baySeq

October 5, 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).
`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

Methods 'new', 'dim', '[' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

Examples

```
library(baySeq)

data(simCount)
data(libsizes)

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))

#create new 'countData' object
CD <- new("countData", data = simCount, replicates = replicates, libsizes = libsizes, grc

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 (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

Examples

```
# See vignette for more examples.

# load test data
data(simCount)
data(libsizes)

# replicate structure of data
replicates <- c(1,1,1,1,1,2,2,2,2,2)

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

# construct 'countData' object
CD <- new("countData", data = simCount, replicates = replicates, libsizes = libsizes, gro
CD[1:10,]

# estimate prior distributions on 'countData' object using Poisson
# method. Other methods are available - see getPriors
CDP.Poi <- getPriors.Pois(CD, samplesize = 20, takemean = TRUE, cl = NULL)

# estimate posterior likelihoods for each row of data belonging to each hypothesis
CDPost.Poi <- getLikelihoods.Pois(CDP.Poi, prs = c(0.5, 0.5), pET = "BIC", cl = NULL)

# display the rows of data showing greatest association with the second
# hypothesis (differential expression)
topCounts(CDPost.Poi, group = 2, number = 10)

# find true positive selection rate
getTPs(CDPost.Poi, group = 2, TPs = 1:100)[1:100]
```

`factCount` *Simulated data for testing the baySeq package methods; simulated counts from a factorial design differential expression analysis*

Description

This data set is a matrix 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 between the first four samples and the second four samples. The second hundred rows of the data are truly differentially expressed between samples 1,2,5,6 and samples 3,4,7,8.

Usage

```
factCount
```

Format

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

Source

Simulation.

References

Hardcastle T.J., and Kelly, K (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

factlibsizes

---

factlibsizes — *Simulated data for testing the baySeq package methods; simulated library sizes from a pairwise differential expression analysis*

Description

This data set is a vector of library sizes for the factCount matrix.

Usage

```
factlibsizes
```

Format

A vector containing library sizes for the ten libraries whose data is given in the factCount matrix.

Source

Simulation.

References

Hardcastle T.J., and Kelly, K (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

factCount

`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", subset = NULL,
priorSubset = NULL, verbose = TRUE, ..., cl)
getLikelihoods.Dirichlet(cD, prs, pET = "BIC", subset = NULL,
priorSubset = NULL, verbose = TRUE, cl)
getLikelihoods.Pois(cD, prs, pET = "BIC", subset = NULL,
priorSubset = NULL, distpriors = FALSE, verbose = TRUE, cl)
getLikelihoods.NB(cD, prs, pET = "BIC", subset = NULL,
priorSubset = NULL, bootStraps = 1, conv = 1e-4, nullData = FALSE,
returnAll = 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.
`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 instead of returning a single countData object, the function returns a list of countData objects; one for each bootstrap. Largely used for debugging purposes.
`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 (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

countData, getPriors, topCounts, getTPs

Examples

```
library(baySeq)

# See vignette for more examples.

# Create a {countData} object and estimate priors for the
# Poisson methods.
data(simCount)
data(libsizes)
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 = simCount, replicates = replicates, libsizes = libsizes, gro
CDP.Poi <- getPriors.Pois(CD, samplesize = 20,
takemean = TRUE, cl = NULL)

# Get likelihoods for data with Poisson method
CDPost.Poi <- getLikelihoods.Pois(CDP.Poi, prs = c(0.5, 0.5),
pET = "BIC", cl = NULL)


# Alternatively, get priors for negative binomial method
CDP.NB <- getPriors.NB(CD, samplesize = 10^5, estimation = "QL", cl = NULL)

# Get likelihoods for data with negative binomial method with bootstrapping

CDPost.NB <- getLikelihoods.NBboot(CDP.NB, prs = c(0.5, 0.5),
pET = "BIC", bootStraps = 1, cl = NULL)

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

cl <- NULL
try(library(snow))
try(cl <- makeCluster(4, "SOCK"))

CDP.NB <- getPriors.NB(CD, samplesize = 10^5, estimation = "QL", cl = cl)
CDPost.NB <- getLikelihoods.NB(CDP.NB, prs = c(0.5, 0.5),
pET = "BIC", cl = 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", groups, priorSubset = NULL, maxit = 100, ac
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.
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 (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

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 = 10^5, perSE = 1e-1, maxit = 10^6,
verbose = TRUE)
getPriors.Pois(cD, samplesize = 10^5, perSE = 1e-1, takemean = TRUE,
maxit = 10^5, verbose = TRUE, cl)
getPriors.NB(cD, samplesize = 10^5, equalDispersions = TRUE, estimation
= "QL", verbose = TRUE, cl, ...)
```

Arguments

`cD`	A `countData` object.
`samplesize`	How large a sample should be taken in estimating the priors?
`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.
`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.
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.

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 (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

countData, getLikelihoods

Examples

```
# See vignette for more examples.

# Create a {countData} object.
data(simCount)
data(libsizes)
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 = simCount, replicates = replicates, libsizes = libsizes, grc
groups)

# If we have the 'snow' package installed we can parallelise the prior
# estimation. This will usually (depending on your parallelisation
# set-up) offer significant performance gains.

cl <- NULL
try(library(snow))
try(cl <- makeCluster(4, "SOCK"))

# Estimate priors using Poisson method.
DP.Poi <- getPriors.Pois(CD, samplesize = 20, takemean = TRUE, cl = cl)

# Alternatively, get priors for negative binomial method.

CDP.NBML <- 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.

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

```
# Create a {countData} object and estimate priors for the Poisson methods.
data(simCount)
data(libsizes)
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 = simCount, replicates = replicates, libsizes = libsizes, gro
CDP.Poi <- getPriors.Pois(CD, samplesize = 20,
takemean = TRUE, cl = NULL)

# Get likelihoods for data with Poisson method
CDPost.Poi <- getLikelihoods.Pois(CDP.Poi, prs = c(0.5, 0.5),
pET = "BIC", cl = NULL)

# If the first hundred rows in the 'simCount' 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.Poi, group = 2, decreasing = TRUE, TPs = 1:100)
```

---

libsizes — *Simulated data for testing the baySeq package methods; simulated library sizes from a pairwise differential expression analysis*

Description

This data set is a vector of library sizes for the simCount matrix.

Usage

```
libsizes
```

Format

A vector containing library sizes for the ten libraries whose data is given in the simCount matrix.

Source

Simulation.

References

Hardcastle T.J., and Kelly, K (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

simCount

---

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.

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.NBboot

Examples

```
# Create a {countData} object and estimate priors for the Poisson methods.
data(simSeg)
data(libsizes)
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 = simSeg[,-1], replicates = replicates, seglens = simSeg[,1],
CDP.NB <- getPriors.NB(CD, samplesize = 1000, estimation = "QL", cl = NULL)

plotPriors(CDP.NB, group = 1)
```

`simCount` *Simulated data for testing the baySeq package methods; simulated counts from a pairwise differential expression analysis*

Description

This data set is a matrix 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. Library sizes for these data sets are given in `libsizes`.

Usage

```
simCount
```

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 (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

`libsizes`

simSeg *Simulated data for testing the baySeq package methods; simulated counts and segment lengths from a pairwise differential expression analysis*

Description

This data set is a matrix of simulated counts from a simple pairwise expression analysis of genomic regions. The first column of the data is the length of the segment, simulated from a negative binomial distribution. The remaining columns of the data are simulated according to a negative binomial distribution with varying parameters for each row, such that the rate of expression is proportional to the length of the segment. The first hundred rows of the data are truly differentially expressed, the second hundred rows have no true expression of any kind, the remainder are expressed but not differentially expressed. Library sizes for these data are given in `libsizes`.

Usage

```
simCount
```

Format

A matrix of which the first column gives the length of a genomic region, and the following ten columns represents the discrete data (acquired by sequencing) observed at each region.

Source

Simulation.

References

Hardcastle T.J., and Kelly, K (2010). Identifying Patterns of Differential Expression in Count Data. In submission.

See Also

`libsizes`

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 log-likelihoods.

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

```
data(simCount)
data(libsizes)

# Make 'countData' object and calculate posterior likelihoods for each
# item belonging to each hypothesis.
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 = simCount, replicates = replicates, libsizes = libsizes, gro
CDP.Poi <- getPriors.Pois(CD, samplesize = 20, cl = NULL)
CDPost.Poi <- getLikelihoods.Pois(CDP.Poi, prs = c(0.5, 0.5), pET = "BIC", cl = NULL)

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

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

Index

∗Topic **classes**
baySeq-classes, 1
∗Topic **datasets**
factCount, 3
factlibsizes, 4
libsizes, 12
simCount, 14
simSeq, 15
∗Topic **distribution**
getLikelihoods, 5
getPriors, 9
∗Topic **manip**
getTPs, 11
plotPriors, 13
∗Topic **models**
getLikelihoods, 5
getPosteriors, 7
getPriors, 9
∗Topic **package**
baySeq-package, 2
∗Topic **print**
topCounts, 15
[,countData-method
(baySeq-classes), 1

baySeq *(baySeq-package)*, 2
baySeq-class *(baySeq-classes)*, 1
baySeq-classes, 1
baySeq-package, 2

countData, *2*, *5–7*, *9–13*, *16*
countData *(baySeq-classes)*, 1
countData-class *(baySeq-classes)*, 1

dim,countData-method
(baySeq-classes), 1

estimateTagwiseDisp, *10*

factCount, 3, *4*
factlibsizes, 4, *4*

getLikelihoods, *1*, *2*, 5, *8*, *10*
getLikelihoods.NBboot, *13*
getPosteriors, 7
getPriors, *1*, *2*, *7*, 9
getPriors.NB, *13*
getTPs, *2*, *7*, 11

libsizes, 12, *14*, *15*

plotPriors, 13

show,countData-method
(baySeq-classes), 1
simCount, *12*, *13*, 14
simSeq, 15

topCounts, *2*, *7*, 15