Package 'limpa'

May 26, 2026

Version 1.5.0

Date 2026-04-26

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. Avoids the need for imputation. Estimates the detection probability curve (DPC), which relates the probability of successful detection to the underlying log-intensity of each precursor ion, and uses it to incorporate missing values into protein quantification and into subsequent differential expression analyses. The package produces objects suitable for downstream analysis in limma. The package accepts precursor (or peptide) intensities including missing values and produces complete protein quantifications without the need for imputation. The uncertainty of the protein quantifications is propagated through to the limma analyses using variance modeling and precision weights, ensuring accurate error rate control. The analysis pipeline can alternatively work with PTM or protein level data. 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 knitr, nanoparquet, BiocStyle

VignetteBuilder knitr

URL https://github.com/SmythLab/limpa

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 8edf301

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-25

Author Mengbo Li [aut] (ORCID: <https://orcid.org/0000-0002-9666-5810>), Pedro Baldoni [ctb] (ORCID: <https://orcid.org/0000-0002-9510-8326>), 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 probabilistic recovery of information from 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. The package accepts precursor (or peptide) intensities including missing values and produces complete protein quantifications without the need for imputation.

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. doi:10.1093/bioinformatics/btad200

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

`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*

Description

Detection probability curve for label-free shotgun proteomics data.

Usage

```
dpc(
  y, model="cn",
  dpc.start = NULL, dpc.slope.start = 0.7,
  iterations = 2, subset = 2000, robust = TRUE, verbose = FALSE
)
```

Arguments

y	numeric matrix of log2-intensities. Rows correspond to features (usually precursor ions) and columns to runs or samples.
model	which model to use. Possibilities are "cn" for the complete-normal model or "on" for the observed-normal model.
dpc.start	numeric vector of length 2 giving starting values for the DPC intercept and slope.
dpc.slope.start	starting value for the DPC slope. Only used if dpc.start is NULL.
iterations	number of outer iterations. Only used if model="cn".
subset	maximum number of rows of y to use in the calculation. A systematic sample is taken stratified by number of missing values and average observed log2-intensity. Only used if model="cn".

robust	if TRUE, outlier rows will be downweighted to produce a robust DPC estimate. Only used if model="on".
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 using either the "complete normal" or the "observed normal" models. The function returns results from either dpcCON() or dpcON(), depending on whether model is equal to "cn" or "on".

Li, Cobbold & Smyth (2025) show that the complete normal and observed normal models, although theoretically different, give similar results in practice.

Value

A list with components

dpc	numeric vector of length 2 giving the estimated DPC intercept and slope.
model	equal to either "CN" or "ON".

For other components of the output list, see the help pages for dpcCN or dpcON.

References

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

See Also

dpc.

Examples

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

Usage

```
dpcCN(
  y, dpc.start = NULL, dpc.slope.start = 0.7,
  iterations = 2, subset = 2000, verbose = FALSE
)
```

Arguments

y	numeric matrix of log2-intensities. Rows correspond to features (usually precursor ions) and columns to runs or samples.
dpc.start	numeric vector of length 2 giving starting values for the DPC intercept and slope.
dpc.slope.start	starting value for the DPC slope. Only used if dpc.start is NULL.
iterations	number of outer iterations.
subset	maximum number of rows of y to use in the calculation. A systematic sample is taken stratified by number of missing values and average observed log2-intensity.
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.

dpcON 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.
model	equal to "CN".

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.

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

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

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

dpcON for DPC estimation using the observed normal model, and dpc for the new main user function.

Examples

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

Details

Calls voomaLmFitWithImputation 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 functionality 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

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

See Also

voomaLmFitWithImputation. Also voomaLmFit in the limma package.

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)
```

---

dpcON *Detection Probability Curve Assuming Observed Normal Model*

Description

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

Usage

```
dpcON(y, dpc.start = NULL, dpc.slope.start = 0.7, robust = TRUE, verbose = FALSE)
dpcLegacy(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.
dpc.start	numeric vector of length 2 giving starting values for the DPC intercept and slope.
dpc.slope.start	starting value for DPC slope. Only used if dpc.start is NULL.
robust	if TRUE, outlier rows will be downweighted to produce a robust DPC estimate.
verbose	if TRUE, successive estimates will be printed to standard output.
maxit	maximum number of iterations.
eps	convergence tolerance.
b1.upper	upper bound for the DPC slope.

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.

dpcLegacy is the original dpc() function demonstrated in Li & Smyth (2023) as part the proDP package and then ported to the limpa package. It was renamed from dpc() to dpcLegacy() in limpa 1.3.11, essentially deprecating the function. dpcON is a newer and faster implementation with a robust option.

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 prior 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.
neg.loglik	minus twice the maximized log-likelihood.
model	equal to "ON".

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

See Also

dpcCN for DPC estimation using the complete normal model, and dpc for the new main user function.

Examples

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

---

dpcQuant *Quantify Proteins Using the DPC*

Description

Use the DPC to quantify protein expression values for a series of samples from precursor ion intensities.

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'
dpcQuantByRow(y, dpc = NULL, dpc.slope = 0.8, verbose = TRUE, chunk = 1000, ...)
```

Arguments

y	a numeric matrix or EList of precursor log2-intensities values. Columns are samples and rows are 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 precursor ion data. More generally, the function can summarize row-wise data to any higher annotation level, for example precursors to peptides or peptides to proteins. The method represents missing values probabilistically using the DPC 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 (or dpcON or dpcCN) 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.

dpcQuantByRow estimates log2-intensities row-wise 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$n.observations	matrix giving the number of missing non-missing precursor observations supporting each protein expression value.
other$standard.error	matrix giving the standard error of each protein expression value.
dpc	DPC used for the quantifications.
prior.mean	prior mean used for the quantification.
prior.sd	prior between-precursor standard deviation used for the quantifications.
prior.logFC	prior within-precursor standard deviation used for the quantifications.

dpcQuant() also adds the following two columns to the genes data.frame:

NPeptides	number of precursors (or features) for each protein.
PropObs	proportion of intensities that are observed (not missing). Averaged over all precursors and samples for each protein.

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

Note

The dpcQuantByRows function was previously called dpcImpute in limpa 3.2.0. The function was renamed in limpa 3.3.0 to clarify that the function is the same as dpcQuant but for each row (precursor or feature) without higher level summarization, and also to clarify that the limpa estimation process is not equivalent to straightforward imputation.

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

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

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

EListFromLongFormatFile

Read Feature Intensities From a Long Format Report File Written By a Mass Spectrometry Quantification Tool

---

Description

Create an EList from a long format report file written by a quantification tool such as Spectronaut or DIA-NN.

Usage

```
EListFromLongFormatFile(
  file = "report.tsv", path = NULL,
  format = NULL, sep = "\t",
  run.column,
  feature.column,
  intensity.column,
  annotation.columns = NULL,
  q.columns = NULL, q.cutoffs = 0.01,
  filter.columns = NULL, filter.values = TRUE,
  censor.value = NULL,
  matrix.columns = NULL,
  log = TRUE,
  verbose = TRUE)
```

Arguments

file	the name or path of the report file. If a text file, it should not be compressed. Alternatively, file can be a long format data.frame, as would arise from reading the report file into R using a function such as read.delim.
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" for a tab-delimited text file or "parquet" for a Parquet format file. By default, the format is detected from the file name extension.
sep	the field separator character for delimited text files. This argument is usually unnecessary, but it can be used in combination with format="tsv" to read comma-separated files.
run.column	name of the column identifying the mass spectrometry runs. Usually each run is a distinct protein sample or, if fractioning is used, a fraction of a sample.
feature.column	name of the column containing feature IDs, or a character vector containing the names of columns, which, when pasted together, will uniquely identify each feature.
intensity.column	character string giving the name of the column containing feature intensities.
annotation.columns	other columns to be read and included in the output genes data.frame annotating the features and proteins.
q.columns	character vector of column names containing q-values for feature identification.
q.cutoffs	cutoffs to apply to the q-values. Either a single value or a numeric vector of the same length as q.columns. Only features with all q-values below the corresponding cutoffs will be retained.
filter.columns	optional columns that will be used to filter observations. The columns may indicated observations that were imputed, or values not used for the final intensity estimates. Any observation that is flagged by any of these columns will be NA in the output EList.
filter.values	values for the filter.columns indicating that the observation should be filtered.
censor.value	any intensities less than or equal to this value will be replaced by NAs.

matrix.columns	character vector of column names containing observation-level covariates. Each such variable will be copied into a matrix of the same dimensions as the intensities and stored in the other component of the output object.
log	logical. If TRUE then intensities will be returned on the log2 scale, otherwise unlogged.
verbose	logical, whether to send informative progress messages.

Details

This function reads report files written in long format by tools wuch as DIA-NN (Demichev et al 2020), Spectronaut (https://biognosys.com/software/spectronaut/), or MSstats convert functions. It produces an EList or EListRaw object with features as rows and samples as columns.

This function is most often used to read precursor ion intensities, but can read intensity data at any summarization level if the intensities are provided in the file. If feature.column contains protein IDs, then protein-level intensities (such as maxLFQ) summaries will be read. If feature.column specifies peptide sequence and charge, then precursor ion intensities will be read. If feature.column specifies fragment ion and fragment charge as well as peptide sequence and precursor charge, then fragment peak intensities can be read.

If file is a Spectronaut v20 report file, then feature.column="FG.ProteinsGroups" and intensity.column="PG.Qua will read protein-level summary intensities, feature.column=c("EG.ModifiedSquence,FG.Charge" and intensity.column="EG.TotalQuantity (Settings)" will read precursor ion estimated intensities, while feature.column=c("EG.ModifiedSequence","FG.Charge","F.FrgIon","F.Charge") and intensity.column="F.PeakArea" will read fragment peak intensities.

This function uses data.table::fread to read text files and nanoparquet::read_parquet to read Parquet files.

Value

If log=FALSE, an EListRaw object containing precursor unlogged intensities, and protein and feature annotation. If log=TRUE, an EList object containing precursor log2 intensities with NAs, and protein and feature annotation. Rows correspond to features and columns to samples. Precursor and protein annotation is stored in the genes output component.

References

BIOGNOSYS Spectronaut manual https://biognosys.com/resources/spectronaut-manual/.

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.

Yu Z, Du A, Xu X, Li Y, Ma X, Zhang W, Zhang Y, Chu IK, Siu KM (2026). Spectronaut and DIA-NN: a comparison of their performance in the analysis of lung adenocarcinoma biopsies. *ACS Omega* 11(5) 8080–8093. doi:10.1021/acsomega.5c10421

estimateDPCIntercept *Estimate DPC Intercept*

Description

Estimate the DPC intercept given a value for the slope.

Usage

```
estimateDPCIntercept(y, dpc.slope = 0.8, verbose = 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.
verbose	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)
```

filterByDetection *Filter Proteins By Detection*

Description

Keep proteins that are detected at least a specified number of times in at least a specified number of samples.

Usage

```
filterByDetection(y, n.samples = 3, n.detections = 1)
```

Arguments

y	EList object produced by dpcQuant().
n.samples	minimum number of samples that protein must be detected in.
n.detections	number of detections (number of non-missing precursors) required for each sample.

Details

This function is used for filtering of proteins after dpcQuant but before normalization or dpeDE. The number of samples required should be chosen to retain proteins that are biological meaningful. For a small experiment, n.samples might be set to the smallest group size. For a large dataset, n.samples would be chosen to keep proteins that are expressed in a sufficiently large subset of samples to be scientifically meaningful in a list of differentially expressed proteins.

Value

Logical vector of length nrow(y) indicating which rows of y to keep in the analysis.

Examples

```
## Not run:
y <- dpcQuant(y.prec, ProteinID)
keep <- filterByDetection(y)
yfilt <- y[keep,]

## End(Not run)
```

---

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

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

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-intensity values. Columns are samples and rows are quantification features, usually 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

Li M, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

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)
```

---

plotAveVsMis *Average Observed Log-intensity Vs Number of Mising Values*

Description

For rows of a matrix containing log-intensities, plot average observed log-intensity vs number of missing values.

Usage

```
plotAveVsMis(y)
```

Arguments

y	matrix or EList containing log-intensities.

Value

A plot is created on the current graphics device.

Examples

```
y <- simProteinDataSet(1000)
plotAveVsMis(y)
```

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", show.start = FALSE, ...)
```

Arguments

dpcfit	object produced by dpc().
add.jitter	logical, whether to add jitter to the detected proportions.
point.cex	relative size of points.
lwd	relative line width.
ylim	limits of the y-axis.
main	main title of plot.
show.start	logical, whether to show starting as well as final estimate of curve.
...	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()
```

---

plotPeptides *Plot Peptide Log-Intensities for One Protein*

Description

Plot the peptide-level log-intensities for a specified protein.

Usage

```
## Default S3 method:
plotPeptides(y, cex = 1.5, lwd = 1.5, col = "blue", las = NULL, step.down = 0.5,
     xlab = "", ylab = "Log-intensity", ...)
## S3 method for class 'EList'
plotPeptides(y, index, cex = 1.5, lwd = 1.5, col = "blue",
     las = NULL, step.down = 0.5, xlab = "", ylab = "Log-intensity", ...)
```

Arguments

`y`	a numeric matrix or EList of peptide-level log2-intensity values. Columns are samples and rows are peptides or precursors.
`index`	index such that `y[index,]` will select rows for one protein.
`cex`	plot symbol size.
`lwd`	line width.
`col`	line color.
`las`	orientation of x-axis sample labels. Use `0` or `1` for horizontal and `2` or `3` for vertical. Defaults to vertical if more than six samples and horizontal if less.
`step.down`	missing values are plotted this amount below the minimum observed value for each peptide.
`xlab`	x-axis label.
`ylab`	y-axis label.
`...`	other arguments are passed to `plot`.

Details

Plots the log-intensities for the peptides or precursors for one protein, connecting the points belonging to the same peptide.

Value

A plot is made on the current graphics device. Closed dots correspond to observed values and open dots to missing values. The matrix of plotting points is also invisibly returned.

See Also

plotProtein.

Examples

```
y.peptide <- simProteinDataSet(20,n.groups=1,samples.per.group=4,peptides.per.protein=4)
plotPeptides(y.peptide, 9:12)
```

---

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, las = NULL,
            xlab = "", ylab = "Estimated log-intensity", ...)
```

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.
las	orientation of x-axis sample labels. Use 0 or 1 for horizontal and 2 or 3 for vertical. Defaults to vertical if more than six samples and horizontal if less.
xlab	x-axis label.
ylab	y-axis label.
...	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:

`y`	numeric vector of estimated log-intensities for the protein
`se`	numeric vector of standard errors

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, Cobbold SA, Smyth GK (2025). Quantification and differential analysis of mass spectrometry proteomics data with probabilistic recovery of information from missing values. *bioRxiv* 2025/651125. doi:10.1101/2025.04.28.651125

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)
```

---

pztbinomSameSizeLogitPBothTails

Zero-Truncated Binomial Distribution P-Values

Description

Left and right p-values for zero-truncated binomial deviates.

Usage

```
pztbinomSameSizeLogitPBothTails(q, size, logit.prob)
```

Arguments

`q`	vector of quantiles.
`size`	number of trials (one or more). Should be a single value.
`logit.prob`	success probabilities on logit scale.

Details

This function returns left and right p-values for zero-truncated binomial deviates. The `size` argument is assumed to be the same for all deviates. This function is slow but accurate, even for very small or very large success probabilities.

Value

A list with components:

left.p.value	left tail p-values.
right.p.value	matrix of complete log2-expression values without NAs.
total.prob	data.frame with columns Protein and DE.Status giving protein ID and true DE status.

Examples

```
x <- 1:3
pztbinomSameSizeLogitPBothTails(x, size=3, logit.prob=0)
```

---

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

Description

Read the DIA-NN report file (report.tsv or report.parquet) into an EList object.

Usage

```
readDIANN(
  file = "report.parquet",
  path = NULL,
  format = NULL,
  sep = "\t",
  run.column = "Run",
  feature.column = "Precursor.Id",
  intensity.column = "Precursor.Normalised",
 annotation.columns = c("Protein.Group", "Protein.Names", "Genes", "Proteotypic"),
  q.columns = c("Q.Value", "Lib.Q.Value", "Lib.PG.Q.Value"),
  q.cutoffs = 0.01,
  log = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

file	name of the Report file from which the data are to be read. The file usually called "report.tsv" or "report.parquet" in the DIA-NN output. Alternatively, file can be a data.frame, as would arise from reading the report file into R using a function such as read.delim.
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" for a delimited text file or "parquet" for a Parquet format file. By default, the format is detected from the file name extension.

sep	the field separator character, used when format="tsv". Text files from DIA-NN are normally tab-delimited, but this argument can be used together with format="tsv" to read comma-separated files if necessary.
run.column	name of the column identifying the mass spectrometry runs. Usually each run is a distinct protein sample or, if fractioning is used, a fraction of a sample.
feature.column	name of column containing precursor IDs, or protein IDs if protein-level data is being read. Can be character vector of length two containing the names of the peptide sequence and charge columns, which will then be read separately and pasted together to form a precursor ID.
intensity.column	name of column containing precursor intensities.
annotation.columns	names of other columns to be read and included in the output genes data.frame annotating the precursors and proteins.
q.columns	names of columns containing q-values for peptide or protein identification.
q.cutoffs	cutoffs to apply to the q-value columns. Either a single value or a numeric vector of the same length as q.columns. Only features with all q-values below the corresponding cutoffs will be retained.
log	logical. If TRUE then intensities will be returned on the log2 scale, otherwise unlogged.
verbose	logical, whether to send informative progress messages. Set this to FALSE if you want the function to run quietly.
...	other arguments are not currently used.

Details

DIA-NN (Demichev et al 2020) writes a main report file in long (data.frame) format, typically called report.tsv or report.parquet, containing normalized intensities for precursors ions. readDIANN reads this file and produces an EList or EListRaw object.

Version 1 of DIA-NN wrote the report file in tab-delimited format. Version 2 of DIA-NN writes the report in Apache Parquet format (https://github.com/vdemichev/DiaNN/releases). In any case, readDIANN can read the report file directly.

An example analysis using this function is shown here: https://smythlab.github.io/limpa/HYE100-DIANN.html.

Value

If log=FALSE, an EListRaw object containing precursor unlogged intensities, with intensities for non-detected precursors equal to 0. If log=TRUE, an EList object containing precursor log2 intensities. Rows are precursor ions and columns are samples. 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()
dpcest <- dpc(ypep)
yprot <- dpcQuant(ypep, dpc=dpcest)

## End(Not run)
```

`readFragPipe` *Read Peptide-Precursor Intensities From FragPipe Output*

Description

Read FragPipe combined peptide, modified peptide or ion output into EList object.

Usage

```
readFragPipe(
    file = "combined_ion.tsv", path = NULL, sep = "\t", log = TRUE,
    peptide.column = c("Modified Sequence", "Charge"),
    qty.column = NULL,
    qty.column.key = " Intensity",
   extra.columns = c("Protein", "Protein ID", "Gene", "Protein Description", "Mapped Proteins"),
    match.type.key = NULL,
    maxlfq = FALSE
)
```

Arguments

`file`	the name of the file from which the data are to be read, typically should be one of "combined_ion.tsv", "combined_peptide.tsv", or "combined_modified_peptide.tsv" output by FragPipe.
`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.
`peptide.column`	column containing peptide IDs. Character vector. If length > 1, these columns will be concatenated to make a unique precursor ID.
`qty.column`	columns containing intensities.
`qty.column.key`	character string of the key that identify columns containing intensities. If not NULL, `qty.column` will be overwritten.
`extra.columns`	extra columns that are appended to the precursor annotation matrix.
`match.type.key`	character string of the key that identify columns containing match type annotation.
`maxlfq`	logical. If `TRUE`, columns containing MaxLFQ intensities will be output as `E` in the output object. Only set to `TRUE` when `qty.column.key` identifies the MaxLFQ columns.

Details

FragPipe (Yu et al 2023) writes a file in wide format called `combine_peptide.tsv` containing normalized intensities for peptide precursors from all experimental samples https://fragpipe.nesvilab.org/docs/tutorial_fragpipe_outputs.html. Rows correspond to precursor ions. The names of the columns containining precursor intensities contain the name of the experiment (biological sample) followed by " `Intensity`", and the `qty.column.key` argument is used to identify these columns. `readFragPipe` reads this file and produces an 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

Yu F, Teo GC, Kong AT, Frölich K, Li GX, Demichev V, Nesvizhskii AI (2023). Analysis of DIA proteomics data using MSFragger-DIA and FragPipe computational platform. *Nature Communications* 14, 4154. doi:10.1038/s41467023398695

Tutorial on FragPipe Outputs: https://fragpipe.nesvilab.org/docs/tutorial_fragpipe_outputs.html.

Examples

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

## End(Not run)
```

`readMaxQuant` *Read Peptide-Precursor Intensities From MaxQuant Output*

Description

Read MaxQuant peptide output into EList object.

Usage

```
readMaxQuant(
    file = "peptides.txt",
    path = NULL, sep = "\t", log = TRUE,
    peptide.column = "Sequence",
    qty.column = NULL,
    qty.column.key = "Intensity ",
    q.columns = c("PEP"), q.cutoffs = 0.01,
    extra.columns = c("Proteins", "Leading razor protein",
    "Gene names", "Unique (Groups)", "Protein group IDs")
)
```

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.
peptide.column	column containing peptide IDs. String of length 1L.
qty.column	columns containing intensities.
qty.column.key	character string of the key that identify columns containing intensities. If not NULL, qty.column will be overwritten.
q.columns	column headings in the 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

readMaxQuant reads standard peptide.txt output from MaxQuant 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 <- readMaxQuant("peptides.txt")
dpcfit <- dpc(y)

## End(Not run)
```

---

readSpectronaut *Read Spectronaut Normal Report File*

Description

Read a Spectronaut Normal Report file into an EList or EListRaw object.

Usage

```
readSpectronaut(
  file = "Report.tsv",
  path = NULL,
  sep = "\t",
  run.column = "R.FileName",
  feature.column = c("EG.ModifiedSequence","FG.Charge"),
  intensity.column = "EG.TotalQuantity (Settings)",
  annotation.columns = c("PG.ProteinAccessions","PG.Genes"),
  q.columns = c("EG.Qvalue", "PG.Qvalue"),
  q.cutoffs = 0.01,
  filter.columns = "EG.IsImputed",
  filter.values = TRUE,
  censor.value = 0,
  run.info = TRUE,
  log = TRUE,
  verbose = TRUE,
  ...
)

readSpectronautRunInfo(
  file="report.tsv",
  path=NULL,
  sep="\t",
  run.column = "R.FileName",
  run.info.columns = c("R.Condition","R.Fraction","R.Label","R.Replicate"),
  verbose = TRUE
)
```

Arguments

`file`	name of the Normal Report file from which the data are to be read. Alternatively, `file` can be a data.frame, as would arise from reading the report file into R using a function such as `read.delim`.
`path`	character string giving the directory containing the file. Defaults to the current working directory.
`sep`	the field separator character. Spectronaut normally writes tab-delimited files, but this argument can be used to read comma-separated files if necessary.
`run.column`	name of the column identifying the mass spectrometry runs. Usually each run is a distinct protein sample or, if fractioning is used, a fraction of a sample.
`feature.column`	name of the column containing feature IDs, or a character vector containing the names of columns, which, when pasted together, will uniquely identify each feature. By default, the features are precursor ions, but other possibilities are fragments, peptides or proteins. See details for examples. This argument and `intensity.column` must correspond to the same feature level.
`intensity.column`	name of column containing the feature intensities. By default, the precursor ion intensities are read, but see details for other possibilities. This argument and `feature.column` must correspond to the same feature level.

annotation.columns
: names of other columns to be read and included in the output genes data.frame annotating the precursors and proteins.

q.columns
: names of columns containing q-values for peptide or protein identification.

q.cutoffs
: cutoffs to apply to the q-value columns. Either a single value or a numeric vector of the same length as q.columns. If any q-value is above the corresponding cutoff, then the intensity will be set to NA.

filter.columns
: optional columns that will be used to filter observations. The columns may indicated observations that were imputed, or values not used for the final intensity estimates at the peptide or protein levels.

filter.values
: values for the filter.columns indicating that the observation should be filtered.

censor.value
: intensities less than or equal to this value will be replaced by NAs. Spectronaut uses precursor intensities of 0 as placeholders to indicate indetermined intensities.

run.info
: should run information columns be read? If TRUE, the run columns on condition and replicate will be read and stored in the targets data.frame.

log
: logical. If TRUE then intensities will be returned on the log2 scale, otherwise unlogged.

verbose
: logical, whether to send informative progress messages. Set this to FALSE if you want the function to run quietly.

run.info.columns
: names of columns containing run-level information, for example information on the experimental conditions. By default, the standard Spectronaut columns are read.

...
: other arguments are not currently used.

Details

The "Normal Report" file from Spectronaut (https://biognosys.com/software/spectronaut/) is a long format file, typically called Report.tsv, which contains normalized intensities for precursor ions, peptides and proteins. readSpectronaut reads this file and produces an EList or EListRaw object.

If file is a Spectronaut v20 report file, then feature.column="FG.ProteinGroups" and intensity.column="PG.Quan will read protein-level summary intensities, feature.column=c("EG.ModifiedSquence,FG.Charge" and intensity.column="EG.TotalQuantity (Settings)" will read precursor ion estimated intensities, while feature.column=c("EG.ModifiedSequence","FG.Charge","F.FrgIon","F.Charge") and intensity.column="F.PeakArea" will read fragment peak intensities. Note that the columns F.FrgIon, F.Charge and F.PeakArea are not included in Spectronaut report files by default, but can be requested when Spectronaut is run. Including fragment fragment information will make the file considerably larger.

Value

If log=FALSE, an EListRaw object containing unlogged intensities with precursor and protein annotation. Unlogged intensities equal to 0 represent non-detected precursors. If log=TRUE, an EList object containing log2 intensities with precursor and protein annotation. Rows are features, usually precursor ions, and columns are runs or samples. Precursor and protein annotation is stored in the genes output component.

readSpectronautRunInfo produces a data.frame with a row for each run (sample) and a column for each run information columns found in the file.

References

BIOGNOSYS Spectronaut manual https://biognosys.com/resources/spectronaut-manual/.

Yu Z, Du A, Xu X, Li Y, Ma X, Zhang W, Zhang Y, Chu IK, Siu KM (2026). Spectronaut and DIA-NN: a comparison of their performance in the analysis of lung adenocarcinoma biopsies. *ACS Omega* 11(5) 8080-8093. doi:10.1021/acsomega.5c10421

See Also

EList-class.

Examples

```
## Not run:
y <- readSpectronaut()
dpcest <- 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, logit.p = FALSE)
pztbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
```

Arguments

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

Details

dztbinom and pztbinom perform similarly to the R stats functions dbinom and pbinom except for the zero-truncation.

Value

Output values give density (dztbinom) or cumulative probability (pztbinom) values 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 negative values for 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