Package: baySeq (via r-universe)

November 1, 2026

Type Package

Title Empirical Bayesian analysis of patterns of differential expression in count data

Version 2.47.0

Date 2012-12-20

Depends R (>= 2.3.0), methods

Imports edgeR, GenomicRanges, abind, parallel, graphics, stats, utils

Suggests BiocStyle, BiocGenerics

Description This package identifies differential expression in high-throughput 'count' data, such as that derived from next-generation sequencing machines, calculating estimated posterior likelihoods of differential expression (or more complex hypotheses) via empirical Bayesian methods.

License GPL-3

LazyLoad yes

biocViews Sequencing, DifferentialExpression, MultipleComparison, SAGE, Bayesian, Coverage

URL https://github.com/samgg/baySeq

BugReports https://github.com/samgg/baySeq/issues

RoxygenNote 7.2.3

Repository https://bioc.r-universe.dev

Date/Publication 2026-04-28 13:02:50 UTC

RemoteUrl https://github.com/bioc/baySeq

RemoteRef HEAD

RemoteSha ec64c53564889db45aafc874d663f5f05f4e165f

Contents

baySeq-package . . . 2
allModels . . . 4
baySeq-classes . . . 5
bimodalSeparator . . . 6
CDPost . . . 7
CDPriors . . . 8
densityFunction-class . . . 9
densityFunctions . . . 10
getLibsizes . . . 10
getLikelihoods . . . 12
getPosteriors . . . 15
getPriors . . . 17
getTPs . . . 19
makeOrderings . . . 21
marginaliseEqual . . . 22
marginalisePairwise . . . 24
methObservables . . . 25
mobAnnotation . . . 26
mobData . . . 27
pairData . . . 28
plotMA.CD . . . 28
plotNullPrior . . . 30
plotPosteriors . . . 30
plotPriors . . . 31
selectTop . . . 32
simData . . . 34
summarisePosteriors . . . 34
topCounts . . . 35
zimData . . . 37

Index **38**

`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", "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
libsizes(CD) <- 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]
```

`allModels` *Function to generate all possible models for a countData object based on the replicate data.*

Description

This function populates the '@groups' slot of the supplied countData object with all possible models for equivalence/non-equivalence of expression between replicate groups.

Usage

```
allModels(CD)
```

Arguments

`CD`	A countData object with a populated '@replicates' slot.

Details

Given a large number of different replicate groups, the total number of possible models listed in the '@groups' slot rises exponentially. This function will attempt to list them all. The use of consensus priors (see `getPriors`) is recommended if the number of models is high.

Value

A `countData` with populated '@groups' slot.

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

getPriors

Examples

```
# load test data
data(simData)

# Create a {countData} object from test data, supposing that there are
# multiple experimental groups present.

replicates <- c("simA", "simA", "simB", "simC", "simC", "simD", "simE", "simE", "simF", "simG")
CD <- new("countData", data = simData, replicates = replicates)
CD <- allModels(CD)

# The total number of models generated is high.
length(CD@groups)
```

---

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 these class 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.
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 (log)-posterior likelihoods for each group (matrix). Calculated by the functions described in getLik
estProps:	Estimated proportion of tags belonging to each group (numeric). Calculated by the functions described in getL
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 vector, o

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', 'rbind' and 'c' have been defined for these classes. The methods 'groups', 'groups<-', 'replicates', 'replicates<-', 'libsizes' and 'libsizes<-' have also been defined in order to access and modify these slots, and their use is recommended. The method 'flatten' can be used to produce a data.frame object containing much of the basic data in a member of this class.

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", "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
libsizes(CD) <- getLibsizes(CD)

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

`bimodalSeparator` *A function that, given a numeric vector, finds the value which splits the data into two sets of minimal total variance using Otsu's method.*

Description

This function takes a numeric vector and finds the value which splits the data into two sets of minimal total variance, weighted by the size of subsets (Otsu's method). It is principally intended to be a quick and easy way of separating bimodally distributed data.

Usage

```
bimodalSeparator(x, weights = NULL, minperc = 0.1, elbow = NULL)
```

Arguments

x	A numeric vector containing the data to be split.
weights	Possible weightings on the values in x for calculating the variance.
minperc	The required minimum size of each of the two subsets, expressed as a percentage of the total size. See Details.
elbow	If set, finds the 'left' or 'right' elbow of variance, instead of the minimum; defaults to NULL. 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 'minperc' parameter can be used to ensure that each subset of the split data will be of some minimum size, avoiding the outlier problem.

If 'elbow = NULL' (the default) then the split occurs at the value that minimises the variance, x0. If 'elbow = left' then we attempt to find the elbow point to the left of the value that minimises the variance, if 'elbow = right' then we find the elbow point to the right of the value that minimises the variance. Elbow points are found by drawing a line from the first point (for the left elbow) or the last point (for the right elbow) to x0, and finding the location on the curve of summed variances which maximises the distance to that line.

Value

Numeric value which splits the data.

Author(s)

Thomas J. Hardcastle

Examples

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

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)

`densityFunction-class` *Class* `"densityFunction"`

Description

This function fills the '@densityFunction' slot of a 'countData' object. It defines the distribution used to estimate posterior likelihoods, and associated values used in these calculations.

Objects from the Class

Objects can be created by calls of the form `new("densityFunction", ...)`.

Slots

`description`**:** A description of the distribution defined.

`density`**:** A `"function"`, defining the likelihood of a data array given observed data and hyperparameters.

`initiatingValues`**:** A `"list"` of functions (may be supplied as numerics) that define initial values of numeric prior discovery.

`equalOverReplicates`**:** A `"logical"`, describing which of the hyperparameters are equally marginally distributed over all groups, and which are not.

`lower`**:** A `"function"`, required to define the lower limit of optimisation in the case where only one hyperparameter is *not* equally marginally distributed over all groups.

`upper`**:** A `"function"`, required to define the upper limit of optimisation, as for 'lower', above.

`stratifyFunction`**:** An optional `"function"`, used to stratify the data for more accurate prior estimation.

`stratifyBreaks`**:** An optional `"numeric"`, used to define the number of strata in a stratification.

`nullFunction`**:** An optional `"function"` on the hyperparameters, used to generate a one-dimensional distribution which can be partitioned to identify 'null' data.

`orderingFunction`**:** An optional `"function"` for ordering the data between groups of a model.

`modifyNullPriors`**:** An optional `"function"` for modifying the priors for the 'null' data.

Methods

No methods defined with class "densityFunction" in the signature.

Author(s)

Thomas J. Hardcastle

Examples

```
showClass("densityFunction")
```

densityFunctions *Lists all currently available densityFunctions.*

Description

The densityFunction objects define the distribution and various other parameters used to analyse the data stored in a countData object.

Usage

```
densityFunctions()
```

Value

Character string giving names of available densityFunction objects.

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

densityFunction

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", "total", "edgeR"), quantile = 0.75, ...)
```

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 and 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

If a \link{countData} object is given, an identical object will be returned with updated library sizes. If only the data and replicate structure are given, a numerical vector of library sizes (scaling factors) for each library in the data will be returned.

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)

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

getLikelihoods *Finds posterior likelihoods for each count or paired count as belonging to some model.*

Description

These functions calculate posterior probabilities for each of the rows in a 'countData' object belonging to each of the models specified in the 'groups' slot.

Usage

```
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, discardSampling =
FALSE, cl, ...)
getLikelihoods.BB(cD, prs, pET = "BIC", marginalise = FALSE, subset =
NULL, priorSubset = NULL, bootStraps = 1, conv = 1e-04, nullData = FALSE,
returnAll = FALSE, returnPD = FALSE, verbose = TRUE, discardSampling =
FALSE, cl, ...)
getLikelihoods(cD, prs, pET = "BIC", marginalise = FALSE, subset = NULL,
priorSubset = NULL, bootStraps = 1, bsNullOnly = TRUE, conv = 1e-4, nullData = FALSE,
 weightByLocLikelihoods = TRUE, modelPriorSets = list(),
modelPriorValues = list(), returnAll = FALSE, returnPD = FALSE, verbose
= TRUE, discardSampling = FALSE, modelLikes = TRUE, cl = NULL, tempFile
= NULL, largeness = 1e+08)
```

Arguments

cD	An object of type countData, or descending from this class.
prs	(Initial) prior probabilities for each of the groups in the 'cD' 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.
bootStraps	How many iterations of bootstrapping should be used in the (re)estimation of priors in the negative binomial method.
bsNullOnly	If TRUE (default, bootstrap hyper-parameters based on the likelihood of the null model and its inverse only; otherwise, on the likelihood of all models.

`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.

`weightByLocLikelihoods`
If a locLikelihoods slot is present in the 'cD' object, and nullData = TRUE, then the initial weighting on nulls will be determined from the locLikelihoods slot. Defaults to TRUE.

`modelPriorSets` If given, a list object, which defines subsets of the data for which different priors on the different models might be expected. See Details.

`modelPriorValues`
If given, a list object which defines priors on the different models. 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.

`discardSampling`
If TRUE, discards information about which data rows are sampled to generate prior information. May slightly degrade the results but reduce computational time required. Defaults to FALSE.

`modelLikes` If TRUE (default), returns likelihoods for each model. If FALSE, returns likelihoods for each hyper-parameter, from which the posterior joint distribution on hyper-parameters can be inferred.

`cl` A SNOW cluster object.

`tempFile` Temporary file prefix for saving data likelihoods. Primarily for debugging purposes at this stage. Defaults to NULL, in which case no temporary data are saved.

`largeness` The maximum size over which data likelihoods are calculated. Objects larger than this are split. This is most useful in combination with the saving of temporary files in the case of excessively large analyses.

`...` Any additional information to be passed to the `'getLikelihoods'` function by the now deprecated functions.

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 input 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.

In certain circumstances, it may be expected that certain subsets of the data are likely to behave differently to others; for example, if a set of genes are expected in advance to be differentially expressed, while the majority of the data are not. In this case, it may be advantageous (in terms of improving false discovery rates) to specify these different subsets in the modelPriorSets variable. However, care should be taken downstream to avoid confirmation bias.

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.

See Hardcastle & Kelly (2010) for a definition of the negative binomial 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.NB' and 'getLikelihoods.BB' functions are now deprecated and will soon be removed.

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", "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)

# set negative binomial density function
densityFunction(CD) <- nbinomDensity

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

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

# 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(CDPriors, pET = "BIC", 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 = NULL, maxit = 100, accuracy =
1e-5, eqOverRep = NULL, 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.
eqOverRep	A boolean describing which prior values are equally marginally distributed over replicates.
cl	Parallelisation 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 specified in the 'densityFunction' slot of the countData object. A special case is maintained for historical reasons; getPriors.NB estimates parameters for a negative binomial distribution using quasi-maximum-likelihood methods.

Usage

```
getPriors(cD, samplesize = 1e5, samplingSubset = NULL,
consensus = FALSE, cl, verbose = TRUE)
getPriors.NB(cD, samplesize = 1e5, samplingSubset = NULL,
equalDispersions = TRUE, estimation = "QL", verbose = TRUE, zeroML =
FALSE, consensus = 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.
consensus	If TRUE, creates a consensus distribution rather than a separate distribution for each member of the groups structure in the 'cD' object. See Details.

cl	A SNOW cluster object.
verbose	Should status messages be displayed? Defaults to TRUE.
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.
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.
...	Additional parameters to be passed to the estimateTagwiseDisp function if 'estimation = "edgeR"'.

Details

These functions empirically estimate prior parameters for the distributions used in estimating posterior likelihoods of each count belonging to a particular group.

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.

If 'consensus = TRUE', then a consensus distribution is created and used for each group in the 'cD' object. This allows faster computation of the priors and likelihoods, but with some degradation of accuracy.

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.

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", "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
libsizes(CD) <- 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)

# 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)
```

`makeOrderings` *Construct orderings for count data given a model structure and an ordering function.*

Description

Given a model structure as defined in the '@groups' slot of a `countData` object containing more than one group, it is often possible to define an ordering on the groups for a given genomic event. To take a simple example, if the average expression of a gene is higher in sample set A then in sample set B, then we might impose an ordering A>B.

Usage

```
makeOrderings(cD, orderingFunction)
```

Arguments

`cD`	A `countData` object, or a descendant thereof.
`orderingFunction`	A function defining the orderings. If not given, will be taken from the '@densityFunction' slot of 'cD'. See Details, and examples below.

Details

The orderingFunction takes 'dat' and 'observables' as arguments. 'dat' is equivalent to the '@data' slot of the 'cD' object, and 'observables' the combined data in the '@sampleObservables', '@rowObservables' and '@cellObservables' slots.

Value

A `countData` with populated '@orderings' slot.

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)

Examples

```
# load test data
data(simData)

# Create a countData object from test data

replicates <- c("simA", "simA", "simA", "simA", "simA", "simB", "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)
libsizes(CD) <- getLibsizes(CD)

# order on expression normalised for library size (scaling factor) and gene length

CD <- makeOrderings(CD, orderingFunction = function(dat, observables) dat / observables$libsizes)

# orderings calculated for DE group
head(CD@orderings)

# load test (paired) data
data(pairData)

# create a countData object from paired data
pairCD <- new("countData", data = list(pairData[,1:4], pairData[,5:8]),
                  replicates = c(1,1,2,2),
                  groups = list(NDE = c(1,1,1,1), DE = c(1,1,2,2)),
               densityFunction = bbDensity)
libsizes(pairCD) <- getLibsizes(pairCD)

# order on (log-)ratio of pairs, with fudge-factor on zeros.

pairCD <- makeOrderings(pairCD, orderingFunction = function(dat, observables) {
                                      data <- dat / observables$libsizes
                                      adjmin <- min(data[data > 0]) / 10
                                      log(data[,,1] + adjmin) - log(data[,,2] + adjmin)
})

# orderings calculated for DE group
head(pairCD@orderings)
```

`marginaliseEqual` *Computes marginal likelihoods that two replicate groups are equal.*

Description

In cases where multiple models are simultaneously evaluated in the 'getLikelihoods' function, the posterior likelihoods for each model in which two conditions are equivalent can be summed to give the marginal likelihood of equivalence for all biomolecular events (i.e., data rows).

Usage

```
marginaliseEqual(cD, r1, r2)
```

Arguments

cD	A countData object with evaluated posterior likelihoods in the '@posteriors' slot.
r1	A defined group name to identify in the '@groups' slot of the countData object 'cD'.
r2	A defined group name to identify in the '@groups' slot of the countData object 'cD'.

Value

A vector of marginal posterior likelihoods defining the probability that the two group identifiers are equal for each row of the data.

Author(s)

Thomas J. Hardcastle

See Also

allModels marginalisePairwise

Examples

```
# load test data
data(simData)

# Create a {countData} object from test data, supposing that there are
# multiple experimental groups present.

replicates <- c("simA", "simA", "simB", "simC", "simC", "simD", "simE", "simE", "simF", "simG")
CD <- new("countData", data = simData, replicates = replicates)
CD <- allModels(CD)

# The total number of models generated is high.
length(CD@groups)

# Priors and likelihoods acquired through standard means.

## Not run: CD <- getPriors(CD, cl = cl)
## Not run: CD <- getLikelihoods(CD, cl = cl)

# Marginal likelihood that 'simA' and 'simD' replicate groups are equal
# for each row of the data.

## Not run: marginaliseEqual(CD, "simA", "simD")
```

`marginalisePairwise` *Computes marginal likelihoods that two replicate groups differ, in a paticular direction.*

Description

In cases where multiple models are simultaneously evaluated in the 'getLikelihoods' function, the posterior likelihoods for each model in which one condition is greater than another can be summed to give the marginal likelihood of (directed) difference for all biomolecular events (i.e., data rows).

Usage

```
marginalisePairwise(cD, greaterThan, lessThan)
```

Arguments

`cD`	A `countData` object with evaluated posterior likelihoods in the '@posteriors' slot.
`greaterThan`	A defined group name (or vector of group names) to identify in the '@groups' slot of the `countData` object 'cD'; the function will identify all models in which these groups are equivalent and greater than that defined in the 'lessThan' variable.
`lessThan`	A defined group name (or vector of group names) to identify in the '@groups' slot of the `countData` object 'cD'; the function will identify all models in which these groups are eqivalent and less than that defined in the 'greaterThan' variable.

Value

A vector of marginal posterior likelihoods defining the probability that the two group identifiers are (directionally) different for each row of the data.

Author(s)

Thomas J. Hardcastle

See Also

`allModels` `marginaliseEqual`

Examples

```
# load test data
data(simData)

# Create a {countData} object from test data, supposing that there are
# multiple experimental groups present.
```

```
replicates <- c("simA", "simA", "simB", "simC", "simC", "simD", "simE", "simE", "simF", "simG")
CD <- new("countData", data = simData, replicates = replicates)
CD <- allModels(CD)

# The total number of models generated is high.
length(CD@groups)

# Priors and likelihoods acquired through standard means.

## Not run: CD <- getPriors(CD, cl = cl)
## Not run: CD <- getLikelihoods(CD, cl = cl)

# Marginal likelihood that 'simA' condition is greater than 'simD' group
# for each row of the data.

## Not run: marginalisePairwise(CD, "simA", "simD")
```

`methObservables` *Generation of intermediate values in likelihood estimation for methylation data.*

Description

Estimating prior and posterior values for methylation data that account for non-conversion rates is a time-consuming process. Significant increases in speed can be made by calculating in advance sets of data that will be re-used at several points of these analyses. This function populates the '@cellObservables' slot of a 'countData' object that contains a 'nonconversion' object in the '@sampleObservables' slot.

Usage

```
methObservables(mD, tail = 0.01)
```

Arguments

`mD`	A `countData` object, or descendant, which contains a numeric vector 'nonconversion' in the '@sampleObservables' slot.
`tail`	A cutoff on the quantile (upper and lower) of the distribution on the non-conversion. Smaller values will give a marginal increase in accuracy at high computational cost. Large values will decrease accuracy somewhat but reduce the time needed for analysis. See Details.

Details

For loci with large numbers of observed cytosines, the full dataset to be pre-computed will be very large. However, only the pre-computations near the average expression level will contribute significantly to the estimated priors and posteriors. The 'tail' parameter sets the quantile at which the distribution is considered to no longer contribute significantly to the results. Values below 0.1 are probably acceptable under nearly all circumstances.

Value

A countData object with the '@cellObservables' slot populated with temporary values useful in the faster calculation of likelihoods.

Author(s)

Thomas J. Hardcastle

---

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

pairData — *Simulated data for testing the baySeq package methods for paired data*

Description

This data set is a matrix ('pairData') of simulated counts from a set of high-throughput sequencing data from a paired experimental design. The first four columns of data are to be paired with the second four columns of data respectively. The first two paired samples form one replicate group, the second two paired samples form another replicate group. The first hundred rows of the data are truly differentially expressed between replicate groups, the second hundred are differentially expressed between pairs, the remainder have no differential expression.

It is simulated according to a set of Poisson distributions whose parameters for each row are determined by a beta distribution on the relative proportions of data in each pairing.

Usage

```
pairData
```

Format

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

Source

Simulation.

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", "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
libsizes(CD) <- getLibsizes(CD)
```

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

`plotNullPrior` *Plots distribution of null function and shows the threshold separator.*

Description

In sequencing expression of various genomic events, it is not uncommon to find a subset of genomic events that are qualitatively different from the remainder of the data. Thus, for some function of the estimated priors, we may observe bimodality or long tails which correlate to this subset.

Usage

```
plotNullPrior(cD, ...)
```

Arguments

`cD`	A `countData` object with a '@nullFunction' slot in its '@densityFunction'.
`...`	Additional arguments to be passed to 'plot'

Value

Invisibly, the numeric value of the threshold.

Author(s)

Thomas J. Hardcastle

`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, par = 1)
```

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.
par	The parameter of the prior that will be plotted.

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.NB.

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)

plotPriors(CDPriors, group = "NDE", par = 1)
```

---

selectTop — *Selects the top genomic events, based on posterior likelihoods, from a 'countData' object.*

Description

This function subsets a countData object by selecting those events that best (or least) represent a model, based on the posterior likelihoods estimated for that model and some threshold. Selection can be done for a specific model (and ordering of the data under that model) or for all models (and all orderings).

Usage

```
selectTop(cD, group, ordering, orderings = TRUE, decreasing = TRUE,
number = 10, likelihood, FDR, FWER, posteriors)
```

Arguments

cD	A countData object, with a populated '@posteriors' slot (unless 'posteriors' is specified; see below).
group	Optionally, the model of interest, as defined in the '@groups' slot of the countData object. If unspecified, subsets for all models will be returned as a list.
ordering	If 'group' is specified, a particular ordering of the data based on that group can also be specified.
orderings	If no group is specified, should the selection of models also be split by the orderings of the data under the models? Defaults to TRUE.
decreasing	If FALSE, considers the data with the lowest posterior likelihoods, rather than the greatest (i.e., selects those data least likely to conform to a particular model.
number	If given, selects the top 'number' of genomic events for each model (and optionally, ordering). Ignored if another selection criteria is chosen, unless this criteria would return no values.
likelihood	If given, selects all genomic events for a given model (and optionally, ordering) with posterior likelihood exceeding this value.
FDR	If given, selects all genomic events for a given model (and optionally, ordering) with false discovery rate below this value. Ignored if likelihood is specified.
FWER	If given, selects all genomic events for a given model (and optionally, ordering) with family-wise error rate below this value. Ignored if likelihood or FDR is specified.
posteriors	If given, a vector of log-posterior likelihoods to use instead of those in the '@posteriors' slot of the 'cD' object.

Value

Either a single countData object (if 'group' is specified), or a named list of countData objects.

Author(s)

Thomas J. Hardcastle

See Also

topCounts

Examples

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

data(CDPost)
```

```
# select from all models and orderings with FDR equal to or lower than 0.01.

selectTop(CDPost, FDR = 0.01)
```

`simData` *Simulated data for testing the baySeq package methods*

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)

`summarisePosteriors` *Summarises expected number of genomic events given the calculated posterior likelihoods of a countData object.*

Description

Given posterior likelihoods for each model, we can calculate the expected number of genomic events corresponding to each model (and to each ordering within each model) by summing the posterior likelihoods.

Usage

```
summarisePosteriors(cD, orderings = TRUE)
```

Arguments

cD	A countData object.
orderings	Indicates whether models should be split by orderings of the data under the model (defaults to TRUE).

Value

Numeric vector of expected number of genomic events belonging to each model (optionally, split by orderings).

Author(s)

Thomas J. Hardcastle

See Also

topCounts, selectTop

Examples

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

data(CDPost)

# summarise the expected number of genomic events in each category
summarisePosteriors(CDPost)
```

---

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, ordering, decreasing = TRUE, number = 10, likelihood, FDR,
FWER, normaliseData = FALSE, posteriors)
```

Arguments

cD	countData object, containing posterior likelihoods for each group (unless 'posteriors' is specified; see below).
group	Which group should we give the counts for? See Details.
ordering	If specified, restricts the analysis to a particlar ordering on the group.
decreasing	Ordering on posterior likelihoods.
number	How many results should be returned?
likelihood	If given, ignores 'number' and returns all results above a certain likelihood (and FDR, and FWER, if given).
FDR	If given, ignores 'number' and returns all results with an FDR lower than this threshold (and likelihood, and FWER, if given).
FWER	If given, ignores 'number' and returns all results with an FWER lower than this threshold (and likelihood, and FDR, if given).
normaliseData	Should the displayed counts be normalised? See details. Defaults to FALSE.
posteriors	If given, a vector of log-posterior likelihoods to use instead of those in the '@posteriors' slot of the 'cD' object.

Details

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).

If a countData object is given, the returned dataframe will contain either the raw counts for that object, or (if 'normaliseData = TRUE' the counts normalised by library size.

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.

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)


# Report the top ten rows of data that have highest likelihood of belonging to
# group 2 of the data (i.e., differentially expressed), with group 1
# being overexpressed compared to group 2.

topCounts(CDPost, group = "DE", ordering = "1>2", number = 10)
```

`zimData` *Simulated data for testing the baySeq package methods*

Description

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

Usage

```
zimData
```

Format

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

Source

Simulation.

Index

∗ **classes**
  baySeq-classes, 5
  densityFunction-class, 9
∗ **datasets**
  CDPost, 7
  CDPriors, 8
  mobAnnotation, 26
  mobData, 27
  pairData, 28
  simData, 34
  zimData, 37
∗ **distribution**
  getLikelihoods, 12
  getPriors, 17
∗ **hplots**
  plotMA.CD, 28
∗ **hplot**
  plotNullPrior, 30
  plotPosteriors, 30
  plotPriors, 31
∗ **mainip**
  marginaliseEqual, 22
∗ **manip**
  allModels, 4
  getLibsizes, 10
  getTPs, 19
  makeOrderings, 21
  marginalisePairwise, 24
  methObservables, 25
  selectTop, 32
∗ **models**
  bimodalSeparator, 6
  getLikelihoods, 12
  getPosteriors, 15
  getPriors, 17
∗ **package**
  baySeq-package, 2
∗ **print**
  summarisePosteriors, 34
  topCounts, 35
∗ **utilities**
  densityFunctions, 10
[,countData,ANY-method *(baySeq-classes)*, 5
[,countData-method *(baySeq-classes)*, 5

allModels, 4, *23, 24*

baySeq *(baySeq-package)*, 2
baySeq-class *(baySeq-classes)*, 5
baySeq-classes, 5
baySeq-package, 2
bbDensity *(densityFunctions)*, 10
bbNCDist *(densityFunctions)*, 10
bimodalSeparator, 6

c,countData-method *(baySeq-classes)*, 5
CDPost, 7
CDPriors, 8
countData, *3, 4*, *10–14*, *17–21*, *23–26*, *29–33*, *35, 36*
countData *(baySeq-classes)*, 5
countData-class *(baySeq-classes)*, 5

densityFunction, *10*
densityFunction *(baySeq-classes)*, 5
densityFunction,countData-method *(baySeq-classes)*, 5
densityFunction-class, 9
densityFunction<- *(baySeq-classes)*, 5
densityFunction<-,countData-method *(baySeq-classes)*, 5
densityFunctions, 10
dim,countData-method *(baySeq-classes)*, 5

estimateTagwiseDisp, *18*

flatten *(baySeq-classes)*, 5
flatten,countData-method *(baySeq-classes)*, 5

getLibsizes, 10
getLikelihoods, *3*, *5*, 12, *17*, *19*
getLikelihoods.NB, *32*
getPosteriors, 15, *31*
getPriors, *3–5*, *14*, 17
getPriors.NB, *32*
getTPs, *3*, *14*, 19
groups *(*baySeq-classes*)*, 5
groups,countData-method
*(*baySeq-classes*)*, 5
groups<- *(*baySeq-classes*)*, 5
groups<-,countData-method
*(*baySeq-classes*)*, 5

libsizes *(*baySeq-classes*)*, 5
libsizes,countData-method
*(*baySeq-classes*)*, 5
libsizes<- *(*baySeq-classes*)*, 5
libsizes<-,countData-method
*(*baySeq-classes*)*, 5

makeOrderings, 21
marginaliseEqual, 22, *24*
marginalisePairwise, *23*, 24
md2Density *(*densityFunctions*)*, 10
md3Density *(*densityFunctions*)*, 10
mdDensity *(*densityFunctions*)*, 10
methObservables, 25
mobAnnotation, 26, *27*
mobData, *26*, 27

nbinomDensity *(*densityFunctions*)*, 10
normDensity *(*densityFunctions*)*, 10

pairData, 28
plot, *29*, *31*
plotMA.CD, 28
plotNullPrior, 30
plotPosteriors, 30
plotPriors, 31

rbind *(*baySeq-classes*)*, 5
rbind,countData-method
*(*baySeq-classes*)*, 5
replicates *(*baySeq-classes*)*, 5
replicates,countData-method
*(*baySeq-classes*)*, 5
replicates<- *(*baySeq-classes*)*, 5
replicates<-,countData-method
*(*baySeq-classes*)*, 5

seglens *(*baySeq-classes*)*, 5
seglens,countData-method
*(*baySeq-classes*)*, 5
seglens<- *(*baySeq-classes*)*, 5
seglens<-,countData-method
*(*baySeq-classes*)*, 5
selectTop, 32, *35*
show,countData-method *(*baySeq-classes*)*,
5
simData, 34
summarisePosteriors, 34

topCounts, *3*, *14*, *33*, 35, *35*

zimData, 37
ZINBDensity *(*densityFunctions*)*, 10