Package: NBPSeq (via r-universe)

September 9, 2024

Type Package

Version 0.3.5

Date 05-17-2014

Title Negative Binomial Models for RNA-Sequencing Data

Author Yanming Di <diy@stat.oregonstate.edu>, Daniel W Schafer <schafer@stat.oregonstate.edu>, with contributions from Jason S Cumbie <cumbiej@onid.orst.edu> and Jeff H Chang <changj@cgrb.oregonstate.edu>.

Maintainer Yanming Di <diy@stat.oregonstate.edu>

Description Negative Binomial (NB) models for two-group comparisons and regression inferences from RNA-Sequencing Data.

Depends R (>= 3.0.0)

Imports splines

License GPL-2

LazyLoad yes

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

RemoteUrl https://github.com/diystat/nbpseq

RemoteRef HEAD

RemoteSha 358f095f4846476c7c9ffe720b502899ea18affb

Contents

`NBPSeq-package` *Negative Binomial Regression Models for Statistical Analysis of RNA-Sequencing Data*

Description

Negative binomial (NB) two-group and regression models for RNA-Sequencing data analysis.

Details

See the examples of `test.coefficient` and `exact.nb.test` for typical workflows of using this package.

`arab` *Arabidopsis RNA-Seq Data Set*

Description

An RNA-Seq dataset from a pilot study of the defense response of Arabidopsis to infection by bacteria. We performed RNA-Seq experiments on three independent biological samples from each of the two treatment groups. The matrix contains the frequencies of RNA-Seq reads mapped to genes in a reference database. Rows correspond to genes and columns correspond to independent biological samples.

Usage

```
data(arab)
```

Format

A 26222 by 6 matrix of RNA-Seq read frequencies.

Details

We challenged leaves of Arabidopsis with the defense-eliciting $\Delta hrcC$ mutant of *Pseudomonas syringae* pathovar *tomato* DC3000. We also infiltrated leaves of Arabidopsis with 10mM MgCl2 as a mock inoculation. RNA was isolated 7 hours after inoculation, enriched for mRNA and prepared for RNA-Seq. We sequenced one replicate per channel on the Illumina Genome Analyzer (http://www.illumina.com). The length of the RNA-Seq reads can vary in length depending on user preference and the sequencing instrument. The dataset used here are derived from a 36-cycle sequencing reaction, that we trimmed to 25mers. We used an in-house computational pipeline to process, align, and assign RNA-Seq reads to genes according to a reference database we developed for Arabidopsis.

Author(s)

Jason S Cumbie <cumbiej@onid.orst.edu> and Jeff H Chang <changj@cgrb.oregonstate.edu>.

References

Di Y, Schafer DW, Cumbie JS, and Chang JH (2011): "The NBP Negative Binomial Model for Assessing Differential Gene Expression from RNA-Seq", Statistical Applications in Genetics and Molecular Biology, 10 (1).

`compute.tail.prob` *(private) Compute the tail probability of a conditional distribution involving a pair of Negative Binomial (NB) random variables given their sum*

Description

Compute the probability of observing values of (S1, S2) that are more extreme than (s1, s2) given that S1+S2=s1+s2 for a pair of Negative Binomial (NB) random variables (S1, S2) with mean and size parameters (mu1, kappa1) and (mu2, kappa2) respectively.

Usage

```
compute.tail.prob(s1, s2, mu1, mu2, kappa1, kappa2)
```

Arguments

`s1`	a number, the observed value of a NB random variable
`s2`	a number, the observed value of a NB random variable
`mu1`	a number, the mean parameter of the NB variable s1
`mu2`	a number, the mean parameter of the NB variable s2
`kappa1`	a number, the size parameter of the NB variable s1
`kappa2`	a number, the size parameter of the NB variable s2

Details

This function computes the probabily of (S1, S2) for all values of S1 and S2 such that S1+S2=s1+s2, then sums over the probabilites that are less than or equal to that of the observed values (s1, s2). In context of DE test using RNA-Seq data after thinning, S1 and S2 are often sums of iid NB random variables (and are thus NB random variables too).

The current implementation can be slow if s1 + s2 is large.

Potential improvements: For computing the one-sided tail probability of Pr(S1<s1 |S1+S2=s1+s2), there might be a faster way. The conditional distribution can be also approximated by saddlepoint methods. If S1 and S2 are sum of two subsets of iid random variables, the saddle point approximation would be very accurate.

Value

a number giving the probability of observing a (S1, S2) that is as or more extreme than (s1, s2) given that S1+S2=s1+s2.

`disp.by.group` *Specify a dispersion model where the parameters of the model will be estimated separately for different groups*

Description

Specify a dispersion model where the parameters of the model will be estimated separately for different groups

Usage

```
disp.by.group(disp.fun, grp.ids, predictor, subset,
  predictor.label = "Predictor", ...)
```

Arguments

`disp.fun`

`grp.ids`

`predictor`

`subset`
`predictor.label`

`...`

Value

a list,

`disp.nbp` *(private) Specify a NBP dispersion model*

Description

Specify a NBP dispersion model. The parameters of the specified model are to be estimated from the data using the function `optim.disp.apl` or `optim.disp.pl`.

Usage

```
disp.nbp(counts, eff.lib.sizes, x, phi.pre = 0.1, mu.lower = 1,
  mu.upper = Inf)
```

Arguments

`counts`	an $m \times n$ matrix of NB counts
`eff.lib.sizes`	a n-vector of estimated effective library sizes
`x`	a nxp matrix, design matrix (specifying the treatment structure)
`phi.pre`	a number, a preliminary constant dispersion value, will be used to get preliminary estimates of mean counts (mu.pre).
`mu.lower`	a number, rows with any component of mu.pre < mu.lower will not be used for estimating the dispersion model
`mu.upper`	a number, rows with any component of mu.pre > mu.upper will not be used for estimating the dispersion model

Details

Under this NBP model, the log dispersion is modeled as a linear function of the preliminary estimates of the log mean realtive frequencies (`pi.pre`):

log(phi) = par[1] + par[2] * log(pi.pre/pi.offset),

where `pi.offset` is 1e-4.

Under this parameterization, par[1] is the dispersion value when the estimated relative frequency is 1e-4 (or 100 RPM).

Value

a list

`fun`	a function that takes a vector, par, as input and outputs a matrix of dispersion values (same dimension as counts)
`par.init`	a numeric vector of length 2, initial values of par
`lower`	a numeric vector of length 2, lower bounds of the parameter values
`upper`	a numeric vector of length 2, upper bounds of the parameter values

subset	a logical vector of length m, specifying the subset of rows to be used when estimating the dispersion model parameters.
pi.pre	a m by n matrix, preliminary estimates of the relative frequencies.
pi.offset	a scalar, fixed to be 1e-4, an offset used in the NBP model (see Details)

---

disp.nbq *(private) Specify a NBQ dispersion model*

Description

Specify a NBQ dispersion model. The unknown parameters in the specified model are to be estimated using the function optim.disp.pl or optim.disp.apl.

Usage

```
disp.nbq(counts, eff.lib.sizes, x, phi.pre = 0.1, mu.lower = 1,
  mu.upper = Inf, pi.offset = median(pi.pre[subset, ]))
```

Arguments

counts	a mxn matrix of NB counts
eff.lib.sizes	a n-vector of estimated effective library sizes
x	a nxp matrix, design matrix (specifying the treatment structure)
phi.pre	a number, a preliminary constant dispersion value
mu.lower	a number, rows with mu.pre < mu.lower will not be used for estimating the dispersion model
mu.upper	a number, rows with mu.pre > mu.upper will not be used for estimating the dispersion model
pi.offset	a scalar, an offset used in the NBQ model (see Details).

Details

Under this NBQ model, the dispersion is modeled as a quadratic function of the preliminary estimates of the log mean realtive frequencies (pi.pre):

log(phi) = par[1] + par[2] * z + par[3] * z^2,

where z = log(pi.pre/pi.offset). By default, pi.offset is the median of pi.pre[subset,].

Value

a list

fun	a function that takes a vector, par, as input and outputs a matrix of dispersion values (same dimension as counts)
par.init	a vector of length 3, initial values of par

lower	a vector of length 3, lower bounds of the parameter values
upper	a vector of length 3, upper bounds of the parameter values
subset	a logical vector of length m, specifying the subset of rows to be used when estimating the dispersion model parameters.
pi.pre	a m by n matrix, preliminary estimates of the relative frequencies.
pi.pre	a m by n matrix, preliminary estimates of the relative frequencies.
pi.offset	a scalar used as an offset in the NBQ model (see Details)

`disp.nbs` *(private) Specify a NBS dispersion model*

Description

Specify a NBS dispersion model. The specified model are to be estimated using the function `optim.disp.pl` or `optim.disp.apl`. Under this NBS model, the dispersion is modeled as a smooth function (a natural cubic spline function) of the preliminary estimates of the log mean realtive frequencies (pi.pre).

Usage

```
disp.nbs(counts, eff.lib.sizes, x, df = 6, phi.pre = 0.1, mu.lower = 1,
  mu.upper = Inf)
```

Arguments

counts	a mxn matrix of NB counts
eff.lib.sizes	a n-vector of estimated effective library sizes
x	a nxp matrix, design matrix (specifying the treatment structure)
df	the number of interior nodes
phi.pre	a number, a preliminary constant dispersion value
mu.lower	a number, rows with mu.pre < mu.lower will not be used for estimating the dispersion model
mu.upper	a number, rows with mu.pre > mu.upper will not be used for estimating the dispersion model

Details

`disp.nbs` calls the function `ns` to generate a set of spline bases, using log(pi.pre) (converted to a vector) as the predictor variable. Linear combinations of these spline bases are smooth functions of log(pi.pre). The return value includes a function, `fun`, to be optimized by `optim.disp.pl` or `optim.disp.apl`. The parameter of that function is a vector of linear combination coefficients of the spline bases.

`df+2` nodes are used when constructing the splint bases. The Boundary.nodes are placed at the min and max values of log(pi.pre). Two nodes are placed at the 0.05 and 0.95th quantiles of log(pi.pre) and an additional `df-2` inner nodes are equally spaced between the two nodes.

It is a challenging issue to determine the optimal number and placement of the nodes.

Value

a list

fun	a function that takes a vector, par, as input and outputs a matrix (same dimension as counts) of dispersion values. par will be used as linear-combination efficients for the spline bases, the estimated dispersion values are a spline function of log(pi.pre).
par.init	initial values of par
subset	a logical vector of length m, specifying the subset of genes to be used when estimating the model parameters. Note that the estimated model will be applied to all rows whenever possible, but only rows specified in subset will be used to estimate the parameters of the dispersion model.
pi.pre	a m by n matrix, preliminary estimates of the relative frequencies.
s	Basis matrix of the natural cubic spline evalued at the z = log(pi.pre)

---

disp.predictor.mu *Dispersion precitor*

Description

Dispersion precitor

Usage

```
disp.predictor.mu(nb.data, x, phi.pre = 0.1, mu.lower = 1, mu.upper = Inf)
```

Arguments

nb.data

x

phi.pre

mu.lower

mu.upper

Value

a logical vector

`disp.step` *(private) Specify a piecewise constant dispersion model*

Description

Specify a piecewise constant (step) dispersion model. The specified model are to be estimated using the function `optim.disp.pl` or `optim.disp.apl`.

Usage

```
disp.step(counts, eff.lib.sizes, x, df = 1, knots = NULL, phi.pre = 0.1,
  mu.lower = 1, mu.upper = Inf)
```

Arguments

`counts`	a mxn matrix of NB counts
`eff.lib.sizes`	a n-vector of estimated effective library sizes
`x`	a nxp matrix, design matrix (specifying the treatment structure)
`df`	the number of steps
`knots`	a numerical vector of length df-1, giving the knots or jump locations.
`phi.pre`	a number, a preliminary constant dispersion value
`mu.lower`	a number, rows with mu.pre < mu.lower will not be used for estimating the dispersion model
`mu.upper`	a number, rows with mu.pre > mu.upper will not be used for estimating the dispersion model

Details

Under this model, the dispersion is modeled as a step (piecewise constant) function.

Value

a list

`fun`	a function that takes par as input and outputs a matrix (same dimension as `counts`) of dispersion values
`par.init`	a vector of length `df`, initial values of par
`subset`	a logical vector of length m, specifying a subset of genes to be used when estimating the model parameters. Note that the estimated model will be applied to all rows whenever possible, but only rows specified in `subset` will be used to estimate the dispersion model parameters.
`pi.pre`	a m by n matrix, preliminary estimates of the relative frequencies.
`knots`	a vector, the break points of the step function

Dispersion Models *(private) Specify a NB2, NBP, NBS, NBS, or STEP dispersion model*

Description

(private) Specify a NB2, NBP, NBS, NBS, or STEP dispersion model

Usage

```
disp.fun.nb2(predictor, subset, offset = NULL,
  predictor.label = "Predictor", par.init = -1)

disp.fun.nbp(predictor, subset, offset = median(predictor[subset, ]),
  predictor.label = "Predictor", par.init = c(log(0.1), 0),
  par.lower = c(log(1e-20), -1.1), par.upper = c(0, 0.1))

disp.fun.nbq(predictor, subset, offset = median(predictor[subset, ]),
  predictor.label = "Predictor", par.init = c(log(0.1), 0, 0),
  par.lower = c(log(1e-20), -1, -0.2), par.upper = c(0, 1, 0.2))

disp.fun.nbs(predictor, subset, offset = NULL,
  predictor.label = "Predictor", df = 6, par.init = rep(-1, df))

disp.fun.step(predictor, subset, offset = NULL,
  predictor.label = "Predictor", df = 6, knots = NULL,
  par.init = rep(-1, df))
```

Arguments

predictor	a m-by-n matrix having the same dimensions as the NB counts, predictor of the dispersion. See Details.
subset	a logical vector of length m, specifying the subset of rows to be used when estimating the dispersion model parameters.
offset	a scalar offset.
predictor.label	a string describing the predictor
par.init	a numeric vector, initial values of par.
label	a string character describing the predictor.
par.lower	a numeric vector, lower bounds of the parameter values.
par.upper	a numeric vector, upper bounds of the parameter values.

Details

Specify a NBP dispersion model. The parameters of the specified model are to be estimated from the data using the function `optim.disp.apl` or `optim.disp.pl`.

Under the NBP model, the log dispersion is modeled as a linear function of specified predictor with a scalar offset,

log(phi) = par[1] + par[2] * log(predictor/offset).

Under this parameterization, par[1] is the dispersion value when the value of predictor equals the offset. This function will return a function (and related settings) to be estimated by either `optim.disp.apl` or `optim.disp.pl`. The logical vector `subset` specifieds which rows will be used when estimating the paramters (`par`) of the dispersion model.

Once estimated, the dispersion function will be applied to all values of the `predictor` matrix. Care needs to be taken to either avoid `NA/Inf` values when preparing the predictor matrix or handle `NA/Inf` values afterwards (e.g., when performing hypothesis tests).

Value

a list

`fun`	a function that takes a vector, `par`, as input and outputs a matrix of dispersion values (same dimension as counts)
`par.init`, `par.lower`, `par.upper`	same as input
`subset`	same as input
`predictor`, `offset`, `predictor.lable`	same as input

`estimate.disp` *Fit a parametric disperison model to thinned counts*

Description

Fit a parametric dispersion model to RNA-Seq counts data prepared by `prepare.nbp`. The model parameters are estimated from the pseudo counts: thinned/down-sampled counts that have the same effective library size.

Usage

```
estimate.disp(obj, model = "NBQ", print.level = 1, ...)
```

Arguments

`obj`	output from `prepare.nbp`.
`model`	a string, one of "NBQ" (default), "NBP" or "NB2".
`print.level`	a number, controls the amount of messages printed: 0 for suppressing all messages, 1 for basic progress messages, larger values for more detailed messages.
`...`	additional parameters controlling the estimation of the parameters.

Details

For each individual gene i, a negative binomial (NB) distribution uses a dispersion parameter ϕ_i to capture the extra-Poisson variation between biological replicates: the NB model imposes a mean-variance relationship $\sigma_i^2 = \mu_i + \phi_i \mu_i^2$. In many RNA-Seq data sets, the dispersion parameter ϕ_i tends to vary with the mean μ_i. We proposed to capture the dispersion-mean dependence using parametric models.

With this function, `estimate.disp`, users can choose from three parametric models: NB2, NBP and NBQ (default).

Under the NB2 model, the dispersion parameter is a constant and does not vary with the mean expression levels.

Under the NBP model, the log dispersion is modeled as a linear function of preliminarily estimated log mean relative frequencies (`pi.pre`):

log(phi) = par[1] + par[2] * log(pi.pre/pi.offset),

Under the NBQ model, the log dispersion is modeled as a quadratic function of preliminarily estimated log mean relative frequencies (`pi.pre`):

log(phi) = par[1] + par[2] * log(pi.pre/pi.offset) + par[3] * (log(pi.pre/pi.offset))^2;

The NBQ model is more flexible than the NBP and NB2 models, and is the current default option.

In the NBP and NBQ models, `pi.offset` is fixed to be 1e-4, so par[1] corresponds to the dispersion level when the relative mean frequency is 100 reads per million (RPM).

The dispersion parameters are estimated from the pseudo counts (counts adjusted to have the same effective library sizes). The parameters are estimated by maximizing the log conditional likelihood of the model parameters given the row sums. The log conditional likelihood is computed for each gene in each treatment group and then summed over genes and treatment groups.

Value

The list `obj` from the input with some added components summarizing the fitted dispersion model. Users can print and plot the output to see brief summaries of the fitted dispersion model. The output is otherwise not intended for use by end users directly.

Note

Users should call `prepare.nbp` before calling this function. The function `prepare.nbp` will normalize the counts and adjust the counts so that the effective library sizes are approximately the same (computing the conditional likelihood requires the library sizes to be the same).

References

Di Y, Schafer DW, Cumbie JS, and Chang JH (2011): "The NBP Negative Binomial Model for Assessing Differential Gene Expression from RNA-Seq", Statistical Applications in Genetics and Molecular Biology, 10 (1).

See Also

`nbp.test`, `exact.nb.test`

Examples

```
## See the example for nb.exact.test
```

`estimate.dispersion` *Estimate Negative Binomial Dispersion*

Description

Estimate NB dispersion by modeling it as a parametric function of preliminarily estimated log mean relative frequencies.

Usage

```
estimate.dispersion(nb.data, x, model = "NBQ", predictor = "pi",
  method = "MAPL", fast = TRUE, ...)
```

Arguments

`nb.data`	output from `prepare.nb.data`.
`x`	a design matrix specifying the mean structure of each row.
`model`	the name of the dispersion model, one of "NB2", "NBP", "NBQ" (default), "NBS" or "step".
`predictor`	
`method`	a character string specifying the method for estimating the dispersion model, one of "ML" or "MAPL" (default).
`fast`	use a faster (but might be less accurate method)
`...`	additional parameters to optim.fun.<method>

Details

We use a negative binomial (NB) distribution to model the read frequency of gene i in sample j. A negative binomial (NB) distribution uses a dispersion parameter ϕ_{ij} to model the extra-Poisson variation between biological replicates. Under the NB model, the mean-variance relationship of a single read count satisfies $\sigma_{ij}^2 = \mu_{ij} + \phi_{ij}\mu_{ij}^2$. Due to the typically small sample sizes of RNA-Seq experiments, estimating the NB dispersion ϕ_{ij} for each gene i separately is not reliable. One can pool information across genes and biological samples by modeling ϕ_{ij} as a function of the mean frequencies and library sizes.

Under the NB2 model, the dispersion is a constant across all genes and samples.

Under the NBP model, the log dispersion is modeled as a linear function of the preliminary estimates of the log mean relative frequencies (`pi.pre`):

log(phi) = par[1] + par[2] * log(pi.pre/pi.offset),

where `pi.offset` is 1e-4.

Under the NBQ model, the dispersion is modeled as a quadratic function of the preliminary estimates of the log mean relative frequencies (pi.pre):

log(phi) = par[1] + par[2] * z + par[3] * z^2,

where z = log(pi.pre/pi.offset). By default, pi.offset is the median of pi.pre[subset,].

Under this NBS model, the dispersion is modeled as a smooth function (a natural cubic spline function) of the preliminary estimates of the log mean relative frequencies (pi.pre).

Under the "step" model, the dispersion is modeled as a step (piecewise constant) function.

Value

a list with following components:

`estimates`	dispersion estimates for each read count, a matrix of the same dimensions as the `counts` matrix in `nb.data`.
`likelihood`	the likelihood of the fitted model.
`model`	details of the estimate dispersion model, NOT intended for use by end users. The name and contents of this component are subject to change in future versions.

Note

Currently, it is unclear whether a dispersion-modeling approach will outperform a more basic approach where regression model is fitted to each gene separately without considering the dispersion-mean dependence. Clarifying the power-robustness of the dispersion-modeling approach is an ongoing research topic.

Examples

```
## See the example for test.coefficient.
```

`estimate.norm.factors` *Estiamte Normalization Factors*

Description

`estimate.norm.factors` estiamtes normalization factors to account for apparent reduction or increase in relative frequencies of non-differentially expressing genes as a result of compensating the increased or decreased relative frequencies of truly differentially expressing genes.

Usage

```
estimate.norm.factors(counts, lib.sizes = colSums(counts),
  method = "AH2010")
```

Arguments

counts	a matrix of RNA-Seq read counts with rows corresponding to gene features and columns corresponding to independent biological samples.
lib.sizes	a vector of observed library sizes, usually and by default estimated by column totals.
method	a character string specifying the method for normalization, currenlty, can be NULL or "AH2010". If method=NULL, the normalization factors will have values of 1 (i.e., no normalization is applied); if method="AH2010" (default), the normalization method proposed by Anders and Huber (2010) will be used.

Details

We take gene expression to be indicated by relative frequency of RNA-Seq reads mapped to a gene, relative to library sizes (column sums of the count matrix). Since the relative frequencies sum to 1 in each library (one column of the count matrix), the increased relative frequencies of truly over expressed genes in each column must be accompanied by decreased relative frequencies of other genes, even when those others do not truly differentially express. If not accounted for, this may give a false impression of biological relevance (see, e.g., Robinson and Oshlack (2010), for some examples.) A simple fix is to compute the relative frequencies relative to effective library sizes—library sizes multiplied by normalization factors.

Value

a vector of normalization factors.

References

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

Robinson, M. D. and A. Oshlack (2010): "A scaling normalization method for differential expression analysis of RNA-seq data," Genome Biol., 11, R25.

Examples

```
## Load Arabidopsis data
data(arab)

## Estimate normalization factors using the method of Anders and Huber (2010)
norm.factors = estimate.norm.factors(arab);
print(norm.factors);
```

`exact.nb.test` *Exact Negative Binomial Test for Differential Gene Expression*

Description

`exact.nb.test` performs the Robinson and Smyth exact negative binomial (NB) test for differential gene expression on each gene and summarizes the results using p-values and q-values (FDR).

Usage

```
exact.nb.test(obj, grp1, grp2, print.level = 1)
```

Arguments

`obj`	output from `estimate.disp`.
`grp1`	Identifier of group 1. A number, character or string (should match at least one of the obj$grp.ids).
`grp2`	Identifier of group 2. A number, character or string (should match at least one of the obj$grp.ids).
`print.level`	a number. Controls the amount of messages printed: 0 for suppressing all messages, 1 for basic progress messages, larger values for more detailed messages.

Details

The negative binomial (NB) distribution offers a more realistic model for RNA-Seq count variability and still permits an exact (non-asymptotic) test for comparing expression levels in two groups.

For each gene, let S_1, S_2 be the sums of gene counts from all biological replicates in each group. The exact NB test is based on the conditional distribution of $S_1|S_1+S_2$: a value of S_1 that is too big or too small, relative to the sum $S_1 + S_2$, indicates evidence for differential gene expression. When the effective library sizes are the same in all replicates and the dispersion parameters are known, we can determine the probability functions of S_1, S_2 explicitly. The exact p-value is computed as the total conditional probability of all possible values of (S_1, S_2) that have the same sum as but are more extreme than the observed values of (S_1, S_2).

Note that we assume that the NB dispersion parameters for the two groups are the same and library sizes (column totals of the count matrix) are the same.

Value

the list `obj` from the input with the following added components:

`grp1`	same as input.
`grp2`	same as input.
`pooled.pie`	estimated pooled mean of relative count frequencies in the two groups being compared.

`expression.levels`
: a matrix of estimated gene expression levels as indicated by mean relative read frequencies. It has three columns `grp1`, `grp2`, `pooled` corresponding to the two treatment groups and the pooled mean.

`log.fc`
: base 2 log fold change in mean relative frequency between two groups.

`p.values`
: p-values of the exact NB test applied to each gene (row).

`q.values`
: q-values (estimated FDR) corresponding to the p-values.

Note

Before calling `exact.nb.test`, the user should call `estimate.norm.factors` to estimate normalization factors, call `prepare.nbp` to adjust library sizes, and call `estimate.disp` to fit a dispersion model. The exact NB test will be performed using `pseudo.counts` in the list `obj`, which are normalized and adjusted to have the same effective library sizes (column sums of the count matrix, multiplied by normalization factors).

Users not interested in fine tuning the underlying statistical model should use `nbp.test` instead. The all-in-one function `nbp.test` uses sensible approaches to normalize the counts, estimate the NBP model parameters and test for differential gene expression.

A test will be performed on a row (a gene) only when the total row count is nonzero, otherwise an NA value will be assigned to the corresponding p-value and q-value.

See Also

`nbp.test`.

Examples

```
## Load Arabidopsis data
data(arab);

## Specify treatment groups
## grp.ids = c(1, 1, 1, 2, 2, 2); # Numbers or strings are both OK
grp.ids = rep(c("mock", "hrcc"), each=3);

## Estimate normalization factors
norm.factors = estimate.norm.factors(arab);
print(norm.factors);

## Prepare an NBP object, adjust the library sizes by thinning the
## counts. For demonstration purpose, only use the first 100 rows of
## the arab data.
set.seed(999);
obj = prepare.nbp(arab[1:100,], grp.ids, lib.size=colSums(arab), norm.factors=norm.factors);
print(obj);

## Fit a dispersion model (NBQ by default)
obj = estimate.disp(obj);
plot(obj);

## Perform exact NB test
```

```
## grp1 = 1;
## grp2 = 2;
grp1 = "mock";
grp2 = "hrcc";

obj = exact.nb.test(obj, grp1, grp2);

## Print and plot results
print(obj);
par(mfrow=c(3,2));
plot(obj);
```

`filter.mu.pre` *Create a logical vector specifyfing the subset of rows to be used when estimating the dispersion model*

Description

Create a logical vector specifyfing the subset of rows to be used when estimating the dispersion model

Usage

```
filter.mu.pre(nb.data, x, mu.lower = 1, mu.upper = Inf, phi.pre = 0.1)
```

Arguments

`nb`

`x`

`mu.lower`

`mu.upper`

Value

a logical vector specifyfing the subset of rows to be used when estimating the dispersion model

`fit.nb.glm.1` *Fit a single negative binomial (NB) log-linear regression model with known dispersion paramreters*

Description

Fit a NB log-linear regression model: find the MLE of the regression coefficients and compute likelihood of the fitted model, the score vector, and the Fisher and observed information.

Usage

```
fit.nb.glm.1(y, s, x, phi, beta0 = rep(NA, dim(x)[2]), ...)
```

Arguments

`y`	an n-vector of NB counts.
`s`	an n-vector of library sizes (multiplicative offset).
`x`	an n by p design matrix.
`phi`	a scalar or an n-vector, the NB dipsersion parameter.
`beta0`	a p-vector specifying the known and unknown components of beta, the regression coefficients. NA values indicate unknown components and non-NA values specify the values of the known components. The default is that all components of beta are unknown.
`...`	furhter arguements to be passed to `irls.nb.1`.

Details

Under the NB regression model, the components of y follow a NB distribution with means mu = s exp(x' beta) and dispersion parameters phi.

The function will call `irls.nb.1` to find MLE of the regression coefficients (which uses the iteratively reweighted least squres (ILRS) algorithm).

Value

a list

`mu`	an n-vector, estimated means (MLE).
`beta`	an p-vector, estimated regression coefficients (MLE).
`iter`	number of iterations performed in the IRLS algorithm.
`zero`	logical, whether any of the estimated `mu` is close to zero.
`l`	log likelihood of the fitted model.
`D`	a p-vector, the score vector
`i`	a p-by-p matrix, fisher information matrix
`j`	a p-by-p matrix, observed information matrix

Note

The information matries, i and j, will be computed for all all components of beta—including known components.

`fit.nb.glm.1u` *Fit a single negative binomial (NB) log-linear regression model with a common unknown dispersion paramreter*

Description

Fit a single negative binomial (NB) log-linear regression model with a common unknown dispersion paramreter.

Usage

```
fit.nb.glm.1u(y, s, x, phi = NA, beta0 = rep(NA, dim(x)[2]),
  kappa = 1/phi, info.kappa = TRUE, ...)
```

Arguments

`y`	a n-vector of NB counts.
`s`	a n-vector of library sizes.
`x`	a n by p design matrix.
`phi`	a scalar, the NB dipsersion parameter.
`beta0`	a p-vector specifying the known and unknown components of beta, the regression coefficients. NA values indicate unknown components and non-NA values specify the values of the known components. The default is that all components of beta are unknown.
`kappa`	a scalar, the size/shape parameter. `kappa` will be set to `1/phi` if `phi` is not `NA` and will be estiamted if both `phi` and `kappa` are NA.
`info.kappa`	
`...`	additional parameters to `irls.nb.1`

Details

Find the MLE of the dipsersion parameter and the regression coefficients in a NB regression model.

Under the NB regression model, the components of y follow a NB distribution with means mu = s exp(x' beta) and a common dispersion parameter phi.

Value

a list

mu	an n-vector, estimated means (MLE).
beta	an p-vector, estimated regression coefficients (MLE).
iter	number of iterations performed in the IRLS algorithm.
zero	logical, whether any of the estimated mu is close to zero.
kappa	a scalar, the size parameter
phi	a scalr, 1/kappa, the dispsersion parameter
l	log likelihood of the fitted model.
D	a p-vector, the score vector
j	a p-by-p matrix, observed information matrix

Note

When the disperison is known, the user should specify only one of phi or kappa. Whenever phi is specified (non-NA), kappa will be set to 1/phi.

The observed information matrix, j, will be computed for all parameters—kappa and all components of beta (including known components). It will be computed at the estimated values of (phi, beta) or (kappa, beta), which can be unconstrained or constrained MLEs depending on how the arguments phi (or kappa) and beta are specified.

TODO: allow computing the information matrix using phi or log(kappa) as parameter

---

get.mean.hat *(private) Extract row means of the pseudo counts for the specified group from an* nbp *object.*

Description

(private) Extract row means of the pseudo counts for the specified group from an nbp object.

Usage

```
get.mean.hat(obj, grp.id)
```

Arguments

obj	a list with class nbp, output prepare.nbp, estimate.disp, exact.nb.test or nbp.test
grp.id	a number or a charater (same type as obj$grp.ids), group id

`get.nbp.pars` *(private) Retrieve nbp parameters for one of the treatment groups from an nbp object*

Description

(private) Retrieve nbp parameters for one of the treatment groups from an nbp object

Usage

```
get.nbp.pars(obj, grp.id)
```

Arguments

`obj`	output form nbp.mcle
`grp.id`	the id of a treatment grp

Value

a list

`n`	number of genes
`r`	number of replicates
`lib.sizes`	library sizes
`pie`	estimated mean relatiev frequenices
`phi, alpha`	dispersion model parameters

`get.rel.mean` *(private) Extract row relative means of the pseudo counts for the specified group from an* nbp *object.*

Description

(private) Extract row relative means of the pseudo counts for the specified group from an nbp object.

Usage

```
get.rel.mean(obj, grp.id)
```

Arguments

`obj`	a list with class nbp, output `prepare.nbp`, `estimate.disp`, `exact.nb.test` or `nbp.test`
`grp.id`	a number or a charater (same type as `obj$grp.ids`), group id

get.var.hat *(private) Extract estimated variance from the oupput of* nbp-mcle *or* nbp-test

Description

(private) Extract estimated variance from the oupput of nbp-mcle or nbp-test

Usage

```
get.var.hat(obj, grp.id)
```

Arguments

obj	a list, output from nbp-mcle or nbp-test
grp.id	a number, group id

hist2d *2-d Histogram*

Description

Commpute a 2-d histogram of the given data values. (not implemented yet: If plot == TRUE, plot the resulting histogram.)

Usage

```
hist2d(x, y, xlim = range(x), ylim = range(y), nbins)
```

Arguments

x	a vector
y	a vector of the same length as x
xlim	a vector of length 2, the range of x values
ylim	a vector of length 2, the range of y values
nbins	a single number giving the number of bins (the same for both x- and y- axes).

Details

This funciton divides the xlim x ylim region into nbins x nbins equal-sized cells and count the number of (x,y) points in each cell.

Value

a list

x	a vector of length nbins, the midpoints of each bin on the x-axis.
y	a vector of length nbins, the midpoints of each bin on the y-axis.
z	a nbins by nbins matrix of of counts. For each cell, the number of (x, y) inside.

The list can be passed to `image()` directly for potting.

Note

Only points inside the region defined by `xlim` x `ylim` (inclusive) will be counted. For each cell, the lower boundaries are closed and upper boundaries are open. A small number will be added to the upper limits in xlim and ylim so that no points will be on the region's upper boundaries.

---

hoa.1d *(private) One-dimensional HOA test for a regression coefficient in an NB GLM model*

Description

(private) One-dimensional HOA test for a regression coefficient in an NB GLM model

Usage

```
hoa.1d(y, s, x, phi, beta0, tol.mu = 0.001/length(y),
  alternative = "two.sided", print.level = 1)
```

Arguments

y	an n vector of counts
s	an n vector of effective library sizes
x	an n by p design matrix
phi	an n vector of dispersion parameters
beta0	a p vector specifying null hypothesis: non NA components are hypothesized values of beta, NA components are free components
tol.mu	convergence criteria
alternative	"less" means phi < 0.
print.level	a number, print level

Value

test statistics and p-values of HOA, LR, and Wald tests

hoa.hd *(private) HOA test for regression coefficients in an NBP GLM model*

Description

(private) HOA test for regression coefficients in an NBP GLM model

Usage

```
hoa.hd(y, s, x, phi, beta0, tol.mu = 0.001/length(y), print.level = 1)
```

Arguments

y	an n vector of counts
s	an n vector of effective library sizes
x	an n by p design matrix
phi	an n vector of dispersion parameters
beta0	a p vector specifying null hypothesis: non NA components are hypothesized values of beta, NA components are free components
tol.mu	convergence criteria
print.level	a number, print level

Value

test statistics and p-values of HOA, LR, and Wald tests

irls.nb *(private) Estiamte the regression coefficients in an NB GLM model*

Description

Estimate the regression coefficients in an NBP GLM model for each gene

Usage

```
irls.nb(y, s, x, phi, beta0 = rep(NA, ncol(x)), mustart = NULL, ...,
  print.level = 0)
```

Arguments

y	an m*n matrix of counts
s	an n vector of effective library sizes
x	an n*p design matrix
phi	a scalar or an m*n matrix of NB2 dispersion coefficients
beta0	a K vector, non NA components are hypothesized values of beta, NA components are free components
mustart	an m*n matrix of starting values of the means
...	other parameters
print.level	a number, print level

Value

beta a K vector, the MLE of the regression coefficients.

---

irls.nb.1 *Estimate the regression coefficients in an NB GLM model*

Description

Estimate the regression coefficients in an NB GLM model with known dispersion parameters

Usage

```
irls.nb.1(y, s, x, phi, beta0 = rep(NA, p), mustart = NULL, maxit = 50,
  tol.mu = 0.001/length(y), print.level = 0)
```

Arguments

y	an n vector of counts
s	a scalar or an n vector of effective library sizes
x	an n by p design matrix
phi	a scalar or an n-vector of dispersion parameters
beta0	a vector specifying known and unknown components of the regression coefficients: non-NA components are hypothesized values of beta, NA components are free components
mustart	starting values for the vector of means
maxit	maximum number of iterations
tol.mu	a number, convergence criteria
tol	a number, will be passed to Cdqrls
print.level	a number, print level

Details

This function estimates the regression coefficients using iterative reweighted least squares (IRLS) algorithm, which is equivalent to Fisher scoring. The implementation is based on `glm.fit`.

Users can choose to fix some regression coefficients by specifying `beta0`. (This is useful when fitting a model under a null hypothesis.)

Value

a list of the following components:

`beta`	a p-vector of estimated regression coefficients
`mu`	an n-vector of estimated mean values
`conv`	logical. Was the IRLS algorithm judged to have converged?
`zero`	logical. Was any of the fitted mean close to 0?

`l.nb` *(private) The Log Likelihood of a NB Model*

Description

The log likelihood of the NB model under the mean shape parameterization

Usage

```
l.nb(kappa, mu, y)
```

Arguments

`kappa`	shape/size parameter
`mu`	mean parameter
`y`	a n-vector of NB counts

Details

This function call dnbinom to compute the log likelihood from each data point and sum the results over all data points. kappa, mu and y should have compatible dimensions.

Value

the log likelihood of the NB model parameterized by `(kappa, mu)`

log.phi.nb2 *(private) A NBP dispersion model*

Description

Specify a NB2 dispersion model. The parameter of the specified model are to be estimated from the data using the function `optim.pcl`.

Usage

```
## S3 method for class 'phi.nb2'
log(par, pi)
```

Arguments

`par`	a number, log dispersion

Details

Under this NB2 model, the log dispersion is a constant.

Value

a vector of length m, log dipserison.

log.phi.nbp *(private) A NBP dispersion model*

Description

Specify a NBP dispersion model. The parameters of the specified model are to be estimated from the data using the function `optim.pcl`.

Usage

```
## S3 method for class 'phi.nbp'
log(par, pi, pi.offset = 1e-04)
```

Arguments

`par`	a vector of length 2, the intercept and the slope of the log linear model (see Details).
`pi`	a vector of length m, estimated mean relative frequencies
`pi.offset`	a number

Details

Under this NBP model, the log dispersion is modeled as a linear function of the log mean realtive frequencies (`pi.pre`):

log(phi) = par[1] + par[2] * log(pi.pre/pi.offset),

where the default value of `pi.offset` is 1e-4.

Value

a vector of length m, log dipserison.

`log.phi.nbq` *(private) A NBQ dispersion model*

Description

Specify a NBQ dispersion model. The parameters of the specified model are to be estimated from the data using the function `optim.pcl`.

Usage

```
## S3 method for class 'phi.nbq'
log(par, pi, pi.offset = 1e-04)
```

Arguments

`par`	a vector of length 3, see Details.
`pi`	a vector of length m, estimated mean relative frequencies
`pi.offset`	a number

Details

Under this NBQ model, the log dispersion is modeled as a quadratic function of the log mean realtive frequencies (`pi`):

log(phi) = par[1] + par[2] * log(pi/pi.offset) + par[3] * (log(pi/pi.offset))^2;

where the (default) value of `pi.offset` is 1e-4.

Value

a vector of length m, log dipserison.

`ma.plot` *(private) MA plot with differently expressed genes highlighted*

Description

Plot log (base 2) fold change vs average expression in RPM (two-group pooled) (i.e., an MA plot) and highlight differentially expressed genes on the plot.

Usage

```
ma.plot(test.out, top = NULL, q.cutoff = NULL, p.cutoff = NULL,
  col.sig = "magenta", main = "MA Plot", ...)
```

Arguments

`test.out`	output from `nbp.test`
`top`	a number indicating the number of genes to be declared as differentially expressed
`q.cutoff`	a number, q-value cutoff
`p.cutoff`	a number, p-value cutoff
`col.sig`	color
`main`	label
`...`	additional parameters to be passed to `smart.plot`

Details

Differentially expressed genes are those with smallest DE test p-values. The user has three options to specify the set of DE genes: the user can specify 1) the number of top genes to be declared as significant; 2) a q-value cutoff; or 3) a p-value cutoff.

The plot is based on the thinned counts. The units on the x-axis is RPM (reads per million mapped reads). We use RPM so that the results are more comparable between experiments with different sequencing depth (and thus different column totals in the count matrix). We exclude rows (genes) with 0 total counts after thinning.

Value

a vector, indices of top genes.

`make.disp` *(private) Specify a dispersion model*

Description

Specify a dispersion model. The parameters of the specified model are to be estimated from the data using the function `optim.disp.apl` or `optim.disp.pl`.

Usage

```
make.disp(nb.data, x, model, predictor, subset = filter.mu.pre(nb.data, x),
  ...)
```

Arguments

`nb`	NB data, output from `prepare.nb.data`
`x`	a matrix, design matrix (specifying the treatment structure).
`model`	a string giving the name of the disperion model, can be one of "NB2", "NBP", "NBQ", "NBS" or "step" (not case sensitive).
`predictor`	a string giving the name of the predictor to use in the dispersion model, can be one of "pi" and "mu", or "rs". `"pi"`, preliminarily estimated mean relative frequencies; `"mu"`, preliminarily estimated mean frequencies; `"rs"`, row sums.
`subset`	a list of logical,
`...`	additional parameter to `disp.fun.*`

Details

This functions calls `disp.fun.<model>` to specify a dispersion model (a list), using output from a call to `disp.predictor.<predictor>` as argument list, where `<model>` is `model` from the input in lower case (one of "nb2", "nbp", "nbq", "nbs" or "step") and `<predictor>` is `predictor` from the input (one of "pi", "mu", or "rs")

Value

a list, output from the call to the funtion `disp.fun.<model>`.

mv.line *(private) Overlay an estimated mean-variance line*

Description

Overlay an estimated mean-variance line on an existing mean-variance plot

Usage

```
mv.line(mu, v, ...)
```

Arguments

mu	a vector of mean values
v	a vector of variance values
...	other

Details

Users should call mv.plot before calling this function.

If the length of theinput vectors (mu, v) is greater than 1000, then we will only use a subset of the input vectors.

mv.line.fitted *(private) Overlay an estimated mean-variance line*

Description

Overlay an estimated mean-variance line on existing plot

Usage

```
mv.line.fitted(obj, ...)
```

Arguments

obj	a list with components mu, a vector of mean values, and v, a vector of variance values.
...	other parameters

Details

This functions is a wrapper of mv.line. It takes a list (rather than two vectors) as input.

Note

Users should call mv.plot before calling this function.

See Also

mv.line

`mv.line.nbp` *(private) Overlay a NBP mean-variance line on an existing plot*

Description

Overlay an estimated mean-variance line on existing plot

Usage

```
mv.line.nbp(nbp.obj, grp.id, ...)
```

Arguments

`nbp.obj`	output from nbp.test or prepare.nbp
`grp.id`	a number, indicates the group of counts to be used (grp.id is passed to `get.mean.hat`
`...`	other parameters

Details

This function extracts the estimated means and variances from an nbp object and then call `mv.line` to draw the mean-variance line on an existing plot

Note

Users should call mv.plot before calling this function.

See Also

`prepare.nbp`, `nbp.test`, `mv.line`

mv.plot *(private) Mean-variance plot*

Description

Mean-variance plot.

Usage

```
mv.plot(counts, xlab = "mean", ylab = "variance",
  main = "variance vs mean", log = "xy", ...)
```

Arguments

counts	a matrix of NB counts
xlab	x label
ylab	y label
main	main, same as in plot
log	same as in plot
...	same as in plot

Details

Rows with mean 0 or variance 0 will not be plotted.

mv.points *(private) Highlight a subset of points on the mean-variance plot*

Description

Highlight a subset of points on the mean-variance plot

Usage

```
mv.points(counts, subset, ...)
```

Arguments

counts	a matrix of NB counts
subset	a numberic or logical vector indicating the subset
...	other
	of rows to be highlighted

nb.glm.test *Fit Negative Binomial Regression Model and Test for a Regression Coefficient*

Description

For each row of the input data matrix, nb.glm.test fits an NB log-linear regression model and performs large-sample tests for a one-dimensional regression coefficient.

Usage

```
nb.glm.test(counts, x, beta0, lib.sizes = colSums(counts),
  normalization.method = "AH2010", dispersion.model = "NBQ",
  tests = c("HOA", "LR", "Wald"), alternative = "two.sided",
  subset = 1:dim(counts)[1])
```

Arguments

counts	an m by n matrix of RNA-Seq read counts with rows corresponding to gene features and columns corresponding to independent biological samples.
x	an n by p design matrix specifying the treatment structure.
beta0	a p-vector specifying the null hypothesis. Non-NA components specify the parameters to test and their null values.
lib.sizes	a p-vector of observed library sizes, usually (and by default) estimated by column totals.
normalization.method	a character string specifying the method for estimating the normalization factors, can be NULL or "AH2010". If method=NULL, the normalization factors will have values of 1 (i.e., no normalization is applied); if method="AH2010", the normalization method proposed by Anders and Huber (2010) will be used.
dispersion.model	a character string specifying the dispersion model, and can be one of "NB2", "NBP", "NBQ" (default), "NBS" or "step".
tests	a character string vector specifying the tests to be performed, can be any subset of "HOA" (higher-order asymptotic test), "LR" (likelihood ratio test), and "Wald" (Wald test).
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
subset	specify a subset of rows to perform the test on

Details

nbp.glm.test provides a simple, one-stop interface to performing a series of core tasks in regression analysis of RNA-Seq data: it calls estimate.norm.factors to estimate normalization factors;

it calls prepare.nb.data to create an NB data structure; it calls estimate.dispersion to estimate the NB dispersion; and it calls test.coefficient to test the regression coefficient.

To keep the interface simple, nbp.glm.test provides limited options for fine tuning models/parameters in each individual step. For more control over individual steps, advanced users can call estimate.norm.factors, prepare.nb.data, estimate.dispersion, and test.coefficient directly, or even substitute one or more of them with their own versions.

Value

A list containing the following components:

data	a list containing the input data matrix with additional summary quantities, output from prepare.nb.data.
dispersion	dispersion estimates and models, output from estimate.dispersion.
test	test results, output from test.coefficient.

Examples

```
## Load Arabidopsis data
data(arab);

## Specify treatment structure
grp.ids = as.factor(c(1, 1, 1, 2, 2, 2));
x = model.matrix(~grp.ids);

## Specify the null hypothesis
## The null hypothesis is beta[1]=0 (beta[1] is the log fold change).
beta0 = c(NA, 0);

## Fit NB regression model and perform large sample tests.
## The step can take long if the number of genes is large
fit = nb.glm.test(arab, x, beta0, subset=1:50);

## The result contains the data, the dispersion estimates and the test results
print(str(fit));

## Show HOA test results for top ten genes
subset = order(fit$test.results$HOA$p.values)[1:10];
cbind(fit$data$counts[subset,], fit$test.results$HOA[subset,]);

## Show LR test results
subset = order(fit$test.results$LR$p.values)[1:10];
cbind(fit$data$counts[subset,], fit$test.results$LR[subset,]);
```

nbp.test *NBP Test for Differential Gene Expression from RNA-Seq Counts*

Description

nbp.test fits an NBP model to the RNA-Seq counts and performs Robinson and Smyth's exact NB test on each gene to assess differential gene expression between two groups.

Usage

```
nbp.test(counts, grp.ids, grp1, grp2, norm.factors = rep(1, dim(counts)[2]),
  model.disp = "NBQ", lib.sizes = colSums(counts), print.level = 1, ...)
```

Arguments

counts	an n by r matrix of RNA-Seq read counts with rows corresponding to genes (exons, gene isoforms, etc) and columns corresponding to libraries (independent biological samples).
grp.ids	an r vector of treatment group identifiers (e.g. integers).
grp1	group 1 id
grp2	group 2 id
norm.factors	an r vector of normalization factors.
model.disp	a string, one of "NB2", "NBP" or "NBQ" (default).
lib.sizes	(unnormalized) library sizes
print.level	a number, controls the amount of messages printed: 0 for suppressing all messages, 1 (default) for basic progress messages, and 2 to 5 for increasingly more detailed messages.
...	optional parameters to be passed to estimate.disp, the function that estimates the dispersion parameters.

Details

nbp.test calls prepare.nbp to create the NBP data structure, perform optional normalization and adjust library sizes, calls estimate.disp to estimate the NBP dispersion parameters and exact.nb.test to perform the exact NB test for differential gene expression on each gene. The results are summarized using p-values and q-values (FDR).

Overview: For assessing evidence for differential gene expression from RNA-Seq read counts, it is critical to adequately model the count variability between independent biological replicates. Negative binomial (NB) distribution offers a more realistic model for RNA-Seq count variability than Poisson distribution and still permits an exact (non-asymptotic) test for comparing two groups.

For each individual gene, an NB distribution uses a dispersion parameter ϕ_i to model the extra-Poisson variation between biological replicates. Across all genes, parameter ϕ_i tends to vary with the mean μ_i. We capture the dispersion-mean dependence using a parametric model: NB2, NBP and NBQ. (See estimate.disp for more details.)

Count Normalization: We take gene expression to be indicated by relative frequency of RNA-Seq reads mapped to a gene, relative to library sizes (column sums of the count matrix). Since the relative frequencies sum to 1 in each library (one column of the count matrix), the increased relative frequencies of truly over expressed genes in each column must be accompanied by decreased relative frequencies of other genes, even when those others do not truly differentially express. Robinson and Oshlack (2010) presented examples where this problem is noticeable.

A simple fix is to compute the relative frequencies relative to effective library sizes—library sizes multiplied by normalization factors. By default, `nbp.test` assumes the normalization factors are 1 (i.e. no normalization is needed). Users can specify normalization factors through the argument `norm.factors`. Many authors (Robinson and Oshlack (2010), Anders and Huber (2010)) propose to estimate the normalization factors based on the assumption that most genes are NOT differentially expressed.

Library Size Adjustment: The exact test requires that the effective library sizes (column sums of the count matrix multiplied by normalization factors) are approximately equal. By default, `nbp.test` will thin (downsample) the counts to make the effective library sizes equal. Thinning may lose statistical efficiency, but is unlikely to introduce bias.

Value

a list with the following components:

`counts`	an n by r matrix of counts, same as input.
`lib.sizes`	an r vector, column sums of the count matrix.
`grp.ids`	an r vector, identifiers of treatment groups, same as input.
`grp1`, `grp2`	identifiers of the two groups to be compared, same as input.
`eff.lib.sizes`	an r vector, effective library sizes, lib.sizes multiplied by the normalization factors.
`pseudo.counts`	count matrix after thinning, same dimension as counts
`pseduo.lib.sizes`	an r vector, effective library sizes of pseudo counts, i.e., column sums of the pseudo count matrix multiplied by the normalization.
`phi`, `alpha`	two numbers, parameters of the dispersion model.
`pie`	a matrix, same dimension as `counts`, estimated mean relative frequencies of RNA-Seq reads mapped to each gene.
`pooled.pie`	a matrix, same dimenions as `counts`, estimated pooled mean of relative frequencies in the two groups being compared.
`expression.levels`	a n by 3 matrix, estimated gene expression levels as indicated by mean relative frequencies of RNA-Seq reads. It has three columns `grp1`, `grp2`, `pooled` corresponding to the two treatment groups and the pooled mean.
`log.fc`	an n-vector, base 2 log fold change in mean relative frequency between two groups.
`p.values`	an n-vector, p-values of the exact NB test applied to each gene (row).
`q.values`	an n-vector, q-values (estimated FDR) corresponding to the p-values.

Note

Due to thinning (random downsampling of counts), two identical calls to nbp.test may yield slightly different results. A random number seed can be used to make the results reproducible. The regression analysis method implemented in nb.glm.test does not require thinning and can also be used to compare expression in two groups.

Advanced users can call estimate.norm.factors, prepare.nbp, estimate.disp, exact.nb.test directly to have more control over modeling and testing.

References

Di, Y, D. W. Schafer, J. S. Cumbie, and J. H. Chang (2011): "The NBP Negative Binomial Model for Assessing Differential Gene Expression from RNA-Seq", Statistical Applications in Genetics and Molecular Biology, 10 (1).

Robinson, M. D. and G. K. Smyth (2007): "Moderated statistical tests for assessing differences in tag abundance," Bioinformatics, 23, 2881-2887.

Robinson, M. D. and G. K. Smyth (2008): "Small-sample estimation of negative binomial dispersion, with applications to SAGE data," Biostatistics, 9, 321-332.

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

Robinson, M. D. and A. Oshlack (2010): "A scaling normalization method for differential expression analysis of RNA-seq data," Genome Biol., 11, R25.

See Also

prepare.nbp, estimate.disp, exact.nb.test.

Examples

```
## Load Arabidopsis data
data(arab);

## Specify treatment groups and ids of the two groups to be compared
grp.ids = c(1, 1, 1, 2, 2, 2);
grp1 = 1;
grp2 = 2;

## Estimate normalization factors
norm.factors = estimate.norm.factors(arab);

## Set a random number seed to make results reproducible
set.seed(999);

## Fit the NBP model and perform exact NB test for differential gene expression.
## For demonstration purpose, we will use the first 100 rows of the arab data.
res = nbp.test(arab[1:100,], grp.ids, grp1, grp2,
  lib.sizes = colSums(arab), norm.factors = norm.factors, print.level=3);

## The argument lib.sizes is needed since we only use a subset of
## rows. If all rows are used, the following will be adequate:
```

```
##
## res = nbp.test(arab, grp.ids, grp1, grp2, norm.factors = norm.factors);

## Show top ten most differentially expressed genes
subset = order(res$p.values)[1:10];
print(res, subset);

## Count the number of differentially expressed genes (e.g. qvalue < 0.05)
alpha = 0.05;
sig.res = res$q.values < alpha;
table(sig.res);

## Show boxplots, MA-plot, mean-variance plot and mean-dispersion plot
par(mfrow=c(3,2));
plot(res);
```

`nll.log.phi.fun` *Negative profile conditional likelihood of the dispersion model*

Description

Negative profile conditional likelihood of the dispersion model

Usage

```
nll.log.phi.fun(par, log.phi.fun, y, ls, n.grps, grps, grp.sizes, mu.lower,
  mu.upper, print.level)
```

Arguments

`par`	parameter of the disperesion model
`log.phi.fun`	the disperison model
`y`	counts
`ls`	library sizes
`n.grps`	number of groups
`grps`	a boolean matrix of group membership
`grp.sizes`	group sizes
`mu.lower`	lower bound for mu
`mu.upper`	upper bound for mu
`print.level`	print level

Value

negative log likelihood of the dispersion function

optim.disp.apl *(private) Estimate the parameters in a dispersion model*

Description

Estimate the parameters in a dispersion model.

Usage

```
optim.disp.apl(disp, counts, eff.lib.sizes, x, method = "L-BFGS-B",
  mustart = NULL, fast = FALSE, print.level = 1, ...)
```

Arguments

disp	a list, output from disp.nbp, disp.nbq.
counts	a matrix, the nb counts
eff.lib.sizes	effective library sizes
x	a desing matrix
method	the optimization method to be used by optim
print.level	print level
mustart	a matrix of the same dimension as counts, starting values of mu
fast	logical, if TRUE will use a faster (but less accurate) method
...	additional parareters, will be passed to optim().

Details

The function will call the R funciton optim to mimimize the negative log adjusted profile likelihood of the dipserison model.

Value

a list with components:

optim.disp.pl *(private) Estimate the parameters in a dispersion model*

Description

Estimate the parameters in a dispersion model.

Usage

```
optim.disp.pl(disp, counts, eff.lib.sizes, x, method = "L-BFGS-B",
  mustart = NULL, fast = FALSE, ...)
```

Arguments

disp	a list, output from disp.nbp, disp.nbq, disp.nbs, and so on.
counts	a matrix, the nb counts
eff.lib.sizes	effective library sizes
x	a desing matrix
method	the optimization method to be used by optim
mustart	a matrix of the same dimension as counts, starting values of mu
fast	logical, if TRUE will use a faster (but less accurate) method
...	additional parareters, will be passed to optim().

Details

The function will call the R funciton optim to mimimize the negative log likelihood of the dipserison model.

Value

a list with components:

---

phi.line *(private) Overlay an mean-dispersion line on an esimtated plot*

Description

Users should call vmr.plot before calling this function.

Usage

```
phi.line(mu, v, alpha = 2, ...)
```

Arguments

mu	a vector of mean values
v	a vector of variance values
alpha	alpha
...	other

Details

If the length of theinput vectors (mu, v) is greater than 1000, then we will only use a subset of the input vectors.

The dispersion is computed from the mean mu and the variance v, using $\phi = (v-\mu)/\mu^a lpha$, where alpha=2 by default.

Note

Currently, we discards genes giving 0 mean or negative dispersion estimate (which can happen if sample variance is smaller than the sample mean).

`phi.line.fitted` *(private) Overlay an estimated mean-dispersion line on an existing plot*

Description

Overlay an estimated mean-dispersion line on an existing plot

Usage

```
phi.line.fitted(obj, alpha = 2, ...)
```

Arguments

`obj`	a list with two components: `mu`, a vector of mean values; `v`, a vector of variance values.
`alpha`	alpha
`...`	other

Details

This function is a wrapper of `phi.line`. It takes a list (rather than two separate vectors) as input.

Note

Users should call phi.plot before calling this function.

See Also

`phi.line`

phi.line.nbp — *(private) Overlay an estimated mean-dispersion line on an existing plot*

Description

Overlay an estimated mean-dispersion line on an existing plot

Usage

```
phi.line.nbp(nbp.obj, grp.id, alpha = 2, ...)
```

Arguments

nbp.obj	output from nbp.test or prepare.nbp
grp.id	a number, indicates the group of counts to be used (grp.id is passed to get.mean.hat)
alpha	alpha
...	other

Details

This function extracts the estimated means and variances from an nbp object and then call phi.line to draw the mean-dispersion curve

Note

Users should call phi.plot before calling this function.

See Also

prepare.nbp, nbp.test, phi.line

phi.plot — *Plot estimated genewise NB2 dispersion parameter versus estimated mean*

Description

Plot estimated NB2 dispersion parameter versus estimated mean

Usage

```
phi.plot(counts, alpha = 2, xlab = "mean", ylab = "phi.hat",
  main = "phi.hat vs mean", log = "xy", ...)
```

Arguments

counts	a matrix of NB counts
alpha	alpha
xlab	x label
ylab	y label
main	main
log	log
...	other

Details

phi.plot estimate the NB2 dispersion parameter for each gene separately by $\phi = (v - \mu)/\mu^a lpha$, where μ and v are sample mean and sample variance. By default, $alpha = 2$.

Note

Currently, we discards genes giving 0 mean or negative dispersion estimate (which can happen if sample variance is smaller than the sample mean).

---

plot.nb.data *Boxplot and scatterplot matrix of relative frequencies (after normalization)*

Description

Boxplot and scatterplot matrix of relative frequencies (after normalization)

Usage

```
## S3 method for class 'nb.data'
plot(x, resolution = 50, hlim = 0.25, clip = 128,
  eps = 0.01, ...)
```

Arguments

x	output from prepare.nb.data
resolution	
hlim	a single number controls the height of the bars in the
clip	
eps	a small positive number added to rpm
...	currently not used histograms

plot.nb.dispersion *Plot the estimated dispersion as a function of the preliminarily estimated mean relative frequencies*

Description

Plot the estimated dispersion as a function of the preliminarily estimated mean relative frequencies

Usage

```
## S3 method for class 'nb.dispersion'
plot(x, ...)
```

Arguments

x	output from estimate.dispersion
...	additional parameters, currently unused

plot.nbp *Diagnostic Plots for an NBP Object*

Description

For output from nbp.test, produce a boxplot, an MA plot, mean-variance plots (one for each group being compared), and mean-dispersion plots (one for each group being compared). On the mean-variance and the mean-dispersion plots, overlay curves corresponding to the estimated NBP model.

Usage

```
## S3 method for class 'nbp'
plot(x, ...)
```

Arguments

x	output from nbp.test.
...	for future use

See Also

nbp.test

Examples

```
## See the example for nbp.test
```

prepare.nb.data *Prepare the NB Data Structure for RNA-Seq Read Counts*

Description

Create a data structure to hold the RNA-Seq read counts and other relevant information.

Usage

```
prepare.nb.data(counts, lib.sizes = colSums(counts),
  norm.factors = estimate.norm.factors(counts), tags = NULL)
```

Arguments

counts	an mxn matrix of RNA-Seq read counts with rows corresponding to gene features and columns corresponding to independent biological samples.
lib.sizes	an n-vector of observed library sizes. By default, library sizes are estimated to the column totals of the matrix counts.
norm.factors	an n-vector of normalization factors. By default, have values 1 (no normalization is applied).
tags	a matrix of tags associated with genes, one row for each gene (having the same number of rows as counts.

Value

A list containing the following components:

counts	the count matrix, same as input.
lib.sizes	observed library sizes, same as input.
norm.factors	normalization factors, same as input.
eff.lib.sizes	effective library sizes (lib.sizes x norm.factors).
rel.frequencies	relative frequencies (counts divided by the effective library sizes).
tags	a matrix of gene tags, same as input.

prepare.nbp *Prepare the Data Structure for Exact NB test for Two-Group Comparison*

Description

Create the NBP data structure, (optionally) normalize the counts, and thin the counts to make the effective library sizes equal.

Usage

```
prepare.nbp(counts, grp.ids, lib.sizes = colSums(counts),
  norm.factors = NULL, thinning = TRUE, print.level = 1)
```

Arguments

counts	an n by r matrix of RNA-Seq read counts with rows corresponding to genes (exons, gene isoforms, etc) and columns corresponding to libraries (independent biological samples).
grp.ids	an r vector of treatment group identifiers (can be a vector of integers, chars or strings).
lib.sizes	library sizes, an r vector of numbers. By default, library sizes are estimated by column sums.
norm.factors	normalization factors, an r vector of numbers. If NULL (default), no normalization will be applied.
thinning	a boolean variable (i.e., logical). If TRUE (default), the counts will be randomly down sampled to make effective library sizes approximately equal.
print.level	a number, controls the amount of messages printed: 0 for suppressing all messages, 1 (default) for basic progress messages, and 2 to 5 for increasingly more detailed messages.

Details

Normalization

We take gene expression to be indicated by relative frequency of RNA-Seq reads mapped to a gene, relative to library sizes (column sums of the count matrix). Since the relative frequencies sum to 1 in each library (one column of the count matrix), the increased relative frequencies of truly over expressed genes in each column must be accompanied by decreased relative frequencies of other genes, even when those others do not truly differently express. Robinson and Oshlack (2010) presented examples where this problem is noticeable.

A simple fix is to compute the relative frequencies relative to effective library sizes—library sizes multiplied by normalization factors. Many authors (Robinson and Oshlack (2010), Anders and Huber (2010)) propose to estimate the normalization factors based on the assumption that most genes are NOT differentially expressed.

By default, prepare.nbp does not estimate the normalization factors, but can incorporate user specified normalization factors through the argument norm.factors.

Library Size Adjustment

The exact test requires that the effective library sizes (column sums of the count matrix multiplied by normalization factors) are approximately equal. By default, prepare.nbp will thin (downsample) the counts to make the effective library sizes equal. Thinning may lose statistical efficiency, but is unlikely to introduce bias.

Value

A list containing the following components:

counts	the count matrix, same as input.
lib.sizes	column sums of the count matrix.
grp.ids	a vector of identifiers of treatment groups, same as input.
eff.lib.sizes	effective library sizes, lib.sizes multiplied by the normalization factors.
pseudo.counts	count matrix after thinning.
pseduo.lib.sizes	effective library sizes of pseudo counts, i.e., column sums of the pseudo count matrix multiplied by the normalization.

Note

Due to thinning (random downsampling of counts), two identical calls to prepare.nbp may yield slightly different results. A random number seed can be used to make the results reproducible.

See Also

nbp.test

Examples

```
## See the example for exact.nb.test
```

---

print.nb.data *Print summary of the nb counts*

Description

Print summary of the nb counts

Usage

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

Arguments

x	output from prepare.nb.data
...	additional parameters, currently not used

---

print.nb.dispersion *Print the estimated dispersion model*

Description

Print the estimated dispersion model

Usage

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

Arguments

x	output from from estimate.dispersion
...	additional parameters, currently unused

---

print.nb.test *Print output from* test.coefficient

Description

We simply print out the structure of x. (Currenlty the method is equivalent to print(str(x)).)

Usage

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

Arguments

x	output from test.coefficient
...	currenty not used

print.nbp *Print summary of an NBP Object*

Description

Print contents of an NBP object, output from prepare.nbp, estimate.disp, or nbp.test.

Usage

```
## S3 method for class 'nbp'
print(x, subset = 1:10, ...)
```

Arguments

x	Output from prepare.nbp, estimate.disp, or nbp.test.
subset	indices of rows of the count matrix to be printed.
...	other parameters (for future use).

See Also

nbp.test.

Examples

```
## See the example for nbp.test
```

smart.plot.new *(private) An alternative to plot.default() for plotting a large number of densely distributed points.*

Description

An alternative to plot.default() for plotting a large number of densely distributed points. This function can produce a visually almost identical plot using only a subset of the points. This is particular useful for reducing output file size when plots are written to eps files.

Usage

```
smart.plot.new(x, y = NULL, xlim = NULL, ylim = NULL, xlab = NULL,
  ylab = NULL, log = "", resolution = 50, col = gray((224:0)/256),
  clip = NULL, col.clipped = rgb(log2(1:256)/log2(256), 0, 0), ...)
```

Arguments

x	x
y	y
xlim	xlim
ylim	ylim
xlab	x label
ylab	y label
log	log
resolution	a number, determines the distance below which points will be considered as overlapping.
plot	logical, whether
col	color
clip	clip
color.clipped	color of clipped points
...	other arguments are the same as in plot.default().

Details

Writing plots with a large number of points to eps files can result in big files and lead to very slow rendering time.

Usually for a large number of points, a lot of them will overlap with each other. Plotting only a subset of selected non-overlapping points can give visually almost identical plots. Further more, the plots can be enhanced if using gray levels (the default setting) that are proportional to the number points overlapping with each plotted point.

This function scans the points sequentially. For each unmarked point that will be plotted, all points that overlap with it will be marked and not to plotted, and the number of overlapping points will be recorded. This is essentially producing a 2d histogram. The freqs of the points will be converted to gray levels, darker colors correspond to higher freqs.

Value

(if plot=FALSE) a list

x, y	the x, y-coordinates of the subset of representative points
id	the indicies of these points in the original data set
freqs	the numbers of points that overlap with each representative point
col	colors determined by the freqs

`smart.plot.old` *(private) An alternative to plot.default() for plotting a large number of densely distributed points.*

Description

An alternative to plot.default() for plotting a large number of densely distributed points. This function can produce a visually almost identical plot using only a subset of the points. This is particular useful for reducing output file size when plots are written to eps files.

Usage

```
smart.plot.old(x, y = NULL, xlim = NULL, ylim = NULL, xlab = NULL,
  ylab = NULL, log = "", resolution = 100, plot = TRUE, col = NULL,
  clip = Inf, color.clipped = TRUE, ...)
```

Arguments

`x`	x
`y`	y
`xlim`	xlim
`ylim`	ylim
`xlab`	x label
`ylab`	y label
`log`	log
`resolution`	a number, determines the distance below which points will be considered as overlapping.
`plot`	logical, whether
`col`	color
`clip`	clip
`color.clipped`	color of clipped points
`...`	other arguments are the same as in plot.default().

Details

Writing plots with a large number of points to eps files can result in big files and lead to very slow rendering time.

Usually for a large number of points, a lot of them will overlap with each other. Plotting only a subset of selected non-overlapping points can give visually almost identical plots. Further more, the plots can be enhanced if using gray levels (the default setting) that are proportional to the number points overlapping with each plotted point.

This function scans the points sequentially. For each unmarked point that will be plotted, all points that overlap with it will be marked and not to plotted, and the number of overlapping points will be recorded. This is essentially producing a 2d histogram. The freqs of the points will be converted to gray levels, darker colors correspond to higher freqs.

Value

(if plot=FALSE) a list

x, y	the x, y-coordinates of the subset of representative points
id	the indicies of these points in the original data set
freqs	the numbers of points that overlap with each representative point
col	colors determined by the freqs

`smart.points` *(private) An alternative to point.default() for plotting a large number of densely distributed points.*

Description

See description of `smart.plot` for more details.

Usage

```
smart.points(x, y = NULL, resolution = 50, col = NULL, clip = Inf,
  color.clipped = TRUE, ...)
```

Arguments

x	x
y	y
resolution	a number, determines the distance below which points will be considered as overlapping.
col	color
clip	clip
color.clipped	color of clipped points
...	other arguments are the same as in plot.default().

`test.coefficient` *Large-sample Test for a Regression Coefficient in an Negative Binomial Regression Model*

Description

`test.coefficient` performs large-sample tests (higher-order asymptotic test, likelihood ratio test, and/or Wald test) for testing regression coefficients in an NB regression model.

Usage

```
test.coefficient(nb, dispersion, x, beta0, tests = c("HOA", "LR", "Wald"),
  alternative = "two.sided", subset = 1:m, print.level = 1)
```

Arguments

`nb`	an NB data object, output from `prepare.nb.data`.
`dispersion`	dispersion estimates, output from `estimate.disp`.
`x`	an n by p design matrix describing the treatment structure
`beta0`	a p-vector specifying the null hypothesis. Non-NA components specify the parameters to test and their null values. (Currently, only one-dimensional test is implemented, so only one non-NA component is allowed).
`tests`	a character string vector specifying the tests to be performed, can be any subset of `"HOA"` (higher-order asymptotic test), `"LR"` (likelihood ratio test), and `"Wald"` (Wald test).
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`subset`	an index vector specifying on which rows should the tests be performed
`print.level`	a number controlling the amount of messages printed: 0 for suppressing all messages, 1 (default) for basic progress messages, and 2 to 5 for increasingly more detailed message.

Details

`test.coefficient` performs large-sample tests for a one-dimensional ($q = 1$) component ψ of the p-dimensional regression coefficient β. The hypothesized value ψ_0 of ψ is specified by the non-NA component of the vector `beta0` in the input.

The likelihood ratio statistic,

$$\lambda = 2(l(\hat{\beta}) - l(\tilde{\beta})),$$

converges in distribution to a chi-square distribution with 1 degree of freedom. The signed square root of the likelihood ratio statistic λ, also called the directed deviance,

$$r = sign(\hat{\psi} - \psi_0)\sqrt{\lambda}$$

converges to a standard normal distribution.

For testing a one-dimensional parameter of interest, Barndorff-Nielsen (1986, 1991) showed that a modified directed

$$r^* = r - \frac{1}{r}\log(z)$$

is, in wide generality, asymptotically standard normally distributed to a higher order of accuracy than the directed deviance r itself, where z is an adjustment term. Tests based on high-order asymptotic adjustment to the likelihood ratio statistic, such as r^* or its approximation, are referred to as higher-order asymptotic (HOA) tests. They generally have better accuracy than corresponding unadjusted likelihood ratio tests, especially in situations where the sample size is small and/or when the number of nuisance parameters $(p-q)$ is large. The implementation here is based on Skovgaard (2001). See Di et al. 2013 for more details.

Value

a list containing the following components:

`beta.hat`	an m by p matrix of regression coefficient under the full model
`mu.hat`	an m by n matrix of fitted mean frequencies under the full model
`beta.tilde`	an m by p matrix of regression coefficient under the null model
`mu.tilde`	an m by n matrix of fitted mean frequencies under the null model.
`HOA`, `LR`, `Wald`	each is a list of two m-vectors, `p.values` and `q.values`, giving p-values and q-values of the corresponding tests when that test is included in `tests`.

References

Barndorff-Nielsen, O. (1986): "Infereni on full or partial parameters based on the standardized signed log likelihood ratio," Biometrika, 73, 307-322

Barndorff-Nielsen, O. (1991): "Modified signed log likelihood ratio," Biometrika, 78, 557-563.

Skovgaard, I. (2001): "Likelihood asymptotics," Scandinavian Journal of Statistics, 28, 3-32.

Di Y, Schafer DW, Emerson SC, Chang JH (2013): "Higher order asymptotics for negative binomial regression inferences from RNA-sequencing data". Stat Appl Genet Mol Biol, 12(1), 49-70.

Examples

```
## Load Arabidopsis data
data(arab);

## Estimate normalization factors (we want to use the entire data set)
norm.factors = estimate.norm.factors(arab);

## Prepare the data
## For demonstration purpose, only the first 50 rows are used
nb.data = prepare.nb.data(arab[1:50,], lib.sizes = colSums(arab), norm.factors = norm.factors);

## For real analysis, we will use the entire data set, and can omit lib.sizes parameter)
## nb.data = prepare.nb.data(arab, norm.factors = norm.factors);

print(nb.data);
plot(nb.data);
```

```
## Specify the model matrix (experimental design)
grp.ids = as.factor(c(1, 1, 1, 2, 2, 2));
x = model.matrix(~grp.ids);

## Estimate dispersion model
dispersion = estimate.dispersion(nb.data, x);

print(dispersion);
plot(dispersion);

## Specify the null hypothesis
## The null hypothesis is beta[2]=0 (beta[2] is the log fold change).
beta0 = c(NA, 0);

## Test regression coefficient
res = test.coefficient(nb.data, dispersion, x, beta0);

## The result contains the data, the dispersion estimates and the test results
print(str(res));

## Show HOA test results for top ten most differentially expressed genes
top = order(res$HOA$p.values)[1:10];
print(cbind(nb.data$counts[top,], res$HOA[top,]));

## Plot log fold change versus the fitted mean of sample 1 (analagous to an MA-plot).
plot(res$mu.tilde[,1], res$beta.hat[,2]/log(2), log="x",
     xlab="Fitted mean of sample 1 under the null",
     ylab="Log (base 2) fold change");

## Highlight top DE genes
points(res$mu.tilde[top,1], res$beta.hat[top,2]/log(2), col="magenta");
```

`thin.counts` *(private) Thin (downsample) counts to make the effective library sizes equal.*

Description

Thin (downsample) counts to make the effective library sizes equal.

Usage

```
thin.counts(y, current.lib.sizes = colSums(y),
  target.lib.sizes = min(current.lib.sizes))
```

Arguments

y	an n by r matrix of counts
current.lib.sizes	an r vector indicating current estimated library sizes
target.lib.sizes	an r vector indicating target library sizes after thinning

Details

The exact NB test for differential gene expression requires that the effective library sizes (column sums of the count matrix multiplied by normalization factors) are approximately equal. This function will thin (downsample) the counts to make the effective library sizes equal. Thinning may lose statistical efficiency, but is unlikely to introduce bias. The reason to use thinning, not scaling, is because Poisson counts after thinning are still Poisson, but Poisson counts after scaling will not be Poisson.

Value

a list

counts	a matrix of thinned counts (same dimension as the input y).
librar.sizes	library sizes after thinning, same as the input target.lib.sizes

---

[.nb.data *hello*

Description

hello

Usage

```
## S3 method for class 'nb.data'
x[i, j, ..., drop = FALSE]
```

Arguments

x
i
j
...
drop

Index