribution, which is calculated by using numerical integration (see also "References"). Since the theoretical variance has no compact forms regarding $d0$, the matching procedure is achieved by using the method of bisection.

Inspired by the ordinary (non-robust) routine for estimating $d0$, we derive the final estimate of $d0$ by separately applying the function $[base]trigamma(x/2)$ to the estimated $d0$ from each bioCond,

taking a weighted average across the results, and applying the inverse of the function (achieved by using Newton iteration). Here the weights are the numbers of genomic intervals (in the bioConds) minus 1 that are used to calculate FZ statistics.

Value

The estimated number of prior degrees of freedom. Note that the function returns NA if there are not sufficient genomic intervals for estimating it.

References

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; estimatePriorDfRobust for an interface to *robustly* estimating the number of prior degrees of freedom on bioCond objects; varRatio for a description of variance ratio factor; scaleMeanVarCurveRobust for *robustly* estimating the variance ratio factor for adjusting a mean-variance curve (or a set of curves).

estimateD0 and scaleMeanVarCurve for the ordinary (non-robust) routines for estimating number of prior degrees of freedom and variance ratio factor, respectively.

Examples

```
## Not run:
## Private functions involved.

# For generating random FZ statistics with outliers. Note that the argument
# scaling controls how extreme outliers are.
rFZ <- function(n, var.ratio, m, d0, p_low, p_up, scaling) {
    z <- list()
    p_low <- p_low * 0.9
    p_up <- p_up * 0.9
    for (i in 1:length(n)) {
        x <- rf(n[i], m[i] - 1, d0)
        q_low <- qf(p_low, m[i] - 1, d0, lower.tail = TRUE)
        q_up <- qf(p_up, m[i] - 1, d0, lower.tail = FALSE)
        f <- x < q_low
        x[f] <- x[f] / runif(sum(f), 1, scaling)
        f <- x > q_up
        x[f] <- x[f] * runif(sum(f), 1, scaling)
        z[[i]] <- log(var.ratio[i]) + log(x)
    }
    z
}

# Settings.
n <- c(30000, 40000)
var.ratio <- c(1.2, 2.5)
```

```
m <- c(2, 3)
d0 <- 17
p_low <- 0.01
p_up <- 0.1

# Compare estimation results from ordinary (non-robust) and robust routines.
# Case 1: no outliers.
set.seed(100)
scaling <- 1
z <- rFZ(n, var.ratio, m, d0, p_low, p_up, scaling)
res1 <- estimateD0(z, m)
res1
scaleMeanVarCurve(z[1], m[1], res1)
scaleMeanVarCurve(z[2], m[2], res1)
res2 <- estimateD0Robust(z, m, p_low, p_up)
res2
scaleMeanVarCurveRobust(z[1], m[1], res2, p_low, p_up)
scaleMeanVarCurveRobust(z[2], m[2], res2, p_low, p_up)

# Case 2: moderate outliers.
scaling <- 3
z <- rFZ(n, var.ratio, m, d0, p_low, p_up, scaling)
res1 <- estimateD0(z, m)
res1
scaleMeanVarCurve(z[1], m[1], res1)
scaleMeanVarCurve(z[2], m[2], res1)
res2 <- estimateD0Robust(z, m, p_low, p_up)
res2
scaleMeanVarCurveRobust(z[1], m[1], res2, p_low, p_up)
scaleMeanVarCurveRobust(z[2], m[2], res2, p_low, p_up)

# Case 3: extreme outliers.
scaling <- 10
z <- rFZ(n, var.ratio, m, d0, p_low, p_up, scaling)
res1 <- estimateD0(z, m)
res1
scaleMeanVarCurve(z[1], m[1], res1)
scaleMeanVarCurve(z[2], m[2], res1)
res2 <- estimateD0Robust(z, m, p_low, p_up)
res2
scaleMeanVarCurveRobust(z[1], m[1], res2, p_low, p_up)
scaleMeanVarCurveRobust(z[2], m[2], res2, p_low, p_up)

## End(Not run)
```

`estimatePriorDf` *Assess the Goodness of Fit of Mean-Variance Curves*

Description

Given a set of bioCond objects of which each has been associated with a mean-variance curve, estimatePriorDf derives a common number of prior degrees of freedom assessing the overall goodness of fit of the mean-variance curves and accordingly adjusts the variance ratio factor of each of the bioConds.

Usage

```
estimatePriorDf(
  conds,
  occupy.only = TRUE,
  return.d0 = FALSE,
  no.rep.rv = NULL,
  .call = TRUE
)
```

Arguments

conds	A list of bioCond objects, of which each has a fit.info field describing its mean-variance curve (see also fitMeanVarCurve).
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to estimate the number of prior degrees of freedom and adjust the variance ratio factors. Otherwise, all intervals are used (see also "Details").
return.d0	A logical scalar. If set to TRUE, the function simply returns the estimated number of prior degrees of freedom.
no.rep.rv	A positive real specifying the variance ratio factor of those bioConds without replicate samples, if any. By default, it's set to the geometric mean of variance ratio factors of the other bioConds.
.call	Never care about this argument.

Details

estimatePriorDf borrows part of the modeling strategy implemented in limma (see "References"). For each bioCond object, the predicted variances from its mean-variance curve serve as the prior variances associated with individual intervals. The common number of prior degrees of freedom of the supplied bioConds quantifies the confidence we have on the associated mean-variance curves. Intuitively, the closer the observed mean-variance points are to the curves, the more prior degrees of freedom there will be. See estimateD0 for technical details about the estimation of number of prior degrees of freedom.

According to the estimated number of prior degrees of freedom, estimatePriorDf separately adjusts the variance ratio factor of each bioCond. Intrinsically, this process is to scale the mean-variance curve of each bioCond so that it passes the "middle" of the observed mean-variance points. See scaleMeanVarCurve for technical details of scaling a mean-variance curve.

ChIP-seq signals located in non-occupied intervals result primarily from background noise, and are therefore associated with less data regularity than signals in occupied intervals. Involving non-occupied intervals in the estimation process may result in an under-estimated number of prior degrees of freedom. Thus, the recommended usage is to set occupy.only to TRUE (i.e., the default).

In most cases, the estimation of number of prior degrees of freedom is automatically handled when fitting or setting a mean-variance curve, and you don't need to call this function explicitly (see also `fitMeanVarCurve` and `setMeanVarCurve`). See "Examples" below for a practical application of this function. Note also that there is a *robust* version of this function that uses Winsorized statistics to protect the estimation procedure against potential outliers (see `estimatePriorDfRobust` for details).

Value

By default, `estimatePriorDf` returns the argument list of `bioCond` objects, with the estimated number of prior degrees of freedom substituted for the `"df.prior"` component of each of them. Besides, their `"ratio.var"` components have been adjusted accordingly, and an attribute named `"no.rep.rv"` is added to the list if it's ever been used as the variance ratio factor of the `bioConds` without replicate samples. A special case is that the estimated number of prior degrees of freedom is 0. In this case, `estimatePriorDf` won't adjust existing variance ratio factors, and you may want to use `setPriorDfVarRatio` to explicitly specify variance ratio factors.

If `return.d0` is set to `TRUE`, `estimatePriorDf` simply returns the estimated number of prior degrees of freedom.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve and using a `fit.info` field to characterize it; `estimatePriorDfRobust` for a *robust* version of `estimatePriorDf`; `setPriorDf` for setting the number of prior degrees of freedom and accordingly adjusting the variance ratio factors of a set of `bioConds`; `diffTest` for calling differential intervals between two `bioCond` objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit a mean-variance curve treating each gender as a biological condition,
## and each individual (i.e., cell line) a replicate.

# First perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
```

```
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Group individuals into bioConds based on their genders.
female <- cmbBioCond(conds[c(1, 3)], name = "female")
male <- cmbBioCond(conds[2], name = "male")

# The dependence of variance of ChIP-seq signal intensity across individuals
# on the mean signal intensity is typically not as regular as could be well
# modeled by an explicit parametric form. Better use the local regression to
# adaptively capture the mean-variance trend.
genders <- list(female = female, male = male)
genders1 <- fitMeanVarCurve(genders, method = "local", occupy.only = TRUE)
genders2 <- fitMeanVarCurve(genders, method = "local", occupy.only = FALSE)

# Suppose the local regression is performed using only the occupied genomic
# intervals as input. Good chances are that the extrapolation algorithm
# implemented in the regression method will produce over-estimated variances
# for the non-occupied intervals.
plotMeanVarCurve(genders1, subset = "all")
plotMeanVarCurve(genders2, subset = "all")
plotMeanVarCurve(genders1, subset = "non-occupied")
plotMeanVarCurve(genders2, subset = "non-occupied")

# On the other hand, applying the local regression on all genomic intervals
# may considerably reduce the estimated number of prior degrees of freedom
# associated with the fitted mean-variance curve, as ChIP-seq signals in the
# non-occupied intervals are generally of less data regularity compared with
# those in the occupied intervals.
summary(genders1$female)
summary(genders2$female)

# To split the difference, fit the mean-variance curve on all genomic
# intervals and re-estimate the number of prior degrees of freedom using
# only the occupied intervals, which is also the most recommended strategy
# in practice.
genders3 <- estimatePriorDf(genders2, occupy.only = TRUE)
plotMeanVarCurve(genders3, subset = "all")
plotMeanVarCurve(genders3, subset = "non-occupied")
summary(genders3$female)
```

`estimatePriorDfRobust` *Assess the Goodness of Fit of Mean-Variance Curves in a Robust Manner*

Description

Given a set of `bioCond` objects of which each has been associated with a mean-variance curve, `estimatePriorDfRobust` derives a common number of prior degrees of freedom assessing the

overall goodness of fit of the mean-variance curves and accordingly adjusts the variance ratio factor of each of the bioConds. Compared with estimatePriorDf, the underlying methods of estimatePriorDfRobust for parameter estimation are *robust* to outliers.

Usage

```
estimatePriorDfRobust(
  conds,
  occupy.only = TRUE,
  p_low = 0.01,
  p_up = 0.1,
  d0_low = 0.001,
  d0_up = 1e+06,
  eps = d0_low,
  nw = gauss.quad(128, kind = "legendre"),
  return.d0 = FALSE,
  no.rep.rv = NULL,
  .call = TRUE
)
```

Arguments

conds	A list of bioCond objects, of which each has a fit.info field describing its mean-variance curve (see also fitMeanVarCurve).
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to estimate the number of prior degrees of freedom and adjust the variance ratio factors. Otherwise, all intervals are used.
p_low, p_up	Lower- and upper-proportions of extreme values to be Winsorized (see "References"). Must be strictly between 0 and 0.5.
d0_low, d0_up	Positive reals specifying the lower and upper bounds of estimated $d0$ (i.e., number of prior degrees of freedom). Inf is *not* allowed. During the estimation process, if $d0$ is sure to be less than or equal to d0_low, it will be considered as 0, and if it is sure to be larger than or equal to d0_up, it will be considered as positive infinity.
eps	The required numeric precision for estimating $d0$.
nw	A list containing nodes and weights variables for calculating the definite integral of a function f over the interval [-1,1], which is approximated by sum(nw$weights * f(nw$nodes)). By default, a set of Gauss-Legendre nodes along with the corresponding weights calculated by gauss.quad is used.
return.d0	A logical scalar. If set to TRUE, the function simply returns the estimated $d0$.
no.rep.rv	A positive real specifying the variance ratio factor of those bioConds without replicate samples, if any. By default, it's set to the geometric mean of variance ratio factors of the other bioConds.
.call	Never care about this argument.

Details

The core function of `estimatePriorDfRobust` is very similar to that of `estimatePriorDf`, except that the former estimates the number of prior degrees of freedom and variance ratio factors *in a robust manner* (see also "References").

Unlike `estimatePriorDf`, you need to call explicitly `estimatePriorDfRobust` if you are intended to perform *robust* parameter estimation after associating a mean-variance curve with a set of `bioCond` objects (via `fitMeanVarCurve` for example; see "Examples" below).

Value

By default, `estimatePriorDfRobust` returns the argument list of `bioCond` objects, with the estimated number of prior degrees of freedom substituted for the `"df.prior"` component of each of them. Besides, their `"ratio.var"` components have been adjusted accordingly, and an attribute named `"no.rep.rv"` is added to the list if it's ever been used as the variance ratio factor of the `bioConds` without replicate samples. A special case is that the estimated number of prior degrees of freedom is 0. In this case, `estimatePriorDfRobust` won't adjust existing variance ratio factors, and you may want to use `setPriorDfVarRatio` to explicitly specify variance ratio factors.

If `return.d0` is set to `TRUE`, `estimatePriorDfRobust` simply returns the estimated number of prior degrees of freedom.

References

Tukey, J.W., *The future of data analysis.* The annals of mathematical statistics, 1962. **33**(1): p. 1-67.

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve and using a `fit.info` field to characterize it; `estimatePriorDf` for the ordinary (non-robust) version of `estimatePriorDfRobust`; `setPriorDfRobust` for setting the number of prior degrees of freedom and accordingly adjusting the variance ratio factors of a set of `bioConds` *in a robust manner*.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Estimate parameters regarding the associated mean-variance curve in a
## robust manner. Here we treat each cell line (i.e., individual) as a
## biological condition.

# Perform MA normalization and construct bioConds to represent cell lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
```

```
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Fit a mean-variance curve by using the parametric method.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)

# Estimate the associated number of prior degrees of freedom and variance
# ratio factors in a robust manner.
conds2 <- estimatePriorDfRobust(conds, occupy.only = TRUE)

# In this case, there is little difference in estimation results between the
# ordinary routine and the robust one.
sapply(conds, function(x) c(x$fit.info$df.prior, x$fit.info$ratio.var))
sapply(conds2, function(x) c(x$fit.info$df.prior, x$fit.info$ratio.var))
```

`estimateSizeFactors` *Estimate Size Factors of ChIP-seq Samples*

Description

Given a table of raw read counts from ChIP-seq experiments, `estimateSizeFactors` returns estimated size factors representing relative sequencing depths of the ChIP-seq samples.

Usage

```
estimateSizeFactors(counts, subset = NULL)
```

Arguments

`counts`	A matrix or data frame consisting of read counts. Each row represents an observation (typically a genomic interval) and each column a ChIP-seq sample. Objects of other types are coerced to a matrix.
`subset`	An optional vector specifying a subset of observations to be used in the estimation process.

Details

This function utilizes the median ratio strategy to deduce size factors (see "References" for details). It's primarily for being used by the MA normalization process to select an optimal baseline sample, and in most cases you don't need to call this function directly. It may help, however, when you want to specify the baseline sample by your own criterion.

Value

`estimateSizeFactors` returns a numeric vector specifying the size factors.

References

Anders, S. and W. Huber, *Differential expression analysis for sequence count data.* Genome Biol, 2010. **11**(10): p. R106.

See Also

`normalize` for the MA normalization process.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

# Use all the genomic intervals.
estimateSizeFactors(H3K27Ac[4:8])

# Use only the genomic intervals occupied by all the ChIP-seq samples.
estimateSizeFactors(H3K27Ac[4:8], subset = apply(H3K27Ac[9:13], 1, all))
```

---

estimateVarRatio *Estimate Relative Variance Ratio Factors of* bioCond *Objects*

Description

Given a set of `bioCond` objects assumed to be associated with the same mean-variance curve, `estimateVarRatio` robustly estimates their relative variance ratio factors, by selecting one of the `bioConds` as the base condition and comparing the others to it.

Usage

```
estimateVarRatio(
  conds,
  base.cond = NULL,
  subset = NULL,
  invariant = NULL,
  no.rep.rv = NULL
)
```

Arguments

`conds`	A list of `bioCond` objects.
`base.cond`	An optional positive integer or character name indexing the base `bioCond` in `conds`. Note that the base condition must contain replicate samples. By default, the base `bioCond` is automatically selected by measuring the variation levels of the `bioConds` (see "Details").

`subset` An optional vector specifying the subset of intervals to be used for measuring the variation levels. Defaults to the intervals occupied by all the `bioConds`. Ignored if `base.cond` is specified.

`invariant` An optional non-negative real specifying the upper bound of difference in mean signal intensity for a genomic interval to be treated as invariant between two `bioCond` objects. By default, intervals occupied by both `bioConds` are treated as invariant between them. Note that `estimateVarRatio` uses exactly the invariant intervals to compare the variance ratio factors of two `bioConds`.

`no.rep.rv` A positive real specifying the (relative) variance ratio factor of those `bioConds` without replicate samples, if any. By default, it's set to be the geometric mean of variance ratio factors of the other `bioConds`.

Details

Technically, `estimateVarRatio` uses 1 as the (relative) variance ratio factor of the base `bioCond`, and estimates the variance ratio factors of the other `bioConds` by separately comparing each of them to the base. Refer to `varRatio` for details about comparing the variance ratio factors of two `bioConds` by using their invariant genomic intervals.

If the base `bioCond` is not explicitly specified by users, `estimateVarRatio` will measure the variation level of each `bioCond` containing replicate samples. Technically, the variation levels are calculated by applying the median ratio strategy to the observed variances of the `bioConds`. This process is rather similar to the one for estimating size factors of ChIP-seq samples (see also `estimateSizeFactors`). After that, the `bioCond` whose variation level is closest to 1 is selected as the base (with the exception that, if there are only two `bioConds` that contain replicate samples, the function will always use the `bioCond` with the lower variation level as the base, for avoiding potential uncertainty in selection results due to limited numerical precision).

Value

A named vector of the estimated relative variance ratio factors, with the names being those of the corresponding `bioCond` objects. Besides, the following attributes are associated with the vector:

`var.level` Variation levels of the `bioCond` objects. Present only when the base `bioCond` is automatically selected by the function.

`base.cond` Name of the base `bioCond`.

`no.rep.rv` Variance ratio factor of the `bioConds` with no replicate samples. Present only when it's ever been used.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve for a set of `bioCond` objects; `varRatio` for a formal description of variance ratio factor.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Estimate the relative variance ratio factors of cell lines.

# Perform the MA normalization and construct bioConds to represent cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Automatically select the base bioCond.
estimateVarRatio(conds)

# Explicitly specify the base bioCond.
estimateVarRatio(conds, base.cond = "GM12891")
```

`estParamHyperChIP` *The Parameter Estimation Framework of HyperChIP*

Description

Given a `bioCond` object with which a mean-variance curve has been associated, `estParamHyperChIP` estimates the related parameters (i.e., the number of prior degrees of freedom and the variance ratio factor) by following the framework designed in HyperChIP.

Usage

```
estParamHyperChIP(
  cond,
  occupy.only = TRUE,
  prob = 0.1,
  subset = NULL,
  p_low = 0.01,
  p_up = 0.1,
  return.d0 = FALSE,
  .call = TRUE,
  ...
)
```

Arguments

cond	A bioCond object with which a mean-variance curve been associated (see also fitMeanVarCurve).
occupy.only	A logical scalar. If it is TRUE (default), only occupied genomic intervals are used for the parameter estimation process. Otherwise, all intervals are used.
prob	The proportion of the intervals with the lowest (observed) mean signal intensities that shall be used in the subsequent Winsorization procedure (see "Details").
subset	Alternatively, you can set this argument to a logical vector to directly specify the intervals to be used in the Winsorization procedure. This option overrides occupy.only and prob.
p_low, p_up	Lower- and upper-proportions of extreme values to be Winsorized. Must be strictly between 0 and 0.5.
return.d0	A logical scalar. If set to TRUE, the function simply returns the estimated number of prior degrees of freedom.
.call	Never care about this argument.
...	Further arguments to be passed to estimatePriorDfRobust.

Details

Technically, estParamHyperChIP first derives a lower quantile of the observed mean signal intensities in different genomic intervals based on the prob argument. It then selects the intervals whose mean intensities are less than or equal to the quantile. Finally, it applies the Winsorization technique to the selected intervals to finish the parameter estimation (see also "References"), by using the estimatePriorDfRobust function as the underlying engine.

estParamHyperChIP is primarily designed for coordinating with varTestBioCond to call hypervariable and lowly variable intervals across samples. See "Examples" for the workflow of a standard HyperChIP analysis.

Value

By default, estParamHyperChIP returns the argument bioCond object, whose "df.prior" and "ratio.var" components have been updated. If return.d0 is set to TRUE, it simply returns the estimated number of prior degrees of freedom.

References

Tukey, J.W., *The future of data analysis.* The annals of mathematical statistics, 1962. **33**(1): p. 1-67.

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve and using a fit.info field to characterize it; estimatePriorDfRobust for estimating the number of prior degrees of freedom and adjusting the variance ratio factors of a set of bioConds *in a robust manner*; varTestBioCond for calling hypervariable and invariant intervals across ChIP-seq samples contained in a bioCond.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Treat all the samples as independent and perform a HyperChIP analysis.

# Use a pseudo-reference profile as baseline in the MA normalization
# process.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
norm <- normalize(H3K27Ac, 4:8, 9:13, baseline = "pseudo-reference",
                  common.peak.regions = autosome)
plot(attr(norm, "MA.cor"), symbreaks = TRUE, margins = c(8, 8))

# Construct a bioCond.
cond <- bioCond(norm[4:8], norm[9:13], occupy.num = 1,
                name = "all")

# Fit a mean-variance curve by using local regression.
cond <- fitMeanVarCurve(list(cond), method = "local",
                        occupy.only = TRUE, args.lp = list(nn = 1))[[1]]
summary(cond)

# Apply the parameter estimation framework of HyperChIP.
cond <- estParamHyperChIP(cond)
summary(cond)

# Perform statistical tests and visualize the results.
res <- varTestBioCond(cond)
head(res)
hist(res$pval, breaks = 100, col = "red")
plot(res)
```

---

extendMeanVarCurve *Extend the Application Scope of a Mean-Variance Curve*

Description

extendMeanVarCurve associates the mean-variance curve of a bioCond object with a set of other bioConds. This function is called most often when ChIP-seq samples stored in some bioConds have a low data regularity (due to, for example, a bad data quality), and you don't want to include them for fitting a mean-variance curve (see "Examples" below and also fitMeanVarCurve).

Usage

```
extendMeanVarCurve(
  conds,
  base.cond,
  occupy.only = TRUE,
  no.rep.rv = NULL,
  invariant = NULL
)
```

Arguments

conds	A list of bioCond objects.
base.cond	An extra bioCond object, from which the mean-variance curve is obtained.
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to estimate variance ratio factors (see also "Details"). Otherwise, all intervals are used.
no.rep.rv	A positive real specifying the variance ratio factor of no-replicate conditions, if any. By default, it's set to be the variance ratio factor of base.cond.
invariant	An optional non-negative real specifying the upper bound of difference in mean signal intensity for a genomic interval to be treated as invariant between two conditions. By default, intervals occupied by both conditions are treated as invariant between them. Note that this argument is only used when the number of prior degrees of freedom of base.cond is 0 (see also "Details").

Details

Technically, extendMeanVarCurve associates the mean-variance curve of base.cond as well as its number of prior degrees of freedom to each bioCond object in conds. Then, for each bioCond in conds, its variance ratio factor is estimated accordingly (see estimatePriorDf for details). Note that, if the inherited number of prior degrees of freedom is 0, the regular routine for estimating variance ratio factors does not apply. In this case, extendMeanVarCurve utilizes an alternative strategy to estimate the variance ratio factor of each bioCond via comparing it with the base.cond (see varRatio for details).

As mentioned, the prior df of each bioCond in conds is inherited from base.cond. Now that there are new bioCond objects that are associated with the same mean-variance curve as is base.cond, you may want to re-assess its goodness of fit incorporating these new datasets. See "Examples" below for using estimatePriorDf to re-estimate the number of prior degrees of freedom.

Another scenario where extendMeanVarCurve could be useful is when each of two bioCond objects to be compared has only one ChIP-seq sample. To make it possible to estimate the variances of individual genomic intervals, a simple solution is to treat the two samples as if they were replicates. Thus, a mean-variance curve can be fitted accordingly and then be associated with the two bioCond objects. See "Examples" for a complete routine for calling differential intervals between two conditions with no replicate samples at all. Notably, this method is most suited when the two conditions being compared are close. Otherwise, the method may lead to an over-conserved *p*-value calculation.

Value

extendMeanVarCurve returns the argument list of bioCond objects, each of which has an added (updated) fit.info field constructed based on the mean-variance curve associated with base.cond.

Specifically, each returned bioCond inherits all the components of its fit.info field from base.cond except the calls and ratio.var (see fitMeanVarCurve for a detailed description of the structure of a fit.info field). All the returned bioConds will have a record of this function call, and their variance ratio factors are separately estimated.

Besides, an attribute named "no.rep.rv" will be added to the returned list if it's ever been used as the variance ratio factor of the bioConds without replicate samples.

Note

You must normalize the bioCond objects in conds together with the base.cond to the same level before invoking this extension process. See normalize and normBioCond for performing MA normalization on ChIP-seq samples and bioCond objects, respectively.

See Also

bioCond for creating a bioCond object from a set of ChIP-seq samples; fitMeanVarCurve for fitting a mean-variance curve; setMeanVarCurve for setting the mean-variance curve of a set of bioConds; plotMeanVarCurve for plotting a mean-variance curve.

estimatePriorDf for estimating number of prior degrees of freedom and the corresponding variance ratio factors; estimatePriorDfRobust for a *robust* version of estimatePriorDf; varRatio for comparing the variance ratio factors of two bioConds.

distBioCond for robustly measuring the distance between each pair of ChIP-seq samples of a bioCond by considering its mean-variance trend; vstBioCond for applying a variance-stabilizing transformation to signal intensities of samples in a bioCond.

diffTest for calling differential intervals between two bioCond objects; aovBioCond for calling differential intervals across multiple bioConds; varTestBioCond for calling hypervariable and invariant intervals across ChIP-seq samples contained in a bioCond.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit a mean-variance curve based on the GM12891 cell line and associate
## the resulting curve with the other two cell lines.

# Perform the MA normalization and construct bioConds to represent cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
```

```
conds <- normBioCond(conds, common.peak.regions = autosome)

# Fit a mean-variance curve using only the GM12891 bioCond.
conds[2] <- fitMeanVarCurve(conds[2], method = "parametric",
                            occupy.only = TRUE)
summary(conds[[2]])
plotMeanVarCurve(conds[2], subset = "occupied")

# Associate the resulting curve with the other two bioConds.
conds[c(1, 3)] <- extendMeanVarCurve(conds[c(1, 3)], conds[[2]],
                                     occupy.only = TRUE)
summary(conds[[1]])
summary(conds[[3]])
plotMeanVarCurve(conds[3], subset = "occupied")

# Re-estimate number of prior degrees of freedom using all the bioConds,
# though the estimation result doesn't change in this example. But note the
# change of variance ratio factor of the bioCond without replicates (i.e.,
# GM12890).
conds2 <- estimatePriorDf(conds, occupy.only = TRUE)
summary(conds2[[1]])

## Make a comparison between GM12891 and GM12892 cell lines using only their
## first replicates.

# Perform MA normalization and construct bioConds to represent the two cell
# lines.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
norm <- normalize(H3K27Ac, c(5, 7), c(10, 12),
                  common.peak.regions = autosome)
conds <- list(GM12891 = bioCond(norm[5], norm[10], name = "GM12891"),
              GM12892 = bioCond(norm[7], norm[12], name = "GM12892"))

# Construct a "blind" bioCond that treats the two samples as replicates and
# fit a mean-variance curve accordingly. Only common peak regions of the two
# samples are considered to be occupied by the "blind" bioCond, and only
# these intervals are used for fitting the mean-variance curve. This setting
# is for capturing underlying non-differential intervals as accurately as
# possible and avoiding over-estimation of prior variances (i.e., variances
# read from a mean-variance curve).
conds$blind <- bioCond(norm[c(5, 7)], norm[c(10, 12)], occupy.num = 2,
                       name = "blind")
conds[3] <- fitMeanVarCurve(conds[3], method = "parametric",
                            occupy.only = TRUE, init.coef = c(0.1, 10))
summary(conds[[3]])
plotMeanVarCurve(conds[3], subset = "occupied")

# Associate the resulting mean-variance curve with the two cell lines.
conds[1:2] <- extendMeanVarCurve(conds[1:2], conds[[3]])
summary(conds[[1]])
summary(conds[[2]])

# Perform differential tests between the two cell lines.
```

```
res <- diffTest(conds[[1]], conds[[2]])
head(res)
MAplot(res, pval = 0.01)
abline(h = 0, lwd = 2, lty = 5, col = "green3")
```

`fitMeanVarCurve` *Fit a Mean-Variance Curve*

Description

Given a set of `bioCond` objects, `fitMeanVarCurve` robustly fits a curve capturing the mean-variance dependence across the genomic intervals contained in them, by iteratively detecting outliers and removing them from a regression procedure.

Usage

```
fitMeanVarCurve(
  conds,
  ratio.var = estimateVarRatio(conds),
  method = c("parametric fit", "local regression"),
  occupy.only = TRUE,
  range.residual = c(1e-04, 15),
  max.iter = 50,
  init.coef = NULL,
  args.lp = list(nn = 0.7),
  args.locfit = list(),
  verbose = TRUE
)
```

Arguments

`conds`	A list of `bioCond` objects, of which at least one should contain replicate samples.
`ratio.var`	A vector giving the initial variance ratio factors of the `bioConds`. Elements are recycled if necessary. By default, it's estimated by calling `estimateVarRatio`. See also "Variance Ratio Factor" below.
`method`	A character string indicating the method to be used for fitting the curve. Either `"parametric fit"` (default) or `"local regression"`. Can be abbreviated.
`occupy.only`	A logical value. If set to `FALSE`, all the genomic intervals contained in the `bioConds` are used to fit the curve. By default, only the occupied intervals are used. See also "Methods for Fitting a Mean-Variance Curve" below.
`range.residual`	A length-two vector specifying the range of residuals of non-outliers.
`max.iter`	Maximum number of iteration times allowed during the fitting procedure.

`init.coef` An optional length-two vector specifying the initial coefficients for applying the parametric fitting scheme. Only used when `method` is `"parametric fit"`.

In practice, chances are that `init.coef` is strictly required for the fitting process to go smoothly, as the underlying algorithm may fail to deduce a proper setting of initial coefficients (see "Examples" below). In this case, try setting `init.coef` to `c(0.1,10)`, which is expected to suit most practical datasets.

`args.lp` A named list of extra arguments to `lp`. Only used when `method` is set to `"local regression"`. Note the default value (see "Methods for Fitting a Mean-Variance Curve" below for an explanation).

`args.locfit` A named list of extra arguments to `locfit`. Only used when `method` is set to `"local regression"`. Note that, due to the internal implementation, the argument `subset` to `locfit` mustn't be specified in it.

`verbose` Whether to print processing messages during fitting the mean-variance curve?

Details

This function performs a regression of the variance of ChIP-seq signal intensity across replicate samples, using the mean intensity as the only predictor. Each genomic interval contained in each of the supplied `bioCond`s that consists of two or more ChIP-seq samples serves as an observation for the regression (the sample mean and sample variance of the interval's signal intensities in the `bioCond` are used as the predictor value and response, respectively).

Note that `bioCond` objects must be normalized to the same level before a mean-variance curve could be fitted for them. You can choose to either normalize the involved ChIP-seq samples all together (see `normalize`) or perform the normalization at the level of `bioCond` objects (see `normBioCond` and also "Examples" below).

Value

`fitMeanVarCurve` returns the argument list of `bioCond` objects, each of which has an added (updated) `fit.info` field describing its mean-variance dependence. The field is itself a list consisting of the following components:

`calls` The two function calls for associating a mean variance curve with this `bioCond` and estimating the related parameters, respectively. The latter is only present if you have made an explicit call to some function (e.g., `estimatePriorDf`) for performing the parameter estimation.

`method` Method used for fitting the mean-variance curve.

`predict` A function representing the fitted curve, which accepts a vector of means and returns the predicted variances.

`mvcID` ID of the fitted mean-variance curve.

`df.prior` Number of prior degrees of freedom assessing the goodness of fit of the mean-variance curve.

`ratio.var` Variance ratio factor of this `bioCond`.

Each `bioCond` object in the returned list has the same values of all these components but the `ratio.var`.

mvcID is used to label each fitted mean-variance curve. Each call to fitMeanVarCurve results in a unique ID. Thus we assert that different bioCond objects are associated with the same mean-variance curve if and only if they have the same mvcID. This is useful if you are to call differential intervals between two conditions via diffTest, which requires the two bioCond objects being compared are associated with the same mean-variance curve.

Besides, if there exist bioCond objects that contain only one ChIP-seq sample, an attribute named "no.rep.rv" will be added to the returned list, recording the variance ratio factor of no-replicate conditions. Note that the method for estimating the variance ratio factor of no-replicate conditions is specifically designed (see estimatePriorDf for details).

Variance Ratio Factor

fitMeanVarCurve applies a regression process to the observed means and variances of signal intensities of genomic intervals. The regression result serves as a model capturing the mean-variance trend across intervals. Notably, each genomic interval in each bioCond object that contains replicate samples serves as an observation point for the regression.

Variance ratio factor is designed to account for the global difference in variation level of signal intensities between conditions. Each bioCond has its own variance ratio factor, and method has been developed to robustly estimate the relative (scaled) variance ratio factors of a given set of bioConds (see estimateVarRatio for details). Technically, observed variances from each bioCond are scaled based on the corresponding (relative) variance ratio factor, so that the scaled variances from different bioConds are comparable to each other. Finally, the scaled variances from all the provided bioConds are pooled together constituting the vector of responses for the regression process. Note that the variance ratio factors will be adjusted afterwards, according to the fitted mean-variance curve and its goodness of fit (see "Assessing Goodness of Fit" below).

Methods for Fitting a Mean-Variance Curve

There are currently two candidate methods for performing the regression: "parametric fit" (default) and "local regression". Besides, the argument occupy.only controls whether to use all genomic intervals or only the occupied ones for the regression process.

Typically, ChIP-seq signal intensities at non-occupied intervals are much lower than those at occupied ones. Accordingly, variation levels of the former are significantly higher than the latter (provided that a log transformation has been applied to raw read counts before performing the normalization, which is the default setting of normalize). This is because, for the genomic intervals having a low-level abundance of ChIP-seq reads, only a little fluctuation of read count could give rise to a dramatic fold change. If a mean-variance scatter plot is drawn mapping all genomic intervals to a plane, the points corresponding to non-occupied intervals will be largely separated from those of occupied intervals.

In practice, the ChIP-seq signals located in non-occupied intervals result primarily from background noise and therefore have much lower signal-to-noise ratios than those in occupied intervals. As a result, signals observed in the two types of intervals almost always have distinct data characteristics from one another. In particular, the mean-variance dependence associated with non-occupied intervals is not as regular as observed from occupied intervals. In light of these observations, the recommended setting of occupy.only may be different across calls of fitMeanVarCurve depending on the exact method chosen for performing the regression. See the following for details.

For the method of "parametric fit", it adopts the parametric form of $var = c1 + c2/(2^{m}ean)$, where $c1$ and $c2$ are coefficients to be estimated. More specifically, it fits a gamma-family generalized linear model with the identity link. The form is deduced by assuming a quadratic mean-variance relationship for raw read counts and applying the delta method to log2 transformation (see also "References"). When using this method, one typically sets occupy.only to TRUE (the default). Otherwise, the GLM fitting procedure may fail to estimate the coefficients, or the estimation results may be significantly biased towards the characteristics of ChIP-seq signals at non-occupied intervals (which is undesired since these signals are mostly background noises). Note also that applying this method is most recommended when ChIP-seq samples within each single bioCond are associated with a low level of signal variation (e.g., when these samples are biological replicates of a cell line; see also "Examples" below), since in such cases ChIP-seq data should be of high regularity and, thus, the parametric form could be safely expected to hold. Moreover, as the variation level across ChIP-seq samples increases, the possibility becomes higher that the GLM fitting procedure fails.

For the method of "local regression", it directly passes the observed means and scaled variances to the locfit function, specifying the family to be "gamma". When using this method, setting occupy.only to TRUE almost certainly leads to an exaggerated variance prediction for small signal intensities (due to the underlying algorithm for extrapolation) and, thus, a reduction in statistical power for detecting differential intervals between conditions. On the other hand, involving non-occupied intervals in the fitting process might result in an underestimated number of prior degrees of freedom (see "Assessing Goodness of Fit" below). This is because the ChIP-seq signals located in non-occupied intervals generally have low signal-to-noise ratios, and are therefore associated with less data regularity than the signals in occupied intervals. One way to compensate that is to re-estimate the prior df using only the occupied intervals after fitting the mean-variance curve (see estimatePriorDf and "Examples" below), which is also the most recommended strategy for employing a local regression. Note also that smoothness of the resulting curve could be adjusted by modifying the nn variable in args.lp (see also lp). By default, nn=0.7 is adopted, which is also the default setting of lp at the time of developing this package.

Iteration Scheme for a Robust Regression

Whichever method is selected, fitMeanVarCurve adopts an iteration scheme to enhance the robustness of fitting the mean-variance trend. More specifically, it iteratively detects and removes outliers from a regression procedure. The process converges as soon as the set of outliers fixes. Residual of each observation is calculated as the ratio of its observed variance to the fitted one, and those observations with a residual falling outside range.residual shall be considered as outliers. The default value of range.residual works well for chi-squared distributions with a broad range of numbers of degrees of freedom (see also "References").

Assessing Goodness of Fit

Each fitted mean-variance curve is associated with a quantity assessing its goodness of fit, which is the number of prior degrees of freedom. Roughly speaking, the closer the observed mean-variance points are to the curve, the larger the resulting prior df of the curve, and we get more confidence in the curve. To be noted, the initial variance ratio factors for scaling the sample variances from different bioCond objects will be adjusted according to the estimated prior df (based on the underlying distributional theory). These adjusted variance ratio factors are exactly the ones stored in the returned bioCond objects. See estimatePriorDf for details about estimating prior df and accordingly adjusting variance ratio factors. Note also that fitMeanVarCurve uses exactly the set

of intervals that are utilized for fitting the mean-variance curve to estimate the prior df and adjust the variance ratio factors (the set is controlled by the argument `occupy.only`; see also "Methods for Fitting a Mean-Variance Curve" above).

Prior df is primarily used for the following differential analysis. We call a variance read from a mean-variance curve a prior one. In cases where you use `diffTest` to call differential intervals between two `bioConds`, the final variance estimate associated with each individual interval is obtained by averaging its observed and prior variances, weighted by their respective numbers of degrees of freedom.

Extending the Application Scope of a Mean-Variance Curve

With a set of `bioCond` objects at hand, you might want to use only part of them to fit the mean-variance curve. For example, suppose ChIP-seq samples stored in some `bioCond` objects are associated with a low data regularity (due to, e.g., bad sample qualities), and you don't want to include these samples when fitting the curve. One way to work around it is to exclude the `bioCond` objects from the fitting process, extend the application scope of the fitted curve (via `extendMeanVarCurve`) so that it applies to the excluded `bioConds` as well, and (optionally) re-assess the overall goodness of fit via `estimatePriorDf` (see also the "Examples" given for `extendMeanVarCurve`).

There is another scenario where extending a mean-variance curve could be useful. In practice, chances are that only one ChIP-seq sample is available for each of two conditions to be compared. To make the analysis possible, one way is to treat the two samples as replicates and fit a mean-variance curve accordingly. The fitted curve can then be associated with the two conditions each containing a single sample (via `extendMeanVarCurve`), and differential intervals between them can be subsequently called following a regular routine (see "Examples" provided in `extendMeanVarCurve`). To be noted, this method is most suited when the two conditions being compared are close. Otherwise, the method may lead to an over-conserved *p*-value calculation.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Anders, S. and W. Huber, *Differential expression analysis for sequence count data.* Genome Biol, 2010. **11**(10): p. R106.

Law, C.W., et al., *voom: Precision weights unlock linear model analysis tools for RNA-seq read counts.* Genome Biol, 2014. **15**(2): p. R29.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object from a set of ChIP-seq samples; `normalize` for performing an MA normalization on ChIP-seq samples; `normalizeBySizeFactors` for normalizing ChIP-seq samples based on their size factors; `normBioCond` for performing an MA normalization on `bioCond` objects; `normBioCondBySizeFactors` for normalizing `bioCond` objects based on their size factors.

`estimateVarRatio` for estimating the relative variance ratio factors of a set of `bioConds`; `varRatio` for a formal description of variance ratio factor; `estimatePriorDf` for estimating the number of

prior degrees of freedom as well as adjusting variance ratio factors accordingly; estimatePriorDfRobust for a *robust* version of estimatePriorDf.

setMeanVarCurve for setting the mean-variance curve of a set of bioCond objects; extendMeanVarCurve for extending the application scope of a fitted mean-variance curve to the bioConds not used to fit it; plotMeanVarCurve for plotting a mean-variance curve.

distBioCond for robustly measuring the distances between ChIP-seq samples in a bioCond by considering its mean-variance trend; vstBioCond for applying a variance-stabilizing transformation to signal intensities of samples in a bioCond.

diffTest for calling differential intervals between two bioCond objects; aovBioCond for calling differential intervals across multiple bioConds; varTestBioCond for calling hypervariable and invariant intervals across ChIP-seq samples contained in a bioCond.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit a mean-variance curve treating each cell line (i.e., individual) as a
## biological condition.

# Perform the MA normalization and construct bioConds to represent cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Variations in ChIP-seq signals across biological replicates of a cell line
# are generally of a low level, and their relationship with the mean signal
# intensities is expected to be well modeled by the presumed parametric
# form.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
plotMeanVarCurve(conds, subset = "occupied")

## Not run:
# Sometimes the parametric fitting algorithm cannot automatically deduce
# proper starting values for estimating the coefficients.
fitMeanVarCurve(conds[3], method = "parametric", occupy.only = TRUE)

## End(Not run)

# In such cases, explicitly specify the initial values of the coefficients.
fitMeanVarCurve(conds[3], method = "parametric", occupy.only = TRUE,
                init.coef = c(0.1, 10))

## Fit a mean-variance curve treating each gender as a biological condition,
```

```
## and each individual a replicate.

# Group individuals into bioConds based on their genders.
female <- cmbBioCond(conds[c(1, 3)], name = "female")
male <- cmbBioCond(conds[2], name = "male")

# The dependence of variance of ChIP-seq signal intensity across individuals
# on the mean signal intensity is not as regular as in the case for modeling
# biological replicates of cell lines. Better use the local regression to
# adaptively capture the mean-variance trend.
genders <- list(female = female, male = male)
genders1 <- fitMeanVarCurve(genders, method = "local", occupy.only = TRUE)
genders2 <- fitMeanVarCurve(genders, method = "local", occupy.only = FALSE)

# Suppose the local regression is performed using only the occupied genomic
# intervals as input. Good chances are that the extrapolation algorithm
# implemented in the regression method will produce over-estimated variances
# for the non-occupied intervals.
plotMeanVarCurve(genders1, subset = "all")
plotMeanVarCurve(genders2, subset = "all")
plotMeanVarCurve(genders1, subset = "non-occupied")
plotMeanVarCurve(genders2, subset = "non-occupied")

# On the other hand, applying the local regression on all genomic intervals
# may considerably reduce the estimated number of prior degrees of freedom
# associated with the fitted mean-variance curve, as ChIP-seq signals in the
# non-occupied intervals are generally of less data regularity compared with
# those in the occupied intervals.
summary(genders1$female)
summary(genders2$female)

# To split the difference, fit the mean-variance curve on all genomic
# intervals and re-estimate the number of prior degrees of freedom using
# only the occupied intervals, which is also the most recommended strategy
# in practice.
genders3 <- estimatePriorDf(genders2, occupy.only = TRUE)
plotMeanVarCurve(genders3, subset = "all")
plotMeanVarCurve(genders3, subset = "non-occupied")
summary(genders3$female)
```

H3K27Ac *ChIP-seq Samples for H3K27Ac in Human Lymphoblastoid Cell Lines*

Description

Benefiting from the associated ChIP-seq samples, this dataset profiles H3K27Ac levels along the whole genome for multiple human lymphoblastoid cell lines, each derived from a separate person. Specifically, a set of genomic intervals of around the same size (2 kb) has been systematically selected to thoroughly cover the part of the genome that is enriched with reads in at least one of

the ChIP-seq samples. And for each of these intervals, this dataset records its raw read count and enrichment status in each of the samples.

Usage

```
H3K27Ac
```

Format

`H3K27Ac` is a data frame that records the features of 73,828 non-overlapping genomic intervals regarding the H3K27Ac ChIP-seq signals in multiple human lymphoblastoid cell lines. It contains the following variables:

`chrom, start, end` Genomic coordinate of each interval. Note that these coordinates are 0-based and correspond to the hg19 genome assembly.

`cellLine_H3K27Ac_num.read_cnt` Each variable whose name is of this form records the number of reads from a ChIP-seq sample that fall within each genomic interval. For example, `GM12891_H3K27Ac_2.read_cnt` corresponds to the 2nd biological replicate of a ChIP-seq experiment that targets H3K27Ac in a cell line named GM12891.

`cellLine_H3K27Ac_num.occupancy` Each variable whose name is of this form records the enrichment status of each genomic interval in a ChIP-seq sample. An enrichment status of 1 indicates that the interval is enriched with reads in the sample; an enrichment status of 0 indicates otherwise. In practice, enrichment status of a genomic interval in a certain ChIP-seq sample could be determined by its overlap with the peaks (see "References" below) of the sample. Note also that variables of this class correspond to the variables of raw read counts one by one.

Each cell line derives from a separate individual of the Caucasian population. Use `attr(H3K27Ac,"metaInfo")` to get a data frame that records meta information about the involved individuals.

Source

Raw sequencing data were obtained from Kasowski et al., 2013 (see "References" below). Adapters and low-sequencing-quality bases were trimmed from 3' ends of reads using `trim_galore`. The resulting reads were then aligned to the hg19 reference genome by `bowtie`. `MACS` was utilized to call peaks for each ChIP-seq sample.

Finally, `MAnorm2_utils` was exploited to integrate the alignment results as well as peaks of ChIP-seq samples into this regular table. `MAnorm2_utils` is specifically designed to create input tables of `MAnorm2`. See the home page of MAnorm2_utils for more information about it. It has also been uploaded to the PyPI repository as a Python package.

References

Zhang, Y., et al., *Model-based analysis of ChIP-Seq (MACS).* Genome Biol, 2008. **9**(9): p. R137.

Kasowski, M., et al., *Extensive variation in chromatin states across humans.* Science, 2013. **342**(6159): p. 750-2.

intervalMeans *Deduce the Sample Mean Signal Intensity*

Description

Given a matrix of normalized signal intensities and the inverse of the corresponding structure matrices, `intervalMeans` returns the sample mean signal intensity of each genomic interval.

Usage

```
intervalMeans(x, inv.strMatrix)
```

Arguments

`x`	A matrix of normalized signal intensities, where each row represents an interval and each column a sample.
`inv.strMatrix`	A list of inversed structure matrices corresponding to the intervals. Elements of it are recycled if necessary.

Value

A numeric vector of the sample mean signal intensities.

See Also

`bioCond` for creating an R object representing a biological condition, and `setWeight` for modifying the structure matrices of such an object.

intervalVars *Sample Variance of Replicated Signal Intensities*

Description

Given a matrix of normalized signal intensities and the inverse of the corresponding structure matrices, `intervalVars` returns the sample variance of signal intensities of each genomic interval.

Usage

```
intervalVars(x, inv.strMatrix)
```

Arguments

`x`	A matrix of normalized signal intensities, where each row represents an interval and each column a sample.
`inv.strMatrix`	A list of inversed structure matrices corresponding to the intervals. Elements of it are recycled if necessary.

Value

A numeric vector of the sample variances.

Note

For the ith interval, $ti * Si$ is the covariance matrix of the signal intensities of the interval, where ti is a scalar quantifying the variation level of these signal intensities (under this biological condition), and Si is the interval's structure matrix (under this biological condition). `intervalVars` returns exactly the sample estimate of each ti.

See Also

`bioCond` for creating an R object representing a biological condition, and `setWeight` for modifying the structure matrices of such an object.

`inv.trigamma` *Inversion of Trigamma Function*

Description

`inv.trigamma` implements the Newton iteration for solving, given `x`, the equation for `y`: `trigamma(y) = x`. See appendix of the `limma` paper (see "References") for a theoretical deduction of the method.

Usage

```
inv.trigamma(x, eps = 1e-08)
```

Arguments

`x`	A positive numeric scalar.
`eps`	The required precision of the solution.

Value

The solution, which is also a positive numeric scalar.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

See Also

`trigamma` for the trigamma function.

Examples

```
x <- trigamma(1:6)
vapply(x, inv.trigamma, numeric(1))
```

`isSymPosDef` *Is a Real Matrix Symmetric and Positive Definite?*

Description

`isSymPosDef` checks if a real matrix is symmetric and positive definite.

Usage

```
isSymPosDef(x, ...)
```

Arguments

`x`	A real matrix.
`...`	Further arguments to `isSymmetric` for deciding on matrix symmetry.

Value

`TRUE` if `x` is both symmetric and positive definite. `FALSE` otherwise.

See Also

`isSymmetric` for testing if a matrix is symmetric.

Examples

```
x <- matrix(c(1, 0.5, 0.5, 2), nrow = 2)
isSymPosDef(x)

# Not positive definite.
x <- matrix(c(1, 0.5, 0.5, 0.2), nrow = 2)
isSymPosDef(x)
```

MA.pcc *Deduce Pearson Correlation Coefficient between M & A Values*

Description

Deduce Pearson Correlation Coefficient between M & A Values

Usage

```
MA.pcc(x, y)
```

Arguments

x, y	Two numeric vectors representing the signal intensities of two samples.

Value

Safely deduced PCC between (x + y) and (y -x).

Examples

```
## Not run:
## Private functions involved.

MA.pcc(1:4, 1:4 + c(1, 2, 4, 9))

# The robustness.
MA.pcc(1, 0)
MA.pcc(1:4, 2:5)

## End(Not run)
```

MAnorm2 *MAnorm2: a Package for Normalizing and Comparing ChIP-seq Samples*

Description

MAnorm2 provides a robust method for normalizing ChIP-seq signals across individual samples or groups of samples. It also designs a self-contained system of statistical models for calling differential ChIP-seq signals between two or more biological conditions as well as for calling hypervariable ChIP-seq signals across samples.

Details

For a typical differential analysis between two biological conditions starting with raw read counts, the standard workflow is to sequentially call normalize, bioCond, normBioCond, fitMeanVarCurve, and diffTest (see the following sections for a rough description of each of these functions). Examples given for diffTest provide specific demonstrations. MAnorm2 is also capable of calling differential ChIP-seq signals across multiple biological conditions. See the section below titled "Comparing ChIP-seq Signals across Multiple Conditions".

For a hypervariable ChIP-seq analysis starting with raw read counts, the standard workflow is to sequentially call normalize, bioCond, fitMeanVarCurve, estParamHyperChIP, and varTestBioCond. Examples given for estParamHyperChIP provide a specific demonstration.

The following sections classify the majority of MAnorm2 functions into different utilities. Basically, these sections also represent the order in which the functions are supposed to be called for a differential/hypervariable analysis. For a complete list of MAnorm2 functions, use `library(help = "MAnorm2")`.

Normalizing ChIP-seq Signals across Individual Samples

normalize Perform MA Normalization on a Set of ChIP-seq Samples

normalizeBySizeFactors Normalize ChIP-seq Samples by Their Size Factors

estimateSizeFactors Estimate Size Factors of ChIP-seq Samples

MAplot.default Create an MA Plot on Two Individual ChIP-seq Samples

Creating bioCond Objects to Represent Biological Conditions

bioCond Create a bioCond Object to Group ChIP-seq Samples

setWeight Set the Weights of Signal Intensities Contained in a bioCond

normBioCond Perform MA Normalization on a Set of bioCond Objects

normBioCondBySizeFactors Normalize bioCond Objects by Their Size Factors

cmbBioCond Combine a Set of bioCond Objects into a Single bioCond

MAplot.bioCond Create an MA Plot on Two bioCond Objects

summary.bioCond Summarize a bioCond Object

Modeling Mean-Variance Trend

fitMeanVarCurve Fit a Mean-Variance Curve

setMeanVarCurve Set the Mean-Variance Curve of a Set of bioCond Objects

extendMeanVarCurve Extend the Application Scope of a Mean-Variance Curve

plotMeanVarCurve Plot a Mean-Variance Curve

plotMVC Plot a Mean-Variance Curve on a Single bioCond Object

estimateVarRatio Estimate Relative Variance Ratio Factors of bioCond Objects

varRatio Compare Variance Ratio Factors of Two bioCond Objects

distBioCond Quantify the Distance between Each Pair of Samples in a bioCond

vstBioCond Apply a Variance-Stabilizing Transformation to a bioCond

Assessing the Goodness of Fit of Mean-Variance Curves

`estimatePriorDf` Assess the Goodness of Fit of Mean-Variance Curves

`estimatePriorDfRobust` Assess the Goodness of Fit of Mean-Variance Curves in a Robust Manner

`setPriorDf` Set the Number of Prior Degrees of Freedom of Mean-Variance Curves

`setPriorDfRobust` The Robust Counterpart of `setPriorDf`

`setPriorDfVarRatio` Set the Number of Prior Degrees of Freedom and Variance Ratio Factors

`estParamHyperChIP` The Parameter Estimation Framework of HyperChIP

Calling Differential ChIP-seq Signals between Two Conditions

`diffTest.bioCond` Compare Two `bioCond` Objects

`MAplot.diffBioCond` Create an MA Plot on Results of Comparing Two `bioCond` Objects

Comparing ChIP-seq Signals across Multiple Conditions

`aovBioCond` Perform a Moderated Analysis of Variance on `bioCond` Objects

`plot.aovBioCond` Plot an `aovBioCond` Object

`varTestBioCond` Call Hypervariable and Invariant Intervals for a `bioCond`

`plot.varTestBioCond` Plot a `varTestBioCond` Object

Author and Maintainer

Shiqi Tu <<tushiqi@picb.ac.cn>>

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

Chen, H., et al., *HyperChIP for identifying hypervariable signals across ChIP/ATAC-seq samples.* bioRxiv, 2021: p. 2021.07.27.453915.

`MAplot` *Generic MA Plotting*

Description

`MAplot` is a generic function used to produce an MA plot. Described here is the default method for plotting on (normalized) signal intensities of two ChIP-seq samples (see also `normalize`).

Usage

```
MAplot(x, ...)

## Default S3 method:
MAplot(
  x,
  y,
  occupy.x,
  occupy.y,
  col = NULL,
  pch = NULL,
  ylim = c(-6, 6),
  xlab = "A value",
  ylab = "M value",
 args.legend = list(x = "topright", legend = c("common", "y specific", "x specific",
    "others")),
  ...
)
```

Arguments

x, y	x is any R object for which a MAplot method has been defined. For the default method, x and y are two numeric vectors representing signal intensities of the 1st and 2nd samples, respectively.
...	Arguments to be passed to specific methods for the S3 generic. For the default method, ... represents further arguments to be passed to plot.
occupy.x, occupy.y	Two logical vectors of occupancy indicators of the two samples.
col, pch	Optional length-4 vectors specifying the colors and point characters of 4 types of genomic intervals: common peak regions, peak regions specific to the 2nd sample, peak regions specific to the 1st sample, and the others. Elements are recycled if necessary.
ylim	A length-two vector specifying the plotting range of Y-axis (i.e., the M value). Each M value falling outside the range will be shrunk to the corresponding limit. Setting the option to NULL to suppress this behavior.
xlab, ylab	Labels for the X and Y axes.
args.legend	A list of arguments to be passed to legend. You may want to modify the default to incorporate actual sample names.

Value

For the default method, MAplot returns NULL.

Note

While it's not strictly required, one typically normalizes the signal intensities (using normalize) prior to calling this function.

Given the typically large number of points to draw, you may want to use alpha to adjust color transparency if you intend to specify col explicitly.

See Also

normalize for performing an MA normalization on ChIP-seq samples; MAplot.bioCond for creating an MA plot on bioCond objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Create MA scatter plots on normalized ChIP-seq samples.

# Perform MA normalization directly on all ChIP-seq samples. Exclude the
# genomic intervals in sex chromosomes from common peak regions, since these
# samples are from different genders.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
norm <- normalize(H3K27Ac, 4:8, 9:13, common.peak.regions = autosome)

# MA plot on two samples from the same individual.
legend <- c("common", "GM12891_2 specific", "GM12891_1 specific", "others")
MAplot(norm[[5]], norm[[6]], norm[[10]], norm[[11]],
       args.legend = list(x = "topright", legend = legend),
       main = "GM12891_rep1 vs. GM12891_rep2")
abline(h = 0, lwd = 2, lty = 5)

# MA plot on two samples from different individuals.
legend <- c("common", "GM12891_1 specific", "GM12890_1 specific", "others")
MAplot(norm[[4]], norm[[5]], norm[[9]], norm[[10]],
       args.legend = list(x = "topright", legend = legend),
       main = "GM12890_rep1 vs. GM12891_rep1")
abline(h = 0, lwd = 2, lty = 5)
```

MAplot.bioCond *Create an MA Plot on Two* bioCond *Objects*

Description

Given two bioCond objects, the function draws an MA plot, which is a scatter plot with signal intensity differences between the two conditions against the average signal intensities across conditions.

Usage

```
## S3 method for class 'bioCond'
MAplot(
  x,
```

```
  y,
  col = NULL,
  pch = NULL,
  ylim = c(-6, 6),
  xlab = "A value",
  ylab = "M value",
  plot.legend = TRUE,
  ...
)
```

Arguments

x, y	Two bioCond objects.
col, pch	Optional length-4 vectors specifying the colors and point characters of 4 types of genomic intervals: common peak regions, peak regions specific to the 2nd condition, peak regions specific to the 1st condition, and the others. Elements are recycled if necessary.
ylim	A length-two vector specifying the plotting range of Y-axis (i.e., the M value). Each M value falling outside the range will be shrunk to the corresponding limit. Setting the option to NULL to suppress this behavior.
xlab, ylab	Labels for the X and Y axes.
plot.legend	A logical value indicating whether to add a legend.
...	Further arguments to be passed to plot.

Details

Genomic intervals are classified based on the occupancy field in each of the two bioCond objects. See bioCond for a full description of the structure of a bioCond object.

Value

The function returns NULL.

Note

While it's not strictly required, ChIP-seq samples contained in the two bioCond objects are expected to have been normalized prior to calling this function. These samples could be normalized all together before being classified into biological conditions (via normalize). Alternatively, normalization can also be performed at the level of bioCond objects (via normBioCond).

Given the typically large number of points to draw, you may want to use alpha to adjust color transparency if you intend to specify col explicitly.

See Also

bioCond for creating a bioCond object; MAplot.default for producing an MA plot on normalized signal intensities of two ChIP-seq samples; normalize for performing an MA normalization on ChIP-seq samples; normBioCond for normalizing a set of bioCond objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Create MA scatter plots for the comparisons between individuals.

# Perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# MA plots on pairs of individuals.
MAplot(conds[[1]], conds[[2]], main = "GM12890 vs. GM12891")
abline(h = 0, lwd = 2, lty = 5)
MAplot(conds[[1]], conds[[3]], main = "GM12890 vs. GM12892")
abline(h = 0, lwd = 2, lty = 5)
MAplot(conds[[2]], conds[[3]], main = "GM12891 vs. GM12892")
abline(h = 0, lwd = 2, lty = 5)
```

MAplot.diffBioCond *Create an MA Plot on Results of Comparing Two* bioCond *Objects*

Description

This method produces an MA plot demonstrating the results of comparing two bioCond objects. More specifically, it draws a scatter plot consisting of the genomic intervals having been compared, and those intervals with differential ChIP-seq signals between the two conditions are explicitly indicated.

Usage

```
## S3 method for class 'diffBioCond'
MAplot(
  x,
  padj = NULL,
  pval = NULL,
  col = alpha(c("black", "red"), 0.1),
  pch = 20,
  ylim = c(-6, 6),
  xlab = "A value",
  ylab = "M value",
```

```
  args.legend = list(x = "topright"),
  ...
)
```

Arguments

x	An object of class "diffBioCond", typically obtained by passing two bioCond objects to diffTest.
padj, pval	Cutoff of adjusted/raw *p*-value for selecting differential intervals. Only one of the two arguments is effectively used; pval is ignored if padj is specified. The default is equivalent to setting padj to 0.1.
col, pch	Optional length-2 vectors specifying the colors and point characters of non-differential and differential intervals, respectively. Elements are recycled if necessary.
ylim	A length-two vector specifying the plotting range of Y-axis (i.e., the M value). Each M value falling outside the range will be shrunk to the corresponding limit. Setting the option to NULL to suppress this behavior.
xlab, ylab	Labels for the X and Y axes.
args.legend	Further arguments to be passed to legend.
...	Further arguments to be passed to plot.

Value

The function returns NULL.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve given a list of bioCond objects; diffTest for making a comparison between two bioCond objects; alpha for adjusting color transparency.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Make a comparison between GM12891 and GM12892 cell lines and create an MA
## plot on the comparison results.

# Perform MA normalization and construct bioConds to represent the two cell
# lines.
norm <- normalize(H3K27Ac, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Variations in ChIP-seq signals across biological replicates of a cell line
```

```
# are generally of a low level, and their relationship with the mean signal
# intensities is expected to be well modeled by the presumed parametric
# form.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
plotMeanVarCurve(conds, subset = "occupied")

# Perform differential tests between the two cell lines.
res <- diffTest(conds[[1]], conds[[2]])
head(res)

# Visualize the overall test results.
MAplot(res, padj = 0.001)
abline(h = 0, lwd = 2, lty = 5, col = "green3")
```

`meanVarLocalFit` *Fit Mean-Variance Trend by Local Regression*

Description

`meanVarLocalFit` fits a mean-variance curve by applying a robust, gamma-family local regression.

Usage

```
meanVarLocalFit(
  x,
  y,
  weight,
  range.residual = c(1e-04, 15),
  max.iter = 50,
  args.lp = list(),
  args.locfit = list(),
  verbose = TRUE
)
```

Arguments

`x, y`	Two numeric vectors of (sample) means and sample variances, respectively.
`weight`	An optional vector of weights to be used in the fitting procedure. It's typically used when sample variances in `y` are associated with different numbers of degrees of freedom.
`range.residual`	A length-two vector specifying the range of residuals of non-outliers.
`max.iter`	Maximum number of iteration times allowed during the fitting procedure.
`args.lp`	A named list of extra arguments to `lp`.
`args.locfit`	A named list of extra arguments to `locfit`.
`verbose`	Whether to print processing messages about iteratively fitting the mean-variance curve?

Details

meanVarLocalFit iteratively detects outliers and applies the local regression procedure to non-outliers. The procedure converges as soon as the set of outlier points fixes.

Value

A prediction function which accepts a vector of means and returns the predicted variances.

Note

Due to the internal implementation, the argument subset to locfit mustn't be specified in args.locfit.

See Also

meanVarParaFit for parametrically fitting a mean-variance curve; fitMeanVarCurve for an interface to modeling the mean-variance dependence on bioCond objects; plotMeanVarCurve for plotting a mean-variance curve.

---

meanVarParaFit *Parametrically Fit a Mean-Variance Curve*

Description

meanVarParaFit fits a mean-variance curve by applying a robust, gamma-family glm regression, taking advantage of the form: $var = c1 + c2/(2^{m}ean)$.

Usage

```
meanVarParaFit(
  x,
  y,
  weight,
  range.residual = c(1e-04, 15),
  max.iter = 50,
  init.coef = NULL,
  verbose = TRUE
)
```

Arguments

x	Two numeric vectors of (sample) means and sample variances, respectively.
y	Two numeric vectors of (sample) means and sample variances, respectively.
weight	An optional vector of weights to be used in the fitting procedure. It's typically used when sample variances in y are associated with different numbers of degrees of freedom.
range.residual	A length-two vector specifying the range of residuals of non-outliers.

`max.iter`	Maximum number of iteration times allowed during the fitting procedure.
`init.coef`	An optional length-two vector specifying the initial values of the coefficients.
`verbose`	Whether to print processing messages about iteratively fitting the mean-variance curve?

Details

`meanVarParaFit` iteratively detects outliers and fits a generalized linear model on non-outliers. The procedure converges as soon as the set of outlier points fixes.

See "References" for the theoretical foundation of the parametric form.

Value

A prediction function which accepts a vector of means and returns the predicted variances, with an attribute named `"coefficients"` attached.

References

Robinson, M.D. and G.K. Smyth, *Small-sample estimation of negative binomial dispersion, with applications to SAGE data.* Biostatistics, 2008. **9**(2): p. 321-32.

Love, M.I., W. Huber, and S. Anders, *Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.* Genome Biol, 2014. **15**(12): p. 550.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`meanVarLocalFit` for using local regression to fit a mean-variance curve; `fitMeanVarCurve` for an interface to modeling the mean-variance dependence on `bioCond` objects; `plotMeanVarCurve` for plotting a mean-variance curve.

`mean_var_logwinf` *Expectation and Variance of Log Winsorized* F *Distribution*

Description

`mean_var_logwinf` calculates the expectation and variance of a log Winsorized F distribution by appealing to methods for numerical integration.

Usage

```
mean_var_logwinf(
  df1,
  df2,
  p_low = 0.01,
  p_up = 0.1,
  nw = gauss.quad(128, kind = "legendre")
)
```

Arguments

df1, df2	Vectors of numbers of numerator and denominator degrees of freedom. Inf is allowed.
p_low, p_up	Vectors of lower- and upper-tail probabilities for Winsorizing. Must be strictly between 0 and 0.5. Note that df1, df2, p_low and p_up are recycled to align with the longest of them.
nw	A list containing nodes and weights variables for calculating the definite integral of a function f over the interval [-1,1], which is approximated by sum(nw$weights * f(nw$nodes)). By default, mean_var_logwinf uses a set of Gauss-Legendre nodes along with the corresponding weights calculated by gauss.quad.

Details

The function implements exactly the method described in Phipson et al., 2016 (see "References").

Value

A list consisting of the following components:

mu Vector of expectations.

v Vector of variances.

References

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

gauss.quad for calculating nodes and weights for Gaussian quadrature.

Examples

```
# Derive the expectation and variance of a log Winsorized F distribution by
# simulation.
random_logwinf <- function(n, df1, df2, p_low, p_up) {
    x <- rf(n, df1, df2)
    q_low <- qf(p_low, df1, df2, lower.tail = TRUE)
    q_up <- qf(p_up, df1, df2, lower.tail = FALSE)
    x[x < q_low] <- q_low
    x[x > q_up] <- q_up
    x <- log(x)
    c(mean(x), var(x))
}

# Set parameters.
n <- 10000
df1 <- 2
df2 <- 2 ^ (0:10)
```

```
p_low <- 0.01
p_up <- 0.1

# Compare simulation results with those from numerical integration.
set.seed(100)
res1 <- vapply(df2, function(x) random_logwinf(n, df1, x, p_low, p_up),
               numeric(2))
res2 <- mean_var_logwinf(df1, df2, p_low, p_up)

# Compare mean.
plot(0:10, res1[1, ], type = "l", lwd = 2, col = "red", xlab = "Log2(df2)",
     ylab = "Mean")
lines(0:10, res2$mu, lty = 5, lwd = 2, col = "red")
legend("topright", c("Simulation", "Numerical integration"), lty = c(1, 5),
       lwd = 2, col = "red")

# Compare variance.
plot(0:10, res1[2, ], type = "l", lwd = 2, col = "red", xlab = "Log2(df2)",
     ylab = "Var")
lines(0:10, res2$v, lty = 5, lwd = 2, col = "red")
legend("topright", c("Simulation", "Numerical integration"), lty = c(1, 5),
       lwd = 2, col = "red")

# When df2 is Inf.
random_logwinf(n, df1, Inf, p_low, p_up)
mean_var_logwinf(df1, Inf, p_low, p_up)
```

`mvcID.new` *Create a New Unique ID for a Mean-Variance Curve*

Description

To achieve that different calls of `mvcID.new` generate distinct IDs, the function returns the current time via `Sys.time`, appended by a randomly generated serial number.

Usage

```
mvcID.new(n = 10)
```

Arguments

`n`	Length of the serial number.

Value

A character scalar representing the created ID.

Examples

```
## Not run:
## Private functions involved.

mvcID.new()
mvcID.new()
Sys.sleep(1.1)
mvcID.new()

## End(Not run)
```

`normalize` *Perform MA Normalization on a Set of ChIP-seq Samples*

Description

Given read counts from a set of ChIP-seq samples in a set of genomic intervals as well as the signal enrichment states of these intervals in each of the samples, this function converts the read counts into signal intensities more of a continuous variable, and normalizes these signal intensities through linear transformations so that the normalized signal intensities in each genomic interval are comparable across samples.

Usage

```
normalize(
  x,
  count,
  occupancy,
  baseline = NULL,
  subset = NULL,
  interval.size = FALSE,
  offset = 0.5,
  convert = NULL,
  common.peak.regions = NULL
)
```

Arguments

`x`	A data frame containing the read count and occupancy indicator variables. Each row should represent a genomic interval. Objects of other types are coerced to a data frame.
`count`	Either an integer vector or a character vector that indexes the read count variables in `x` to be normalized. Each of these variables represents a ChIP-seq sample. Elements of `count` must be unique.

`occupancy` Either an integer or character vector indexing occupancy indicator variables in `x`. Must correspond to `count` one by one with the same order. These variables are interpreted as logical, where `TRUE` indicates being occupied by peaks (i.e., showing an enrichment for reads) of the corresponding ChIP-seq sample.

`baseline` Either an integer scalar or a character scalar referring to the baseline sample. Must be an element of `count` if specified. By default, the baseline is automatically selected by the function (see "Details").

A special option for this argument is `"pseudo-reference"`, in which case the function constructs a pseudo ChIP-seq sample as baseline by "averaging" the samples to be normalized (see "Details"). This option is especially recommended when the number of samples to be normalized is large (e.g., >5).

`subset` An optional vector specifying the subset of intervals to be used for estimating size factors and selecting the baseline (see "Details" and `estimateSizeFactors`). Defaults to the intervals occupied by all the samples. Ignored if `baseline` is specified.

`interval.size` A numeric vector of interval sizes or a logical scalar to specify whether to use interval sizes for converting read counts into signal intensities (see "Details"). If set to `TRUE`, the function will look for the `"start"` and `"end"` variables in `x`, and use them to calculate interval sizes. By default, interval sizes are not used.

`offset` The offset value used for converting read counts into signal intensities (see "Details").

`convert` An optional function specifying the way that read counts are converted into signal intensities. It should accept a vector of read counts and return the corresponding signal intensities. If set, `interval.size` and `offset` are ignored.

`common.peak.regions`
An optional logical vector specifying the intervals that could possibly be common peak regions for each pair of samples. By default, for each pair of samples, all the intervals occupied by both samples are considered as their common peak regions. See "Details" for an application of this argument.

Details

The function first determines a baseline ChIP-seq sample from the given set. The baseline could either be specified by the user or automatically selected by the function. In the latter case, the function deduces the size factor of each sample using `estimateSizeFactors`, and selects the sample as baseline whose $log2$ size factor is closest to 0 (with the exception that, if there are only two samples to be normalized, the function will always use the sample with the smaller size factor as baseline, for avoiding potential uncertainty in selection results due to limited numerical precision). A special case is setting the `baseline` argument to `"pseudo-reference"`, in which case the function constructs a pseudo ChIP-seq sample as baseline. Technically, for an individual genomic interval in the pseudo sample, the function derives its signal intensity (rather than read count; see below) by taking the average across those samples that occupy it, and it is considered to be a peak region as long as it is occupied by any of the samples to be normalized. We don't need to care about the signal intensities of those intervals that are not occupied by any sample, since they are never used in the normalization process (see below). Using such a pseudo sample as baseline is especially recommended when the number of samples to be normalized is large, for avoiding computation artifacts resulting from an arbitrary selection of baseline sample.

Then, the function converts each read count into a signal intensity through the equation $log2(count + offset)$, or $log2(count/intervalSize + offset)$ if sizes of the genomic intervals are provided. To be noted, while the interval sizes (either specified by users or calculated from the data frame) are considered as number of base pairs, the $intervalSize$ variable used in the latter equation has a unit of kilo base pairs (so that 0.5 still serves as a generally appropriate offset).

In most cases, simply using the former equation is recommended. You may, however, want to involve the interval sizes when ChIP-seq samples to be classified into the same biological condition are associated with a large variation (e.g., when they are from different individuals; see also `bioCond`). Besides, the goodness of fit of mean-variance curve (see also `fitMeanVarCurve`) could serve as one of the principles for selecting an appropriate converting equation.

The `convert` argument serves as an optional function for converting read counts into signal intensities. The function is expected to operate on the read count vector of each sample, and should return the converted signal intensities. `convert` is barely used, exceptions including applying a variance stabilizing transformation or shrinking potential outlier counts.

Finally, the function normalizes each ChIP-seq sample to the baseline. Basically, it applies a linear transformation to the signal intensities of each non-baseline sample, so that M and A values calculated from common peak regions (the genomic intervals occupied by both the sample to be normalized and the baseline) are not correlated. The argument `common.peak.regions` can be used to narrow down the set of intervals that could possibly be considered as common peak regions. You may, for example, use it to remove the intervals located on sex chromosomes from common peak regions when the involved ChIP-seq samples come from different genders (see also "Examples" below).

Value

`normalize` returns the provided data frame, with the read counts replaced by the corresponding normalized signal intensities. Besides, the following attributes are added to the data frame:

`size.factor` Size factors of the specified read count variables. Only present when `baseline` is not explicitly specified by the user.

`baseline` Name of the read count variable used as the baseline sample or `"pseudo-reference"` if the `baseline` argument is specified so.

`norm.coef` A data frame recording the linear transformation coefficients of each sample as well as the number of common peak regions between each sample and the baseline.

`MA.cor` A real matrix recording the Pearson correlation coefficient between M & A values calculated from common peak regions of each pair of samples. The upper and lower triangles of this matrix are deduced from raw and normalized signal intensities, respectively. Note that M values are always calculated as the column sample minus the row sample.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`normalizeBySizeFactors` for normalizing ChIP-seq samples based on their size factors; `estimateSizeFactors` for estimating size factors of ChIP-seq samples; `MAplot` for creating an MA plot on normalized sig-

nal intensities of two samples; bioCond for creating an object to represent a biological condition given a set of normalized ChIP-seq samples, and normBioCond for performing an MA normalization on such objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Perform MA normalization on the whole set of ChIP-seq samples once for
## all.

# Exclude the genomic intervals in sex chromosomes from the common peak
# regions, since the ChIP-seq samples to be normalized are associated with
# different genders.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
norm <- normalize(H3K27Ac, 4:8, 9:13, common.peak.regions = autosome)

# Inspect the normalization effects.
attributes(norm)[5:8]
plot(attr(norm, "MA.cor"), symbreaks = TRUE, margins = c(8, 8))
MAplot(norm[[4]], norm[[5]], norm[[9]], norm[[10]],
       main = "GM12890_rep1 vs. GM12891_rep1")
abline(h = 0, lwd = 2, lty = 5)

## Alternatively, apply MA normalization first within each cell line, and
## then normalize across cell lines. In practice, this strategy is more
## recommended than the aforementioned one.

# Normalize samples separately for each cell line.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)

# Construct separately a bioCond object for each cell line, and perform MA
# normalization on the resulting bioConds. Genomic intervals in sex
# chromosomes are not allowed to be common peak regions, since the cell
# lines are from different genders.
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Inspect the normalization effects.
attributes(conds)
plot(attr(conds, "MA.cor"), symbreaks = TRUE, margins = c(8, 8))
MAplot(conds[[1]], conds[[2]], main = "GM12890 vs. GM12891")
abline(h = 0, lwd = 2, lty = 5)
```

normalizeBySizeFactors *Normalize ChIP-seq Samples by Their Size Factors*

Description

Given read counts from a set of ChIP-seq samples in a set of genomic intervals, this function normalizes the counts using size factors of the samples, and converts the normalized read counts into normalized signal intensities more of a continuous variable. The function can also be used to normalize RNA-seq samples, in which case each genomic interval refers to a gene. In fact, the normalization method implemented in this function is most suited to RNA-seq datasets. See `normalize` for a more robust method for normalizing ChIP-seq samples.

Usage

```
normalizeBySizeFactors(
  x,
  count,
  subset = NULL,
  interval.size = FALSE,
  offset = 0.5,
  convert = NULL
)
```

Arguments

`x`	A data frame containing the read count variables. Each row should represent a genomic interval or a gene. Objects of other types are coerced to a data frame.
`count`	A vector of either integers or characters indexing the read count variables in `x` to be normalized. Each of these variables represents a ChIP-seq/RNA-seq sample. Elements of `count` must be unique.
`subset`	An optional vector specifying the subset of intervals or genes to be used for estimating size factors. For ChIP-seq samples, you may want to use only the intervals occupied by all the samples to estimate their size factors (see "Examples" below). By default, all genomic intervals or genes are used.
`interval.size`	A numeric vector of interval sizes or a logical scalar to specify whether to use interval sizes for converting normalized read counts into normalized signal intensities (see "Details"). If set to `TRUE`, the function will look for the `"start"` and `"end"` variables in `x`, and use them to calculate interval sizes. By default, interval sizes are not used. In cases of analyzing RNA-seq samples, interval sizes, if used, should be the corresponding gene lengths (or sums of exon lengths).
`offset`	The offset value used for converting normalized read counts into normalized signal intensities (see "Details"). The default value is suited to most cases. If you are analyzing RNA-seq samples and intended to use gene lengths, however, a smaller offset value (e.g., 0.01) is recommended.

`convert` An optional function specifying the way that normalized read counts are converted into normalized signal intensities. It should accept a vector of inputs and return a vector of the corresponding signal intensities. If set, `interval.size` and `offset` are ignored.

Details

This function first estimates the size factor of each sample specified, which quantifies the sample's relative sequencing depth. Technically, the function applies the median ratio method to the raw read counts, which is originally devised to normalize RNA-seq samples (see "References"). Then, normalized read counts are deduced by dividing the raw counts of each sample by its size factor.

These normalized read counts are then converted into normalized signal intensities more of a continuous variable. By default, the function uses the equation $log2(normCnt + offset)$, or $log2(normCnt/intervalSize + offset)$ if interval sizes (or gene lengths) are provided. To be noted, while the interval sizes (either specified by users or calculated from the data frame) are considered as number of base pairs, the $intervalSize$ variable used in the latter equation has a unit of kilo base pairs. In this case, 0.5 still serves as a generally appropriate offset for ChIP-seq samples. For RNA-seq samples, however, a smaller offset value (e.g., 0.01) should be adopted.

In most cases, simply using the former equation is recommended. You may, however, want to involve the interval sizes (or gene lengths) when the samples to be classified into the same biological condition are associated with a large variation (e.g., when they are from different individuals; see also `bioCond`). Besides, the goodness of fit of mean-variance curve (see also `fitMeanVarCurve`) could serve as one of the principles for selecting an appropriate converting equation.

The `convert` argument serves as an optional function for converting normalized read counts into normalized signal intensities. The function is expected to operate on the vector of normalized counts of each sample, and should return the converted signal intensities. `convert` is barely used, exceptions including applying a variance stabilizing transformation or shrinking potential outliers.

Value

`normalizeBySizeFactors` returns the provided data frame, with the read counts replaced by the corresponding normalized signal intensities. Besides, an attribute named `"size.factor"` is added to the data frame, recording the size factor of each specified sample.

References

Anders, S. and W. Huber, *Differential expression analysis for sequence count data.* Genome Biol, 2010. **11**(10): p. R106.

See Also

`normalize` for performing an MA normalization on ChIP-seq samples; `estimateSizeFactors` for estimating size factors of ChIP-seq/RNA-seq samples; `MAplot` for creating an MA plot on normalized signal intensities of two samples; `bioCond` for creating an object to represent a biological condition given a set of normalized samples, and `normBioCondBySizeFactors` for normalizing such objects based on their size factors.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Normalize directly the whole set of ChIP-seq samples by their size
## factors.

# Use only the genomic intervals that are occupied by all the ChIP-seq
# samples to be normalized to estimate the size factors.
norm <- normalizeBySizeFactors(H3K27Ac, 4:8,
                               subset = apply(H3K27Ac[9:13], 1, all))

# Inspect the normalization effects.
attr(norm, "size.factor")
MAplot(norm[[4]], norm[[5]], norm[[9]], norm[[10]],
       main = "GM12890_rep1 vs. GM12891_rep1")
abline(h = 0, lwd = 2, lty = 5)

## Alternatively, perform the normalization first within each cell line, and
## then normalize across cell lines. In practice, this strategy is more
## recommended than the aforementioned one.

# Normalize samples separately for each cell line.
norm <- normalizeBySizeFactors(H3K27Ac, 4)
norm <- normalizeBySizeFactors(norm, 5:6,
                               subset = apply(norm[10:11], 1, all))
norm <- normalizeBySizeFactors(norm, 7:8,
                               subset = apply(norm[12:13], 1, all))

# Construct separately a bioCond object for each cell line, and normalize
# the resulting bioConds by their size factors.
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
conds <- normBioCondBySizeFactors(conds)

# Inspect the normalization effects.
attr(conds, "size.factor")
MAplot(conds[[1]], conds[[2]], main = "GM12890 vs. GM12891")
abline(h = 0, lwd = 2, lty = 5)
```

---

normBioCond *Perform MA Normalization on a Set of* bioCond *Objects*

Description

Given a list of bioCond objects, normBioCond performs an MA normalization on the signal intensities stored in them so that these objects are comparable to each other.

Usage

```
normBioCond(conds, baseline = NULL, subset = NULL, common.peak.regions = NULL)
```

Arguments

conds	A list of bioCond objects to be normalized.
baseline	A positive integer or character name indexing the baseline bioCond in conds. By default, the baseline is automatically selected by estimating the size factor of each bioCond (see normalize and estimateSizeFactors for details). Note that normBioCond treats the signal intensities contained in the supplied bioConds as in the scale of $log2$ read counts, which is consistent with the default behavior of normalize. Note also that baseline can be set to "pseudo-reference" as in normalize. And we recommend using this setting when the number of bioConds to be normalized is large (e.g., >5).
subset	An optional vector specifying the subset of intervals to be used for estimating size factors and selecting the baseline. Defaults to the intervals occupied by all the bioCond objects. Ignored if baseline is specified.
common.peak.regions	An optional logical vector specifying the intervals that could possibly be considered as common peak regions for each pair of bioCond objects. See also normalize.

Details

Technically, normBioCond treats each bioCond object as a ChIP-seq sample. It extracts the sample.mean and occupancy variables stored in each bioCond to represent its signal intensities and occupancy indicators, respectively. See bioCond for a description of the structure of a bioCond object.

Next, MA normalization on these bioCond objects is performed exactly as described in normalize. Specifically, we get a linear transformation for each bioCond object, which is subsequently applied to each of the ChIP-seq samples contained in it.

normBioCond is an effort to reduce potential biases introduced by the MA normalization process. The idea comes from the principle that the more similar two samples are to each other, the fewer biases are expected to introduce when normalizing them. With this function, instead of performing an overall normalization on all the ChIP-seq samples involved, you may choose to first perform a normalization within each biological condition, and then normalize between the resulting bioCond objects (see "Examples" below).

Value

A list of bioCond objects with normalized signal intensities, corresponding to the argument conds. To be noted, information about the mean-variance dependence stored in the original bioCond objects, if any, will be removed from the returned bioConds. You can re-fit a mean-variance curve for them by, for example, calling fitMeanVarCurve. Note also that the original structure matrices are retained for each bioCond in the returned list (see setWeight for a detailed description of structure matrix).

Besides, the following attributes are added to the list describing the MA normalization performed:

`size.factor` Size factors of provided `bioCond` objects. Only present when `baseline` is not explicitly specified by the user.

`baseline` Condition name of the `bioCond` object used as baseline or `"pseudo-reference"` if the `baseline` argument is specified so.

`norm.coef` A data frame recording the MA normalization coefficients for each `bioCond`.

`MA.cor` A real matrix recording the Pearson correlation coefficient between M & A values calculated from common peak regions of each pair of `bioCond` objects. The upper and lower triangle of the matrix are deduced from raw and normalized signal intensities, respectively. Note that M values are always calculated as the column `bioCond` minus the row one.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`normalize` for performing an MA normalization on ChIP-seq samples; `bioCond` for creating a `bioCond` object; `normBioCondBySizeFactors` for normalizing `bioCond` objects based on their size factors; `cmbBioCond` for combining a set of `bioCond` objects into a single one; `MAplot.bioCond` for creating an MA plot on two normalized `bioCond` objects; `fitMeanVarCurve` for modeling the mean-variance dependence across intervals in `bioCond` objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Apply MA normalization first within each cell line, and then normalize
## across cell lines.

# Normalize samples separately for each cell line.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)

# Construct separately a bioCond object for each cell line, and perform MA
# normalization on the resulting bioConds. Genomic intervals in sex
# chromosomes are not allowed to be common ones, since the cell lines are
# from different genders.
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Inspect the normalization effects.
attributes(conds)
plot(attr(conds, "MA.cor"), symbreaks = TRUE, margins = c(8, 8))
MAplot(conds[[1]], conds[[2]], main = "GM12890 vs. GM12891")
```

```
abline(h = 0, lwd = 2, lty = 5)
```

---

normBioCondBySizeFactors *Normalize* bioCond *Objects by Their Size Factors*

Description

Given a list of bioCond objects, normBioCondBySizeFactors normalizes the signal intensities stored in them based on their respective size factors, so that these bioConds become comparable to each other. Note that the normalization method implemented in this function is most suited to the bioConds comprised of RNA-seq samples. See normBioCond for a more robust method for normalizing the bioConds consisting of ChIP-seq samples.

Usage

```
normBioCondBySizeFactors(conds, subset = NULL)
```

Arguments

conds	A list of bioCond objects to be normalized.
subset	An optional vector specifying the subset of intervals or genes to be used for estimating size factors. Defaults to the intervals/genes occupied by all the bioCond objects. See normalize and bioCond for more information about occupancy states of intervals/genes in a biological condition.

Details

Technically, normBioCondBySizeFactors considers each bioCond object to be a single ChIP-seq/RNA-seq sample. It treats the sample.mean variable of each bioCond as in the scale of log2 read count, and applies the median ratio strategy to estimate their respective size factors (see "References"). Finally, each bioCond object is normalized by subtracting its log2 size factor from each of its samples.

The idea of normBioCondBySizeFactors comes from the principle that the more similar a set of samples are to each other, the fewer biases are expected to introduce when normalizing them. With this function, instead of performing an overall normalization on all the samples involved, you may choose to first normalize the samples within each biological condition, and then perform a normalization between the resulting bioCond objects (see "Examples" below).

Value

A list of bioCond objects with normalized signal intensities, corresponding to the argument conds. To be noted, information about the mean-variance dependence stored in the original bioCond objects, if any, will be removed from the returned bioConds. You can re-fit a mean-variance curve for them by, for example, calling fitMeanVarCurve. Note also that the original structure matrices are

retained for each bioCond in the returned list (see setWeight for a detailed description of structure matrix).

Besides, an attribute named "size.factor" is added to the returned list, recording the size factor of each bioCond object.

References

Anders, S. and W. Huber, *Differential expression analysis for sequence count data.* Genome Biol, 2010. **11**(10): p. R106.

See Also

normalizeBySizeFactors for normalizing ChIP-seq/RNA-seq samples based on their size factors; bioCond for creating a bioCond object; normBioCond for performing an MA normalization on bioCond objects; cmbBioCond for combining a set of bioCond objects into a single one; MAplot.bioCond for creating an MA plot on two normalized bioCond objects; fitMeanVarCurve for modeling the mean-variance dependence across intervals in bioCond objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## First perform a normalization within each cell line, and then normalize
## across cell lines.

# Normalize samples separately for each cell line.
norm <- normalizeBySizeFactors(H3K27Ac, 4)
norm <- normalizeBySizeFactors(norm, 5:6,
                               subset = apply(norm[10:11], 1, all))
norm <- normalizeBySizeFactors(norm, 7:8,
                               subset = apply(norm[12:13], 1, all))

# Construct separately a bioCond object for each cell line, and normalize
# the resulting bioConds by their size factors.
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
conds <- normBioCondBySizeFactors(conds)

# Inspect the normalization effects.
attr(conds, "size.factor")
MAplot(conds[[1]], conds[[2]], main = "GM12890 vs. GM12891")
abline(h = 0, lwd = 2, lty = 5)
```

normCoef *Deduce MA Normalization Coefficients*

Description

Deduce MA Normalization Coefficients

Usage

```
normCoef(baseline, to.norm)
```

Arguments

baseline	A numeric vector representing the baseline signal intensity.
to.norm	A numeric vector representing the sample to be normalized.

Value

`c(slope,intercept)`

plot.aovBioCond *Plot an* aovBioCond *Object*

Description

Given an aovBioCond object, which records the results of calling differential genomic intervals across a set of bioCond objects, this method creates a scatter plot of `(conds.mean,log10(between.ms))` pairs from all genomic intervals, marking specifically the ones that show a statistical significance. See aovBioCond for a description of the two variables and the associated hypothesis testing. The mean-variance curve associated with the bioCond objects is also added to the plot, serving as a baseline to which the `between.ms` variable of each interval could be compared.

Usage

```
## S3 method for class 'aovBioCond'
plot(
  x,
  padj = NULL,
  pval = NULL,
  col = alpha(c("black", "red"), 0.04),
  pch = 20,
  xlab = "Mean",
  ylab = "log10(Var)",
  args.legend = list(x = "bottomleft"),
  args.lines = list(col = "green3", lwd = 2),
  ...
)
```

Arguments

x	An object of class "aovBioCond", typically a returned value from aovBioCond.
padj, pval	Cutoff of adjusted/raw *p*-value for selecting significant intervals. Only one of the two arguments is effectively used; pval is ignored if padj is specified. The default is equivalent to setting padj to 0.1.
col, pch	Optional length-2 vectors specifying the colors and point characters of non-significant and significant intervals, respectively. Elements are recycled if necessary.
xlab, ylab	Labels for the X and Y axes.
args.legend	Further arguments to be passed to legend.
args.lines	Further arguments to be passed to lines.
...	Further arguments to be passed to plot.

Value

The function returns NULL.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve for a set of bioCond objects; aovBioCond for calling differential intervals across multiple bioConds.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Call differential genomic intervals among GM12890, GM12891 and GM12892
## cell lines and visualize the overall analysis results.

# Perform MA normalization and construct bioConds to represent the cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Variations in ChIP-seq signals across biological replicates of a cell line
# are generally of a low level, and their relationship with the mean signal
# intensities is expected to be well modeled by the presumed parametric
# form.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
plotMeanVarCurve(conds, subset = "occupied")
```

```
# Perform a moderated ANOVA on these cell lines.
res <- aovBioCond(conds)
head(res)

# Visualize the overall analysis results.
plot(res, padj = 1e-6)
```

`plot.matrix` *Visualize a Matrix of Numeric Values*

Description

This method draws a heat map to demonstrate a numeric matrix, using `gplots::heatmap.2` as the underlying engine. Note that the method retains the original (unscaled) values in the matrix, as well as the orders of rows and columns of the matrix.

Usage

```
## S3 method for class 'matrix'
plot(
  x,
  breaks = 101,
  symbreaks = FALSE,
  col = NULL,
  low = "blue",
  mid = "white",
  high = "red",
  na.color = "black",
  lmat = NULL,
  ...
)
```

Arguments

`x`	The matrix of numeric values to be plotted.
`breaks`	Either a numeric vector indicating the splitting points for binning `x` into colors, or an integer number of break points to be used, in which case the break points will be spaced equally across the data range.
`symbreaks`	Logical value indicating whether the break points should be made symmetric about 0. Ignored if `breaks` is specified as a numeric vector.
`col`	Colors used for the heat map. Must have a length equal to the number of break points minus 1. By default, colors are generated by `colorpanel`.
`low, mid, high`	Arguments to be passed to `colorpanel` to generate colors. Ignored if `col` is explicitly specified. Note that setting `mid` to `NA` suppresses the usage of this argument.

na.color	Color to be used for missing (NA) values.
lmat	Position matrix for the layout of color key and heat map. To be passed to heatmap.2. By default, the color key lies above the heat map.
...	Further arguments to be passed to heatmap.2.

Value

The value returned from heatmap.2.

See Also

colorpanel for generating a sequence of colors that varies smoothly; heatmap.2 for drawing a heat map.

Examples

```
set.seed(17)
x <- matrix(rnorm(30, sd = 2), nrow = 5)
x[2, 5] <- NA

# Use the default setting.
plot(x)

# Use break points symmetric about 0.
plot(x, symbreaks = TRUE)
```

---

plot.varTestBioCond *Plot a* varTestBioCond *Object*

Description

Given a varTestBioCond object, which records the results of calling hypervariable and invariant genomic intervals across ChIP-seq samples of a bioCond object, this method creates a scatter plot of observed (mean,log10(variance)) pairs from all genomic intervals, marking specifically the ones that have a significantly large or small variance. Besides, the mean-variance curve associated with the bioCond is also added to the plot, serving as a baseline to which each observed variance could be compared.

Usage

```
## S3 method for class 'varTestBioCond'
plot(
  x,
  padj = NULL,
  pval = NULL,
  col = alpha(c("black", "red"), 0.04),
  pch = 20,
```

```
  xlab = "Mean",
  ylab = "log10(Var)",
  args.legend = list(x = "bottomleft"),
  args.lines = list(col = "green3", lwd = 2),
  ...
)
```

Arguments

x	An object of class "varTestBioCond", typically a returned value from varTestBioCond.
padj, pval	Cutoff of adjusted/raw *p*-value for selecting significant intervals. Only one of the two arguments is effectively used; pval is ignored if padj is specified. The default is equivalent to setting padj to 0.1.
col, pch	Optional length-2 vectors specifying the colors and point characters of non-significant and significant intervals, respectively. Elements are recycled if necessary.
xlab, ylab	Labels for the X and Y axes.
args.legend	Further arguments to be passed to legend.
args.lines	Further arguments to be passed to lines.
...	Further arguments to be passed to plot.

Details

Those genomic intervals considered to be significant are actually the ones that significantly deviate from the mean-variance curve in the plot. See varTestBioCond for technical details of the associated hypothesis testing.

Value

The function returns NULL.

See Also

bioCond for creating a bioCond object from a set of ChIP-seq samples; fitMeanVarCurve for fitting a mean-variance curve; varTestBioCond for calling hypervariable and invariant intervals across ChIP-seq samples contained in a bioCond object.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Call hypervariable and invariant genomic intervals across biological
## replicates of the GM12891 cell line.

# Perform MA normalization and construct a bioCond to represent GM12891.
norm <- normalize(H3K27Ac, 5:6, 10:11)
GM12891 <- bioCond(norm[5:6], norm[10:11], name = "GM12891")
```

```
# Fit a mean-variance curve for GM12891 using the parametric method.
GM12891 <- fitMeanVarCurve(list(GM12891), method = "parametric",
                           occupy.only = TRUE)[[1]]
summary(GM12891)
plotMeanVarCurve(list(GM12891), subset = "occupied")

# Assess the observed variances of ChIP-seq signal intensities in GM12891.
res <- varTestBioCond(GM12891)
head(res)

# Inspect only the test results of occupied genomic intervals.
res <- res[GM12891$occupancy, ]
res$padj <- p.adjust(res$pval, method = "BH")
plot(res, col = scales::alpha(c("black", "red"), c(0.04, 0.5)))
```

---

plotMeanVarCurve *Plot a Mean-Variance Curve*

Description

Given a list of bioCond objects associated with a common mean-variance curve, plotMeanVarCurve draws a scatter plot of observed (mean,log10(variance)) pairs from the genomic intervals contained in them. It also adds the mean-variance curve to the plot.

Usage

```
plotMeanVarCurve(
  conds,
  subset = c("all", "occupied", "non-occupied"),
  col = alpha("blue", 0.02),
  pch = 20,
  xlab = "Mean",
  ylab = "log10(Var)",
  args.legend = list(x = "bottomleft"),
  args.lines = list(col = "red", lwd = 2),
  only.add.line = FALSE,
  ...
)
```

Arguments

conds	A list of bioCond objects with which a mean-variance curve has been associated.
subset	A character string indicating the subset of genomic intervals used for the scatter plot (see "Details"). Must be one of "all" (default), "occupied", or "non-occupied". Can be abbreviated.
col, pch	Optional vectors specifying the color and point character for genomic intervals in each bioCond. Elements are recycled if necessary.

xlab, ylab	Labels for the X and Y axes.
args.legend	Further arguments to be passed to legend.
args.lines	Further arguments to be passed to lines.
only.add.line	A logical value. If set to TRUE, only the mean-variance curve is added to the current plot.
...	Further arguments to be passed to plot.

Details

All bioCond objects supplied in conds should be associated with the same mean-variance curve. Thus, they must have the same "mvcID" (see fitMeanVarCurve for the data structure stored in a bioCond object describing its fit of mean-variance trend). Typically, conds is a returned value from fitMeanVarCurve, setMeanVarCurve or extendMeanVarCurve.

Notably, to make the observed variance of each genomic interval in each bioCond object comparable to the mean-variance curve, all variance values used for the scatter plot have been adjusted for the variance ratio factor specific to each bioCond. See fitMeanVarCurve and estimatePriorDf for a description of variance ratio factor. Note also that there is a function named plotMVC that is specifically designed for plotting a mean-variance curve on a single bioCond. This function scales mean-variance curve by the associated variance ratio factor and leaves observed variances unadjusted.

By default, each genomic interval in each bioCond object that contains replicate samples provides one point for the scatter plot. Setting subset to "occupied" ("non-occupied") makes the function use only those intervals occupied (not occupied) by their bioConds to draw the plot (see normalize and bioCond for more information about occupancy states of genomic intervals).

Value

The function returns NULL.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve given a list of bioCond objects; extendMeanVarCurve for extending the application scope of a fitted mean-variance curve to additional bioCond objects; varRatio for a formal description of variance ratio factor; plotMVC for plotting a mean-variance curve on a single bioCond object; normalize for using occupancy states of genomic intervals to normalize ChIP-seq samples; alpha for adjusting color transparency.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")
```

```
## Fit and plot a mean-variance curve for GM12891 and GM12892 cell lines.

# Perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Fit mean-variance trend based on the presumed parametric form.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])

# Plot the fitted mean-variance curve.
plotMeanVarCurve(conds, subset = "occupied")

# Use different colors for the two bioConds, to see if the mean-variance
# points from the two cell lines mix uniformly with each other.
plotMeanVarCurve(conds, subset = "occupied",
                 col = scales::alpha(c("blue", "green3"), 0.02))
```

---

plotMVC *Plot a Mean-Variance Curve on a Single* bioCond *Object*

Description

Given an individual bioCond object associated with a mean-variance curve, plotMVC draws a scatter plot of observed (mean,log10(variance)) pairs from the genomic intervals contained in the bioCond. It also adds the mean-variance curve to the plot. Notably, unlike plotMeanVarCurve, here the observed variances used for plotting are not adjusted but the mean-variance curve is scaled based on the associated variance ratio factor (see fitMeanVarCurve and estimatePriorDf for a description of variance ratio factor).

Usage

```
plotMVC(
  cond,
  subset = c("all", "occupied", "non-occupied"),
  col = alpha("blue", 0.02),
  pch = 20,
  add = FALSE,
  xlab = "Mean",
  ylab = "log10(Var)",
  args.lines = list(col = "red", lwd = 2),
  only.add.line = FALSE,
  ...
)
```

Arguments

cond	An individual bioCond object with which a mean-variance curve has been associated.
subset	A character string indicating the subset of genomic intervals used for the scatter plot. Must be one of "all" (default), "occupied", or "non-occupied". Can be abbreviated.
col, pch	Optional vectors specifying the colors and point characters of the genomic intervals in cond, respectively. Elements are recycled to match the total number of intervals and are then subject to the subsetting operation specified by subset.
add	Whether to add points to existing graphics (by calling points) or to create new graphics (by calling plot)?
xlab, ylab	Labels for the X and Y axes.
args.lines	Further arguments to be passed to lines.
only.add.line	A logical value. If set to TRUE, only the mean-variance curve is added to existing graphics.
...	Further arguments to be passed to plot or points, depending on the setting of add.

Value

The function returns NULL.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve on a list of bioCond objects; varRatio for a formal description of variance ratio factor; plotMeanVarCurve for plotting a mean-variance curve on a list of bioCond objects; alpha for adjusting color transparency.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit and plot a mean-variance curve for the GM12892 cell line (i.e.,
## individual).

# Perform the MA normalization and construct a bioCond to represent GM12892.
norm <- normalize(H3K27Ac, 7:8, 12:13)
GM12892 <- bioCond(norm[7:8], norm[12:13], name = "GM12892")

# Fit a mean-variance curve by using the parametric method.
GM12892 <- fitMeanVarCurve(list(GM12892), method = "parametric",
```

```
                            occupy.only = TRUE, init.coef = c(0.1, 10))[[1]]

# Draw a mean-variance scatter plot with adjusting observed variances.
plotMeanVarCurve(list(GM12892), subset = "occupied")

# Draw a mean-variance scatter plot with scaling the mean-variance curve.
plotMVC(GM12892, subset = "occupied")
```

`print.bioCond` *Print a* `bioCond` *Object*

Description

This function prints its argument, which is a `bioCond` object, and returns it invisibly (via `invisible(x)`).

Usage

```
## S3 method for class 'bioCond'
print(x, ...)
```

Arguments

`x`	A `bioCond` object.
`...`	Arguments passed from other methods.

Details

This function implements the `print` method for the "`bioCond`" class.

Value

The function returns `x` invisibly.

See Also

`bioCond` for creating a `bioCond` object.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Print bioConds that correspond to individuals.

# Perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
```

```
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Print these bioConds.
print(conds[[1]])
print(conds[[2]])
print(conds[[3]])
```

`print.summaryBioCond` *Print a* `summaryBioCond` *Object*

Description

This function prints the `summary` result of a `bioCond` object.

Usage

```
## S3 method for class 'summaryBioCond'
print(x, ...)
```

Arguments

`x`	An object of class `"summaryBioCond"`, typically obtained by passing a `bioCond` object to the `summary` function.
`...`	Arguments passed from other methods.

Details

This function implements the `print` method for the `"summaryBioCond"` class.

Value

The function returns `x` invisibly.

See Also

`bioCond` for creating a `bioCond` object; `summary.bioCond` for summarizing a `bioCond` object.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Print summary results of bioConds that correspond to individuals.

# Perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Print summary results of these bioConds.
print(summary(conds[[1]]))
print(summary(conds[[2]]))
print(summary(conds[[3]]))

# Print summary results of these bioConds after fitting a mean-variance
# curve for them.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
print(summary(conds[[1]]))
print(summary(conds[[2]]))
print(summary(conds[[3]]))
```

`scaleMeanVarCurve` *Scale a Mean-Variance Curve*

Description

`scaleMeanVarCurve` underlies other interface functions for estimating the variance ratio factor of an unadjusted mean-variance curve (or a set of unadjusted mean-variance curves).

Usage

```
scaleMeanVarCurve(z, m, d0)
```

Arguments

`z`	A list of which each element is a vector of FZ statistics corresponding to a `bioCond` object (see also "Details").
`m`	A vector of numbers of replicates in `bioCond` objects. Must correspond to `z` one by one in the same order.

d0 A positive real specifying the number of prior degrees of freedom of the mean-variance curve(s). Inf is allowed. Note that d0 is typically estimated via estimateD0.

Details

For each bioCond object with replicate samples, a vector of FZ statistics can be deduced from the unadjusted mean-variance curve associated with it. More specifically, for each genomic interval in a bioCond with replicate samples, its FZ statistic is defined to be $log(t_hat/v0)$, where t_hat is the observed variance of signal intensities of the interval, and $v0$ is the interval's prior variance read from the corresponding mean-variance curve.

Theoretically, each FZ statistic follows a scaled Fisher's Z distribution plus a constant (since the mean-variance curve is not adjusted yet), and we can use the sample mean (plus a constant that depends on the number of prior degrees of freedom) of the FZ statistics of each single bioCond to get an estimate of log variance ratio factor.

The final estimate of log variance ratio factor is a weighted mean of estimates across bioCond objects, with the weights being their respective numbers of genomic intervals that are used to calculate FZ statistics. This should be appropriate, as Fisher's Z distribution is roughly normal (see also "References"). The weighted mean is actually a plain (unweighted) mean across all the involved genomic intervals.

Finally, we get an estimate of variance ratio factor by taking an exponential.

Value

The estimated variance ratio factor for adjusting the mean-variance curve(s). Note that the function returns NA if there are not sufficient genomic intervals for estimating it.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; varRatio for a formal description of variance ratio factor; estimateD0 for estimating the number of prior degrees of freedom associated with a mean-variance curve (or a set of curves); estimatePriorDf for an interface to estimating the number of prior degrees of freedom on bioCond objects as well as adjusting their mean-variance curve(s) accordingly.

estimateD0Robust and scaleMeanVarCurveRobust for estimating number of prior degrees of freedom and variance ratio factor *in a robust manner*, respectively.

scaleMeanVarCurveRobust *Scale a Mean-Variance Curve in a Robust Manner*

Description

scaleMeanVarCurveRobust underlies other interface functions for estimating the variance ratio factor of an unadjusted mean-variance curve (or a set of unadjusted mean-variance curves) *in a robust manner*.

Usage

```
scaleMeanVarCurveRobust(
  z,
  m,
  d0,
  p_low = 0.01,
  p_up = 0.1,
  nw = gauss.quad(128, kind = "legendre")
)
```

Arguments

z	A list of which each element is a vector of FZ statistics corresponding to a bioCond object (see also "Details").
m	A vector of numbers of replicates in bioCond objects. Must correspond to z one by one in the same order.
d0	A positive real specifying the number of prior degrees of freedom of the mean-variance curve(s). Inf is allowed. Note that d0 could be robustly estimated by estimateD0Robust.
p_low, p_up	Lower- and upper-tail probabilities for Winsorizing the FZ statistics associated with each bioCond.
nw	A list containing nodes and weights variables for calculating the definite integral of a function f over the interval [-1,1], which is approximated by sum(nw$weights * f(nw$nodes)). By default, a set of Gauss-Legendre nodes along with the corresponding weights calculated by gauss.quad is used.

Details

For each bioCond object with replicate samples, a vector of FZ statistics can be deduced from the unadjusted mean-variance curve associated with it. More specifically, for each genomic interval in a bioCond with replicate samples, its FZ statistic is defined to be $log(t_hat/v0)$, where t_hat is the observed variance of signal intensities of the interval, and $v0$ is the interval's prior variance read from the corresponding mean-variance curve.

Theoretically, each FZ statistic follows a scaled Fisher's Z distribution plus a constant (since the mean-variance curve is not adjusted yet), and we derive a robust estimation of log variance ratio

factor by Winsorizing the FZ statistics of each bioCond and matching the resulting sample mean with the theoretical expectation of the Winsorized distribution, which is calculated by using numerical integration (see also "References").

The final estimate of log variance ratio factor is a weighted mean of estimates across bioCond objects, with the weights being their respective numbers of genomic intervals that are used to calculate FZ statistics.

Finally, we get an estimate of variance ratio factor by taking an exponential.

Value

The estimated variance ratio factor for adjusting the mean-variance curve(s). Note that the function returns NA if there are not sufficient genomic intervals for estimating it.

References

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; varRatio for a formal description of variance ratio factor; estimateD0Robust for estimating the number of prior degrees of freedom associated with a mean-variance curve (or a set of curves) *in a robust manner*; estimatePriorDfRobust for an interface to *robustly* estimating the number of prior degrees of freedom on bioCond objects as well as *robustly* adjusting their mean-variance curve(s) accordingly.

estimateD0 and scaleMeanVarCurve for the ordinary (non-robust) routines for estimating number of prior degrees of freedom and variance ratio factor, respectively.

Examples

```
# Refer to "Examples" given in the help page for the function
# estimateD0Robust.
```

---

setMeanVarCurve *Set the Mean-Variance Curve of a Set of* bioCond *Objects*

Description

Given a set of bioCond objects, setMeanVarCurve associates a common mean-variance curve with each of them, assesses the overall goodness of fit by estimating the number of prior degrees of freedom, and accordingly estimates their variance ratio factors (see also fitMeanVarCurve).

Usage

```
setMeanVarCurve(
  conds,
  predict,
  occupy.only = TRUE,
  method = "NA",
  ratio.var = estimateVarRatio(conds),
  .call = NULL
)
```

Arguments

conds	A list of bioCond objects, of which at least one should contain replicate samples.
predict	A function representing the mean-variance curve to be associated with the bioConds. It should accept a vector of means and return the predicted variances.
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to estimate the number of prior degrees of freedom and the variance ratio factors. Otherwise, all intervals are used.
method	A character string giving the method for fitting the mean-variance curve. Used only for constructing the fit.info fields (see "Value" below).
ratio.var	Backup variance ratio factors of the bioConds. Only used when the estimated number of prior degrees of freedom is 0, which in practice rarely happens.
.call	Never care about this argument.

Details

The specific behavior of this function is pretty much the same as fitMeanVarCurve, except that the mean-variance curve is directly specified by users rather than fitted based on the observed means and variances. Refer to fitMeanVarCurve for a detailed description of related terms.

Interestingly, if a positive constant function is supplied as the mean-variance curve, the resulting statistical model will be rather similar to the one implemented in the limma package (see also "References"). Notably, using a constant function as the mean-variance curve is particularly suited to bioCond objects that have gone through a variance-stabilizing transformation (see vstBioCond for details and "Examples" below) as well as bioConds whose structure matrices have been specifically designed (see "References").

Value

setMeanVarCurve returns the argument list of bioCond objects, each of which has an added (updated) fit.info field constructed based on the supplied mean-variance curve. The field is itself a list consisting of the following components:

calls The two function calls for associating a mean variance curve with this bioCond and estimating the related parameters, respectively. The latter is only present if you have made an explicit call to some function (e.g., estimatePriorDf) for performing the parameter estimation.

method Method used for fitting the mean-variance curve.

predict The supplied mean-variance function.

`mvcID` ID of the mean-variance curve.

`df.prior` Number of prior degrees of freedom assessing the goodness of fit of the mean-variance curve.

`ratio.var` Variance ratio factor of this `bioCond`.

Each `bioCond` object in the returned list has the same values of all these components but the `ratio.var`. `mvcID` is automatically generated by the function to label the supplied mean-variance curve. Each call to `setMeanVarCurve` results in a unique `mvcID`.

Besides, if there exist `bioCond` objects that contain only one ChIP-seq sample, an attribute named `"no.rep.rv"` will be added to the returned list, recording the variance ratio factor of no-replicate conditions.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

Law, C.W., et al., *voom: Precision weights unlock linear model analysis tools for RNA-seq read counts.* Genome Biol, 2014. **15**(2): p. R29.

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object from a set of ChIP-seq samples; `fitMeanVarCurve` for fitting a mean-variance curve for a set of `bioCond` objects; `estimateVarRatio` for estimating the relative variance ratio factors of a set of `bioConds`; `varRatio` for a formal description of variance ratio factor; `estimatePriorDf` for estimating the number of prior degrees of freedom and the corresponding variance ratio factors; `estimatePriorDfRobust` for a *robust* version of `estimatePriorDf`.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Perform differential analysis on bioConds that have gone through a
## variance-stabilizing transformation.

# Perform MA normalization and construct bioConds to represent cell lines
# (i.e., individuals).
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)
```

```
# Fit a mean-variance curve.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
plotMeanVarCurve(conds, subset = "occupied")

# Apply a variance-stabilizing transformation.
vst_conds <- list(GM12890 = vstBioCond(conds$GM12890))
vst.func <- attr(vst_conds$GM12890, "vst.func")
temp <- matrix(vst.func(as.numeric(conds$GM12891$norm.signal)),
               nrow = nrow(norm))
vst_conds$GM12891 <- bioCond(temp, norm[10:11], name = "GM12891")
temp <- matrix(vst.func(as.numeric(conds$GM12892$norm.signal)),
               nrow = nrow(norm))
vst_conds$GM12892 <- bioCond(temp, norm[12:13], name = "GM12892")

# Associate a constant function with the resulting bioConds as their
# mean-variance curve.
vst_conds <- setMeanVarCurve(vst_conds, function(x) rep_len(1, length(x)),
                             occupy.only = TRUE, method = "constant prior")
plotMeanVarCurve(vst_conds, subset = "occupied")

# Make a comparison between GM12891 and GM12892.
res1 <- diffTest(conds$GM12891, conds$GM12892)
res2 <- diffTest(vst_conds$GM12891, vst_conds$GM12892)

# Examine the consistency of analysis results between using ordinary and
# VSTed signal intensities. Here we map p-values together with observed
# directions of signal changes to the standard normal distribution.
z1 <- qnorm(res1$pval / 2)
z1[res1$Mval > 0] <- -z1[res1$Mval > 0]
z2 <- qnorm(res2$pval / 2)
z2[res2$Mval > 0] <- -z2[res2$Mval > 0]
plot(z1, z2, xlab = "Ordinary", ylab = "VSTed")
abline(a = 0, b = 1, lwd = 2, lty = 5, col = "red")
cor(z1, z2)
cor(z1, z2, method = "sp")

# Simultaneously compare GM12890, GM12891 and GM12892 cell lines.
res1 <- aovBioCond(conds)
res2 <- aovBioCond(vst_conds)

# Examine the consistency of analysis results between using ordinary and
# VSTed signal intensities by mapping p-values to the standard normal
# distribution.
z1 <- qnorm(res1$pval, lower.tail = FALSE)
z1[z1 == Inf] <- 39
z2 <- qnorm(res2$pval, lower.tail = FALSE)
z2[z2 == Inf] <- 39
plot(z1, z2, xlab = "Ordinary", ylab = "VSTed")
abline(a = 0, b = 1, lwd = 2, lty = 5, col = "red")
cor(z1, z2)
cor(z1, z2, method = "sp")
```

setPriorDf *Set the Number of Prior Degrees of Freedom of Mean-Variance Curves*

Description

Given a set of bioCond objects of which each has been associated with a mean-variance curve, setPriorDf assigns a common number of prior degrees of freedom to all the bioConds and accordingly adjusts their variance ratio factors.

Usage

```
setPriorDf(conds, d0, occupy.only = TRUE, no.rep.rv = NULL, .call = TRUE)
```

Arguments

conds	A list of bioCond objects, of which each has a fit.info field describing its mean-variance curve (see also fitMeanVarCurve).
d0	A non-negative real specifying the number of prior degrees of freedom. Inf is allowed.
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to adjust the variance ratio factors. Otherwise, all intervals are used.
no.rep.rv	A positive real specifying the variance ratio factor of those bioConds without replicate samples, if any. By default, it's set to the geometric mean of variance ratio factors of the other bioConds.
.call	Never care about this argument.

Details

The specific behavior of this function is pretty much the same as estimatePriorDf, except that the number of prior degrees of freedom is directly specified by users rather than estimated based on the observed data. Refer to estimatePriorDf for more information.

Note also that there is a *robust* version of this function that uses Winsorized statistics to derive variance ratio factors (see setPriorDfRobust for details).

Value

setPriorDf returns the argument list of bioCond objects, with the specified number of prior degrees of freedom substituted for the "df.prior" component of each of them. Besides, their "ratio.var" components have been adjusted accordingly, and an attribute named "no.rep.rv" is added to the list if it's ever been used as the variance ratio factor of the bioConds without replicate samples.

To be noted, if the specified number of prior degrees of freedom is 0, setPriorDf won't adjust existing variance ratio factors. In this case, you may want to use setPriorDfVarRatio to explicitly specify variance ratio factors.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve and using a fit.info field to characterize it; estimatePriorDf for estimating the number of prior degrees of freedom and adjusting the variance ratio factors of a set of bioConds; setPriorDfRobust for a *robust* version of setPriorDf; diffTest for calling differential intervals between two bioCond objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit a mean-variance curve for the GM12892 cell line (i.e., individual)
## and set the number of prior degrees of freedom of the curve to Inf.

# Perform the MA normalization and construct a bioCond to represent GM12892.
norm <- normalize(H3K27Ac, 7:8, 12:13)
GM12892 <- bioCond(norm[7:8], norm[12:13], name = "GM12892")

# Variations in ChIP-seq signals across biological replicates of a cell line
# are generally of a low level, and typically their relationship with the
# mean signal intensities could be well modeled by the presumed parametric
# form.
GM12892 <- fitMeanVarCurve(list(GM12892), method = "parametric",
                           occupy.only = TRUE, init.coef = c(0.1, 10))[[1]]

# In the vast majority of cases for modeling biological replicates of cell
# lines, the associated variance structure is so regular that variances of
# individual genomic intervals could be reliably estimated by fully
# depending on the mean-variance curve.
GM12892_2 <- setPriorDf(list(GM12892), Inf, occupy.only = TRUE)[[1]]

# The resulting model makes few differences from the original one, though.
# This is because MAnorm2 will adaptively deduce a large number of prior
# degrees of freedom for the mean-variance curve if the underlying variance
# structure is of high regularity. In practice, we recommend leaving the
# specification of prior df to the estimation method implemented in MAnorm2
# all the time.
summary(GM12892)
summary(GM12892_2)
```

---

setPriorDfRobust *The Robust Counterpart of* setPriorDf

Description

Given a set of bioCond objects of which each has been associated with a mean-variance curve, setPriorDfRobust assigns a common number of prior degrees of freedom to all of them and accordingly adjusts their variance ratio factors *in a robust manner*.

Usage

```
setPriorDfRobust(
  conds,
  d0,
  occupy.only = TRUE,
  p_low = 0.01,
  p_up = 0.1,
  nw = gauss.quad(128, kind = "legendre"),
  no.rep.rv = NULL,
  .call = TRUE
)
```

Arguments

conds	A list of bioCond objects, of which each has a fit.info field describing its mean-variance curve (see also fitMeanVarCurve).
d0	A non-negative real specifying the number of prior degrees of freedom. Inf is allowed.
occupy.only	A logical scalar. If it is TRUE (default), only occupied intervals are used to adjust the variance ratio factors. Otherwise, all intervals are used.
p_low	Lower- and upper-proportions of extreme values to be Winsorized (see "References"). Must be strictly between 0 and 0.5.
p_up	Lower- and upper-proportions of extreme values to be Winsorized (see "References"). Must be strictly between 0 and 0.5.
nw	A list containing nodes and weights variables for calculating the definite integral of a function f over the interval [-1,1], which is approximated by sum(nw$weights * f(nw$nodes)). By default, a set of Gauss-Legendre nodes along with the corresponding weights calculated by gauss.quad is used.
no.rep.rv	A positive real specifying the variance ratio factor of those bioConds without replicate samples, if any. By default, it's set to the geometric mean of variance ratio factors of the other bioConds.
.call	Never care about this argument.

Details

The specific behavior of this function is pretty much the same as setPriorDf, except that this function adjusts variance ratio factors in a manner that is *robust* to potential outliers (see also "References").

Value

setPriorDfRobust returns the argument list of bioCond objects, with the specified number of prior degrees of freedom substituted for the "df.prior" component of each of them. Besides, their "ratio.var" components have been adjusted accordingly, and an attribute named "no.rep.rv" is added to the list if it's ever been used as the variance ratio factor of the bioConds without replicate samples.

To be noted, if the specified number of prior degrees of freedom is 0, `setPriorDfRobust` won't adjust existing variance ratio factors. In this case, you may want to use `setPriorDfVarRatio` to explicitly specify variance ratio factors.

References

Tukey, J.W., *The future of data analysis.* The annals of mathematical statistics, 1962. **33**(1): p. 1-67.

Phipson, B., et al., *Robust Hyperparameter Estimation Protects against Hypervariable Genes and Improves Power to Detect Differential Expression.* Annals of Applied Statistics, 2016. **10**(2): p. 946-963.

See Also

`bioCond` for creating a `bioCond` object; `fitMeanVarCurve` for fitting a mean-variance curve and using a `fit.info` field to characterize it; `estimatePriorDfRobust` for estimating the number of prior degrees of freedom and adjusting the variance ratio factors of a set of `bioConds` *in a robust manner*; `setPriorDf` for the ordinary (non-robust) version of `setPriorDfRobust`; `diffTest` for calling differential intervals between two `bioCond` objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Fit a mean-variance curve for the GM12892 cell line (i.e., individual)
## and set the number of prior degrees of freedom of the curve to Inf.

# Perform the MA normalization and construct a bioCond to represent GM12892.
norm <- normalize(H3K27Ac, 7:8, 12:13)
GM12892 <- bioCond(norm[7:8], norm[12:13], name = "GM12892")

# Fit a mean-variance curve by using the parametric method.
GM12892 <- fitMeanVarCurve(list(GM12892), method = "parametric",
                           occupy.only = TRUE, init.coef = c(0.1, 10))[[1]]

# Set the number of prior degrees of freedom to Inf.
GM12892_2 <- setPriorDf(list(GM12892), Inf, occupy.only = TRUE)[[1]]

# Use the robust version of setPriorDf.
GM12892_3 <- setPriorDfRobust(list(GM12892), Inf, occupy.only = TRUE)[[1]]

# In this case, there is little difference in estimated variance ratio
# factor between the ordinary routine and the robust one.
summary(GM12892_2)
summary(GM12892_3)
```

`setPriorDfVarRatio` *Set the Number of Prior Degrees of Freedom and Variance Ratio Factors*

Description

Given a set of `bioCond` objects of which each has been associated with a mean-variance curve, `setPriorDfVarRatio` assigns a common number of prior degrees of freedom to all of them and sets their variance ratio factors based on user-provided values. There are few scenarios where you need to call this function (see "Details").

Usage

```
setPriorDfVarRatio(
  conds,
  d0 = 0,
  ratio.var = estimateVarRatio(conds),
  .call = TRUE
)
```

Arguments

`conds`	A list of `bioCond` objects, of which each has a `fit.info` field describing its mean-variance curve (see also `fitMeanVarCurve`).
`d0`	A non-negative real specifying the number of prior degrees of freedom. Specifying a value other than 0 will lead to a warning (see also "Details").
`ratio.var`	A vector giving the variance ratio factors of the `bioConds`. Elements are recycled if necessary. By default, it's estimated by calling `estimateVarRatio`.
`.call`	Never care about this argument.

Details

Basically, the only reason for which you need to call this function is that you don't want to borrow information between genomic intervals to improve variance estimation. Therefore, this function should be in principle called always with the default value (i.e., 0) for `d0`, in which case you can still account for potential differences in global within-group variability between groups of samples. Otherwise, you should empirically estimate `d0` via, for example, `estimatePriorDf` or `estimatePriorDfRobust`.

There are two typical scenarios in which you don't want to borrow information between genomic intervals. In the first one, the estimated `d0` derived by `estimatePriorDfRobust` is 0 because the underlying variance structure is highly irregular. In the second one, there are sufficient replicate samples (e.g., >7 profiles in each group) such that observed variances alone could provide reliable variance estimates.

Value

The argument list of `bioCond` objects, with updated `"df.prior"` and `"ratio.var"` components.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve and using a fit.info field to characterize it; estimatePriorDf and estimatePriorDfRobust for estimating the number of prior degrees of freedom and adjusting the variance ratio factors of a set of bioConds; setPriorDf and setPriorDfRobust for setting the number of prior degrees of freedom and accordingly adjusting the variance ratio factors of a set of bioConds.

---

setWeight *Set the Weights of Signal Intensities Contained in a* bioCond

---

Description

setWeight modifies the relative precisions of signal intensities stored in a bioCond object. One typically uses this function in the form of x <-setWeight(x,weight), where x is a bioCond object and weight is a matrix of positive weights.

Usage

```
setWeight(x, weight = NULL, strMatrix = NULL)
```

Arguments

x	A bioCond object.
weight	A matrix or data frame specifying the relative precisions of signal intensities contained in x. Must have the same number of columns as x$norm.signal. A vector is interpreted as a matrix having a single row. Note that rows of weight are recycled if necessary. By default, the same weight is assigned to each measurement in x$norm.signal.
strMatrix	An optional list of symmetric matrices specifying directly the structure matrix of each genomic interval. Elements of it are recycled if necessary. This argument, if set, overrides the weight argument. See "Details" for more information about structure matrix.

Details

For each genomic interval in a bioCond object, MAnorm2 models the signal intensities of it as having a common mean and a covariance matrix proportional to the interval's structure matrix. Put it formally, $cov(Xi|ti) = ti * Si$, where Xi is the vector of signal intensities of the ith interval, ti is a positive scalar quantifying the variation level of this interval and Si is a symmetric matrix denoting the interval's structure matrix.

Naturally, assuming there are no correlations between ChIP-seq samples, each Si is a diagonal matrix, with its diagonal elements being the reciprocal of the corresponding weights.

The structure matrices will be used to derive the sample mean and sample variance (i.e., estimate of ti) of signal intensities of each interval, using the GLS (generalized least squares) estimation. See also fitMeanVarCurve for modeling their relationship across intervals.

Value

A bioCond object with an updated strMatrix field. To be noted, information about the mean-variance dependence of the original bioCond object, if any, will be removed in the returned bioCond. You can re-fit it by, for example, calling fitMeanVarCurve.

Warning

Do not directly modify the strMatrix field in a bioCond object. Instead, use this function.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

bioCond for creating a bioCond object based on normalized signal intensities; fitMeanVarCurve for fitting the mean-variance trend across genomic intervals.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Set the weights of replicate ChIP-seq samples in a bioCond.

# Construct a bioCond object for the GM12891 cell line. By default, all the
# ChIP-seq samples belonging to the bioCond have the same weight for
# estimating the mean signal intensities of genomic intervals in the cell
# line.
norm <- normalize(H3K27Ac, 5:6, 10:11)
GM12891 <- bioCond(norm[5:6], norm[10:11], name = "GM12891")

# Now we set the weight of the 2nd sample to half of the 1st one.
GM12891_2 <- setWeight(GM12891, weight = c(1, 0.5))

# Equivalently, you can achieve the same effect by setting the strMatrix
# parameter.
GM12891_3 <- setWeight(GM12891, strMatrix = list(diag(c(1, 2))))
```

---

summary.bioCond *Summarize a* bioCond *Object*

Description

The method produces an object that summarizes the data and fit information of mean-variance dependence (if available) stored in a bioCond object.

Usage

```
## S3 method for class 'bioCond'
summary(object, ...)
```

Arguments

object	A bioCond object.
...	Arguments passed from other methods.

Details

This function implements the summary method for the "bioCond" class.

Value

The method returns an object of class "summaryBioCond", for which a specialized print method has been defined.

See Also

bioCond for creating a bioCond object. fitMeanVarCurve for fitting a mean-variance curve on bioCond objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Summarize bioConds that correspond to individuals.

# Perform the MA normalization and construct bioConds to represent
# individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Summarize these bioConds.
summary(conds[[1]])
summary(conds[[2]])
summary(conds[[3]])
str(summary(conds[[3]]))

# Summarize these bioConds after fitting a mean-variance curve for them.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
summary(conds[[1]])
summary(conds[[2]])
```

```
summary(conds[[3]])
str(summary(conds[[3]]))
```

`util.trigamma` *Utility Trigamma Function*

Description

`util.trigamma` is essentially the same as the `trigamma` function but is for being consistent with the `inv.trigamma` function at very small or very large input values.

Usage

```
util.trigamma(y)
```

Arguments

`y`	A positive numeric scalar. `Inf` is allowed.

Value

A positive numeric scalar, which is essentially the same as `trigamma(y)` but could be a little different at very small or very large `y` values.

See Also

`inv.trigamma` for an implementation of the inversion of the `trigamma` function.

Examples

```
trigamma(1:6)
vapply(1:6, util.trigamma, numeric(1))

trigamma(1e-4)
util.trigamma(1e-4)

trigamma(1e8)
util.trigamma(1e8)

trigamma(Inf)
util.trigamma(Inf)
```

varRatio *Compare Variance Ratio Factors of Two* bioCond *Objects*

Description

Given two bioCond objects, varRatio robustly estimates the ratio between their variance ratio factors, assuming they are associated with the same mean-variance curve and using the genomic intervals expected to have invariant signal intensities across the two biological conditions (see "Details").

Usage

```
varRatio(cond1, cond2, invariant = NULL)
```

Arguments

cond1, cond2	Two bioCond objects.
invariant	An optional non-negative real specifying the upper bound of difference in mean signal intensity for a genomic interval to be treated as invariant between cond1 and cond2. By default, intervals occupied by both conditions are treated as invariant.

Details

MAnorm2 models ChIP-seq samples as grouped by biological conditions. It constructs a bioCond object to represent each biological condition, which contains a set of ChIP-seq samples belonging to the condition.

Given multiple bioCond objects, MAnorm2 could fit a single curve to model the mean-variance dependence across genomic intervals. Each genomic interval in each bioCond object that contains replicate samples serves as an observation for the fitting process.

To account for the global difference in variation level of signal intensities between two conditions, MAnorm2 involves a "variance ratio factor" for each condition. Specifically, given two bioCond objects associated with the same mean-variance curve (say condition 1 and 2), we have

$$cov(Xi, 1|vi) = (r1 * vi) * Si, 1$$

and

$$cov(Xi, 2|vi) = (r2 * vi) * Si, 2$$

for any genomic interval i that is *not* differentially represented between the two conditions. Here, Xi, j is the vector of signal intensities of interval i in condition j, rj is the variance ratio factor (a scalar) of condition j, vi is the unscaled variance (a scalar) of signal intensities in interval i, and Si, j is the structure matrix of interval i in condition j (see bioCond and setWeight for a detailed description of structure matrix).

Under this formulation, varRatio estimates the ratio of the variance ratio factor of cond2 to that of cond1, using the intervals with invariant signal intensities across the two conditions. The argument invariant controls the set of such intervals. By default, intervals occupied by both conditions

constitute the set. Alternatively, giving `invariant` a non-negative value specifies these intervals to be invariant that have a difference in average signal intensity between the two conditions less than or equal to the value.

In most cases, you don't need to call this function directly. It's typically used by `fitMeanVarCurve` for fitting a mean-variance trend on a set of `bioCond` objects.

Value

The estimated ratio of the variance ratio factor of `cond2` to that of `cond1`. Note that the function returns `NA` if there are not sufficient invariant intervals for estimating it.

References

Tu, S., et al., *MAnorm2 for quantitatively comparing groups of ChIP-seq samples.* Genome Res, 2021. **31**(1): p. 131-145.

See Also

`bioCond` for creating a `bioCond` object; `setWeight` for a detailed description of structure matrix; `fitMeanVarCurve` for fitting a mean-variance curve given a set of `bioCond` objects.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Compare variance ratio factor between cell lines.

# Perform the MA normalization and construct bioConds to represent cell
# lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Compare the variance ratio factor of GM12892 to that of GM12891.
varRatio(conds$GM12891, conds$GM12892)

# Such a comparison is only possible when both bioConds have replicate
# samples.
varRatio(conds$GM12891, conds$GM12890)
```

varTestBioCond *Call Hypervariable and Invariant Intervals for a* bioCond

Description

Given a bioCond object with which a mean-variance curve is associated (see fitMeanVarCurve), varTestBioCond tests for each genomic interval if the observed variation of its signal intensity across ChIP-seq samples in the bioCond is significantly greater or less than is implied by the curve. This function is typically used in combination with estParamHyperChIP to call hypervariable and invariant intervals in a bioCond (see also "Examples").

Usage

```
varTestBioCond(cond, min.var = 0, df.prior = NULL)
```

Arguments

cond	A bioCond object with which a mean-variance curve has been associated (see also fitMeanVarCurve).
min.var	Lower bound of variances read from the mean-variance curve. Any variance read from the curve less than min.var will be adjusted to this value. It's primarily used for safely getting the prior variances and taking into account the practical significance of a signal variation.
df.prior	Number of prior degrees of freedom associated with the mean-variance curve. Must be positive. Can be set to Inf (see "Details"). The default value should be used in most cases, which is extracted from the "df.prior" component of cond.

Details

varTestBioCond adopts the modeling strategy implemented in limma (see "References"), except that each genomic interval has its own prior variance, which is read from the mean-variance curve associated with the bioCond object. The argument df.prior could be used to specify the common number of degrees of freedom of all the prior variances, which also effectively assesses the overall goodness of fit of the mean-variance curve. Technically, varTestBioCond uses the ratio of the observed variance of each interval to its prior variance as key statistic, which under the null hypothesis follows an F distribution, with its two numbers of degrees of freedom being those of the two variances, respectively. (Hence the statistic follows a scaled chi-squared distribution when the prior df is Inf.) To be noted, the prior df can be empirically estimated for each mean-variance curve by specifically designed statistical methods (see also fitMeanVarCurve, setMeanVarCurve, estimatePriorDf, and estParamHyperChIP) and, by default, the function uses the estimation result to perform the tests. It's highly not recommended to specify df.prior explicitly when calling varTestBioCond, unless you know what you are really doing. Besides, varTestBioCond won't adjust the variance ratio factor of the provided bioCond based on the specified prior df (see estimatePriorDf for a description of variance ratio factor).

Any bioCond object passed to varTestBioCond must contain at least two ChIP-seq samples; the observed variances of individual genomic intervals cannot be calculated otherwise. Besides, a

mean-variance curve must be associated with the bioCond for deducing the prior variances before varTestBioCond could work. Notably, when fitting a mean-variance curve for a bioCond object to be passed to varTestBioCond, it's recommended to pass it alone to fitMeanVarCurve (not involving other bioCond objects). Also, if you have set occupy.only to TRUE when calling fitMeanVarCurve, you should accordingly inspect only the test results of those genomic intervals that are occupied by the bioCond, and should re-adjust *p*-values within this set of intervals (see "Examples" below).

varTestBioCond can also be used to call hypervariable and invariant intervals across biological conditions, by first combining multiple bioCond objects into a single one (see "Examples" below). Note that ChIP-seq samples in those bioConds to be combined must be first normalized to the same level (see cmbBioCond for details).

Value

This function returns an object of class c("varTestBioCond","data.frame"), recording the test results for each genomic interval by each row. The data frame consists of the following variables:

observed.mean Sample mean of the observed signal intensities.

observed.var Sample variance of the observed signal intensities.

prior.var Prior variance corresponding to the observed mean signal intensity.

fold.change Ratio of observed.var to prior.var.

pval Two sided *p*-value for the statistical significance of this fold change.

padj *P*-value adjusted for multiple testing with the "BH" method (see p.adjust), which controls false discovery rate.

Row names of the returned data frame inherit from those of cond$norm.signal. Besides, several attributes are added to the returned object:

bioCond.name Name of the bioCond object.

mean.var.curve A function representing the mean-variance curve. It accepts a vector of mean signal intensities and returns the corresponding prior variances. Note that this function has incorporated variance ratio factor of the bioCond and the min.var argument.

df A length-2 vector giving the numbers of degrees of freedom of the observed and prior variances.

References

Smyth, G.K., *Linear models and empirical bayes methods for assessing differential expression in microarray experiments.* Stat Appl Genet Mol Biol, 2004. **3**: p. Article3.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve for a set of bioCond objects; setMeanVarCurve for setting the mean-variance curve of a set of bioConds; estimatePriorDf for estimating number of prior degrees of freedom as well as adjusting variance ratio factors accordingly; estParamHyperChIP for applying the parameter estimation framework of HyperChIP; cmbBioCond for combining multiple bioConds into a single one.

plot.varTestBioCond for creating a plot to demonstrate a varTestBioCond object; diffTest for calling differential intervals between two bioCond objects; aovBioCond for calling differential intervals across multiple bioConds.

Examples

```
library(scales)
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Call hypervariable and invariant genomic intervals across biological
## replicates of the GM12891 cell line.

# Perform MA normalization and construct a bioCond to represent GM12891.
norm <- normalize(H3K27Ac, 5:6, 10:11)
GM12891 <- bioCond(norm[5:6], norm[10:11], name = "GM12891")

# Fit a mean-variance curve for GM12891 using the parametric method.
GM12891 <- fitMeanVarCurve(list(GM12891), method = "parametric",
                           occupy.only = TRUE)[[1]]
summary(GM12891)
plotMeanVarCurve(list(GM12891), subset = "occupied")

# Assess the observed variances of ChIP-seq signal intensities in GM12891.
res <- varTestBioCond(GM12891)
head(res)

# Inspect only the test results of occupied genomic intervals.
res <- res[GM12891$occupancy, ]
res$padj <- p.adjust(res$pval, method = "BH")
plot(res, padj = 0.2, col = alpha(c("black", "red"), c(0.04, 0.5)))

## Call hypervariable and invariant genomic intervals across cell lines.

# Perform MA normalization and construct bioConds to represent cell lines.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))

# Normalize the cell lines.
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Combine the cell lines into a single bioCond and use local regression to
# adaptively capture the mean-variance trend. Only genomic intervals that
# are occupied by each of the cell lines are considered to be occupied by
# the combined bioCond, which is for avoiding over-estimation of the prior
# variances.
LCLs <- cmbBioCond(conds, occupy.num = length(conds),
                   name = "lymphoblastoid_cell_lines")
LCLs <- fitMeanVarCurve(list(LCLs), method = "local",
                        occupy.only = FALSE)[[1]]
LCLs <- estimatePriorDf(list(LCLs), occupy.only = TRUE)[[1]]
summary(LCLs)
```

```
plotMeanVarCurve(list(LCLs), subset = "all")

# Assess the observed variances of ChIP-seq signal intensities across these
# cell lines.
res <- varTestBioCond(LCLs)
head(res)
plot(res, pval = 0.01, col = alpha(c("black", "red"), c(0.04, 0.5)))

# Non-occupied intervals are occupied by some of the cell lines but not all
# of them. These intervals tend to be more variable across the cell lines
# and more significant in the tests than occupied intervals.
f <- !(LCLs$occupancy)
wilcox.test(res$fold.change[f], res$fold.change[!f],
            alternative = "greater")
wilcox.test(res$pval[f], res$pval[!f], alternative = "less")

# Intervals in Y chromosome tend to be more variable across the cell lines
# and more significant in the tests than the other intervals, since the cell
# lines are of different genders.
f <- H3K27Ac$chrom == "chrY"
wilcox.test(res$fold.change[f], res$fold.change[!f],
            alternative = "greater")
wilcox.test(res$pval[f], res$pval[!f], alternative = "less")

# Make a comparison with HyperChIP.
LCLs2 <- estParamHyperChIP(LCLs, occupy.only = FALSE, prob = 0.1)
summary(LCLs)
summary(LCLs2)
res2 <- varTestBioCond(LCLs2)
hist(res$pval, breaks = 100, col = "red")
hist(res2$pval, breaks = 100, col = "red")
```

---

vstBioCond *Apply a Variance-Stabilizing Transformation to a* bioCond

---

Description

Given a bioCond object with which a mean-variance curve is associated, vstBioCond deduces a variance-stabilizing transformation (VST) based on the curve, and applies it to the signal intensities of samples contained in the bioCond, so that variances of individual genomic intervals are comparable between each other.

Usage

```
vstBioCond(x, min.var = 0, integrate.func = integrate, ...)
```

Arguments

x	A bioCond object with which a mean-variance curve has been associated (see also fitMeanVarCurve).
min.var	Lower bound of variances read from the mean-variance curve. Any variance read from the curve less than min.var will be adjusted to this value. It's primarily used for safely reading positive values from the curve and taking into account the practical significance of a signal variation.
integrate.func	A function for quadrature of functions of one variable. Any function passed to this argument must mimic the behavior of integrate (the default argument). See "Details".
...	Additional arguments to integrate.func.

Details

vstBioCond deduces the VST by applying the standard delta method to the mean-variance curve associated with the bioCond object. To be noted, applying the VST to the bioCond retains its structure matrices. More specifically, the transformed signal intensities of each genomic interval will have a covariance matrix approximately proportional to its structure matrix in the bioCond. See setWeight for a detailed description of structure matrix.

Technically, applying the VST requires the quadrature of a one-variable function, which in vstBioCond is achieved numerically. One can specify the numerical integration routine used by vstBioCond via the argument integrate.func, as long as the provided function mimics the behavior of integrate. Specifically, supposing the first three arguments to the function are f, a and b, then ret$value should be the integral of f from a to b, where ret is the object returned from the function. See integrate for details.

One of the applications of applying a VST to a bioCond is for clustering the samples contained in it. Since variances of transformed signals are comparable across genomic intervals, performing a clustering analysis on the transformed data is expected to give more reliable results than those from the original signals. Notably, to apply a clustering analysis to the VSTed signals, one typically passes the returned object from vstBioCond to distBioCond setting the method argument to "none", by which you can get a dist object recording the distance between each pair of samples of the bioCond. This procedure is specifically designed to handle VSTed bioConds and has considered the possibility that different genomic intervals may be associated with different structure matrices (see distBioCond for details). The resulting dist object can then be passed to hclust to perform a hierarchical clustering (see also "Examples").

From this perspective, vstBioCond could also be used to cluster a set of bioCond objects, by first combining them into a single bioCond and fitting a mean-variance curve for it (see "Examples" below and also cmbBioCond).

Value

vstBioCond returns a bioCond object with an extra attribute named "vst.func", which represents the VST applied to x. Signal intensities contained in the returned bioCond are obtained by applying the VST to the signal intensities in x.

The returned bioCond has the same biological condition name and occupancy states of genomic intervals as x. Besides, the structure matrix of each interval in the returned bioCond inherits from

x as well, since performing the designed VST approximately retains the original structure matrices (see "Details").

The vst.func attribute is a function that accepts a vector of signal intensities and returns the VSTed signals. To be noted, vst.func has been scaled so that the resulting transformed signals in the returned bioCond have a similar numerical range and variation level to the signal intensities in x. More specifically, the sample.mean and sample.var fields of the returned bioCond have the same arithmetic mean and geometric mean as x$sample.mean and x$sample.var, respectively. See bioCond for a detailed description of these fields.

Note also that, in principle, applying the vst.func to any bioCond object that is associated with the same mean-variance curve as is x (i.e., has the same mvcID as that of x; see also fitMeanVarCurve) effectively stabilizes the variances of its signal intensities across genomic intervals. For future reference, the vst.func itself has an attribute named "mvcID" recording the mvcID of x.

See Also

bioCond for creating a bioCond object; fitMeanVarCurve for fitting a mean-variance curve; integrate for a numerical integration routine; setWeight for a detailed description of structure matrix; cmbBioCond for combining a set of bioCond objects into a single one; distBioCond for robustly measuring the distances between samples in a bioCond; hclust for performing a hierarchical clustering on a dist object.

Examples

```
data(H3K27Ac, package = "MAnorm2")
attr(H3K27Ac, "metaInfo")

## Cluster a set of ChIP-seq samples from different cell lines (i.e.,
## individuals).

# Perform MA normalization and construct a bioCond.
norm <- normalize(H3K27Ac, 4:8, 9:13)
cond <- bioCond(norm[4:8], norm[9:13], name = "all")

# Fit a mean-variance curve.
cond <- fitMeanVarCurve(list(cond), method = "local",
                        occupy.only = FALSE)[[1]]
plotMeanVarCurve(list(cond), subset = "all")

# Apply a variance-stabilizing transformation and associate a constant
# function with the resulting bioCond as its mean-variance curve.
vst_cond <- vstBioCond(cond)
vst_cond <- setMeanVarCurve(list(vst_cond), function(x)
                            rep_len(1, length(x)), occupy.only = FALSE,
                            method = "constant prior")[[1]]
plotMeanVarCurve(list(vst_cond), subset = "all")

# Measure the distance between each pair of samples and accordingly perform
# a hierarchical clustering. Note that biological replicates of each cell
# line are clustered together.
d1 <- distBioCond(vst_cond, method = "none")
plot(hclust(d1, method = "average"), hang = -1)
```

```
# Measure the distances using only hypervariable genomic intervals. Note the
# change of scale of the distances.
res <- varTestBioCond(vst_cond)
f <- res$fold.change > 1 & res$pval < 0.05
d2 <- distBioCond(vst_cond, subset = f, method = "none")
plot(hclust(d2, method = "average"), hang = -1)

## Cluster a set of individuals.

# Perform MA normalization and construct bioConds to represent individuals.
norm <- normalize(H3K27Ac, 4, 9)
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
conds <- normBioCond(conds)

# Group the individuals into a single bioCond and fit a mean-variance curve
# for it.
cond <- cmbBioCond(conds, name = "all")
cond <- fitMeanVarCurve(list(cond), method = "local",
                        occupy.only = FALSE)[[1]]
plotMeanVarCurve(list(cond), subset = "all")

# Apply a variance-stabilizing transformation and associate a constant
# function with the resulting bioCond as its mean-variance curve.
vst_cond <- vstBioCond(cond)
vst_cond <- setMeanVarCurve(list(vst_cond), function(x)
                            rep_len(1, length(x)), occupy.only = FALSE,
                            method = "constant prior")[[1]]
plotMeanVarCurve(list(vst_cond), subset = "all")

# Measure the distance between each pair of individuals and accordingly
# perform a hierarchical clustering. Note that GM12891 and GM12892 are
# actually a couple and they are clustered together.
d1 <- distBioCond(vst_cond, method = "none")
plot(hclust(d1, method = "average"), hang = -1)

# Measure the distances using only hypervariable genomic intervals. Note the
# change of scale of the distances.
res <- varTestBioCond(vst_cond)
f <- res$fold.change > 1 & res$pval < 0.05
d2 <- distBioCond(vst_cond, subset = f, method = "none")
plot(hclust(d2, method = "average"), hang = -1)

## Perform differential analysis on bioConds that have gone through a
## variance-stabilizing transformation.

# Perform MA normalization and construct bioConds to represent cell lines
# (i.e., individuals).
norm <- normalize(H3K27Ac, 4, 9)
```

```
norm <- normalize(norm, 5:6, 10:11)
norm <- normalize(norm, 7:8, 12:13)
conds <- list(GM12890 = bioCond(norm[4], norm[9], name = "GM12890"),
              GM12891 = bioCond(norm[5:6], norm[10:11], name = "GM12891"),
              GM12892 = bioCond(norm[7:8], norm[12:13], name = "GM12892"))
autosome <- !(H3K27Ac$chrom %in% c("chrX", "chrY"))
conds <- normBioCond(conds, common.peak.regions = autosome)

# Fit a mean-variance curve.
conds <- fitMeanVarCurve(conds, method = "parametric", occupy.only = TRUE)
plotMeanVarCurve(conds, subset = "occupied")

# Apply a variance-stabilizing transformation.
vst_conds <- list(GM12890 = vstBioCond(conds$GM12890))
vst.func <- attr(vst_conds$GM12890, "vst.func")
temp <- matrix(vst.func(as.numeric(conds$GM12891$norm.signal)),
               nrow = nrow(norm))
vst_conds$GM12891 <- bioCond(temp, norm[10:11], name = "GM12891")
temp <- matrix(vst.func(as.numeric(conds$GM12892$norm.signal)),
               nrow = nrow(norm))
vst_conds$GM12892 <- bioCond(temp, norm[12:13], name = "GM12892")

# Associate a constant function with the resulting bioConds as their
# mean-variance curve.
vst_conds <- setMeanVarCurve(vst_conds, function(x) rep_len(1, length(x)),
                             occupy.only = TRUE, method = "constant prior")
plotMeanVarCurve(vst_conds, subset = "occupied")

# Make a comparison between GM12891 and GM12892.
res1 <- diffTest(conds$GM12891, conds$GM12892)
res2 <- diffTest(vst_conds$GM12891, vst_conds$GM12892)

# Examine the consistency of analysis results between using ordinary and
# VSTed signal intensities. Here we map p-values together with observed
# directions of signal changes to the standard normal distribution.
z1 <- qnorm(res1$pval / 2)
z1[res1$Mval > 0] <- -z1[res1$Mval > 0]
z2 <- qnorm(res2$pval / 2)
z2[res2$Mval > 0] <- -z2[res2$Mval > 0]
plot(z1, z2, xlab = "Ordinary", ylab = "VSTed")
abline(a = 0, b = 1, lwd = 2, lty = 5, col = "red")
cor(z1, z2)
cor(z1, z2, method = "sp")

# Simultaneously compare GM12890, GM12891 and GM12892 cell lines.
res1 <- aovBioCond(conds)
res2 <- aovBioCond(vst_conds)

# Examine the consistency of analysis results between using ordinary and
# VSTed signal intensities by mapping p-values to the standard normal
# distribution.
z1 <- qnorm(res1$pval, lower.tail = FALSE)
z1[z1 == Inf] <- 39
```

```
z2 <- qnorm(res2$pval, lower.tail = FALSE)
z2[z2 == Inf] <- 39
plot(z1, z2, xlab = "Ordinary", ylab = "VSTed")
abline(a = 0, b = 1, lwd = 2, lty = 5, col = "red")
cor(z1, z2)
cor(z1, z2, method = "sp")
```

Index