baySeq

March 24, 2012

`CDPost`	*'countData' object derived from data file 'simData' with estimated likelihoods of differential expression.*

Description

This 'countData' object is derived from the data set 'simData' and contains the estimated likelihoods of differential expression. This data set is intended to be used to speed the processing of the examples.

Usage

```
CDPost
```

Format

A 'countData' object.

Source

Simulation.

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

`CDPriors` *'countData' object derived from data file 'simData' with estimated priors.*

Description

This 'countData' object is derived from the data set 'simData' and contains the estimated priors. This data set is intended to be used to speed the processing of the examples.

Usage

```
CDPriors
```

Format

A 'countData' object.

Source

Simulation.

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

`baySeq-classes` *baySeq - classes*

Description

The `countData` class is used to define summaries of count data and establishing prior and posterior parameters on distributions defined upon the count data.

Slots

Objects of the `'countData'` class should contain the following components:

`data`:	Count data (matrix).
`replicates`:	The replicate structure of the data. Stored as a factor, but can be given in any form.
`libsizes`:	Vector of library size for each sample.
`groups`:	Group (model) structure to test on the data (list).
`annotation`:	Annotation data for each count (data.frame).
`priorType`:	Character string describing the type of prior information available in slot `'priors'`.
`priors`:	Prior parameter information. Calculated by the functions described in `getPriors`.
`posteriors`:	Estimated posterior likelihoods for each group (matrix). Calculated by the functions described in `getL`
`estProps`:	Estimated proportion of tags belonging to each group (numeric). Calculated by the functions described
`nullPosts`:	If calculated, the posterior likelihoods for the data having no true expression of any kind.
`seglens`:	Lengths of segments containing the counts described in `data`. A matrix, but may be initialised with a v

Details

The `seglens` slot describes, for each row of the `data` object, the length of the 'segment' that contains the number of counts described by that row. For example, if we are looking at the number of hits matching genes, the `seglens` object would consist of transcript lengths. Exceptionally, we may want to use different segment lengths for different samples and so the slot takes the form of a matrix. If the matrix has only one column, it is duplicated for all samples. Otherwise, it should have the same number of columns as the '@data' slot. If the slot is the empty matrix, then it is assumed that all segments have the same length.

Methods

The standard methods 'new', 'dim', '[', 'show' and 'rbind' have been defined for this class. The methods 'groups', 'groups<-', 'replicates' and 'replicates<-' have also been defined in order to access and modify these slots, and their use is recommended.

Author(s)

Thomas J. Hardcastle

Examples

```
#load test data
data(simData)

# Create a 'countData' object from test data.
replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "simB", "simB", "simB", "
groups <- list(NDE = c(1,1,1,1,1,1,1,1,1,1), DE = c(1,1,1,1,1,2,2,2,2,2))
CD <- new("countData", data = simData, replicates = replicates, groups = groups)

#estimate library sizes for countData object
CD@libsizes <- getLibsizes(CD)

CD[1:10,]
dim(CD)
```

---

baySeq-package *Empirical Bayesian analysis of patterns of differential expression in count data.*

Description

This package is intended to identify differential expression in high-throughput 'count' data, such as that derived from next-generation sequencing machines. We achieve this by empirical bayesian methods, first bootstrapping to estimate prior parameters from the data and then assessing posterior likelihoods of the models proposed.

Details

Package:	baySeq
Type:	Package
Version:	1.1.1
Date:	2009-16-05
License:	GPL-3
LazyLoad:	yes

To use the package, construct a `countData` object and use the functions documented in getPriors to empirically determine priors on the data. Then use the functions documented in getLikelihoods to establish posterior likelihoods for the models proposed. A few convenience functions, `getTPs` and `topCounts` are also included.

The package (optionally) makes use of the 'snow' package for parallelisation of computationally intensive functions. This is highly recommended for large data sets.

See the vignette for more details.

Author(s)

Thomas J. Hardcastle

Maintainer: Thomas J. Hardcastle <tjh48@cam.ac.uk>

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

Examples

```
# See vignette for more examples.

# load test data
data(simData)


# replicate structure of data
replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "simB", "simB", "simB", "

# define hypotheses on data
groups <- list(NDE = c(1,1,1,1,1,1,1,1,1,1), DE = c(1,1,1,1,1,2,2,2,2,2))

# construct 'countData' object
CD <- new("countData", data = simData, replicates = replicates, groups =
groups)

#estimate library sizes for countData object
CD@libsizes <- getLibsizes(CD)

# estimate prior distributions on 'countData' object using negative binomial
# method. Other methods are available - see getPriors
CDPriors <- getPriors.NB(CD, cl = NULL)
```

```
# estimate posterior likelihoods for each row of data belonging to each hypothesis
CDPost <- getLikelihoods(CDPriors, cl = NULL)

# display the rows of data showing greatest association with the second
# hypothesis (differential expression)
topCounts(CDPost, group = "DE", number = 10)

# find true positive selection rate
getTPs(CDPost, group = "DE", TPs = 1:100)[1:100]
```

`bimodalSep` *A function that, given a numeric vector, finds the value which splits the data into two sets of minimal total variance.*

Description

This function takes a numeric vector and finds the value which splits the data into two sets of minimal total variance. It is principally intended to be a quick and easy way of separating bimodally distributed data.

Usage

```
bimodalSep(z, weights = NULL, bQ = c(0,1))
```

Arguments

`z`	A numeric vector containing the data to be split.
`weights`	Possible weightings on the values in z for calculating the variance.
`bQ`	Lower and upper limits on the quartile of z in which a separating value is sought. See Details.

Details

This function is intended to give a quick and easy way of splitting bimodally distributed data. Where there are large outliers in the data, it may be that the value which minimises the variance does not split the bimodal data but isolates the outliers. The `'bQ'` parameter can be used to ensure that the split occurs within some range of quantiles of the data.

Value

Numeric.

Author(s)

Thomas J. Hardcastle

Examples

```
bimodalSep(c(rnorm(200, mean = c(5,7), sd = 1)))
```

`getLibsizes` *Estimates library scaling factors (library sizes) for count data.*

Description

This function estimates the library scaling factors that should be used for either a 'countData', or a matrix of counts and replicate information.

Usage

```
getLibsizes(cD, data, replicates, subset = NULL, estimationType = c("quantile",
```

Arguments

`cD`	A `countData` object.
`data`	A matrix of count values. Ignored if 'cD' is given.
`replicates`	A replicate structure for the data given in 'data'. Ignored if 'cD' is given.
`subset`	A numerical vector indicating the rows of the 'countData' object that should be used to estimate library scaling factors.
`estimationType`	One of 'quantile', 'total', or 'edgeR'. Partial matching is allowed. See Details.
`quantile`	A value between 0 and 1 indicating the level of trimming that should take place. See Details.
`...`	Additional parameters to be passed to the 'edgeR' calcNormFactors function.

Details

This function estimates the library scaling factors (surrogates for library size) in one of several ways, depending on the 'estimationType' argument. 'total' will give the library sizes by summing all counts in each sample. 'quantile' will give a library scaling factor by the method of Bullard et al (Bioinformatics 2010), summing all counts in each sample whose value below the qth quantile of non-zero counts for that sample. 'edgeR' uses the Trimmed Mean of M-vales (TMM) method of Robinson \& Oshlack (Genome Biology, 2010) via the 'edgeR' calcNormFactors function; other options are available through this function.

If a `countData` object 'cD' is given, the library sizes will be inferred from this. Alternatively, a matrix of count values (columns are libraries) and a replicate structure (a vector defining which samples belong to which replicate group) can be given.

Value

A numerical vector of library sizes (scaling factors) for each library in the data.

Author(s)

Thomas J. Hardcastle

See Also

`countData`

Examples

```
data(simData)
replicates <- c(1,1,1,1,1,2,2,2,2,2)
groups <- list(c(1,1,1,1,1,1,1,1,1,1), c(1,1,1,1,1,2,2,2,2,2))
CD <- new("countData", data = simData, replicates = replicates, groups = groups)

CD@libsizes <- getLibsizes(CD)
```

`getLikelihoods` *Finds posterior likelihoods for each count as belonging to some hypothesis.*

Description

These functions calculate posterior probabilities for each of the 'counts' in the countDP object belonging to each of the groups specified. The choice of function depends on the prior belief about the underlying distribution of the data. It is essential that the method used for calculating priors matches the method used for calculating the posterior probabilites.

For a comparison of the methods, see Hardcastle & Kelly, 2009.

Usage

```
getLikelihoods(cD, prs, pET = "BIC", marginalise = FALSE, subset = NULL,
priorSubset = NULL, verbose = TRUE, ..., cl)
getLikelihoods.Dirichlet(cD, prs, pET = "BIC", marginalise = FALSE, subset = NUL
priorSubset = NULL, verbose = TRUE, cl)
getLikelihoods.Pois(cD, prs, pET = "BIC", marginalise = FALSE, subset = NULL,
priorSubset = NULL, distpriors = FALSE, verbose = TRUE, cl)
getLikelihoods.NB(cD, prs, pET = "BIC", marginalise = FALSE, subset = NULL,
priorSubset = NULL, bootStraps = 1, conv = 1e-4, nullData = FALSE,
returnAll = FALSE, returnPD = FALSE, verbose = TRUE, cl)
```

Arguments

`cD`	An object of type `countData`, or descending from this class.
`prs`	(Initial) prior probabilities for each of the groups in the 'countDP' object. Should sum to 1, unless nullData is TRUE, in which case it should sum to less than 1.
`pET`	What type of prior re-estimation should be attempted? Defaults to "BIC"; "none" and "iteratively" are also available.
`marginalise`	Should an attempt be made to numerically marginalise over a prior distribution iteratively estimated from the posterior distribution? Defaults to FALSE, as in general offers little performance gain and increases computational cost considerably.
`subset`	Numeric vector giving the subset of counts for which posterior likelihoods should be estimated.
`priorSubset`	Numeric vector giving the subset of counts which may be used to estimate prior probabilities on each of the groups. See Details.
`distpriors`	Should the Poisson method use an empirically derived distribution on the prior parameters of the Poisson distribution, or use the mean of the maximum likelihood estimates (default).

bootStraps	How many iterations of bootstrapping should be used in the (re)estimation of priors in the negative binomial method.
conv	If not null, bootstrapping iterations will cease if the mean squared difference between posterior likelihoods of consecutive bootstraps drops below this value.
nullData	If TRUE, looks for segments or counts with no true expression. See Details.
returnAll	If TRUE, and bootStraps > 1, then instead of returning a single countData object, the function returns a list of countData objects; one for each bootstrap. Largely used for debugging purposes.
returnPD	If TRUE, then the function returns the (log) likelihoods of the data given the models, rather than the posterior (log) likelihoods of the models given the data. Not recommended for general use.
verbose	Should status messages be displayed? Defaults to TRUE.
cl	A SNOW cluster object.
...	Any additional information to be passed by the 'getLikelihoods' wrapper function to the individual functions which calculate the likelihoods.

Details

These functions estimate, under the assumption of various distributions, the (log) posterior likelihoods that each count belongs to a group defined by the @group slot of the countData object. The posterior likelihoods are stored on the natural log scale in the @posteriors slot of the countData object generated by this function. This is because the posterior likelihoods are calculated in this form, and ordering of the counts is better done on these log-likelihoods than on the likelihoods.

If 'pET = "none"' then no attempt is made to re-estimate the prior likelihoods given in the 'prs' variable. However, if 'pET = "BIC"', then the function will attempt to estimate the prior likelihoods by using the Bayesian Information Criterion to identify the proportion of the data best explained by each model and taking these proportions as prior. Alternatively, an iterative re-estimation of priors is possible ('pET = "iteratively"'), in which an inital estimate for the prior likelihoods of the models is used to calculated the posteriors and then the priors are updated by taking the mean of the posterior likelihoods for each model across all data. This often works well, particularly if the 'BIC' method is used (see Hardcastle & Kelly 2010 for details). However, if the data are sufficiently non-independent, this approach may substantially mis-estimate the true priors. If it is possible to select a representative subset of the data by setting the variable 'subsetPriors' that is sufficiently independent, then better estimates may be acquired.

The Dirichlet and Poisson methods produce almost identical results in simulation. The Negative Binomial method produces results with much lower false discovery rates, but takes considerably longer to run.

Filtering the data may be extremely advantageous in reducing run time. This can be done by passing a numeric vector to 'subset' defining a subset of the data for which posterior likelihoods are required.

If 'nullData = TRUE', the algorithm attempts to find those counts or segments that have no true expression in all samples. This means that there is another, implied group where all samples are equal. The prior likelihoods given in the 'prs' object must thus sum to less than 1, with the residual going to this group.

See Hardcastle & Kelly (2010) for a full comparison of the methods.

A 'cluster' object is strongly recommended in order to parallelise the estimation of posterior likelihoods, particularly for the negative binomial method. However, passing NULL to the cl variable will allow the functions to run in non-parallel mode.

The 'getLikelihoods' function will infer the correct distribution to use from the information stored in the '@priors' slot of the countData object 'sD' and call the appropriate function.

Value

A countData object.

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

See Also

countData, getPriors, topCounts, getTPs

Examples

```
# See vignette for more examples.

# If we do not wish to parallelise the functions we set the cluster
# object to NULL.

cl <- NULL

# Alternatively, if we have the 'snow' package installed we
# can parallelise the functions. This will usually (not always) offer
# significant performance gain.

## Not run: try(library(snow))
## Not run: try(cl <- makeCluster(4, "SOCK"))

# load test data
data(simData)

# Create a {countData} object from test data.

replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "simB", "simB", "simB", "
groups <- list(NDE = c(1,1,1,1,1,1,1,1,1,1), DE = c(1,1,1,1,1,2,2,2,2,2))
CD <- new("countData", data = simData, replicates = replicates, groups = groups)

#estimate library sizes for countData object
CD@libsizes <- getLibsizes(CD)

# Get priors for negative binomial method
## Not run: CDPriors <- getPriors.NB(CD, samplesize = 10^5, estimation =
"QL", cl = cl)
## End(Not run)

# To speed up the processing of this example, we have already created
# the `CDPriors' object.
data(CDPriors)

# Get likelihoods for data with negative binomial method.
```

```
CDPost <- getLikelihoods.NB(CDPriors, prs = c(0.5, 0.5),
pET = "BIC", marginalise = FALSE, bootStraps = 1, cl = cl)

try(stopCluster(cl))
```

`getPosteriors` *An internal function in the baySeq package for calculating posterior likelihoods given likelihoods of the data.*

Description

For likelihoods of the data given a set of models, this function calculates the posterior likelihoods of the models given the data. An internal function of baySeq, which should not in general be called by the user.

Usage

```
getPosteriors(ps, prs, pET = "none", marginalise = FALSE, groups, priorSubset =
1e-5, cl = cl)
```

Arguments

`ps`	A matrix containing likelihoods of the data for each count (rows) under each model (columns).
`prs`	(Initial) prior probabilities for each of the models.
`pET`	What type of prior re-estimation should be attempted? Defaults to "none"; "BIC" and "iteratively" are also available.
`marginalise`	Should an attempt be made to numerically marginalise over a prior distribution iteratively estimated from the posterior distribution? Defaults to FALSE, as in general offers little performance gain and increases computational cost considerably.
`groups`	Group structure from which likelihoods in `'ps'` were defined.
`priorSubset`	If `'estimatePriors = TRUE'`, what subset of the data should be used to re-estimate the priors? Defaults to NULL, implying all data will be used.
`maxit`	What is the maximum number of iterations that should be tried if we are bootstrapping prior probabilities from the data?
`accuracy`	How small should the difference in estimated priors be before we stop bootstrapping.
`cl`	A SNOW cluster object.

Details

An internal function, that will not in general be called by the user. It takes the log-likelihoods of the data given the models being tested and returns the posterior likelihoods of the models.

The function may attempt to estimate the prior likelihoods either by using the Bayesian Information Criterion (`'pET = "BIC"'`) to identify the proportion of the data best explained by each model and taking these proportions as prior. Alternatively, an iterative re-estimation of priors is possible (`'pET = "iteratively"'`, in which an inital estimate for the prior likelihoods of the models is used to calculated the posteriors and then the priors are updated by taking the mean of the posterior likelihoods for each model across all data.

Value

A list containing posteriors: estimated posterior likelihoods of the model for each count (log-scale) priors: estimated (or given) prior probabilities of the model

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

See Also

getLikelihoods

Examples

```
# Simulate some log-likeihoods of data given models (each model
# describes one column of the 'ps' object).
ps <- log(rbind(
                cbind(runif(10000, 0, 0.1), runif(10000, 0.3, 0.9)),
                cbind(runif(10000, 0.4, 0.9), runif(1000, 0, 0.2))))

# get posterior log-likelihoods of model, estimating prior likelihoods
# of each model from the data.

pps <- getPosteriors(ps, prs <- c(0.5, 0.5), pET = "none", cl =
NULL)

pps$priors

pps$posteriors[1:10,]
```

---

getPriors *Estimates prior parameters for the underlying distributions of 'count' data.*

Description

These functions estimate, via maximum likelihood methods, the parameters of the underlying distributions for the different methods of describing the 'count' data.

Usage

```
getPriors.Dirichlet(cD, samplesize = 1e5, perSE = 1e-1, maxit = 1e6,
verbose = TRUE)
getPriors.Pois(cD, samplesize = 1e5, perSE = 1e-1, takemean = TRUE,
maxit = 1e5, weights = NULL, verbose = TRUE, cl)
getPriors.NB(cD, samplesize = 1e5, samplingSubset = NULL,
```

```
equalDispersions = TRUE, estimation = "QL", verbose = TRUE, zeroML =
FALSE, cl, ...)
```

Arguments

cD	A countData object.
samplesize	How large a sample should be taken in estimating the priors?
samplingSubset	If given, the priors will be sampled only from the subset specified.
perSE	What should the relative standard error of the estimated parameters fall below?
maxit	Over how many iterations (at most) should we take samples and re-estimate the priors in order to achieve convergence?
takemean	If TRUE (recommended), we take the mean of the estimated priors to define a gamma distribution. If FALSE, we use all estimated priors to define an empirical distribtion on the parameters of the gamma distribution.
weights	If not NULL, specifies a weighting on the random sampling of rows of the 'cD' object used to estimate parameters of the underlying distribution.
equalDispersions	Should we assume equal dispersions of data across all groups in the 'cD' object? Defaults to TRUE; see Details.
estimation	Defaults to "QL", indicating quasi-likelihood estimation of priors. Currently, the only other possibilities are "ML", a maximum-likelihood method, and "edgeR", the moderated dispersion estimates produced by the 'edgeR' package. See Details.
verbose	Should status messages be displayed? Defaults to TRUE.
zeroML	Should parameters from zero data (rows that within a group are all zeros) be estimated using maximum likelihood methods (which will result in zeros in the parameters? See Details.
cl	A SNOW cluster object.
...	Additional parameters to be passed to the estimateTagwiseDisp function if 'estimation = "edgeR"'.

Details

These functions empirically estimate prior parameters for different distributions used in estimating posterior likelihoods of each count belonging to a particular group. The choice of which function to use for estimating the prior parameters will depend on the choice of which method is being used to estimate the posterior likelihoods (see getLikelihoods).

For priors estimated for the negative binomial methods, three options are available. Differences in the options focus on the way in which the dispersion is estimated for the data. In simulation studies, quasi-likelihood methods ('estimation = "QL"') performed best and so these are used by default. Alternatives are maximum-likelihood methods ('estimation = "ML"'), and the 'edgeR' packages moderated dispersion estimates ('estimation = "edgeR"').

The priors estimated for the negative binomial methods ('getPriors.NB') may assume that the dispersion of data for a given row is identical for all group structures defined in 'cD@groups' ('equalDispersions = TRUE'). Alternatively, the dispersions may be estimated individually for each group structure ('equalDispersions = FALSE'). Unless there is a strong reason for believing that the data are differently dispersed between groups, 'equalDispersions

= TRUE' is recommended. If 'estimation = "edgeR"' then this parameter is ignored and dispersion is assumed identical for all group structures.

If all counts in a given row for a given group are zero, then maximum and quasi-likelihood estimation methods will result in a zero parameter for the mean. In analyses where segment length is a factor, this makes it hard to differentiate between (for example) a region which contains no reads but is only ten bases long and one which likewise contains no reads but is ten megabases long. If 'zeroML' is FALSE, therefore, the dispersion is set to 1 and the mean estimated as the value that leaves the likelihood of zero data at fifty percent.

A 'cluster' object is recommended in order to estimate the priors for the negative binomial distribution. Passing NULL to this variable will cause the function to run in non-parallel mode.

getPriors.Dirichlet and getPriors.Pois will issue warnings if the estimation of any priors fails to achieve less than the relative standard error specified in the maximum number of iterations.

Value

A countData object.

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

See Also

countData, getLikelihoods

Examples

```
# See vignette for more examples.


# If we do not wish to parallelise the functions we set the cluster
# object to NULL.

cl <- NULL

# Alternatively, if we have the 'snow' package installed we
# can parallelise the functions. This will usually (not always) offer
# significant performance gain.

## Not run: try(library(snow))
## Not run: try(cl <- makeCluster(4, "SOCK"))

# load test data
data(simData)

# Create a {countData} object from test data.

replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "simB", "simB", "simB", "
groups <- list(NDE = c(1,1,1,1,1,1,1,1,1,1), DE = c(1,1,1,1,1,2,2,2,2,2))
CD <- new("countData", data = simData, replicates = replicates, groups = groups)
```

```
#estimate library sizes for countData object
CD@libsizes <- getLibsizes(CD)

# Get priors for negative binomial method
CDPriors <- getPriors.NB(CD, samplesize = 10^5, estimation = "QL", cl = cl)
```

`getTPs` *Gets the number of true positives in the top n counts selected by ranked posterior likelihoods*

Description

If the true positives are known, this function will return a vector, the ith member of which gives the number of true positives identified if the top i counts, based on estimated posterior likelihoods, are chosen.

Usage

```
getTPs(cD, group, decreasing = TRUE, TPs)
```

Arguments

`cD`	`countData` object, containing posterior likelihoods for each group.
`group`	Which group should we give the counts for? See Details.
`decreasing`	Ordering on posterior likelihoods.
`TPs`	Known true positives.

Details

In the rare (or simulated) cases where the true positives are known, this function will calculate the number of true positives selected at any cutoff.

The 'group' can be defined either as the number of the element in `'cD@groups'` or as a string which will be partially matched to the names of the `'cD@groups'` elements.If group = NULL, then the function looks at the posterior likelihoods that the data have no true differential expression (if calculated).

Value

A vector, the ith member of which gives the number of true positives identified if the top i counts are chosen.

Author(s)

Thomas J. Hardcastle

See Also

`countData`

Examples

```
# See vignette for more examples.

# We load in a `countData' object containing the estimated posterior
# likelihoods of expression (see `getLikelihoods').

data(CDPost)

try(stopCluster(cl))

# If the first hundred rows in the 'simData' matrix are known to be
# truly differentially expressed (the second hypothesis defined in the
# 'groups' list) then we find the number of true positives for the top n
# genes selected as the nth member of

getTPs(CDPost, group = "DE", decreasing = TRUE, TPs = 1:100)
```

`mobAnnotation` *Annotation data for a set of small RNA loci derived from sequencing of grafts of Arabidopsis thaliana intended for differential expression analyses.*

Description

This data set is a data.frame ('mobAnnotation') describing three thousand small RNA loci identified in a set of Arabidopsis grafting experiments.

The data acquired through sequencing for these loci is found in data file 'mobData'.

Usage

```
mobAnnotation
```

Format

A data.frame defining chromosome and position of the sRNA loci.

Source

Illumina sequencing.

References

Molnar A. and Melnyk C.W. et al. Small silencing RNAs in plants are mobile and direct epigenetic modification in recipient cells. Science (2010)

See Also

`mobData`

mobData *Data from a set of small RNA sequencing experiments carried out on grafts of Arabidopsis thaliana intended for differential expression analyses.*

Description

This data set is a matrix ('mobData') of counts acquired for three thousand small RNA loci from a set of Arabidopsis grafting experiments. Three different biological conditions exist within these data; one in which a Dicer 2,3,4 triple mutant shoot is grafted onto a Dicer 2,3,4 triple mutant root (SL236 and SL260), one in which a wild-type shoot is grafted onto a wild-type root (SL239 and SL240), and one in which a wild-type shoot is grafted onto a Dicer 2,3,4 triple mutant root (SL237 and SL238). Dicer 2,3,4 is required for the production of 22nt and 24nt small RNAs, as well as some 21nt ones. Consequently, if we detect differentially expressed sRNA loci in the root stock of the grafts, we can make inferences about the mobility of small RNAs.

The annotation of the loci from which these data derive is in data file 'mobAnnotation'.

Usage

```
mobData
```

Format

A matrix of which each of the six columns represents a sample, and each row an sRNA locus (acquired by sequencing).

Source

Illumina sequencing.

References

Molnar A. and Melnyk C.W. et al. Small silencing RNAs in plants are mobile and direct epigenetic modification in recipient cells. Science (2010)

See Also

mobAnnotation

plotMA.CD *'MA'-plot for count data.*

Description

This function creates an MA-plot from two sets of samples. For those data where the log-ratio is infinite (because in one set of sample data all observed counts are zero), we plot instead the log-values of the other group.

Usage

```
plotMA.CD(cD, samplesA, samplesB, normaliseData = TRUE, scale = NULL,
xlab = "A", ylab = "M", ...)
```

Arguments

cD	A countData object.
samplesA	Either a character vector, identifying sample set A by either replicate name or sample name, or a numerical vector giving the columns of data in the 'countData' object that forms sample set A. See Details.
samplesB	Either a character vector, identifying sample set B by either replicate name or sample name, or a numerical vector giving the columns of data in the 'countData' object that forms sample set B. See Details.
normaliseData	Should the data be normalised by library size before computing log-ratios? Defaults to TRUE.
scale	If given, defines the scale on which the log-ratios will be plotted. Defaults to NULL, implying that the scale will be calculated by the function.
xlab	Label for the X-axis. Defaults to "A".
ylab	Label for the Y-axis. Defaults to "M".
...	Any other parameters to be passed to the plot function.

Details

The samples sets can be identified either by a numeric vector which specifies the columns of data from the countData object 'cD', or by a character vector. If a character vector is used, the members of the character vector will first be searched for in the @replicates slot of the 'cD' object. Any members of the vector not found in the replicates slot, will be searched for in the column names of the @data slot of the 'cD' object. Different classes of vector can be used for 'samplesA' and 'samplesB', as shown in the example below.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

countData

Examples

```
data(simData)

replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "simB", "simB", "simB", "
groups <- list(NDE = c(1,1,1,1,1,1,1,1,1,1), DE = c(1,1,1,1,1,2,2,2,2,2))
CD <- new("countData", data = simData, replicates = replicates, groups = groups)

#estimate library sizes for countData object
```

```
CD@libsizes <- getLibsizes(CD)

#MA-plot comparing replicate groups
plotMA.CD(CD, samplesA = "simA", samplesB = 6:10)
```

`plotPosteriors` *Plots the posterior likelihoods estimated for a 'countData' object against the log-ratios observed between two sets of sample data.*

Description

This function plots the posterior likelihoods estimated for a 'countData' object against the log-ratios observed between two sets of sample data. For those data where the log-ratio is infinite (because in one set of sample data all observed counts are zero), we plot instead the log-values of the other group.

Usage

```
plotPosteriors(cD, group, samplesA, samplesB, ...)
```

Arguments

`cD`	A `countData` object, for which posterior likelihoods have been estimated (see `getPosteriors`).
`group`	From which group (as defined in the `'cD@groups'` slot) should posterior likelihoods be shown? Can be defined either as the number of the element in `'cD@groups'` or as a string which will be partially matched to the names of the `'cD@groups'` elements.
`samplesA`	A numerical vector giving the columns of data in the `'countData'` object that forms sample set A.
`samplesB`	A numerical vector giving the columns of data in the `'countData'` object that forms sample set B.
`...`	Any other parameters to be passed to the `plot` function.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

`getPosteriors`

Examples

```
# We load in a `countData' object containing the estimated posterior
# likelihoods of expression (see `getLikelihoods').

data(CDPost)

plotPosteriors(CDPost, group = "DE", samplesA = 1:5, samplesB = 6:10)

# equivalent to plotPosteriors(CDPost, group = 2, samplesA = 1:5, samplesB = 6:10)
```

`plotPriors` *Plots the density of the log values estimated for the mean rate in the prior data for the Negative Binomial approach to detecting differential expression*

Description

This function plots the density of the log values estimated for the mean rate in the data used to estimate a prior distribution for data under the assumption of a Negative Binomial distribution. This function is useful for looking for bimodality of the distributions, and thus determining whether we should try and identify data with no true expression.

Usage

```
plotPriors(cD, group)
```

Arguments

`cD`	`countData` object, for which priors have been estimated using the assumption of a Negative Binomial distribution (see `getPriors.NB`).
`group`	Which group should we plot the priors for? In general, should be the group that defines non-differentially expressed data. Can be defined either as the number of the element in `'cD@groups'` or as a string which will be partially matched to the names of the `'cD@groups'` elements.

Details

If the plot of the data appears bimodal, then it may be sensible to try and look for data with no true expression by using the option `nullPosts = TRUE` in `getLikelihoods.NBboot`.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

`getPriors.NB`, `getLikelihoods.NB`

Examples

```
# We load in a `countData' object containing the estimated priors (see `getPriors').

data(CDPriors)

try(stopCluster(cl))

plotPriors(CDPriors, group = "NDE")
```

`simData` *Simulated data for testing the baySeq package methods; simulated counts and library sizes from a pairwise differential expression analysis.*

Description

This data set is a matrix ('simData') of simulated counts from a simple pairwise expression analysis. It is simulated according to a negative binomial distribution with varying parameters for each row. The first hundred rows of the data are truly differentially expressed, the remainder have no differential expression.

Usage

```
simData
```

Format

A matrix of which each of the ten columns represents a sample, and each row some discrete data (acquired by sequencing).

Source

Simulation.

References

Hardcastle T.J., and Kelly, K. baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data. BMC Bioinformatics (2010)

`topCounts` *Get the top counts corresponding to some group from a 'countData' object*

Description

Takes posterior likelihoods and returns the counts with highest (or lowest) likelihood of association with a given group.

Usage

```
topCounts(cD, group, decreasing = TRUE, number = 10, normaliseData = FALSE)
```

Arguments

cD	countData object, containing posterior likelihoods for each group.
group	Which group should we give the counts for? See Details.
decreasing	Ordering on posterior likelihoods.
number	How many results should be returned?
normaliseData	Should the displayed counts be normalised by library size? Defaults to FALSE.

Value

A dataframe of the top counts associated with some model (group), described by annotation drawn from the '@annotation' slot of the 'cD' object and the raw data from the '@data' slot, together with the posterior likelihoods and false discovery rates.

The argument 'group' can be specified either as a number, giving the index of an element in the cD@groups list, or as a character string identifying an element by name. Partial matching is allowed. If group = NULL, then the function looks at the posterior likelihoods that the data have no true differential expression (if calculated).

Author(s)

Thomas J. Hardcastle

See Also

countData

Examples

```
# We load in a `countData' object containing the estimated posterior
# likelihoods of expression (see `getLikelihoods').

data(CDPost)

# Report the top ten rows of data that have highest likelihood of belonging to
# group 2 of the data (i.e., differentially expressed)

topCounts(CDPost, group = "DE", number = 10)

# equivalently...
topCounts(CDPost, group = 2, number = 10)
```

Index