Package ‘zinbwave’

May 11, 2024

Type Package

Title Zero-Inflated Negative Binomial Model for RNA-Seq Data

Version 1.27.0

Description Implements a general and flexible zero-inflated negative binomial model that can be used to provide a low-dimensional representations of single-cell RNA-seq data. The model accounts for zero inflation (dropouts), over-dispersion, and the count nature of the data. The model also accounts for the difference in library sizes and optionally for batch effects and/or other covariates, avoiding the need for pre-normalize the data.

License Artistic-2.0

Depends R (>= 3.4), methods, SummarizedExperiment, SingleCellExperiment

Imports BiocParallel, softImpute, stats, genefilter, edgeR, Matrix

Suggests knitr, rmarkdown, testthat, matrixStats, magrittr, scRNAseq, ggplot2, biomaRt, BiocStyle, Rtsne, DESeq2, sparseMatrixStats

VignetteBuilder knitr

LazyData TRUE

RoxygenNote 7.3.1

biocViews ImmunoOncology, DimensionReduction, GeneExpression, RNASeq, Software, Transcriptomics, Sequencing, SingleCell

BugReports https://github.com/drisso/zinbwave/issues

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

git_branch devel

git_last_commit 82cf6db

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-05-10

Author Davide Risso [aut, cre, cph],
Svetlana Gribkova [aut],
Fanny Perraudeau [aut],

Jean-Philippe Vert [aut],
Clara Bagatin [aut]

Maintainer Davide Risso <risso.davide@gmail.com>

Contents

`computeDevianceResiduals`
Deviance residuals of the zero-inflated negative binomial model

---

Description

Given a matrix of counts, this function computes the deviance residuals under a zero-inflated negative binomial (ZINB) model.

Usage

```
computeDevianceResiduals(model, x, ignoreW = TRUE)
```

Arguments

`model`	the zinb model
`x`	the matrix of counts n cells by J genes
`ignoreW`	logical, if true matrix `W` is ignored. Default is TRUE.

Value

the matrix of deviance residuals of the model.

Examples

```
se <- SummarizedExperiment(matrix(rpois(60, lambda=5), nrow=10, ncol=6),
    colData = data.frame(bio = gl(2, 3)))
m <- zinbFit(se, X=model.matrix(~bio, data=colData(se)),
    BPPARAM=BiocParallel::SerialParam())
computeDevianceResiduals(m, t(assay(se)))
```

computeObservationalWeights

Observational weights of the zero-inflated negative binomial model for each entry in the matrix of counts

Description

Given a matrix of counts, this function computes the observational weights of the counts under a zero-inflated negative binomial (ZINB) model. For each count, the ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.

Usage

```
computeObservationalWeights(model, x)
```

Arguments

model	the zinb model
x	the matrix of counts

Value

the matrix of observational weights computed from the model.

Examples

```
se <- SummarizedExperiment(matrix(rpois(60, lambda=1), nrow=10, ncol=6),
    colData = data.frame(bio = gl(2, 3)))
m <- zinbFit(se, X=model.matrix(~bio, data=colData(se)),
    BPPARAM=BiocParallel::SerialParam())
computeObservationalWeights(m, assay(se))
```

getAlpha_mu

Returns the matrix of paramters alpha_mu

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with W for the mean part (mu)

Usage

```
getAlpha_mu(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of alpha_mu parameters

Examples

```
a <- zinbModel(n=5, J=10)
getAlpha_mu(a)
```

`getAlpha_pi` *Returns the matrix of paramters alpha_pi*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with W for the zero part (pi)

Usage

```
getAlpha_pi(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of alpha_pi parameters

Examples

```
a <- zinbModel(n=5, J=10)
getAlpha_pi(a)
```

getBeta_mu — *Returns the matrix of paramters beta_mu*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with X_mu

Usage

```
getBeta_mu(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of beta_mu parameters

Examples

```
a <- zinbModel(n=5, J=10)
getBeta_mu(a)
```

getBeta_pi — *Returns the matrix of paramters beta_pi*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with X_pi

Usage

```
getBeta_pi(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of beta_pi parameters

Examples

```
a <- zinbModel(n=5, J=10)
getBeta_pi(a)
```

---

getEpsilon_alpha *Returns the vector of regularization parameter for alpha*

Description

Given an object describing a ZINB model, returns a vector of size the number of rows in the parameter `alpha` with the regularization parameters associated to each row. Here `alpha` refers to both `alpha_mu` and `alpha_pi`, which have the same size and have the same regularization.

Usage

```
getEpsilon_alpha(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `alpha_mu` and `alpha_pi`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_alpha(a)
```

---

getEpsilon_beta_mu *Returns the vector of regularization parameter for beta_mu*

Description

Given an object describing a ZINB model, returns a vector of size the number of rows in the parameter `beta_mu` with the regularization parameters associated to each row.

Usage

```
getEpsilon_beta_mu(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `beta_mu`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_beta_mu(a)
```

---

getEpsilon_beta_pi *Returns the vector of regularization parameter for beta_pi*

Description

Given an object describing a ZINB model, returns a vector of size the number of rows in the parameter `beta_pi` with the regularization parameters associated to each row.

Usage

```
getEpsilon_beta_pi(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `beta_pi`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_beta_pi(a)
```

---

getEpsilon_gamma_mu *Returns the vector of regularization parameter for gamma_mu*

Description

Given an object describing a ZINB model, returns a vector of size the number of columns in the parameter `gamma_mu` with the regularization parameters associated to each row.

Usage

```
getEpsilon_gamma_mu(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `gamma_mu`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_gamma_mu(a)
```

`getEpsilon_gamma_pi` *Returns the vector of regularization parameter for gamma_pi*

Description

Given an object describing a ZINB model, returns a vector of size the number of columns in the parameter `gamma_pi` with the regularization parameters associated to each column.

Usage

```
getEpsilon_gamma_pi(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `gamma_pi`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_gamma_pi(a)
```

getEpsilon_W — *Returns the vector of regularization parameter for W*

Description

Given an object describing a ZINB model, returns a vector of size the number of columns in the parameter `W` with the regularization parameters associated to each column.

Usage

```
getEpsilon_W(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `W`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_W(a)
```

getEpsilon_zeta — *Returns the regularization parameter for the dispersion parameter*

Description

The regularization parameter penalizes the variance of zeta, the log of the dispersion parameters across samples.

Usage

```
getEpsilon_zeta(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the regularization parameters for `zeta`.

Examples

```
a <- zinbModel(n=5, J=10)
getEpsilon_zeta(a)
```

getGamma_mu — *Returns the matrix of paramters gamma_mu*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with V_mu

Usage

```
getGamma_mu(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of gamma_mu parameters

Examples

```
a <- zinbModel(n=5, J=10)
getGamma_mu(a)
```

getGamma_pi — *Returns the matrix of paramters gamma_pi*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of parameters associated with V_pi

Usage

```
getGamma_pi(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the matrix of gamma_pi parameters

Examples

```
a <- zinbModel(n=5, J=10)
getGamma_pi(a)
```

`getLogitPi` *Returns the matrix of logit of probabilities of zero*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of logit of probabilities of 0.

Usage

```
getLogitPi(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Details

Note that although the user interface of `zinbFit` requires a J x n matrix, internally this is stored as a n x J matrix (i.e., samples in row and genes in column). Hence the parameter matrix returned by this function is of n x J dimensions.

Value

the matrix of logit-probabilities of 0

Examples

```
a <- zinbModel(n=5, J=10)
getLogitPi(a)
```

getLogMu — *Returns the matrix of logarithm of mean parameters*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of logarithm of mean parameters.

Usage

```
getLogMu(object)
```

Arguments

object	an object that describes a matrix of zero-inflated distributions.

Details

Note that although the user interface of `zinbFit` requires a J x n matrix, internally this is stored as a n x J matrix (i.e., samples in row and genes in column). Hence the parameter matrix returned by this function is of n x J dimensions.

Value

the matrix of logarithms of mean parameters

Examples

```
a <- zinbModel(n=5, J=10)
getLogMu(a)
```

getMu — *Returns the matrix of mean parameters*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of mean parameters.

Usage

```
getMu(object)
```

Arguments

object	an object that describes a matrix of zero-inflated distributions.

Details

Note that although the user interface of zinbFit requires a J x n matrix, internally this is stored as a n x J matrix (i.e., samples in row and genes in column). Hence the parameter matrix returned by this function is of n x J dimensions.

Value

the matrix of mean parameters

Examples

```
a <- zinbModel(n=5, J=10)
getMu(a)
```

---

getPhi *Returns the vector of dispersion parameters*

Description

Given an object that describes a matrix of zero-inflated negative binomial distributions, returns the vector of dispersion parameters phi.

Usage

```
getPhi(object)
```

Arguments

object	an object that describes a matrix of zero-inflated. distributions.

Value

the vector of dispersion parameters

Examples

```
a <- zinbModel(n=5, J=10)
getPhi(a)
```

getPi — *Returns the matrix of probabilities of zero*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the matrix of probabilities of 0.

Usage

```
getPi(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Details

Note that although the user interface of `zinbFit` requires a J x n matrix, internally this is stored as a n x J matrix (i.e., samples in row and genes in column). Hence the parameter matrix returned by this function is of n x J dimensions.

Value

the matrix of probabilities of 0

Examples

```
a <- zinbModel(n=5, J=10)
getPi(a)
```

getTheta — *Returns the vector of inverse dispersion parameters*

Description

Given an object that describes a matrix of zero-inflated negative binomial distributions, returns the vector of inverse dispersion parameters `theta`.

Usage

```
getTheta(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the vector of inverse dispersion parameters theta

Examples

```
a <- zinbModel(n=5, J=10)
getTheta(a)
```

---

getV_mu — *Returns the gene-level design matrix for mu*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the gene-level design matrix for mu

Usage

```
getV_mu(object, ...)
```

Arguments

object	an object that describes a matrix of zero-inflated distributions.
...	Additional parameters.

Value

the gene-level design matrix for mu

Examples

```
a <- zinbModel(n=5, J=10)
getV_mu(a)
```

---

getV_pi — *Returns the gene-level design matrix for pi*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the gene-level design matrix for pi

Usage

```
getV_pi(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the gene-level design matrix for pi

Examples

```
a <- zinbModel(n=5, J=10)
getV_pi(a)
```

---

getW — *Returns the low-dimensional matrix of inferred sample-level covariates W*

Description

Given an object that contains the fit of a ZINB-WaVE model, returns the matrix `W` of low-dimensional matrix of inferred sample-level covariates.

Usage

```
getW(object)
```

Arguments

`object`	a `ZinbModel` object, typically the result of `zinbFit`.

Value

the matrix `W` of inferred sample-level covariates.

Examples

```
a <- zinbModel(n=5, J=10)
getW(a)
```

getX_mu — *Returns the sample-level design matrix for mu*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the sample-level design matrix for mu

Usage

```
getX_mu(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the sample-level design matrix for mu

Examples

```
a <- zinbModel(n=5, J=10)
getX_mu(a)
```

getX_pi — *Returns the sample-level design matrix for pi*

Description

Given an object that describes a matrix of zero-inflated distributions, returns the sample-level design matrix for pi

Usage

```
getX_pi(object, ...)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.
`...`	Additional parameters.

Value

the sample-level design matrix for pi

Examples

```
a <- zinbModel(n=5, J=10)
getX_pi(a)
```

---

getZeta — *Returns the vector of log of inverse dispersion parameters*

Description

Given an object that describes a matrix of zero-inflated negative binomial distributions, returns the vector `zeta` of log of inverse dispersion parameters

Usage

```
getZeta(object)
```

Arguments

`object`	an object that describes a matrix of zero-inflated distributions.

Value

the vector `zeta` of log of inverse dispersion parameters

Examples

```
a <- zinbModel(n=5, J=10)
getZeta(a)
```

---

glmWeightedF — *Zero-inflation adjusted statistical tests for assessing differential expression.*

Description

This function recycles an old version of the `glmLRT` method that allows an F-test with adjusted denominator degrees of freedom to account for the downweighting in the zero-inflation model.

Usage

```
glmWeightedF(
  glmfit,
  coef = ncol(glmfit$design),
  contrast = NULL,
  ZI = TRUE,
  independentFiltering = TRUE,
  filter = NULL
)
```

Arguments

- `glmfit` — a `DGEGLM-class` object, usually output from `glmFit`.
- `coef` — integer or character vector indicating which coefficients of the linear model are to be tested equal to zero. Values must be columns or column names of design. Defaults to the last coefficient. Ignored if `contrast` is specified.
- `contrast` — numeric vector or matrix specifying one or more contrasts of the linear model coefficients to be tested equal to zero. Number of rows must equal to the number of columns of `design`. If specified, then takes precedence over `coef`.
- `ZI` — logical, specifying whether the degrees of freedom in the statistical test should be adjusted according to the weights in the `fit` object to account for the down-weighting. Defaults to TRUE and this option is highly recommended.
- `independentFiltering` — logical, specifying whether independent filtering should be performed.
- `filter` — vector of values to perform filtering on. Default is the mean of the fitted values from glmfit.

Details

When ‘independentFiltering=TRUE‘ (default) an independent filtering step is applied prior to the multiple testing procedure, as described in great details in the ‘DESeq2“ vignette. The values in the ‘padjFilter‘ column refer to this procedure. They are identical to the ‘FDR‘ values if the filtering step does not remove any gene, since the function uses the Benjamini-Hochberg correction by default. If the procedure filters some genes, the adjusted p-values will typically result in greater power to detect DE genes. The theory behind independent filtering is described in Bourgon et al. (2010).

Note

This function uses an adapted version of the `glmLRT` function that was originally written by Gordon Smyth, Davis McCarthy and Yunshun Chen as part of the edgeR package. Koen Van den Berge wrote code to adjust residual degree of freedoom and added the independent filtering step.

References

McCarthy, DJ, Chen, Y, Smyth, GK (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. Nucleic Acids Research 40, 4288-4297. Bourgon, Richard, Robert Gentleman, and Wolfgang Huber (2010) Independent Filtering Increases Detection Power for High-Throughput Experiments. PNAS 107 (21): 9546-51.

See Also

`glmLRT`

imputeZeros — *Impute the zeros using the estimated parameters from the ZINB model.*

Description

Given a matrix of counts and a zinb model, this function computes the imputed counts under a zero-inflated negative binomial (ZINB) model.

Usage

```
imputeZeros(model, x)
```

Arguments

model	the zinb model
x	the matrix of counts n cells by J genes

Value

the matrix of imputed counts.

Examples

```
se <- SummarizedExperiment(matrix(rpois(60, lambda=5), nrow=10, ncol=6),
    colData = data.frame(bio = gl(2, 3)))
m <- zinbFit(se, X=model.matrix(~bio, data=colData(se)),
    BPPARAM=BiocParallel::SerialParam())
imputeZeros(m, t(assay(se)))
```

independentFiltering — *Perform independent filtering in differential expression analysis.*

Description

This function uses the DESeq2 independent filtering method to increase detection power in high throughput gene expression studies.

Usage

```
independentFiltering(object, filter, objectType = c("edgeR", "limma"))
```

Arguments

object	Either a DGELRT-class object or a data.frame with differential expression results.
filter	The characteristic to use for filtering, usually a measure of normalized mean expression for the features.
objectType	Either "edgeR" or "limma". If "edgeR", it is assumed that object is of class DGELRT-class, the output of glmLRT. If "limma", it is assumed that object is a data.frame and the output of a limma-voom analysis.

Author(s)

Koen Van den Berge

References

Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12):550, dec 2014.

See Also

results

---

loglik — *Compute the log-likelihood of a model given some data*

Description

Given a statistical model and some data, this function computes the log-likelihood of the model given the data, i.e., the log-probability of the data under the model.

Usage

```
loglik(model, x, ...)

## S4 method for signature 'ZinbModel,matrix'
loglik(model, x)
```

Arguments

model	an object that describes a statistical model.
x	an object that describes data.
...	additional arguments.

Value

The log-likelihood of the model given the data.

Methods (by class)

- `loglik(model = ZinbModel, x = matrix)`: return the log-likelihood of the ZINB model.

Examples

```
m <- zinbModel(n=5, J=10)
x <- zinbSim(m)
loglik(m, x$counts)
```

---

nFactors — *Generic function that returns the number of latent factors*

Description

Given an object that describes a dataset or a model involving latent factors, this function returns the number of latent factors.

Usage

```
nFactors(x)
```

Arguments

`x`	an object that describes a dataset or a model involving latent factors

Value

the number of latent factors

Examples

```
a <- zinbModel()
nFactors(a)
```

---

nFeatures — *Generic function that returns the number of features*

Description

Given an object that describes a dataset or a model, it returns the number of features.

Usage

```
nFeatures(x)
```

Arguments

x	an object that describes a dataset or a model.

Value

the number of features.

Examples

```
a <- zinbModel()
nFeatures(a)
```

---

nParams — *Generic function that returns the total number of parameters of the model*

Description

Given an object that describes a model or a dataset, it returns total number of parameters of the model.

Usage

```
nParams(model)

## S4 method for signature 'ZinbModel'
nParams(model)
```

Arguments

model	an object that describes a dataset or a model.

Value

the total number of parameters of the model.

Functions

- `nParams(ZinbModel)`: returns the total number of parameters in the model.

Examples

```
a <- zinbModel()
nParams(a)
```

nSamples — *Generic function that returns the number of samples*

Description

Given an object that describes a model or a dataset, it returns the number of samples.

Usage

```
nSamples(x)
```

Arguments

x	an object that describes a dataset or a model.

Value

the number of samples.

Examples

```
a <- zinbModel()
nSamples(a)
```

orthogonalizeTraceNorm — *Orthogonalize matrices to minimize trace norm of their product*

Description

Given two matrices U and V that can be multiplied, this function finds two new matrices U2 and V2 such that their product is conserved (U*V = U2*V2) and such that a||U||^2 + b||V||^2 is minimized.

Usage

```
orthogonalizeTraceNorm(U, V, a = 1, b = 1)
```

Arguments

U	left matrix
V	right matrix
a	weight of the norm of U (default=1)
b	weight of the norm of V (default=1)

Value

A list with the two matrices that solve the problem in the slots U and V.

Examples

```
U <- matrix(rnorm(15),5,3)
V <- matrix(rnorm(12),3,4)
o <- orthogonalizeTraceNorm(U,V)
norm( U%*%V - o$U%*%o$V) # should be zero
sum(U^2)+sum(V^2)
sum(o$U^2)+sum(o$V^2) # should be smaller
```

---

penalty *Compute the penalty of a model*

Description

Given a statistical model with regularization parameters, compute the penalty.

Usage

```
penalty(model)

## S4 method for signature 'ZinbModel'
penalty(model)
```

Arguments

model	an object that describes a statistical model with regularization parameters.

Value

The penalty of the model.

Methods (by class)

- `penalty(ZinbModel)`: return the penalization.

Examples

```
m <- zinbModel(K=2)
penalty(m)
```

`pvalueAdjustment` *Perform independent filtering in differential expression analysis.*

Description

This function performs independent filtering to increase detection power in high throughput gene expression studies.

Usage

```
pvalueAdjustment(
  baseMean,
  filter,
  pValue,
  theta,
  alpha = 0.05,
  pAdjustMethod = "BH"
)
```

Arguments

`baseMean`	A vector of mean values.
`filter`	A vector of stage-one filter statistics.
`pValue`	A vector of unadjusted p-values, or a function which is able to compute this vector from the filtered portion of data, if data is supplied. The option to supply a function is useful when the value of the test statistic depends on which hypotheses are filtered out at stage one. (The limma t-statistic is an example.)
`theta`	A vector with one or more filtering fractions to consider. Actual cutoffs are then computed internally by applying quantile to the filter statistics contained in (or produced by) the filter argument.
`alpha`	A cutoff to which p-values, possibly adjusted for multiple testing, will be compared. Default is 0.05.
`pAdjustMethod`	The unadjusted p-values contained in (or produced by) test will be adjusted for multiple testing after filtering. Default is "BH".

Value

a list with pvalues, filtering threshold, theta, number of rejections, and alpha.

Note

This function is an adapted version of the `pvalueAdjustment` function that was originally written by Michael I. Love as part of the DESeq2 package. Koen Van den Berge adapted the function.

`solveRidgeRegression` *Solve ridge regression or logistic regression problems*

Description

This function solves a regression or logistic regression problem regularized by a L2 or weighted L2 penalty. Contrary to `lm.ridge` or `glmnet`, it works for any number of predictors.

Usage

```
solveRidgeRegression(
  x,
  y,
  beta = rep(0, NCOL(x)),
  epsilon = 1e-06,
  family = c("gaussian", "binomial"),
  offset = rep(0, NROW(x))
)
```

Arguments

`x`	a matrix of covariates, one sample per row, one covariate per column.
`y`	a vector of response (continuous for regression, 0/1 binary for logistic regression)
`beta`	an initial solution where optimization starts (null vector by default)
`epsilon`	a scalar or vector of regularization parameters (default `1e-6`)
`family`	a string to choose the type of regression (default `family="gaussian"`)
`offset`	a vector of offsets (default null vector)

Details

When `family="gaussian"`, we solve the ridge regression problem that finds the β that minimizes:

$$||y - x\beta||^2 + \epsilon||\beta||^2/2.$$

When `family="binomial"` we solve the ridge logistic regression problem

$$min \sum_i [-y_i(x\beta)_i + log(1 + exp(x\beta)_i))] + \epsilon||\beta||^2/2.$$

When `epsilon` is a vector of size equal to the size of `beta`, then the penalty is a weighted L2 norm $\sum_i \epsilon_i \beta_i^2/2$.

Value

A vector solution of the regression problem

toydata — *Toy dataset to check the model*

Description

Toy dataset to check the model

Format

A matrix of integers (counts) with 96 samples (rows) and 500 genes (columns).

zinb.loglik — *Log-likelihood of the zero-inflated negative binomial model*

Description

Given a vector of counts, this function computes the sum of the log-probabilities of the counts under a zero-inflated negative binomial (ZINB) model. For each count, the ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.

Usage

```
zinb.loglik(Y, mu, theta, logitPi)
```

Arguments

`Y`	the vector of counts
`mu`	the vector of mean parameters of the negative binomial
`theta`	the vector of dispersion parameters of the negative binomial, or a single scalar is also possible if the dispersion parameter is constant. Note that theta is sometimes called inverse dispersion parameter (and phi=1/theta is then called the dispersion parameter). We follow the convention that the variance of the NB variable with mean mu and dispersion theta is mu + mu^2/theta.
`logitPi`	the vector of logit of the probabilities of the zero component

Value

the log-likelihood of the model.

Examples

```
n <- 10
mu <- seq(10,50,length.out=n)
logitPi <- rnorm(10)
zeta <- rnorm(10)
Y <- rnbinom(n=n, size=exp(zeta), mu=mu)
zinb.loglik(Y, mu, exp(zeta), logitPi)
zinb.loglik(Y, mu, 1, logitPi)
```

---

zinb.loglik.dispersion

Log-likelihood of the zero-inflated negative binomial model, for a fixed dispersion parameter

---

Description

Given a unique dispersion parameter and a set of counts, together with a corresponding set of mean parameters and probabilities of zero components, this function computes the sum of the log-probabilities of the counts under the ZINB model. The dispersion parameter is provided to the function through zeta = log(theta), where theta is sometimes called the inverse dispersion parameter. The probabilities of the zero components are provided through their logit, in order to better numerical stability.

Usage

```
zinb.loglik.dispersion(zeta, Y, mu, logitPi)
```

Arguments

zeta	a scalar, the log of the inverse dispersion parameters of the negative binomial model
Y	a vector of counts
mu	a vector of mean parameters of the negative binomial
logitPi	a vector of logit of the probabilities of the zero component

Value

the log-likelihood of the model.

See Also

zinb.loglik.

Examples

```
mu <- seq(10,50,length.out=10)
logitPi <- rnorm(10)
zeta <- rnorm(10)
Y <- rnbinom(n=10, size=exp(zeta), mu=mu)
zinb.loglik.dispersion(zeta, Y, mu, logitPi)
```

---

zinb.loglik.dispersion.gradient

Derivative of the log-likelihood of the zero-inflated negative binomial model with respect to the log of the inverse dispersion parameter

---

Description

Derivative of the log-likelihood of the zero-inflated negative binomial model with respect to the log of the inverse dispersion parameter

Usage

```
zinb.loglik.dispersion.gradient(zeta, Y, mu, logitPi)
```

Arguments

zeta	the log of the inverse dispersion parameters of the negative binomial
Y	a vector of counts
mu	a vector of mean parameters of the negative binomial
logitPi	a vector of the logit of the probability of the zero component

Value

the gradient of the inverse dispersion parameters.

See Also

zinb.loglik, zinb.loglik.dispersion.

zinb.loglik.matrix *Log-likelihood of the zero-inflated negative binomial model for each entry in the matrix of counts*

Description

Given a matrix of counts, this function computes the log-probabilities of the counts under a zero-inflated negative binomial (ZINB) model. For each count, the ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.

Usage

```
zinb.loglik.matrix(model, x)
```

Arguments

model	the zinb model
x	the matrix of counts

Value

the matrix of log-likelihood of the model.

zinb.loglik.regression *Penalized log-likelihood of the ZINB regression model*

Description

This function computes the penalized log-likelihood of a ZINB regression model given a vector of counts.

Usage

```
zinb.loglik.regression(
  alpha,
  Y,
  A.mu = matrix(nrow = length(Y), ncol = 0),
  B.mu = matrix(nrow = length(Y), ncol = 0),
  C.mu = matrix(0, nrow = length(Y), ncol = 1),
  A.pi = matrix(nrow = length(Y), ncol = 0),
  B.pi = matrix(nrow = length(Y), ncol = 0),
  C.pi = matrix(0, nrow = length(Y), ncol = 1),
  C.theta = matrix(0, nrow = length(Y), ncol = 1),
  epsilon = 0
)
```

Arguments

`alpha`	the vectors of parameters c(a.mu, a.pi, b) concatenated
`Y`	the vector of counts
`A.mu`	matrix of the model (see Details, default=empty)
`B.mu`	matrix of the model (see Details, default=empty)
`C.mu`	matrix of the model (see Details, default=zero)
`A.pi`	matrix of the model (see Details, default=empty)
`B.pi`	matrix of the model (see Details, default=empty)
`C.pi`	matrix of the model (see Details, default=zero)
`C.theta`	matrix of the model (see Details, default=zero)
`epsilon`	regularization parameter. A vector of the same length as `alpha` if each coordinate of `alpha` has a specific regularization, or just a scalar is the regularization is the same for all coordinates of `alpha`. Default=`0`.

Details

The regression model is parametrized as follows:

$$log(\mu) = A_\mu * a_\mu + B_\mu * b + C_\mu$$

$$logit(\Pi) = A_\pi * a_\pi + B_\pi * b$$

$$log(\theta) = C_\theta$$

where μ, Π, θ are respectively the vector of mean parameters of the NB distribution, the vector of probabilities of the zero component, and the vector of inverse dispersion parameters. Note that the b vector is shared between the mean of the negative binomial and the probability of zero. The log-likelihood of a vector of parameters $\alpha = (a_\mu; a_\pi; b)$ is penalized by a regularization term $\epsilon||\alpha||^2/2$ is ϵ is a scalar, or $\sum_i \epsilon_i \alpha_i^2/2$ is ϵ is a vector of the same size as α to allow for differential regularization among the parameters.

Value

the penalized log-likelihood.

`zinb.loglik.regression.gradient`

Gradient of the penalized log-likelihood of the ZINB regression model

---

Description

This function computes the gradient of the penalized log-likelihood of a ZINB regression model given a vector of counts.

Usage

```
zinb.loglik.regression.gradient(
  alpha,
  Y,
  A.mu = matrix(nrow = length(Y), ncol = 0),
  B.mu = matrix(nrow = length(Y), ncol = 0),
  C.mu = matrix(0, nrow = length(Y), ncol = 1),
  A.pi = matrix(nrow = length(Y), ncol = 0),
  B.pi = matrix(nrow = length(Y), ncol = 0),
  C.pi = matrix(0, nrow = length(Y), ncol = 1),
  C.theta = matrix(0, nrow = length(Y), ncol = 1),
  epsilon = 0
)
```

Arguments

`alpha`	the vectors of parameters c(a.mu, a.pi, b) concatenated
`Y`	the vector of counts
`A.mu`	matrix of the model (see Details, default=empty)
`B.mu`	matrix of the model (see Details, default=empty)
`C.mu`	matrix of the model (see Details, default=zero)
`A.pi`	matrix of the model (see Details, default=empty)
`B.pi`	matrix of the model (see Details, default=empty)
`C.pi`	matrix of the model (see Details, default=zero)
`C.theta`	matrix of the model (see Details, default=zero)
`epsilon`	regularization parameter. A vector of the same length as `alpha` if each coordinate of `alpha` has a specific regularization, or just a scalar is the regularization is the same for all coordinates of `alpha`. Default=`0`.

Details

The regression model is described in `zinb.loglik.regression`.

Value

The gradient of the penalized log-likelihood.

See Also

`zinb.loglik.regression`

zinb.regression.parseModel

Parse ZINB regression model

Description

Given the parameters of a ZINB regression model, this function parses the model and computes the vector of log(mu), logit(pi), and the dimensions of the different components of the vector of parameters. See `zinb.loglik.regression` for details of the ZINB regression model and its parameters.

Usage

```
zinb.regression.parseModel(alpha, A.mu, B.mu, C.mu, A.pi, B.pi, C.pi)
```

Arguments

`alpha`	the vectors of parameters c(a.mu, a.pi, b) concatenated
`A.mu`	matrix of the model (see above, default=empty)
`B.mu`	matrix of the model (see above, default=empty)
`C.mu`	matrix of the model (see above, default=zero)
`A.pi`	matrix of the model (see above, default=empty)
`B.pi`	matrix of the model (see above, default=empty)
`C.pi`	matrix of the model (see above, default=zero)

Value

A list with slots `logMu`, `logitPi`, `dim.alpha` (a vector of length 3 with the dimension of each of the vectors `a.mu`, `a.pi` and `b` in `alpha`), and `start.alpha` (a vector of length 3 with the starting indices of the 3 vectors in `alpha`)

See Also

`zinb.loglik.regression`

zinbAIC

Compute the AIC or BIC of a model given some data

Description

Given a statistical model and some data, these functions compute the AIC or BIC of the model given the data, i.e., the AIC/BIC of the data under the model.

Usage

```
zinbAIC(model, x)

zinbBIC(model, x)

## S4 method for signature 'ZinbModel,matrix'
zinbAIC(model, x)

## S4 method for signature 'ZinbModel,matrix'
zinbBIC(model, x)
```

Arguments

model	an object that describes a statistical model.
x	an object that describes data.

Value

the AIC/BIC of the model.

Functions

- `zinbAIC(model = ZinbModel, x = matrix)`: returns the AIC of the ZINB model.
- `zinbBIC(model = ZinbModel, x = matrix)`: returns the BIC of the ZINB model.

Examples

```
se <- SummarizedExperiment(matrix(rpois(60, lambda=5), nrow=10, ncol=6),
    colData = data.frame(bio = gl(2, 3)))
m <- zinbFit(se, X=model.matrix(~bio, data=colData(se)),
    BPPARAM=BiocParallel::SerialParam())
zinbAIC(m, t(assay(se)))
zinbBIC(m, t(assay(se)))
```

---

zinbFit *Fit a ZINB regression model*

Description

Given an object with the data, it fits a ZINB model.

Usage

```
zinbFit(Y, ...)

## S4 method for signature 'SummarizedExperiment'
zinbFit(
  Y,
  X,
  V,
  K,
  which_assay,
  commondispersion = TRUE,
  zeroinflation = TRUE,
  verbose = FALSE,
  nb.repeat.initialize = 2,
  maxiter.optimize = 25,
  stop.epsilon.optimize = 1e-04,
  BPPARAM = BiocParallel::bpparam(),
  ...
)

## S4 method for signature 'matrix'
zinbFit(
  Y,
  X,
  V,
  K,
  commondispersion = TRUE,
  zeroinflation = TRUE,
  verbose = FALSE,
  nb.repeat.initialize = 2,
  maxiter.optimize = 25,
  stop.epsilon.optimize = 1e-04,
  BPPARAM = BiocParallel::bpparam(),
  ...
)

## S4 method for signature 'dgCMatrix'
zinbFit(Y, ...)
```

Arguments

Y	The data (genes in rows, samples in columns).
...	Additional parameters to describe the model, see zinbModel.
X	The design matrix containing sample-level covariates, one sample per row. If missing, X will contain only an intercept. If Y is a SummarizedExperiment object, X can be a formula using the variables in the colData slot of Y.
V	The design matrix containing gene-level covariates, one gene per row. If missing, V will contain only an intercept. If Y is a SummarizedExperiment object,

V can be a formula using the variables in the rowData slot of Y.

`K` integer. Number of latent factors.

`which_assay` numeric or character. Which assay of Y to use (only if Y is a SummarizedExperiment).

`commondispersion` Whether or not a single dispersion for all features is estimated (default TRUE).

`zeroinflation` Whether or not a ZINB model should be fitted. If FALSE, a negative binomial model is fitted instead.

`verbose` Print helpful messages.

`nb.repeat.initialize` Number of iterations for the initialization of beta_mu and gamma_mu.

`maxiter.optimize` maximum number of iterations for the optimization step (default 25).

`stop.epsilon.optimize` stopping criterion in the optimization step, when the relative gain in likelihood is below epsilon (default 0.0001).

`BPPARAM` object of class `bpparamClass` that specifies the back-end to be used for computations. See `bpparam` for details.

Details

By default, i.e., if no arguments other than `Y` are passed, the model is fitted with an intercept for the regression across-samples and one intercept for the regression across genes, both for mu and for pi.

This means that by default the model is fitted with `X_mu = X_pi = 1_n` and `V_mu = V_pi = 1_J`. If the user explicitly passes the design matrices, this behavior is overwritten, i.e., the user needs to explicitly include the intercept in the design matrices.

If Y is a Summarized experiment, the function uses the assay named "counts", if any, or the first assay.

Currently, if Y is a sparseMatrix, this calls the zinbFit method on as.matrix(Y)

Value

An object of class `ZinbModel` that has been fitted by penalized maximum likelihood on the data.

Methods (by class)

- `zinbFit(SummarizedExperiment)`: Y is a `SummarizedExperiment`.
- `zinbFit(matrix)`: Y is a matrix of counts (genes in rows).
- `zinbFit(dgCMatrix)`: Y is a sparse matrix of counts (genes in rows).

See Also

`model.matrix`.

Examples

```
se <- SummarizedExperiment(matrix(rpois(60, lambda=5), nrow=10, ncol=6),
                           colData = data.frame(bio = gl(2, 3)))

m <- zinbFit(se, X=model.matrix(~bio, data=colData(se)),
             BPPARAM=BiocParallel::SerialParam())
bio <- gl(2, 3)
m <- zinbFit(matrix(rpois(60, lambda=5), nrow=10, ncol=6),
             X=model.matrix(~bio), BPPARAM=BiocParallel::SerialParam())
```

`zinbInitialize` *Initialize the parameters of a ZINB regression model*

Description

The initialization performs quick optimization of the parameters with several simplifying assumptions compared to the true model: non-zero counts are models as log-Gaussian, zeros are modeled as dropouts. The dispersion parameter is not modified.

Usage

```
zinbInitialize(
  m,
  Y,
  nb.repeat = 2,
  it.max = 100,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

`m`	The model of class ZinbModel
`Y`	The matrix of counts.
`nb.repeat`	Number of iterations for the estimation of beta_mu and gamma_mu.
`it.max`	Maximum number of iterations in softImpute.
`BPPARAM`	object of class `bpparamClass` that specifies the back-end to be used for computations. See `bpparam` for details.

Value

An object of class ZinbModel similar to the one given as argument with modified parameters alpha_mu, alpha_pi, beta_mu, beta_pi, gamma_mu, gamma_pi, W.

Examples

```
Y <- matrix(rpois(60, lambda=2), 6, 10)
bio <- gl(2, 3)
time <- rnorm(6)
gc <- rnorm(10)
m <- zinbModel(Y, X=model.matrix(~bio + time), V=model.matrix(~gc),
             which_X_pi=1L, which_V_mu=1L, K=1)
m <- zinbInitialize(m, Y, BPPARAM=BiocParallel::SerialParam())
```

---

zinbModel *Initialize an object of class ZinbModel*

Description

Initialize an object of class ZinbModel

Usage

```
zinbModel(
  X,
  V,
  O_mu,
  O_pi,
  which_X_mu,
  which_X_pi,
  which_V_mu,
  which_V_pi,
  W,
  beta_mu,
  beta_pi,
  gamma_mu,
  gamma_pi,
  alpha_mu,
  alpha_pi,
  zeta,
  epsilon,
  epsilon_beta_mu,
  epsilon_gamma_mu,
  epsilon_beta_pi,
  epsilon_gamma_pi,
  epsilon_W,
  epsilon_alpha,
  epsilon_zeta,
  epsilon_min_logit,
  n,
  J,
  K
)
```

Arguments

X	matrix. The design matrix containing sample-level covariates, one sample per row.
V	matrix. The design matrix containing gene-level covariates, one gene per row.
O_mu	matrix. The offset matrix for mu.
O_pi	matrix. The offset matrix for pi.
which_X_mu	integer. Indeces of which columns of X to use in the regression of mu.
which_X_pi	integer. Indeces of which columns of X to use in the regression of pi.
which_V_mu	integer. Indeces of which columns of V to use in the regression of mu.
which_V_pi	integer. Indeces of which columns of V to use in the regression of pi.
W	matrix. The factors of sample-level latent factors.
beta_mu	matrix or NULL. The coefficients of X in the regression of mu.
beta_pi	matrix or NULL. The coefficients of X in the regression of pi.
gamma_mu	matrix or NULL. The coefficients of V in the regression of mu.
gamma_pi	matrix or NULL. The coefficients of V in the regression of pi.
alpha_mu	matrix or NULL. The coefficients of W in the regression of mu.
alpha_pi	matrix or NULL. The coefficients of W in the regression of pi.
zeta	numeric. A vector of log of inverse dispersion parameters.
epsilon	nonnegative scalar. Regularization parameter.
epsilon_beta_mu	nonnegative scalar. Regularization parameter for beta_mu.
epsilon_gamma_mu	nonnegative scalar. Regularization parameter for gamma_mu.
epsilon_beta_pi	nonnegative scalar. Regularization parameter for beta_pi.
epsilon_gamma_pi	nonnegative scalar. Regularization parameter for gamma_pi.
epsilon_W	nonnegative scalar. Regularization parameter for W.
epsilon_alpha	nonnegative scalar. Regularization parameter for alpha (both alpha_mu and alpha_pi).
epsilon_zeta	nonnegative scalar. Regularization parameter for zeta.
epsilon_min_logit	scalar. Minimum regularization parameter for parameters of the logit model, including the intercept.
n	integer. Number of samples.
J	integer. Number of genes.
K	integer. Number of latent factors.

Details

This is a wrapper around the new() function to create an instance of class `ZinbModel`. Rarely, the user will need to create a `ZinbModel` object from scratch, as tipically this is the result of `zinbFit`.

If any of `X`, `V`, `W` matrices are passed, `n`, `J`, and `K` are inferred. Alternatively, the user can specify one or more of `n`, `J`, and `K`.

The regularization parameters can be set by a unique parameter `epsilon` or specific values for the different regularization parameters can also be provided. If only `epsilon` is specified, the other parameters take the following values:

- epsilon_beta = epsilon/J
- epsilon_gamma = epsilon/n
- epsilon_W = epsilon/n
- epsilon_alpha = epsilon/J
- epsilon_zeta = epsilon

We empirically found that large values of `epsilon` provide a more stable estimation of `W`.

A call with no argument has the following default values: `n = 50`, `J = 100`, `K = 0`, `epsilon=J`.

Although it is possible to create new instances of the class by calling this function, this is not the most common way of creating `ZinbModel` objects. The main use of the class is within the `zinbFit` function.

Value

an object of class `ZinbModel`.

Examples

```
a <- zinbModel()
nSamples(a)
nFeatures(a)
nFactors(a)
nParams(a)
```

`ZinbModel-class` *Class ZinbModel*

Description

Objects of this class store all the values needed to work with a zero-inflated negative binomial (ZINB) model, as described in the vignette. They contain all information to fit a model by penalized maximum likelihood or simulate data from a model.

Usage

```
## S4 method for signature 'ZinbModel'
show(object)

## S4 method for signature 'ZinbModel'
nSamples(x)

## S4 method for signature 'ZinbModel'
nFeatures(x)

## S4 method for signature 'ZinbModel'
nFactors(x)

## S4 method for signature 'ZinbModel'
getX_mu(object, intercept = TRUE)

## S4 method for signature 'ZinbModel'
getX_pi(object, intercept = TRUE)

## S4 method for signature 'ZinbModel'
getV_mu(object, intercept = TRUE)

## S4 method for signature 'ZinbModel'
getV_pi(object, intercept = TRUE)

## S4 method for signature 'ZinbModel'
getLogMu(object)

## S4 method for signature 'ZinbModel'
getMu(object)

## S4 method for signature 'ZinbModel'
getLogitPi(object)

## S4 method for signature 'ZinbModel'
getPi(object)

## S4 method for signature 'ZinbModel'
getZeta(object)

## S4 method for signature 'ZinbModel'
getPhi(object)

## S4 method for signature 'ZinbModel'
getTheta(object)

## S4 method for signature 'ZinbModel'
getEpsilon_beta_mu(object)
```

```
## S4 method for signature 'ZinbModel'
getEpsilon_gamma_mu(object)

## S4 method for signature 'ZinbModel'
getEpsilon_beta_pi(object)

## S4 method for signature 'ZinbModel'
getEpsilon_gamma_pi(object)

## S4 method for signature 'ZinbModel'
getEpsilon_W(object)

## S4 method for signature 'ZinbModel'
getEpsilon_alpha(object)

## S4 method for signature 'ZinbModel'
getEpsilon_zeta(object)

## S4 method for signature 'ZinbModel'
getW(object)

## S4 method for signature 'ZinbModel'
getBeta_mu(object)

## S4 method for signature 'ZinbModel'
getBeta_pi(object)

## S4 method for signature 'ZinbModel'
getGamma_mu(object)

## S4 method for signature 'ZinbModel'
getGamma_pi(object)

## S4 method for signature 'ZinbModel'
getAlpha_mu(object)

## S4 method for signature 'ZinbModel'
getAlpha_pi(object)
```

Arguments

`object`	an object of class `ZinbModel`.
`x`	an object of class `ZinbModel`.
`intercept`	logical. Whether to return the intercept (ignored if the design matrix has no intercept). Default `TRUE`

Details

For the full description of the model see the model vignette. Internally, the slots are checked so that the matrices are of the appropriate dimensions: in particular, `X`, `O_mu`, `O_pi`, and `W` need to have `n` rows, `V` needs to have `J` rows, `zeta` must be of length `J`.

Value

`nSamples` returns the number of samples; `nFeatures` returns the number of features; `nFactors` returns the number of latent factors.

Methods (by generic)

- `show(ZinbModel)`: show useful info on the object.
- `nSamples(ZinbModel)`: returns the number of samples.
- `nFeatures(ZinbModel)`: returns the number of features.
- `nFactors(ZinbModel)`: returns the number of latent factors.
- `getX_mu(ZinbModel)`: returns the sample-level design matrix for mu.
- `getX_pi(ZinbModel)`: returns the sample-level design matrix for pi.
- `getV_mu(ZinbModel)`: returns the gene-level design matrix for mu.
- `getV_pi(ZinbModel)`: returns the sample-level design matrix for pi.
- `getLogMu(ZinbModel)`: returns the logarithm of the mean of the non-zero component.
- `getMu(ZinbModel)`: returns the mean of the non-zero component.
- `getLogitPi(ZinbModel)`: returns the logit-probability of zero.
- `getPi(ZinbModel)`: returns the probability of zero.
- `getZeta(ZinbModel)`: returns the log of the inverse of the dispersion parameter.
- `getPhi(ZinbModel)`: returns the dispersion parameter.
- `getTheta(ZinbModel)`: returns the inverse of the dispersion parameter.
- `getEpsilon_beta_mu(ZinbModel)`: returns the regularization parameters for `beta_mu`.
- `getEpsilon_gamma_mu(ZinbModel)`: returns the regularization parameters for `gamma_mu`.
- `getEpsilon_beta_pi(ZinbModel)`: returns the regularization parameters for `beta_pi`.
- `getEpsilon_gamma_pi(ZinbModel)`: returns the regularization parameters for `gamma_pi`.
- `getEpsilon_W(ZinbModel)`: returns the regularization parameters for `W`.
- `getEpsilon_alpha(ZinbModel)`: returns the regularization parameters for `alpha`.
- `getEpsilon_zeta(ZinbModel)`: returns the regularization parameters for `zeta`.
- `getW(ZinbModel)`: returns the matrix W of inferred sample-level covariates.
- `getBeta_mu(ZinbModel)`: returns the matrix beta_mu of inferred parameters.
- `getBeta_pi(ZinbModel)`: returns the matrix beta_pi of inferred parameters.
- `getGamma_mu(ZinbModel)`: returns the matrix gamma_mu of inferred parameters.
- `getGamma_pi(ZinbModel)`: returns the matrix gamma_pi of inferred parameters.
- `getAlpha_mu(ZinbModel)`: returns the matrix alpha_mu of inferred parameters.
- `getAlpha_pi(ZinbModel)`: returns the matrix alpha_pi of inferred parameters.

Slots

`X` matrix. The design matrix containing sample-level covariates, one sample per row.

`V` matrix. The design matrix containing gene-level covariates, one gene per row.

`O_mu` matrix. The offset matrix for mu.

`O_pi` matrix. The offset matrix for pi.

`which_X_mu` integer. Indeces of which columns of X to use in the regression of mu.

`which_V_mu` integer. Indeces of which columns of V to use in the regression of mu.

`which_X_pi` integer. Indeces of which columns of X to use in the regression of pi.

`which_V_pi` integer. Indeces of which columns of V to use in the regression of pi.

`X_mu_intercept` logical. TRUE if X_mu contains an intercept.

`X_pi_intercept` logical. TRUE if X_pi contains an intercept.

`V_mu_intercept` logical. TRUE if V_mu contains an intercept.

`V_pi_intercept` logical. TRUE if V_pi contains an intercept.

`W` matrix. The factors of sample-level latent factors.

`beta_mu` matrix or NULL. The coefficients of X in the regression of mu.

`gamma_mu` matrix or NULL. The coefficients of V in the regression of mu.

`alpha_mu` matrix or NULL. The coefficients of W in the regression of mu.

`beta_pi` matrix or NULL. The coefficients of X in the regression of pi.

`gamma_pi` matrix or NULL. The coefficients of V in the regression of pi.

`alpha_pi` matrix or NULL. The coefficients of W in the regression of pi.

`zeta` numeric. A vector of log of inverse dispersion parameters.

`epsilon_beta_mu` nonnegative scalar. Regularization parameter for beta_mu

`epsilon_gamma_mu` nonnegative scalar. Regularization parameter for gamma_mu

`epsilon_beta_pi` nonnegative scalar. Regularization parameter for beta_pi

`epsilon_gamma_pi` nonnegative scalar. Regularization parameter for gamma_pi

`epsilon_W` nonnegative scalar. Regularization parameter for W

`epsilon_alpha` nonnegative scalar. Regularization parameter for alpha (both alpha_mu and alpha_pi)

`epsilon_zeta` nonnegative scalar. Regularization parameter for zeta

`epsilon_min_logit` scalar. Minimum regularization parameter for parameters of the logit model, including the intercept.

zinbOptimize *Optimize the parameters of a ZINB regression model*

Description

The parameters of the model given as argument are optimized by penalized maximum likelihood on the count matrix given as argument. It is recommended to call zinb_initialize before this function to have good starting point for optimization, since the optimization problem is not convex and can only converge to a local minimum.

Usage

```
zinbOptimize(
  m,
  Y,
  commondispersion = TRUE,
  maxiter = 25,
  stop.epsilon = 1e-04,
  verbose = FALSE,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

m	The model of class ZinbModel
Y	The matrix of counts.
commondispersion	Whether the dispersion is the same for all features (default=TRUE)
maxiter	maximum number of iterations (default 25)
stop.epsilon	stopping criterion, when the relative gain in likelihood is below epsilon (default 0.0001)
verbose	print information (default FALSE)
BPPARAM	object of class bpparamClass that specifies the back-end to be used for computations. See bpparam for details.

Value

An object of class ZinbModel similar to the one given as argument with modified parameters alpha_mu, alpha_pi, beta_mu, beta_pi, gamma_mu, gamma_pi, W.

Examples

```
Y = matrix(10, 3, 5)
m = zinbModel(n=NROW(Y), J=NCOL(Y))
m = zinbInitialize(m, Y, BPPARAM=BiocParallel::SerialParam())
m = zinbOptimize(m, Y, BPPARAM=BiocParallel::SerialParam())
```

zinbOptimizeDispersion

Optimize the dispersion parameters of a ZINB regression model

Description

The dispersion parameters of the model are optimized by penalized maximum likelihood on the count matrix given as argument.

Usage

```
zinbOptimizeDispersion(
  J,
  mu,
  logitPi,
  epsilon,
  Y,
  commondispersion = TRUE,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

J	The number of genes.
mu	the matrix containing the mean of the negative binomial.
logitPi	the matrix containing the logit of the probability parameter of the zero-inflation part of the model.
epsilon	the regularization parameter.
Y	The matrix of counts.
commondispersion	Whether or not a single dispersion for all features is estimated (default TRUE)
BPPARAM	object of class `bpparamClass` that specifies the back-end to be used for computations. See `bpparam` for details.

Value

An object of class ZinbModel similar to the one given as argument with modified parameters zeta.

Examples

```
Y = matrix(10, 3, 5)
m = zinbModel(n=NROW(Y), J=NCOL(Y))
m = zinbInitialize(m, Y, BPPARAM=BiocParallel::SerialParam())
m = zinbOptimizeDispersion(NROW(Y), getMu(m), getLogitPi(m),
     getEpsilon_zeta(m), Y, BPPARAM=BiocParallel::SerialParam())
```

`zinbSim` *Simulate counts from a zero-inflated negative binomial model*

Description

Given an object that describes zero-inflated negative binomial distribution, simulate counts from the distribution.

Usage

```
zinbSim(object, seed, ...)

## S4 method for signature 'ZinbModel'
zinbSim(object, seed)
```

Arguments

`object`	an object that describes a matrix of zero-inflated negative binomial.
`seed`	an optional integer to specify how the random number generator should be initialized with a call to `set.seed`. If missing, the random generator state is not changed.
`...`	additional arguments.

Value

A list with the following elements.

- countsthe matrix with the simulated counts.
- dataNBthe data simulated from the negative binomial.
- dataDropoutsthe data simulated from the binomial process.
- zeroFractionthe fraction of zeros.

Methods (by class)

- `zinbSim(ZinbModel)`: simulate from a ZINB distribution.

Examples

```
a <- zinbModel(n=5, J=10)
zinbSim(a)
```

zinbsurf *Perform dimensionality reduction using a ZINB regression model for large datasets.*

Description

Given an object with the data, it performs dimensionality reduction using a ZINB regression model with gene and cell-level covariates on a random subset of the data. It then projects the remaining data onto the lower dimensional space.

Usage

```
zinbsurf(Y, ...)

## S4 method for signature 'SummarizedExperiment'
zinbsurf(
  Y,
  X,
  V,
  K,
  which_assay,
  which_genes,
  zeroinflation = TRUE,
  prop_fit = 0.1,
  BPPARAM = BiocParallel::bpparam(),
  verbose = FALSE,
  ...
)
```

Arguments

`Y`	The data (genes in rows, samples in columns). Currently implemented only for `SummarizedExperiment`.
`...`	Additional parameters to describe the model, see `zinbModel`.
`X`	The design matrix containing sample-level covariates, one sample per row. If missing, X will contain only an intercept. If Y is a SummarizedExperiment object, X can be a formula using the variables in the colData slot of Y.
`V`	The design matrix containing gene-level covariates, one gene per row. If missing, V will contain only an intercept. If Y is a SummarizedExperiment object, V can be a formula using the variables in the rowData slot of Y.
`K`	integer. Number of latent factors. Specify `K = 0` if only computing observational weights.
`which_assay`	numeric or character. Which assay of Y to use. If missing, if ‘assayNames(Y)‘ contains "counts" then that is used. Otherwise, the first assay is used.

which_genes	character. Which genes to use to estimate W (see details). Ignored if fitted_model is provided.
zeroinflation	Whether or not a ZINB model should be fitted. If FALSE, a negative binomial model is fitted instead.
prop_fit	numeric between 0 and 1. The proportion of cells to use for the zinbwave fit.
BPPARAM	object of class bpparamClass that specifies the back-end to be used for computations. See bpparam for details.
verbose	Print helpful messages.

Details

This function implements an approximate strategy, in which the full zinbwave model is fit only on a random subset of the data (controlled by the prop_fit parameter). The rest of the samples are subsequently projected onto the low-rank space. This strategy is much faster and uses less memory than the full zinbwave method. It is recommended with extremely large datasets.

By default zinbsurf uses all genes to estimate W. However, we recommend to use the top 1,000 most variable genes for this step. In general, a user can specify any custom set of genes to be used to estimate W, by specifying either a vector of gene names, or a single character string corresponding to a column of the rowData.

Value

An object of class SingleCellExperiment; the dimensionality reduced matrix is stored in the reducedDims slot.

Methods (by class)

- zinbsurf(SummarizedExperiment): Y is a SummarizedExperiment.

Examples

```
se <- SingleCellExperiment(assays = list(counts = matrix(rpois(60, lambda=5),
                                                         nrow=10, ncol=6)),
                           colData = data.frame(bio = gl(2, 3)))
colnames(se) <- paste0("sample", 1:6)
m <- zinbsurf(se, X="~bio", K = 1, prop_fit = .5, which_assay = 1,
              BPPARAM=BiocParallel::SerialParam())
```

---

zinbwave *Perform dimensionality reduction using a ZINB regression model with gene and cell-level covariates.*

Description

Given an object with the data, it performs dimensionality reduction using a ZINB regression model with gene and cell-level covariates.

Usage

```
zinbwave(Y, ...)

## S4 method for signature 'SummarizedExperiment'
zinbwave(
  Y,
  X,
  V,
  K = 2,
  fitted_model,
  which_assay,
  which_genes,
  commondispersion = TRUE,
  zeroinflation = TRUE,
  verbose = FALSE,
  nb.repeat.initialize = 2,
  maxiter.optimize = 25,
  stop.epsilon.optimize = 1e-04,
  BPPARAM = BiocParallel::bpparam(),
  normalizedValues = FALSE,
  residuals = FALSE,
  imputedValues = FALSE,
  observationalWeights = FALSE,
  ...
)
```

Arguments

`Y`	The data (genes in rows, samples in columns). Currently implemented only for `SummarizedExperiment`.
`...`	Additional parameters to describe the model, see `zinbModel`.
`X`	The design matrix containing sample-level covariates, one sample per row. If missing, X will contain only an intercept. If Y is a SummarizedExperiment object, X can be a formula using the variables in the colData slot of Y.
`V`	The design matrix containing gene-level covariates, one gene per row. If missing, V will contain only an intercept. If Y is a SummarizedExperiment object, V can be a formula using the variables in the rowData slot of Y.
`K`	integer. Number of latent factors. Specify `K = 0` if only computing observational weights.
`fitted_model`	a `ZinbModel` object.
`which_assay`	numeric or character. Which assay of Y to use. If missing, if ‘assayNames(Y)‘ contains "counts" then that is used. Otherwise, the first assay is used.
`which_genes`	character. Which genes to use to estimate W (see details). Ignored if `fitted_model` is provided.
`commondispersion`	Whether or not a single dispersion for all features is estimated (default TRUE).

`zeroinflation` Whether or not a ZINB model should be fitted. If FALSE, a negative binomial model is fitted instead.

`verbose` Print helpful messages.

`nb.repeat.initialize` Number of iterations for the initialization of beta_mu and gamma_mu.

`maxiter.optimize` maximum number of iterations for the optimization step (default 25).

`stop.epsilon.optimize` stopping criterion in the optimization step, when the relative gain in likelihood is below epsilon (default 0.0001).

`BPPARAM` object of class `bpparamClass` that specifies the back-end to be used for computations. See `bpparam` for details.

`normalizedValues` indicates wether or not you want to compute normalized values for the counts after adjusting for gene and cell-level covariates.

`residuals` indicates wether or not you want to compute the residuals of the ZINB model. Deviance residuals are computed.

`imputedValues` indicates wether or not you want to compute the imputed counts of the ZINB model.

`observationalWeights` indicates whether to compute the observational weights for differential expression (see vignette).

Details

For visualization (heatmaps, ...), please use the normalized values. It corresponds to the deviance residuals when the `W` is not included in the model but the gene and cell-level covariates are. As a results, when `W` is not included in the model, the deviance residuals should capture the biology. Note that we do not recommend to use the normalized values for any downstream analysis (such as clustering, or differential expression), but only for visualization.

If one has already fitted a model using `ZinbModel`, the object containing such model can be used as input of `zinbwave` to save the resulting W into a `SummarizedExperiment` and optionally compute residuals and normalized values, without the need for re-fitting the model.

By default `zinbwave` uses all genes to estimate `W`. However, we recommend to use the top 1,000 most variable genes for this step. In general, a user can specify any custom set of genes to be used to estimate `W`, by specifying either a vector of gene names, or a single character string corresponding to a column of the `rowData`.

Note that if both `which_genes` is specified and at least one among `observationalWeights`, `imputedValues`, `residuals`, and `normalizedValues` is `TRUE`, the model needs to be fit twice.

Value

An object of class `SingleCellExperiment`; the dimensionality reduced matrix is stored in the `reducedDims` slot and optionally normalized values and residuals are added in the list of assays.

Methods (by class)

- zinbwave(SummarizedExperiment): Y is a SummarizedExperiment.

Examples

```
se <- SingleCellExperiment(assays = list(counts = matrix(rpois(60, lambda=5),
                                                         nrow=10, ncol=6)),
                           colData = data.frame(bio = gl(2, 3)))

m <- zinbwave(se, X="~bio", BPPARAM=BiocParallel::SerialParam())
```

Index