Package 'limpa'

April 14, 2025

Version 0.99.13

Date 2025-04-10

Title Quantification and Differential Analysis of Proteomics Data

Description Quantification and differential analysis of mass-spectrometry proteomics data, with probabilistic recovery of information from missing values. Estimates the detection probability curve (DPC), which relates the probability of successful detection to the underlying expression level of each peptide, and uses it to incorporate peptide missing values into protein quantification and into subsequent differential expression analyses. The package produces objects suitable for downstream analysis in limma. The package accepts peptide-level data with missing values and produces complete protein quantifications without missing values. The uncertainty introduced by missing value imputation is propagated through to the limma analyses using variance modeling and precision weights. The package name ``limpa" is an acronym for ``Linear Models for Proteomics Data".

License GPL (>=2)

Depends limma

Imports methods, stats, data.table, statmod

Suggests arrow, knitr, BiocStyle

VignetteBuilder knitr

biocViews Bayesian, BiologicalQuestion, DataImport, DifferentialExpression, GeneExpression, MassSpectrometry, Preprocessing, Proteomics, Regression, Software

git_url https://git.bioconductor.org/packages/limpa

git_branch devel

git_last_commit 61ca22d

git_last_commit_date 2025-04-09

Repository Bioconductor 3.21

Date/Publication 2025-04-14

Author Mengbo Li [aut] (ORCID: <https://orcid.org/0000-0002-9666-5810>), Gordon Smyth [cre, aut] (ORCID: <https://orcid.org/0000-0001-9221-2892>)

Maintainer Gordon Smyth <smyth@wehi.edu.au>

Contents

`limpa-package` *Linear Models for Proteomics Data (Accounting for Missing Values)*

Description

This package implements a pipeline for quantification and differential expression analysis of mass-spectrometry-based proteomics data.

Mass spectrometry (MS)-based proteomics is a powerful tool in biomedical research. Recent advancements in label-free methods and MS instruments have enabled the quantitative characterisation of large-scale complex biological samples with the increasingly deeper coverage of the proteome. However, missing values are still ubiquitous in MS-based proteomics data. We observe from a wide range of real datasets that missingness in label-free data is intensity-dependent, so that the missing values are missing not at random or, in other words, are non-ignorable.

This package implements statistical and computational methods for analysing MS-based label-free proteomics data with non-ignorable missing values. The package use the observed proteomics data to estimate the detection probability curve (DPC), which provides a formal probabilistic model for the intensity-dependent missingness. Based on exponential tilting, the DPC estimates the detection

probabilities given the underlying intensity of each observation, observed or unobserved. Importantly, the DPC evaluates how much statistical information can or cannot be recovered from the missing value pattern, and can be used to inform downstream analyses such as differential expression (DE) analysis.

Next, the package implements a novel protein quantification method, called DPC-quant, where missing values are represented by the DPC. An empirical Bayes scheme is employed to borrow information across the tens of thousands of peptides measured in a typical experiment. A multivariate normal prior is estimated empirically from data to describe the variability in log-intensities across the samples and across the peptides.

Finally, quantification uncertainty is incorporated into the differential expression analysis using precision weights. Leveraging the limma package, a new variance modelling approach with multiple predictors is used, which allows the DPC-quant precisions to be propagated to the differential expression analysis while simultaneously assuming a mean-variance relationship. The new differential expression pipeline has been implemented in the limma R package in the `vooma()` function.

The limpa package is fully compatible with limma pipelines, allowing any arbitrarily complex experimental design and other downstream tasks such as the gene ontology or pathway analysis.

Author(s)

Mengbo Li and Gordon K Smyth

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Li M, Smyth GK (2023). Neither random nor censored: estimating intensity-dependent probabilities for missing values in label-free proteomics. *Bioinformatics* 39(5), btad200. 10.1093/bioinformatics/btad200)

`completeMomentsON` *Complete Distribution Moments from Observed Normal Model*

Description

Mean and standard-deviation of the complete data distribution under the observed normal model.

Usage

```
completeMomentsON(mean.obs=6, sd.obs=1, dpc=c(-4,0.7))
```

Arguments

`mean.obs`	mean of observed normal distribution.
`sd.obs`	standard deviation of observed normal distribution.
`dpc`	numeric vector of length 2 giving the DPC intercept and slope.

Details

Under the observed normal model, calculate the mean and standard deviation of the complete data distribution that would have occurred if the missing value mechanism hadn't operated.

Value

A list with compoenents

`mean.comp`	mean of complete data distribution.
`sd.comp`	standard deviation of complete data distribution.
`prob.obs`	unconditional probability that values are observed.

Examples

```
completeMomentsON(mean.obs=6, sd.obs=2)
```

---

dpc *Detection Probability Curve Assuming Observed Normal Model*

Description

Detection probability curve for label free shotgun proteomics data assuming observed normal intensities.

Usage

```
dpc(y, maxit = 100, eps = 1e-04, b1.upper = 1)
```

Arguments

`y`	numeric matrix of log2-transformed intensities. Rows correspond to peptide precursors and columns to samples. Any object such as an EList that can be coerced to a matrix is also acceptable.
`maxit`	maximum number of iterations.
`eps`	convergence tolerance.
`b1.upper`	upper bound for beta1.

Details

Estimate the detection probability curve (DPC) for label-free shotgun proteomics data using the method described by Li & Smyth (2023). This function assumes that the observed log-intensities are normally distributed (the "observed normal" model), and uses exponential tilting to reformulate the DPC in terms of observed statistics instead of in terms of unobserved quantities.

Value

A list with components

dpc	estimated DPC coefficients.
history	iteration history.
dpc.start	initial values estimated for the DPC coefficients.
prop.detected	proportion of observed values for each row.
mu.prior	prior value for row-wise means for observed values.
n.prior	precision of prior for row-wise means, expressed as effective number of observations.
s2.prior	prior value for row-wise variances for observed values.
df.prior	precision of prior for row-wise variances, expressed as effective degrees of freedom.
mu.obs	posterior row-wise means for observed values.
s2.obs	posterior row-wise variances for observed values.
mu.mis	posterior row-wise means for values that are missing.

References

Li M, Smyth GK (2023). Neither random nor censored: estimating intensity-dependent probabilities for missing values in label-free proteomics. *Bioinformatics* 39(5), btad200. 10.1093/bioinformatics/btad200

See Also

dpcCN

Examples

```
y <- simProteinDataSet(n.peptides=100, n.groups=1)
out <- dpc(y)
out$dpc
```

---

dpcCN *Detection Probability Curve Assuming Complete Normal Model*

Description

Detection probability curve for label-free shotgun proteomics data assuming a complete normal model for the peptide intensities.

Usage

```
dpcCN(y, dpc.start= c(-4,0.7), iterations = 3, verbose = TRUE)
```

Arguments

y	numeric matrix of log2-intensities. Rows correspond to peptide precursors and columns to samples.
dpc.start	numeric vector of length 2 giving starting estimates for the DPC intercept and slope.
iterations	number of outer iterations.
verbose	if TRUE, then progress information will be printed from each iteration.

Details

Estimate the detection probability curve (DPC) for label-free shotgun proteomics data by maximum posterior assuming that the complete log-intensities are normally distributed (the "complete normal" model). The complete log-intensities are the values that would have been observed if the missing value mechanism had not operated.

The algorithm uses an alternating iteration (Smyth, 1996), alternately estimating the row-wise means and standard deviations (mu and sigma) for fixed DPC and estimating the DPC for fixed mu and sigma. The inner estimations use the BFGS algorithm implemented in the optim function. Three outer iterations are usually sufficient.

dpc estimates the DPC by a different method, described in Li & Smyth (2023), based on exponential tilting and assuming that only the observed values are normally distributed (the "observed normal" model).

Value

A list with components

dpc	numeric vector of length 2 giving estimated DPC coefficients.
mu	numeric vector of length nrow(y) giving estimated complete data row-wise means.
sigma	numeric vector of length nrow(y) giving estimated complete data row-wise standard deviations.

Note

This function may underestimate the DPC slope if entirely missing peptides are omitted and the proportion of peptides that are entirely missing by chance is not small.

dpcCN can take several minutes on large datasets so, by default, progress information is turned on with verbose=TRUE. The function will run quietly if verbose=FALSE is set.

References

Li M, Smyth GK (2023). Neither random nor censored: estimating intensity-dependent probabilities for missing values in label-free proteomics. *Bioinformatics* 39(5), btad200. 10.1093/bioinformatics/btad200

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Smyth GK (1996). Partitioned algorithms for maximum likelihood and other non-linear estimation. *Statistics and Computing* 6, 201-216. doi:10.1007/BF00140865 https://gksmyth.github.io/pubs/partitio.pdf

See Also

dpc.

Examples

```
y <- simProteinDataSet(n.peptides=100, n.groups=1)
out <- dpcCN(y)
out$dpc
```

---

dpcDE *Fit Linear Model With Precision Weights*

Description

Fit linear models and make precision weights from the DPC-Quant standard errors.

Usage

```
dpcDE(y, design, plot=TRUE, ...)
```

Arguments

y	protein-level EList produced by dpcQuant().
design	design matrix.
plot	should the variance trend be plotted?
...	other arguments are passed to voomaLmFit.

Details

Calls voomaLmFit to compute vooma precision weights from the DPC-Quant standard errors stored in y and to use those weights to fit protein-wise linear models. Any voomaLmFit functinality can be used, giving access to optional empirical sample weights or random blocks.

Value

An MArrayLM object suitable for analysis in limma.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

See Also

voomaLmFit

Examples

```
y.peptide <- simProteinDataSet()
y.protein <- dpcQuant(y.peptide, "Protein", dpc=c(-4,0.7))
Group <- factor(y.peptide$targets$Group)
design <- model.matrix(~Group)
fit <- dpcDE(y.protein, design)
```

---

dpcQuant *Quantify Proteins*

Description

Use the DPC to quantify protein expression values.

Usage

```
## S3 method for class 'EList'
dpcQuant(y, protein.id = "Protein.Group", dpc = NULL, dpc.slope = 0.8,
     verbose = TRUE, chunk = 1000, ...)
## S3 method for class 'EList'
dpcImpute(y, dpc = NULL, dpc.slope = 0.8, verbose = TRUE, chunk = 1000, ...)
```

Arguments

y	a numeric matrix or EList of peptide-level log2-expression values. Columns are samples and rows are peptides or precursors.
protein.id	protein IDs. Either an annotation column name (if y is an EList) or a character vector of length nrow(y).
dpc	numeric vector giving intercept and slope of DPC. Alternatively the output objects from dpc or dpcCN are also acceptable.
dpc.slope	slope coefficient of DPC. Only used if dpc is NULL.
verbose	should progress information be output? If TRUE, then progress information is output every 1000 proteins.
chunk	When verbose=TRUE, how often to output progress information. By default, reports every 1000 proteins.
...	other arguments are passed to dpcQuantHyparam.

Details

Implements the DPC-Quant method, which quantifies protein log2-expression values from peptide data. The method represents missing values probabilistically using the PDC and returns maximum posterior estimates for all the protein log2-expression values, so that there are no missing values in the final summary.

The `dpc` function is usually used to estimate the detection probability curve (DPC) before running `dpcQuant`, however a preset DPC slope can also be used. If the `dpc` argument is `NULL`, then `dpc.slope` will be used as the DPC together with a DPC intercept estimated by `estimateDPCIntercept`.

The output from `dpcQuant` can be input to `dpcDE`.

`dpcImpute` performs imputation without summarization by treating each row as a separate protein.

Value

`dpcQuant()` produces an `EList` object with a row for each protein, with the following extra components:

`other$n.observations`
: matrix giving the number of missing non-missing peptide observations supporting each protein expression value.

`other$standard.error`
: matrix giving the standard error of each protein expression value.

`dpcImpute()` produces an `EList` object with the same number of rows as `y`.

Note

`dpcQuant` can take several minutes on large datasets so, by default, progress information is turned on with `verbose=TRUE`. The function will run quietly if `verbose=FALSE` is set.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

See Also

`dpc`, `dpcQuantHyperparam`, `dpcDE`, `EList-class`.

Examples

```
y.peptide <- simProteinDataSet(n.groups=1,samples.per.group=4,prop.missing=0.2)
y.protein <- dpcQuant(y.peptide, "Protein", dpc.slope=0.7)
```

dpcQuantHyperparam *Estimate Hyperparameters for DPC-Quant*

Description

Estimate hyperparameters for the DPC-based protein quantification method (DPC-Quant).

Usage

```
dpcQuantHyperparam(y, protein.id, dpc.slope = 0.7,
     sd.quantile.for.logFC = 0.9, robust = FALSE, ...)
dpcImputeHyperparam(y, dpc.slope = 0.7,
     sd.quantile.for.logFC = 0.9, robust = FALSE, ...)
```

Arguments

y	a numeric matrix of peptide-level log2-expression values. Columns are samples and rows are peptides or precursors.
protein.id	a character vector of length nrow(y) giving protein IDs.
dpc.slope	slope of the DPC.
sd.quantile.for.logFC	a number between 0 and 1. The quantile of the precursor-level variances to represent the typical between-sample variation.
robust	should robust empirical Bayes moderation be applied to the protein standard deviations? robust=TRUE will cause very large standard deviations to be squeezed less strongly towards the prior value.
...	other arguments are passed to imputeByExpTilt.

Details

Estimates and returns the empirical Bayes hyperparameters required for DPC-Quant protein quantification. dpcQuantHyperparam is called by dpcQuant function, and dpcImputeHyperparam is called by dpcImpute.

Value

A list with components

prior.mean	mean of the global prior distribution for protein log-expression values. Represents the typical average log-expression of a protein.
prior.sd	standard deviation of the global prior distribution for protein log-expression values. Represents the standard deviation of average log-expression across proteins.
prior.logFC	standard deviation to be expected between log-expression values for the same protein across conditions.

sigma	protein standard deviations from additive model fitted to peptide log expression values. Numeric vector of same length as unique(protein.id)).

The last component is omitted in the dpcImputeHyperparam output.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

See Also

dpcQuant, imputeByExpTilt

---

estimateDPCIntercept *Estimate DPC Intercept*

Description

Estimate the DPC intercept given a value for the slope.

Usage

```
estimateDPCIntercept(y, dpc.slope = 0.8, trace = FALSE)
```

Arguments

y	numeric matrix of log2-intensities, or any data object than can be coerced to a matrix. Includes NAs. Rows correspond to peptide precursors and columns to samples.
dpc.slope	DPC slope.
trace	if TRUE, then progress information will be printed from each glm iteration.

Details

Estimates the intercept coefficient of the detection probability curve (DPC) by using imputeByExpTilt to impute complete data, then fitting a binomial glm model with the slope as an offset vector. If the dataset is large, then similar y values are aggregated before fitting the glm.

Value

A single numeric value giving the intercept.

See Also

imputeByExpTilt.

Examples

```
y <- simProteinDataSet(n.peptides=100, n.groups=1, dpc.slope=0.7)
estimateDPCIntercept(y, dpc.slope=0.7)
```

`filterCompoundProteins`

Filtering Based On Protein Annotation

Description

Filter peptides or proteins from the dataset based on uniqueness of annotation.

Usage

```
## Default S3 method:
filterCompoundProteins(y, protein.group, ...)
## S3 method for class 'EList'
filterCompoundProteins(y, protein.group="Protein.Group", ...)
## Default S3 method:
filterSingletonPeptides(y, protein.group, min.n.peptides = 2, ...)
## S3 method for class 'EList'
filterSingletonPeptides(y, protein.group="Protein.Group", min.n.peptides = 2, ...)
## Default S3 method:
filterNonProteotypicPeptides(y, proteotypic, ...)
## S3 method for class 'EList'
filterNonProteotypicPeptides(y, proteotypic="Proteotypic", ...)
```

Arguments

`y`	a matrix, `EList` object or `EListRaw` object containing log2-expression values.
`protein.group`	protein group for each row of `y`. Can be either a character vector of length `nrow(y)` or the name of an annotation column.
`proteotypic`	indicates whether each peptide is proteotypic (detectable and unique to one protein). Should contain 0/1 or TRUE/FALSE values. Can be either a vector of length `nrow(y)` or the name of an annotation column.
`min.n.peptides`	minimum number of peptides required in a protein.
`...`	other arguments are not currently used.

Details

Filter peptide or proteins from the dataset based on uniqueness of annotation. `filterCompoundProteins` removes compound protein groups consisting of multiple proteins separated by ";" delimiters. `filterSingletonPeptides` removes proteins with only one peptide. `filterNonProteotypicPeptides` removes peptides that belong to more than one protein, using the "Proteotypic" annotation column that is returned by DIA-NN and other proteomics quantification software.

Value

An object the same as y but with non-compliant rows removed.

See Also

readDIANN

`fitZTLogit` *Fit Capped Logistic Regression To Zero-Truncated Binomial Data*

Description

Estimate a logistic regression, with optionally capped probabilities, by maximum likelihood with zero-truncated data.

Usage

```
fitZTLogit(n.successes, n.trials, X = NULL, capped = FALSE,
     beta.start = NULL, alpha.start = 0.95)
```

Arguments

`n.successes`	number of binomial successes (numeric vector). Should be bounded below by 1 and bounded above by `n.trials`.
`n.trials`	number of binomial trials (numeric vector).
`X`	the regression design matrix. Number of rows should match `length(n.successes)`.
`capped`	if `TRUE`, then probability of a success will be capped at alpha < 1, where alpha is to be estimated.
`beta.start`	starting values for the regression coefficients. Of same length as `ncol(X)`.
`alpha.start`	starting value for alpha.

Details

Estimates a logistic regression equation for zero-truncated binomial observations. Optionally estimates a limiting value for the probabilities that may be less than one.

The function maximizes the zero-truncated binomial likelihood using the `optim` function with `method="BFGS"`. The fitted probabilities are equal to `alpha * plogis(X %*% beta)`.

Value

A list with components

`beta`	linear predictor coefficients.
`alpha`	capping parameter, maximum or asymptotic value for the probabilities.
`p`	fitted probabilities.
`deviance`	minus twice the maximized log-likelihood.
`calls`	number of function calls used in the optimization.

References

Li M, Smyth GK (2023). Neither random nor censored: estimating intensity-dependent probabilities for missing values in label-free proteomics. *Bioinformatics* 39(5), btad200. 10.1093/bioinformatics/btad200

Examples

```
# Generate binomial data
n <- 30
n.trials <- rep(4,n)
x <- seq(from=3, to=9, length.out=n)
X <- model.matrix(~x)
beta <- c(-4,0.7)
p <- plogis(X %*% beta)
n.successes <- rbinom(n, size=n.trials, prob=p)

# Zero truncation
is.pos <- (n.successes > 0)
n.successes <- n.successes[is.pos]
n.trials <- n.trials[is.pos]
x <- x[is.pos]
X <- X[is.pos,]

# Zero-truncated regression
fit <- fitZTLogit(n.successes, n.trials, X)
p.observed <- n.successes / n.trials
plot(x, p.observed)
lines(x, fit$p)
```

---

imputeByExpTilt *Impute Missing Values by Exponential Tilting*

Description

Impute missing values in a log-expression matrix by applying exponential tilting to rows, columns or both.

Usage

```
## Default S3 method:
imputeByExpTilt(y, dpc.slope = 0.7, prior.logfc = NULL, by = "both", ...)
expTiltByRows(y, dpc.slope = 0.7, sigma.obs = NULL)
expTiltByColumns(y, dpc.slope = 0.7)
```

Arguments

y	an EList object or a numeric matrix of log-expression values. Columns are samples and rows are peptides or proteins. For expTiltByRows or expTiltByRows, should be a numeric matrix.

dpc.slope	slope of detection probability curve.
prior.logfc, sigma.obs	simple standard deviation to be expected between observed values for the same peptide or protein. Can a single value or vector of length nrow(y). By default is estimated from the data.
by	character value. Should imputation by rows ("rows"), by columns ("columns") or both ("both")?
...	other arguments are not used.

Details

Implements exponential tilting strategy outlined by Li & Smyth (2023). The imputed values are the expected values of the missing value distribution.

The strategy can be applied to rows or columns. If by="both", the imputated values are an average of the row and column imputations, weighted inversely by the prediction variances.

Value

An object of the same class as y but with NAs imputed.

References

Li M, Smyth GK (2023). Neither random nor censored: estimating intensity-dependent probabilities for missing values in label-free proteomics. *Bioinformatics* 39(5), btad200. doi:10.1093/bioinformatics/btad200

Examples

```
y <- matrix(rnorm(25),5,5)
y[1,1] <- NA
imputeByExpTilt(y)
```

---

observedMomentsCN *Observed Distribution Moments from Complete Normal Model*

Description

Mean and standard-deviation of the observed data distribution under the complete normal model.

Usage

```
observedMomentsCN(mean.comp=6, sd.comp=1, dpc=c(-4,0.7))
```

Arguments

mean.comp	mean of complete normal distribution.
sd.comp	standard deviation of complete normal distribution.
dpc	numeric vector of length 2 giving the DPC intercept and slope.

Details

Under the complete normal model, calculate the mean and standard deviation of the observed data distribution.

Value

A list with compoenents

mean.obs	mean of observed data distribution.
sd.obs	standard deviation of observed data distribution.
prob.obs	unconditional probability that values are observed.

Examples

```
observedMomentsCN(mean.comp=6, sd.comp=2)
```

---

peptides2ProteinBFGS *DPC-Quant for One Protein*

Description

Convert a matrix of peptide log-expression values for one protein to protein-level expression values by the DPC-Quant method.

Usage

```
peptides2ProteinBFGS(y, sigma = 0.5, weights = NULL, dpc = c(-4, 0.7),
        prior.mean = 6, prior.sd = 10, prior.logFC = 2,
        standard.errors = TRUE, newton.polish = TRUE, start = NULL)
peptides2ProteinNewton(y, sigma = 0.5, weights = NULL, dpc = c(-4, 0.7),
        prior.mean = 6, prior.sd = 10, prior.logFC = 2,
       standard.errors = TRUE, tol=1e-6, maxit=10, start = NULL, verbose = FALSE)
peptides2ProteinWithoutNAs(y, sigma = 0.5, weights = NULL, dpc = c(-4, 0.7),
        prior.mean = 6, prior.sd = 10, prior.logFC = 2)
```

Arguments

y	a numeric matrix of log-expression values. Columns are samples and rows are peptides or precursors. Typically contains NAs, but NAs are not allowed for peptides2ProteinWithoutNAs.
sigma	standard deviation of peptide-level expression values after allowing for peptide and sample baseline differences.
weights	numeric matrix of same size as y containing positive precision weights. The precision of the log-expression values is summarized by sigma/sqrt(weights).
dpc	numeric vector giving intercept and slope of the detection probability curve (DPC).

prior.mean	mean of the global prior distribution for protein log-expression values. Represents the typical average log-expression of a protein.
prior.sd	standard deviation of the global prior distribution for protein log-expression values. Represents the standard deviation of average log-expression across proteins.
prior.logFC	standard deviation to be expected between log-expression values for the same protein.
standard.errors	logical, should standard errors for the protein expression values be returned?
newton.polish	logical. If TRUE then one Newton iteration will be done to refine the optimization after the BFGS algorithm has finished. Ignored if standard.errors=FALSE.
start	numeric vector of starting values for the linear model coefficients. Of length ncol(y)+nrow(y)-1).
tol	stopping criterion tolerance for Newton's method, to be achieved by the average local slope statistic.
maxit	maximum number of iterations for Newton's method.
verbose	logical. If TRUE, progress will be output at each iteration.

Details

Implements the DPC-Quant method, which returns maximum posterior estimates for protein expression values.

peptides2ProteinBFGS maximizes the posterior using the BFGS algorithm with analytic first derivatives. The standard errors are computed from analytic second derivatives.

peptides2ProteinNewton maximizes the posterior using Newton's method.

Value

peptides2ProteinBFGS and peptides2ProteinNewton return a list with components.

protein.expression	numeric vector giving the estimated protein log-expression value for each sample.
standard.error	numeric vector giving standard errors for the protein log-expression values.
value	the minimized objective function, minus twice the log-posterior distribution.

peptides2ProteinWithoutNAs returns a numeric vector of protein expression values.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Smyth GK (2005). Optimization and nonlinear equations. In: *Encyclopedia of Biostatistics Second Edition*, Volume 6, P. Armitage and T. Colton (eds.), Wiley, London, pages 3857-3863. https://gksmyth.github.io/pubs/OptimNonlinEqnPreprint.pdf

Examples

```
y <- matrix(rnorm(12),3,4)
y[1:2,1] <- NA
y[1,2] <- NA
peptides2ProteinBFGS(y)
```

`peptides2Proteins` *DPC-Quant for Many Proteins*

Description

Quantify protein expression values by the DPC-Quant method.

Usage

```
peptides2Proteins(y, protein.id, sigma = 0.5, dpc = c(-4, 0.7),
        prior.mean = 6, prior.sd = 10, prior.logFC = 2,
     standard.errors = FALSE, newton.polish = FALSE, verbose = FALSE, chunk = 1000L)
```

Arguments

`y`	a numeric matrix of log-expression values. Columns are samples and rows are peptides or precursors.
`protein.id`	protein IDs. Character vector of length `nrow(y)`.
`sigma`	standard deviations of peptide-level expression values. Numeric vector of same length as `unique(protein.id)`.
`dpc`	numeric vector giving intercept and slope of detection probability curve (DPC).
`prior.mean`	mean of the global prior distribution for protein log-expression values. Represents the typical average log-expression of a protein.
`prior.sd`	standard deviation of the global prior distribution for protein log-expression values. Represents the standard deviation of average log-expression across proteins.
`prior.logFC`	standard deviation to be expected between log-expression values for the same protein.
`standard.errors`	logical, should standard errors for the protein expression values be returned?
`newton.polish`	logical. If `TRUE` then one Newton iteration will be done to refine the optimization after the BFGS algorithm has finished. Ignored if `standard.errors=FALSE`.
`verbose`	should progress information be output? If `TRUE`, then progress information is output every `chunk` proteins.
`chunk`	When `verbose=TRUE`, how often to output progress information. By default, reports every 1000 proteins.

Details

Implements the DPC-Quant method, which returns maximum posterior estimates for protein expression values.

Value

An `EList` object with a row for each protein.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Examples

```
y.peptide <- simProteinDataSet(8,n.groups=1,samples.per.group=4,prop.missing=0.2)
y.protein <- peptides2Proteins(y.peptide$E, y.peptide$genes$Protein)
```

---

plotDPC *Plot the Detection Probability Curve*

Description

Plot the detection probability curve using output from the dpc function.

Usage

```
plotDPC(dpcfit, add.jitter = TRUE,
        point.cex = 0.2, lwd = 2, ylim = c(0, 1),
        main = "Detection probability curve", ...)
```

Arguments

`dpcfit`	object produced by `dpc()`.
`add.jitter`	logical, whether to add jitter to the detected proportion axis.
`point.cex`	relative size of points.
`lwd`	relative line width.
`ylim`	limits of the y-axis.
`main`	main title of plot.
`...`	other arguments are passed to `plot`.

Value

A plot is produced on the current device. A list with components `x` and `y` is also invisibly returned.

Examples

```
y <- simProteinDataSet(n.peptides=100, n.groups=1)
dpcfit <- dpc(y)
plotDPC(dpcfit)
```

`plotMDSUsingSEs` *Multidimensional Scaling Plot of Gene Expression Profiles, Using Standard Errors*

Description

Plot samples on a two-dimensional scatterplot so that distances on the plot approximate the typical z-statistic of differences between the samples.

Usage

```
plotMDSUsingSEs(y, top = 500, labels = NULL, pch = NULL, cex = 1,
     dim.plot = c(1,2), gene.selection = "pairwise",
     xlab = NULL, ylab = NULL, plot = TRUE, var.explained = TRUE, ...)
```

Arguments

`y`	`EList` produced by `dpcQuant` or `dpcImpute`.
`top`	number of top genes used to calculate pairwise distances.
`labels`	character vector of sample names or labels. Defaults to `colnames(x)`.
`pch`	plotting symbol or symbols. See `points` for possible values. Ignored if `labels` is non-`NULL`.
`cex`	numeric vector of plot symbol expansions.
`dim.plot`	integer vector of length two specifying which principal components should be plotted.
`gene.selection`	character, `"pairwise"` to choose the top genes separately for each pairwise comparison between the samples or `"common"` to select the same genes for all comparisons.
`xlab`	title for the x-axis.
`ylab`	title for the y-axis.
`plot`	logical. If `TRUE` then a plot is created on the current graphics device.
`var.explained`	logical. If `TRUE` then the percentage variation explained is included in the axis labels.
`...`	any other arguments are passed to `plot`, and also to `text` (if `pch` is `NULL`).

Details

This function uses multidimensional scaling (MDS) to produce a principal coordinate (PCoA) plot showing the relationships between the expression profiles represented by the columns of `x`. Distances on the plot represent the *leading z-statistic*. The leading log-fold-change between a pair of samples is defined as the root-mean-square average of the `top` largest z-statistics between those two samples.

If `pch=NULL`, then each sample is represented by a text label, defaulting to the column names of `x`. If `pch` is not `NULL`, then plotting symbols are used.

See `text` for possible values for `col` and `cex`.

Value

If `plot=TRUE` or if `x` is an object of class `"MDS"`, then a plot is created on the current graphics device.

An object of class `"MDS"` is also invisibly returned. This is a list containing the following components:

`eigen.values`	eigen values
`eigen.vectors`	eigen vectors
`var.explained`	proportion of variance explained by each dimension
`distance.matrix.squared`	numeric matrix of squared pairwise distances between columns of `x`
`dim.plot`	dimensions plotted
`x`	x-xordinates of plotted points
`y`	y-cordinates of plotted points
`gene.selection`	gene selection method

Author(s)

Gordon Smyth

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, and Smyth GK (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* 43, e47. http://nar.oxfordjournals.org/content/43/7/e47

See Also

`plotMDS` in the limma package.

Examples

```
# See dpcQuant()
```

plotProtein *Plot protein summary with error bars by DPC-Quant*

Description

Plot the log-intensity of a protein summarized by DPC-Quant for each sample with error bars.

Usage

```
plotProtein(y, protein, col = "black", cex = 2, lwd = 2, ...)
```

Arguments

y	protein-level EList produced by dpcQuant().
protein	A vector of length 1. Can be the name of the protein or the numeric index that locates the protein to plot from rows of y.
col	Color for the points and error bars.
cex	Size for the points.
lwd	Line width for the error bars.
...	other arguments are passed to plot().

Details

Plot the sample-wise protein quantification results from dpcQuant() for a specified protein. The error bars (standard errors) indicate the quantification uncertainty associated with each estimate. Typically within a dataset, the larger the error bar is, the more missing values there are in the precursor/peptide-level data for that protein.

Value

A plot is created on the current graphics device. A list with components y and se is also invisibly returned.

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Examples

```
y.peptide <- simProteinDataSet()
y.protein <- dpcQuant(y.peptide, "Protein", dpc=c(-4,0.7))
plotProtein(y.protein, protein = "Protein01", col = rep(c("blue", "red"), each = 5))
y.protein$other$standard.error["Protein01",]
```

proteinResVarFromCompletePeptideData
Protein Residual Variances From Complete Peptide Data

Description

Get protein-wise residual variances by fitting a two-way additive model to the complete (imputed) peptide data for each protein.

Usage

```
proteinResVarFromCompletePeptideData(y, protein.id, reorder=FALSE)
```

Arguments

y	a numeric matrix of complete peptide log2-expression values without NAs. Columns are samples and rows are peptides or precursors.
protein.id	a character vector of length nrow(y) giving protein IDs.
reorder	does the data need to sorted into protein order? If TRUE, then the rows of y will be sorted so that peptides for the same protein are in consecutive rows. If FALSE, the rows are assumed to be already sorted.

Details

This function operates on complete data after imputation of missing values, and is used to get the sigma hyperparameters required by peptides2Proteins and dpcQuant. The function fits an additive linear model (~ sample + peptide) to the peptide data for each protein and returns the residual variances.

Value

A list with components

prior.mean	mean of the global prior distribution for protein log-expression values. Represents the typical average log-expression of a protein.
prior.sd	standard deviation of the global prior distribution for protein log-expression values. Represents the standard deviation of average log-expression across proteins.
prior.logFC	standard deviation to be expected between log-expression values for the same protein across conditions.
sigma	protein standard deviations from additive model fitted to peptide log expression values. Numeric vector of same length as unique(protein.id)).

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

See Also

dpcQuant, peptides2Proteins

Examples

```
y <- simProteinDataSet(8, n.groups=1, samples.per.group=4, prop.missing=0)
proteinResVarFromCompletePeptideData(y$E, y$genes$Protein)
```

---

readDIANN *Read Peptide-Precursor Intensities From DIA-NN Output*

Description

Read DIA-NN Reports.tsv file into EList object.

Usage

```
readDIANN(file = "Report.tsv", path = NULL, format = "tsv", sep = "\t", log = TRUE,
      q.columns = c("Global.Q.Value", "Lib.Q.Value"), q.cutoffs = c(0.01, 0.01))
```

Arguments

file	the name of the file from which the data are to be read. Or it can also be the data.frame read from the report in the long format, where each row is an observation.
path	character string giving the directory containing the file. Defaults to the current working directory.
format	character string giving the format of the file. Possible values are "tsv" or "parquet". Default is "tsv".
sep	the field separator character
log	logical. If TRUE then intensities will be returned on the log2 scale, otherwise unlogged with zeros.
q.columns	column headings in the DIA-NN output containing Q-values for peptide identification. Character vector.
q.cutoffs	cutoffs to apply to the Q-value columns. Only peptides with values below the cutoffs will be retained. Numeric vector of same length as q.columns.

Details

DIA-NN (Demichev et al 2020) writes a file in long (data.frame) format, typically called Report.tsv, containing normalized intensities for peptide precursors. readDIANN reads this file and produces an object in limma EList or EListRaw format. From version 2.0, DIA-NN returns the main report in Apache Parquet format (https://github.com/vdemichev/DiaNN/releases). readDIANN can read the Parquet file directly or, alternatively, one can read the Parquet file into a data.frame, and use readDIANN to process the long-format data.frame into a limma EList or EListRaw object.

Value

If log=FALSE, an EListRaw object containing precursor-level unlogged intensities with zeros and protein annotation. If log=TRUE, an EList object containing precursor-level log2 intensities with NAs and protein annotation. Rows are peptide-precursors and columns are samples. Peptide precursor and protein annotation is stored in the 'genes' output component.

References

Demichev V, Messner CB, Vernardis SI, Lilley KS, Ralser M (2020). DIA-NN: neural networks and interference correction enable deep proteome coverage in high throughput. *Nature Methods* 17(1), 41-44.

Examples

```
## Not run:
ypep <- readDIAN()
ypep <- filterCompoundProteins(ypep)
ypep <- filterNonProteotypicPeptides(ypep)
dpcfit <- dpc(ypep)
yprot <- dpcQuant(ypep, dpc=dpcfit)

## End(Not run)
```

---

readSpectronaut *Read Peptide-Precursor Intensities From Spectronaut Output*

Description

Read Spectronaut Reports.tsv file into EList object.

Usage

```
readSpectronaut(
  file = "Report.tsv", path = NULL, sep = "\t", log = TRUE,
  run.column = "R.Raw File Name",
  precursor.column = "EG.PrecursorId",
  qty.column = "EG.TotalQuantity (Settings)",
  q.columns = c("EG.Qvalue", "PG.Qvalue"), q.cutoffs = 0.01,
  extra.columns = c("PG.ProteinAccessions", "EG.IsImputed")
)
```

Arguments

file	the name of the file from which the data are to be read.
path	character string giving the directory containing the file. Defaults to the current working directory.
sep	the field separator character

log	logical. If TRUE then intensities will be returned on the log2 scale, otherwise unlogged with zeros.
run.column	column containing run. String of length 1L.
precursor.column	column containing precursor IDs. String of length 1L.
qty.column	column containing intensities. String of length 1L.
q.columns	column headings in the Spectronaut output containing Q-values for peptide identification. Character vector.
q.cutoffs	cutoffs to apply to the Q-value columns. Only peptides with values below the cutoffs will be retained. Numeric vector of same length as q.columns.
extra.columns	extra columns that are appended to the precursor annotation matrix.

Details

Spectronaut (https://biognosys.com/software/spectronaut/) writes a file in long (data.frame) format, typically called Report.tsv, containing normalized intensities for peptide precursors. readSpectronaut reads this file and produces an object in limma EList or EListRaw format.

Value

If log=FALSE, an EListRaw object containing precursor-level unlogged intensities with zeros and protein annotation. If log=TRUE, an EList object containing precursor-level log2 intensities with NAs and protein annotation. Rows are peptide-precursors and columns are samples. Peptide precursor and protein annotation is stored in the genes output component.

Examples

```
## Not run:
y <- readSpectronaut()
dpcfit <- dpc(y)

## End(Not run)
```

---

removeNARows *Remove Entirely NA Rows from Matrix or EList*

Description

Remove rows from a matrix that have fewer than a user-specified minimum number of non-missing observations.

Usage

```
## Default S3 method:
removeNARows(y, nobs.min = 1, ...)
```

Arguments

y	a matrix or an EList object.
nobs.min	minimum number of non-missing observations for rows to be kept.
...	other arguments are not currently used.

Details

Produces a new matrix keeping only those rows that have at least the specified number of non-missing values.

Value

A matrix or EList the same as y but with entirely or mostly missing rows removed.

Examples

```
y <- matrix(rnorm(25),5,5)
y[y < -0.5] <- NA
removeNARows(y)
```

---

simCompleteDataON *Simulate Complete Data From Complete or Observed Normal Models*

Description

Simulate a vector complete data together with the associated missing value events, under two different models.

Usage

```
simCompleteDataCN(n, mean.comp=6, sd.comp=1, dpc=c(-4,0.7))
simCompleteDataON(n, mean.obs=6, sd.obs=1, dpc=c(-4,0.7))
```

Arguments

n	number of values to simulate.
mean.comp	mean of complete normal distribution.
sd.comp	standard deviation of complete normal distribution.
mean.obs	mean of observed normal distribution.
sd.obs	standard deviation of observed normal distribution.
dpc	numeric vector of length 2 giving the DPC intercept and slope.

Details

These functions simulate a vector of complete log2-expression data and identify which will be observed and which will be missing. The complete values themselves are all non-missing, but some will be undetected in a hypothetical real dataset. `simCompleteDataCN` simulates data according to the complete normal model (CN), while `simCompleteDataON` simulates data according to the observed normal model (ON).

These functions can be used to explore the differences between the complete and observed normal models. Under the CN model, the complete values (including both observed and unobserved) are exactly normally distributed, while the subset that are observed are only approximately normal. Under the ON model, the opposite is true. The observed values are exactly normal while the complete values are only approximately normal.

Value

A list with compoenents

`y.complete`	vector of complete values.
`is.missing`	vector of `TRUE` or `FALSE` values indicating whether each value will be missing.
`prob.missing`	conditional probability given `y.complete` that each value will be missing.

Examples

```
# Complete values are only approximately normal under the ON model.
out <- simCompleteDataON(100, mean.obs=6, sd.obs=1)
mean(out$prob.missing)
qqnorm(out$y.complete)
qqline(out$y.complete)
```

`simProteinDataSet` *Simulate Peptide Data with NAs By Complete Normal Model*

Description

Simulate peptide-level log2-expression values from a mass spectrometry experiment.

Usage

```
simProteinDataSet(n.peptides = 100,
     n.groups = 2, samples.per.group = 5, peptides.per.protein = 4,
     mu.range = c(2,10), sigma = 0.4, prop.de = 0.2, fc = 2,
     dpc.intercept = NULL, dpc.slope = 0.7, prop.missing = 0.4)
```

Arguments

n.peptides	number of peptides (rows of output).
n.groups	number of experimental groups (conditions).
samples.per.group	number of samples per group.
peptides.per.protein	number of peptides per protein.
mu.range	range of log2-expression values, in terms of expected value per peptide.
sigma	standard deviation of log2-expression values for each peptide in each group.
prop.de	proportion of differentially expressed proteins.
fc	true fold-change for differentially expressed proteins.
dpc.intercept	intercept of detection probability curve. Usually determined from dpc.slope and prop.missing.
dpc.slope	slope of detection probability curve.
prop.missing	proportion of missing values (at average log2-expression). Ignored if dpc.intercept is not NULL.

Details

Simulate peptide-level log2-expression values (log2-intensities) from a mass spectrometry experiment. Values are generated and missing values assigned according to the complete normal model.

Each group of successive peptides is assumed to belong to one protein. If the protein is differentially expressed (DE), then each peptide belonging to that protein is also DE with the same fold-change.

If dpc.intercept is not specified, then it is chosen to ensure that the proportion of missing values is equal to prop.missing at the average log2-expression value.

The simulated data is stored in an EList object, the standard limma package data class for log-expression values. Peptides are ordered by average expected expression level. Some of the more lowly expressed peptides may be entirely NA, depending on the argument settings.

Value

EList containing simulated log2-expression values with n.peptides rows and n.groups * n.samples.per.group columns. The EList contains the following components:

E	matrix of peptide log2-expression values with NAs.
other$E.complete	matrix of complete log2-expression values without NAs.
genes	data.frame with columns Protein and DE.Status giving protein ID and true DE status.
targets	data.frame with column Group giving group identity for each sample.

Examples

```
y <- simProteinDataSet(n.peptides=10, n.groups=1)
show(y)
```

voomaLmFitWithImputation *Apply vooma-lmFit Pipeline With Automatic Estimation of Sample Weights and Block Correlation*

Description

Estimate the variance trend, use it to compute observational weights and use the weights to a fit a linear model. Includes automatic estimation of sample weights and block correlation. Equivalent to calling vooma(), arrayWeights(), duplicateCorrelation() and lmFit() iteratively.

Usage

```
voomaLmFitWithImputation(y, design = NULL,
     prior.weights = NULL, imputed = NULL, block = NULL,
     sample.weights = FALSE, var.design = NULL, var.group = NULL, prior.n = 10,
     predictor = NULL, span = NULL, legacy.span = FALSE,
     plot = FALSE, save.plot = FALSE, keep.EList = TRUE)
```

Arguments

y	a numeric matrix, EList object, or any object containing log-expression data that can be coerced to a matrix. Rows correspond to genes and columns to samples.
design	design matrix with rows corresponding to samples and columns to coefficients to be estimated. Defaults to the unit vector meaning that samples are treated as replicates.
prior.weights	prior weights. Can be a numeric matrix of individual weights of same dimensions as the counts, or a numeric vector of sample weights with length equal to ncol(counts), or a numeric vector of gene weights with length equal to nrow(counts).
imputed	logical matrix of the same size as y indicating whether each observation was entirely imputed.
block	vector or factor specifying a blocking variable on the arrays. Has length equal to ncol(y).
sample.weights	logical value. If TRUE then empirical sample quality weights will be estimated.
var.design	design matrix for predicting the sample variances. Defaults to the sample-specific model whereby each sample has a different variance.
var.group	vector or factor indicating groups to have different sample weights. This is another way to specify var.design for groupwise sample weights.
prior.n	prior number of genes for squeezing the weights towards equality. Larger values squeeze the sample weights more strongly towards equality.

predictor	precision predictor. Either a column vector of length nrow(y) or a numeric matrix of the same dimensions as y that predicts the precision of each log-expression value. Is used as a second covariate together with the log-intensities to predict the variances and produce the final precision weights.
span	width of the smoothing window, as a proportion of the data set. Defaults to a value between 0.3 and 1 that depends the number of genes (nrow(y)). Equal to 1 if the number of genes is less than or equal to 50, then decreases slowly to 0.3 if the number of genes is very large.
legacy.span	logical. If TRUE, then the original default setting will be used for span, which is slightly smaller than the new default.
plot	logical. If TRUE, a plot of the mean-variance trend is displayed.
save.plot	logical, should the coordinates and line of the plot be saved in the output?
keep.EList	logical. If TRUE, then the EList object containing log-expression values and observation weights will be saved in the component EList of the output object.

Details

This function is a modification of voomaLmFit in the limma package, to give special treatment to imputed values. This function gives more accurate estimation of the row-wise variances because it discounts fitted values and associated residuals that are determined entirely by imputed values that are all identical. In a regular limma pipeline, such residuals will be structurally zero and will cause underestimation of the residual variance for that gene. In voomaLmFit, such residuals do not contribute to the genewise variances and the genewise residual degrees of freedom (df) are correspondingly reduced. The principle is the same as for Lun & Smyth (2017), but here the loss of df is from imputed values instead of from zero counts.

This function is analogous to voomLmFit in the edgeR package but for continuous log-expression values instead of count data. voomLmFit is a refinement of voom adjusting for loss of residual df from all zero groups, whereas voomaLmFitWithImputation is a refinement of voomaLmFit adjusting for loss of residual df from all imputed groups. The results from voomaLmFitWithImputation are similar to those from voomaLmFit, but the df.residual values are equal or smaller and the sigma values are equal or larger.

voomaLmFitWithImputation is similar to calling vooma() followed by lmFit(), optionally with arrayWeights() and duplicateCorrelation() to estimate sample weights and block correlation. The function finishes with lmFit() and returns a fitted model object.

Like vooma, voomaLmFitWithImputation estimates the mean-variance relationship in the data and uses it to compute appropriate precision weights for each observation. The mean-variance trend is estimated from gene-level data but is extrapolated back to individual observations to obtain a precision weight (inverse variance) for each observation. The weights are then used by lmFit() to adjust for heteroscedasticity.

Like voomLmFit, which corrects for loss of residual degrees of freedom due to entirely zero counts in a group (Lun & Smyth 2017), voomaLmFitWithImputation corrects for loss of residual degrees of freedom due to entirely imputed values in a group. This adjustment prevents from the residual standard deviations from being underestimated due to zero variance between identical imputed values in a group.

If span=NULL, then an optimal span value is estimated depending on nrow(y). The span is chosen by chooseLowessSpan with n=nrow(y), small.n=50, min.span=0.3 and power=1/3. If legacy.span

= TRUE, then the chooseLowessSpan arguments are reset to small.n=10, min.span=0.3 and power = 0.5 to match the settings used by vooma in limma version 3.59.1 and earlier.

If predictor is not NULL, then the variance trend is modeled as a function of both the mean log-expression and the predictor using a multiple linear regression with the two predictors. In this case, the predictor is assumed to be some prior predictor of the precision or standard deviation of each log-expression value. Any predictor that is correlated with the precision of each observation should give good results. This ability to model the variance trend using two covariates (mean log-expression and the predictor covariate) was described for the first time by Li (2024).

Sample weights will be estimated using arrayWeights if sample.weights = TRUE or if either var.design or var.group are non-NULL. An intra-block correlation will be estimated using duplicateCorrelation if block is non-NULL. In either case, the whole estimation pipeline will be repeated twice to update the sample weights and/or block correlation.

Value

An MArrayLM object containing linear model fits for each row of data. If sample weights are estimated, then the output object will include a targets data.frame component with the sample weights as a column with heading "sample.weights".

If save.plot=TRUE then the output object will include components voom.xy and voom.line. voom.xy contains the x and y coordinates of the points in the vooma variance-trend plot and voom.line contains the estimated trend line.

If keep.EList=TRUE, then the output includes component EList with sub-components Elist$E and EList$weights. If y was an EList object, then the output EList preserves all the components of y and adds the weights.

Author(s)

Mengbo Li and Gordon Smyth

References

Li M (2024). Linear Models and Empirical Bayes Methods for Mass Spectrometry-based Proteomics Data. PhD Thesis, University of Melbourne. http://hdl.handle.net/11343/351600

Lun ATL, Smyth GK (2017). No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data. *Statistical Applications in Genetics and Molecular Biology* 16(2), 83-93. doi:10.1515/sagmb20170010

See Also

vooma, lmFit, voomLmFit (in the edgeR package).

Examples

```
# Example with a precision predictor
group <- gl(2,4)
design <- model.matrix(~group)
y <- matrix(rnorm(500*8),500,8)
u <- matrix(runif(length(y)),500,8)
```

```
yu <- y*u
fit <- voomaLmFitWithImputation(yu,design,plot=TRUE,predictor=u)

# Reproducing vooma plot from output object
fit <- voomaLmFitWithImputation(yu,design,predictor=u,save.plot=TRUE)
do.call(plot,fit$voom.xy)
do.call(lines,fit$voom.line)
```

`ztbinom` *Zero-Truncated Binomial Distribution*

Description

Density and distribution function for the zero-truncated binomial distribution, using the same arguments as for the R stats binomial distribution functions.

Usage

```
dztbinom(x, size, prob, log = FALSE)
pztbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
```

Arguments

`x`, `q`	vector of quantiles.
`p`	vector of probabilities.
`size`	number of trials (zero or more).
`prob`	probability of success on each trial.
`log`	logical; if `TRUE`, the log-density is returned.
`lower.tail`	logical; if `TRUE`, probabilities are P(X<q) otherwise P(X>q).
`log.p`	logical; if `TRUE`, probabilities are on the log-scale.

Details

These functions perform simmilarly to the R stats functions `dbinom` and `pbinom` except for the zero-truncation.

Value

Output values give density (`dztbinom`) or cumulative probability (`pztbinom`) for the zero-truncated binomial distribution with parameters `size` and `prob`. Output is a vector of length equal to the maximum length of any of the arguments `x`, `q`, `size` or `prob`. If the first argument is the longest, then all the attributes of the input argument are preserved on output, for example, a matrix `x` will give a matrix on output. Elements of input vectors that are missing will cause the corresponding elements of the result to be missing, as will non-positive values for `size` or `prob`.

Examples

```
# Compare to binomial
x <- 1:3
dztbinom(x, size=3, prob=0.5)
dbinom(x, size=3, prob=0.5)
pztbinom(x, size=3, prob=0.5)
pbinom(x, size=3, prob=0.5)
```

Index