Package ‘glmGamPoi’

March 18, 2024

Type Package
Title Fit a Gamma-Poisson Generalized Linear Model
Version 1.14.3
Description Fit linear models to overdispersed count data.
   The package can estimate the overdispersion and fit repeated models
   for matrix input. It is designed to handle large input datasets as they
   typically occur in single cell RNA-seq experiments.
License GPL-3
Encoding UTF-8
SystemRequirements C++11
Suggests testthat (>= 2.1.0), zoo, DESeq2, edgeR, limma, beachmat,
   MASS, statmod, ggplot2, bench, BiocParallel, knitr, rmarkdown,
   BiocStyle, TENxPBMCData, muscData, scran, Matrix, dplyr
LinkingTo Rcpp, RcppArmadillo, beachmat
Imports Rcpp, DelayedMatrixStats, matrixStats, MatrixGenerics,
   DelayedArray, HDF5Array, SummarizedExperiment,
   SingleCellExperiment, BiocGenerics, methods, stats, utils,
   splines, rlang, vctrs
Roxygen list(markdown = TRUE)
RoxygenNote 7.2.3
URL https://github.com/const-ae/glmGamPoi
BugReports https://github.com/const-ae/glmGamPoi/issues
biocViews Regression, RNASEq, Software, SingleCell
VignetteBuilder knitr
git_url https://git.bioconductor.org/packages/glmGamPoi
git_branch RELEASE_3_18
git_last_commit 5fd20c7
git_last_commit_date 2024-02-10
Repository Bioconductor 3.18
**as.list.glmGamPoi**

Convert glmGamPoi object to a list

**Description**

Convert glmGamPoi object to a list

**Usage**

```
## S3 method for class 'glmGamPoi'
as.list(x, ...)
```

**Arguments**

- `x` an object with class glmGamPoi
- `...` not used
estimate_betas_fisher_scoring

Value

The method returns a list with the following elements:

Beta a matrix with dimensions nrow(data) x n_coefficients where n_coefficients is based on the design argument. It contains the estimated coefficients for each gene.

overdispersions a vector with length nrow(data). The overdispersion parameter for each gene. It describes how much more the counts vary than one would expect according to the Poisson model.

Mu a matrix with the same dimensions as dim(data). If the calculation happened on disk, than Mu is a HDF5Matrix. It contains the estimated mean value for each gene and sample.

size_factors a vector with length ncol(data). The size factors are the inferred correction factors for different sizes of each sample. They are also sometimes called the exposure factor.

model_matrix a matrix with dimensions ncol(data) x n_coefficients. It is build based on the design argument.

description

Estimate the Betas for Fixed Dispersions

Usage

estimate_betas_fisher_scoring( Y, model_matrix, offset_matrix, dispersions, beta_mat_init, ridge_penalty, try_recovering_convergence_problems = TRUE )

Value

a list with two elements

- Beta a matrix with one column for each coefficient
- iterations the number of iterations
estimate_betas_group_wise

Estimate the Betas for Fixed Dispensions

Description

Estimate the Betas for Fixed Dispensions

Usage

estimate_betas_group_wise(
  Y,
  offset_matrix,
  dispersions,
  beta_group_init = NULL,
  beta_mat_init = NULL,
  groups,
  model_matrix
)

Value

a list with three elements

- Beta a matrix with one column per group and a row for each gene
- iterations the number of iterations from the Newton-Raphson method
- deviances the deviance for each gene (sum of the deviance per group)

estimate_betas_roughly

Make a quick first guess where reasonable beta would be

Description

Make a quick first guess where reasonable beta would be

Usage

estimate_betas_roughly(
  Y,
  model_matrix,
  offset_matrix,
  pseudo_count = 1,
  ridge_penalty = NULL
)
**estimate_betas_roughly_group_wise**

Make a quick first guess where reasonable beta would be for a set of groups

**Value**

a matrix with one column for each coefficient

---

**estimate_size_factors**  
*Estimate the Size Factors*

**Description**

Estimate the Size Factors

**Usage**

`estimate_size_factors(Y, method, verbose = FALSE)`

**Arguments**

- **Y**  
  any matrix-like object (`base::matrix()`, `DelayedArray`, `HDF5Matrix`, `Matrix::Matrix()`, etc.)

- **method**  
  one of `c("normed_sum", "deconvolution", "poscounts")`

**Value**

a vector with one size factor per column of `Y`
format_matrix  

*Helper to format a matrix nicely*

**Description**

Helper to format a matrix nicely

**Usage**

```r
format_matrix(matrix, digits = NULL)
```

**Value**

a string

---

glm_gp  

*Fit a Gamma-Poisson Generalized Linear Model*

**Description**

This function provides a simple to use interface to fit Gamma-Poisson generalized linear models. It works equally well for small scale (a single model) and large scale data (e.g. thousands of rows and columns, potentially stored on disk). The function automatically determines the appropriate size factors for each sample and efficiently finds the best overdispersion parameter for each gene.

**Usage**

```r
glm_gp(
  data,
  design = ~1,
  col_data = NULL,
  reference_level = NULL,
  offset = 0,
  size_factors = c("normed_sum", "deconvolution", "poscounts", "ratio"),
  overdispersion = TRUE,
  overdispersion_shrinkage = TRUE,
  ridge_penalty = 0,
  do_cox_reid_adjustment = TRUE,
  subsample = FALSE,
  on_disk = NULL,
  use_assay = NULL,
  verbose = FALSE
)
```
Arguments

data | any matrix-like object (e.g. matrix, DelayedArray, HDF5Matrix) or anything that can be cast to a SummarizedExperiment (e.g. MsnSet, eSet etc.) with one column per sample and row per gene.

design | a specification of the experimental design used to fit the Gamma-Poisson GLM. It can be a model.matrix() with one row for each sample and one column for each coefficient.

Alternatively, design can be a formula. The entries in the formula can refer to global objects, columns in the col_data parameter, or the colData(data) of data if it is a SummarizedExperiment.

The third option is that design is a vector where each element specifies to which condition a sample belongs.
Default: design = ~ 1, which means that all samples are treated as if they belong to the same condition. Note that this is the fastest option.

col_data | a dataframe with one row for each sample in data. Default: NULL.

reference_level | a single string that specifies which level is used as reference when the model matrix is created. The reference level becomes the intercept and all other coefficients are calculated with respect to the reference_level. Default: NULL.

offset | Constant offset in the model in addition to log(size_factors). It can either be a single number, a vector of length ncol(data) or a matrix with the same dimensions as dim(data). Note that if data is a DelayedArray or HDF5Matrix, offset must be as well. Default: 0.

size_factors | in large scale experiments, each sample is typically of different size (for example different sequencing depths). A size factor is an internal mechanism of GLMs to correct for this effect.

size_factors is either a numeric vector with positive entries that has the same lengths as columns in the data that specifies the size factors that are used. Or it can be a string that species the method that is used to estimate the size factors (one of c("normed_sum", "deconvolution", "poscounts", "ratio"). Note that "normed_sum" and "poscounts" are fairly simple methods and can lead to suboptimal results. For the best performance on data with many zeros, I recommend to use size_factors = "deconvolution" which calls scran::calculateSumFactors(). However, you need to separately install the scran package from Bioconductor for this method to work. For small datasets common for bulk RNA-seq experiments, I recommend to use size_factors = "ratio", which uses the same procedure as DESeq2 and edgeR. Also note that size_factors = 1 and size_factors = FALSE are equivalent. If only a single gene is given, no size factor is estimated (ie. size_factors = 1). Default: "normed_sum".

overdispersion | the simplest count model is the Poisson model. However, the Poisson model assumes that variance = mean. For many applications this is too rigid and the Gamma-Poisson allows a more flexible mean-variance relation (variance = mean + mean^2 * overdispersion).

overdispersion can either be

- a single boolean that indicates if an overdispersion is estimated for each gene.
• a numeric vector of length nrow(data) fixing the overdispersion to those values.

• the string "global" to indicate that one dispersion is fit across all genes.

Note that overdispersion = 0 and overdispersion = FALSE are equivalent and both reduce the Gamma-Poisson to the classical Poisson model. Default: TRUE.

overdispersion_shrinkage

the overdispersion can be difficult to estimate with few replicates. To improve the overdispersion estimates, we can share information across genes and shrink each individual overdispersion estimate towards a global overdispersion estimate. Empirical studies show however that the overdispersion varies based on the mean expression level (lower expression level => higher dispersion). If overdispersion_shrinkage = TRUE, a median trend of dispersion and expression level is fit and used to estimate the variances of a quasi Gamma Poisson model (Lund et al. 2012). Default: TRUE.

ridge_penalty

to avoid overfitting, we can penalize fits with large coefficient estimates. Instead of directly minimizing the deviance per gene \( \sum \text{deviance}(y_i, X_i b) \), we will minimize \( \sum \text{deviance}(y_i, X_i b) + N * \sum \text{penalty}*b_p^2 \).

ridge_penalty can be

• a scalar in which case all parameters except the intercept are penalized.

• a vector which has to have the same length as columns in the model matrix

• a matrix with the same number of columns as columns in the model matrix. This gives maximum flexibility for expert users and allows for full Tikhonov regularization.

Default: ridge_penalty = 0, which is internally replaced with a small positive number for numerical stability.

do_cox_reid_adjustment

the classical maximum likelihood estimator of the overdisperion is biased towards small values. McCarthy et al. (2012) showed that it is preferable to optimize the Cox-Reid adjusted profile likelihood.

do_cox_reid_adjustment can be either be TRUE or FALSE to indicate if the adjustment is added during the optimization of the overdispersion parameter. Default: TRUE.

subsample

the estimation of the overdispersion is the slowest step when fitting a Gamma-Poisson GLM. For datasets with many samples, the estimation can be considerably sped up without loosing much precision by fitting the overdispersion only on a random subset of the samples. Default: FALSE which means that the data is not subsampled. If set to TRUE, at most 1,000 samples are considered. Otherwise the parameter just specifies the number of samples that are considered for each gene to estimate the overdispersion.

on_disk

a boolean that indicates if the dataset is loaded into memory or if it is kept on disk to reduce the memory usage. Processing in memory can be significantly faster than on disk. Default: NULL which means that the data is only processed in memory if data is an in-memory data structure.

use_assay

Specify which assay to use. Default: NULL, which means that if available ’counts’ are used. Otherwise an error is thrown except if there is only a single assay.
Verbose  a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: FALSE.

Details

The method follows the following steps:

1. The size factors are estimated.
   If size_factors = "normed_sum", the column-sum for each cell is calculated and the resulting size factors are normalized so that their geometric mean is 1. If size_factors = "poscounts", a slightly adapted version of the procedure proposed by Anders and Huber (2010) in equation (5) is used. To handle the large number of zeros the geometric means are calculated for $Y + 0.5$ and ignored during the calculation of the median. Columns with all zeros get a default size factor of 0.001. If size_factors = "deconvolution", the method scran::calculateSumFactors() is called. If size_factors = "ratio", the unmodified procedure from Anders and Huber (2010) in equation (5) is used.

2. The dispersion estimates are initialized based on the moments of each row of $Y$.

3. The coefficients of the model are estimated.
   If all samples belong to the same condition (i.e. design = ~ 1), the betas are estimated using a quick Newton-Raphson algorithm. This is similar to the behavior of edgeR. For more complex designs, the general Fisher-scoring algorithm is used. Here, the code is based on a fork of the internal function fitBeta() from DESeq2. It does however contain some modification to make the fit more robust and faster.

4. The mean for each gene and sample is calculate.
   Note that this step can be very IO intensive if data is or contains a DelayedArray.

5. The overdispersion is estimated.
   The classical method for estimating the overdispersion for each gene is to maximize the Gamma-Poisson log-likelihood by iterating over each count and summing the the corresponding log-likelihood. It is however, much more efficient for genes with many small counts to work on the contingency table of the counts. Originally, this approach had already been used by Anscombe (1950). In this package, I have implemented an extension of their method that can handle general offsets.
   See also overdispersion_mle().

6. The beta coefficients are estimated once more with the updated overdispersion estimates

7. The mean for each gene and sample is calculated again.

This method can handle not just in memory data, but also data stored on disk. This is essential for large scale datasets with thousands of samples, as they sometimes encountered in modern single-cell RNA-seq analysis. glmGamPoi relies on the DelayedArray and beachmat package to efficiently implement the access to the on-disk data.

Value

The method returns a list with the following elements:

Beta a matrix with dimensions nrow(data) x n_coefficients where n_coefficients is based on the design argument. It contains the estimated coefficients for each gene.
overdispersions a vector with length nrow(data). The overdispersion parameter for each gene. It describes how much more the counts vary than one would expect according to the Poisson model.

overdispersion_shrinkage_list a list with additional information from the quasi-likelihood shrinkage. For details see overdispersion_shrinkage().

deviances a vector with the deviance of the fit for each row. The deviance is a measure how well the data is fit by the model. A smaller deviance means a better fit.

Mu a matrix with the same dimensions as dim(data). If the calculation happened on disk, than Mu is a HDF5Matrix. It contains the estimated mean value for each gene and sample.

size_factors a vector with length ncol(data). The size factors are the inferred correction factors for different sizes of each sample. They are also sometimes called the exposure factor.

Offset a matrix with the same dimensions as dim(data). If the calculation happened on disk, than Offset is a HDF5Matrix. It contains the log(size_factors) + offset from the function call.

data a SummarizedExperiment that contains the input counts and the col_data

model_matrix a matrix with dimensions ncol(data) x n_coefficients. It is build based on the design argument.

design_formula the formula that used to fit the model, or NULL otherwise

ridge_penalty a vector with the specification of the ridge penalty.

References

See Also

overdispersion_mle() and overdispersion_shrinkage() for the internal functions that do the work. For differential expression analysis, see test_de().

Examples

```r
set.seed(1)
# The simplest example
y <- rnbinom(n = 10, mu = 3, size = 1/2.4)
c(glm_gp(y, size_factors = FALSE))

# Fitting a whole matrix
model_matrix <- cbind(1, rnorm(5))
true_Beta <- cbind(rnorm(n = 30), rnorm(n = 30, mean = 3))
sf <- exp(rnorm(n = 5, mean = 0.7))
model_matrix
Y <- matrix(rnbinom(n = 30 * 5, mu = sf * exp(true_Beta %*% t(model_matrix)), size = 1/2.4),
nrow = 30, ncol = 5)
fit <- glm_gp(Y, design = model_matrix, size_factors = sf, verbose = TRUE)
summary(fit)

# Fitting a model with covariates
data <- data.frame(fav_food = sample(c("apple", "banana", "cherry"), size = 50, replace = TRUE),
city = sample(c("heidelberg", "paris", "new york"), size = 50, replace = TRUE),
age = rnorm(n = 50, mean = 40, sd = 15))
Y <- matrix(rnbinom(n = 100 * 50, mu = 3, size = 1/3.1), nrow = 100, ncol = 50)
fit <- glm_gp(Y, design = ~ fav_food + city + age, col_data = data)
summary(fit)

# Specify 'ridge_penalty' to penalize extreme Beta coefficients
fit_reg <- glm_gp(Y, design = ~ fav_food + city + age, col_data = data, ridge_penalty = 1.5)
summary(fit_reg)
```

---

### glm_gp_impl

**Internal Function to Fit a Gamma-Poisson GLM**

**Description**

Internal Function to Fit a Gamma-Poisson GLM

**Usage**

```r
glm_gp_impl(
  Y,
  model_matrix,
  offset = 0,
```
size_factors = c("normed_sum", "deconvolution", "poscounts", "ratio"),
overdispersion = TRUE,
overdispersion_shrinkage = TRUE,
ridge_penalty = 0,
do_cox_reid_adjustment = TRUE,
subsample = FALSE,
verbose = FALSE
)

Arguments

Y
any matrix-like object (e.g. matrix(), DelayedArray(), HDF5Matrix()) with
one column per sample and row per gene.

model_matrix
a numeric matrix that specifies the experimental design. It can be produced
using stats::model.matrix(). Default: NULL

offset
Constant offset in the model in addition to log(size_factors). It can either
be a single number, a vector of length ncol(data) or a matrix with the same
dimensions as dim(data). Note that if data is a DelayedArray or HDF5Matrix,
offset must be as well. Default: 0.

size_factors
in large scale experiments, each sample is typically of different size (for example
different sequencing depths). A size factor is an internal mechanism of GLMs
to correct for this effect.

size_factors is either a numeric vector with positive entries that has the same
lengths as columns in the data that specifies the size factors that are used. Or
it can be a string that species the method that is used to estimate the size fac-
tors (one of c("normed_sum", "deconvolution", "poscounts", "ratio").
Note that "normed_sum" and "poscounts" are fairly simple methods and can
lead to suboptimal results. For the best performance on data with many zeros, I
recommend to use size_factors = "deconvolution" which calls scran::calculateSumFactors().
However, you need to separately install the scran package from Bioconduc-
tor for this method to work. For small datasets common for bulk RNA-seq
experiments, I recommend to use size_factors = "ratio", which uses the
same procedure as DESeq2 and edgeR. Also note that size_factors = 1 and
size_factors = FALSE are equivalent. If only a single gene is given, no size
factor is estimated (ie. size_factors = 1). Default: "normed_sum".

overdispersion
the simplest count model is the Poisson model. However, the Poisson model
assumes that variance = mean. For many applications this is too rigid and the
Gamma-Poisson allows a more flexible mean-variance relation (variance =
mean + mean^2 * overdispersion).

overdispersion can either be
• a single boolean that indicates if an overdispersion is estimated for each
gene.
• a numeric vector of length nrow(data) fixing the overdispersion to those
values.
• the string "global" to indicate that one dispersion is fit across all genes.

Note that overdispersion = 0 and overdispersion = FALSE are equivalent
and both reduce the Gamma-Poisson to the classical Poisson model. Default:
The overdispersion can be difficult to estimate with few replicates. To improve the overdispersion estimates, we can share information across genes and shrink each individual overdispersion estimate towards a global overdispersion estimate. Empirical studies show however that the overdispersion varies based on the mean expression level (lower expression level => higher dispersion). If `overdispersion_shrinkage = TRUE`, a median trend of dispersion and expression level is fit and used to estimate the variances of a quasi Gamma Poisson model (Lund et al. 2012). Default: TRUE.

To avoid overfitting, we can penalize fits with large coefficient estimates. Instead of directly minimizing the deviance per gene ($\text{Sumdev}(y_i, X_i b)$), we will minimize $\text{Sumdev}(y_i, X_i b) + \text{N} \times \text{Sum}(\text{penalty}_p \times b_p)^2$.

`ridge_penalty` can be
- a scalar in which case all parameters except the intercept are penalized.
- a vector which has to have the same length as columns in the model matrix
- a matrix with the same number of columns as columns in the model matrix. This gives maximum flexibility for expert users and allows for full Tikhonov regularization.

Default: `ridge_penalty = 0`, which is internally replaced with a small positive number for numerical stability.

The classical maximum likelihood estimator of the overdispersion is biased towards small values. McCarthy et al. (2012) showed that it is preferable to optimize the Cox-Reid adjusted profile likelihood. `do_cox_reid_adjustment` can be either be TRUE or FALSE to indicate if the adjustment is added during the optimization of the overdispersion parameter. Default: TRUE.

The estimation of the overdispersion is the slowest step when fitting a Gamma-Poisson GLM. For datasets with many samples, the estimation can be considerably sped up without losing much precision by fitting the overdispersion only on a random subset of the samples. Default: FALSE which means that the data is not subsampled. If set to TRUE, at most 1,000 samples are considered. Otherwise the parameter just specifies the number of samples that are considered for each gene to estimate the overdispersion.

`verbose` a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: FALSE.

Value

a list with four elements
- `Beta` the coefficient matrix
- `overdispersion` the vector with the estimated overdispersions
- `Mu` a matrix with the corresponding means for each gene and sample
- `size_factors` a vector with the size factor for each sample
- `ridge_penalty` a vector with the ridge penalty
loc_median_fit

Estimate local median fit

Description

This function fits y based on x through a (weighted) median using the npoints/2 neighborhood.

Usage

loc_median_fit(
  x, y,
  fraction = 0.1,
  npoints = max(1, round(length(x) * fraction)),
  weighted = TRUE,
  ignore_zeros = FALSE
)

Arguments

x, y the x and y coordinates of the points.

fraction, npoints the fraction / number of the points that are considered for each fit. npoints is the argument that is used in the end it is at least one. Default: fraction = 0.1 and npoints = length(x) * fraction.

weighted a boolean that indicates if a weighted median is calculated.

ignore_zeros should the zeros be excluded from the fit

Details

This function is low-level implementation detail and should usually not be called by the user.

See Also

locfit: a package dedicated to local regression.

Examples

x <- runif(n = 1000, max = 4)
y <- rpois(n = 1000, lambda = x * 10)
plot(x, y)
fit <- loc_median_fit(x, y, fraction = 0.1)
points(x, fit, col = "red")

See Also

glm_gp() and overdispersion_mle()
overdispersion_mle

Estimate the Overdispersion for a Vector of Counts

Description
Estimate the Overdispersion for a Vector of Counts

Usage

overdispersion_mle(
  y,
  mean,  
  model_matrix = NULL,
  do_cox_reid_adjustment = is.null(model_matrix),
  global_estimate = FALSE,
  subsample = FALSE,
  max_iter = 200,
  verbose = FALSE
)

Arguments

y
a numeric or integer vector or matrix with the counts for which the overdispersion is estimated

mean
a numeric vector of either length 1 or length(y) or if y is a matrix, a matrix with the same dimensions. Contains the predicted value for that sample. If missing: mean(y) / rowMeans(y)

model_matrix
a numeric matrix that specifies the experimental design. It can be produced using stats::model.matrix(). Default: NULL

do_cox_reid_adjustment
the classical maximum likelihood estimator of the overdispersion is biased towards small values. McCarthy et al. (2012) showed that it is preferable to optimize the Cox-Reid adjusted profile likelihood.
do_cox_reid_adjustment can be either be TRUE or FALSE to indicate if the adjustment is added during the optimization of the overdispersion parameter. Default: TRUE if a model matrix is provided, otherwise FALSE

global_estimate
flag to decide if a single overdispersion for a whole matrix is calculated instead of one estimate per row. This parameter has no affect if y is a vector. Default: FALSE

subsample
the estimation of the overdispersion is the slowest step when fitting a Gamma-Poisson GLM. For datasets with many samples, the estimation can be considerably sped up without losing much precision by fitting the overdispersion only on a random subset of the samples. Default: FALSE which means that the data is not subsampled. If set to TRUE, at most 1,000 samples are considered. Otherwise the parameter just specifies the number of samples that are considered for each gene to estimate the overdispersion.
overdispersion_mle

max_iter  the maximum number of iterations for each gene
verbose   a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: FALSE.

Details

The function is optimized to be fast on many small counts. To achieve this, the frequency table of the counts is calculated and used to avoid repetitive calculations. If there are probably many unique counts the optimization is skipped. Currently the heuristic is to skip if more than half of the counts are expected to be unique. The estimation is based on the largest observed count in y.

An earlier version of this package (< 1.1.1) used a separate set of functions for the case of many small counts based on a paper by Bandara et al. (2019). However, this didn’t bring a sufficient performance increase and meant an additional maintenance burden.

Value

The function returns a list with the following elements:

estimate the numerical estimate of the overdispersion.
iterations the number of iterations it took to calculate the result.
message additional information about the fitting process.

See Also

glm_gp()

Examples

set.seed(1)
# true overdispersion = 2.4
y <- rnbinom(n = 10, mu = 3, size = 1/2.4)
# estimate = 1.7
overdispersion_mle(y)

# true overdispersion = 0
y <- rpois(n = 10, lambda = 3)
# estimate = 0
overdispersion_mle(y)
# with different mu, overdispersion estimate changes
overdispersion_mle(y, mean = 15)
# Cox-Reid adjustment changes the result
overdispersion_mle(y, mean = 15, do_cox_reid_adjustment = FALSE)

# Many very small counts, true overdispersion = 50
y <- rnbinom(n = 1000, mu = 0.01, size = 1/50)
summary(y)
# estimate = 31
overdispersion_mle(y, do_cox_reid_adjustment = TRUE)
# Function can also handle matrix input
Y <- matrix(rnbinom(n = 10 * 3, mu = 4, size = 1/2.2), nrow = 10, ncol = 3)
Y
as.data.frame(overdispersion_mle(Y))

overdispersion_shrinkage

*Shrink the overdispersion estimates*

Description

Low-level function to shrink a set of overdispersion estimates following the quasi-likelihood and Empirical Bayesian framework.

Usage

```r
overdispersion_shrinkage(
  disp_est,
  gene_means,
  df,
  disp_trend = TRUE,
  ql_disp_trend = NULL,
  ..., 
  verbose = FALSE
)
```

Arguments

- `disp_est`: vector of overdispersion estimates
- `gene_means`: vector of average gene expression values. Used to fit `disp_trend` if that is `NULL`.
- `df`: degrees of freedom for estimating the Empirical Bayesian variance prior. Can be length 1 or same length as `disp_est` and `gene_means`.
- `disp_trend`: vector with the dispersion trend. If `NULL` or `TRUE` the dispersion trend is fitted using a (weighted) local median fit. Default: `TRUE`.
- `ql_disp_trend`: a logical to indicate if a second abundance trend using splines is fitted for the quasi-likelihood dispersions. Default: `NULL` which means that the extra fit is only done if enough observations are present.
- `...`: additional parameters for the `loc_median_fit()` function
- `verbose`: a boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: `FALSE`. 
Details

The function goes through the following steps

1. Fit trend between overdispersion MLE’s and the average gene expression. Per default it uses the `loc_median_fit()` function.
2. Convert the overdispersion MLE’s to quasi-likelihood dispersion estimates by fixing the trended dispersion as the "true" dispersion value: \( \text{disp}_\text{ql} = \frac{(1 + \mu \cdot \text{disp}_\text{mle})}{(1 + \mu \cdot \text{disp}_\text{trend})} \)

Value

the function returns a list with the following elements

- **dispersion_trend** the dispersion trend provided by `disp_trend` or the local median fit.
- **ql_disp_estimate** the quasi-likelihood dispersion estimates based on the dispersion trend, `disp_est`, and `gene_means`
- **ql_disp_trend** the `ql_disp_estimate` still might show a trend with respect to `gene_means`. If `ql_disp_trend = TRUE` a spline is used to remove this secondary trend. If `ql_disp_trend = FALSE` it corresponds directly to the dispersion prior
- **ql_disp_shrunken** the shrunken quasi-likelihood dispersion estimates. They are shrunken towards `ql_disp_trend`
- **ql_df0** the degrees of freedom of the empirical Bayesian shrinkage. They correspond to spread of the `ql_disp_estimate`'s

References


See Also

`limma::squeezeVar()`

Examples

```r
Y <- matrix(rnbinom(n = 300 * 4, mu = 6, size = 1/4.2), nrow = 30, ncol = 4)
disps <- sapply(seq_len(nrow(Y)), function(idx) {
  overdispersion_mle(Y[idx, ])$estimate
})
shrink_list <- overdispersion_shrinkage(disps, rowMeans(Y), df = ncol(Y) - 1,
  disp_trend = FALSE, ql_disp_trend = FALSE)
```
predict.glmGamPoi

plot(rowMeans(Y), shrink_list$ql_disp_estimate)
lines(sort(rowMeans(Y)), shrink_list$ql_disp_trend[order(rowMeans(Y))], col = "red")
points(rowMeans(Y), shrink_list$ql_disp_shrunken, col = "blue", pch = 16, cex = 0.5)

---

predict.glmGamPoi

Predict 'link' or 'response' values for Gamma-Poisson GLMs

Description

Predict mu (i.e., type = "response") or log(mu) (i.e., type = "link") from a 'glmGamPoi' fit (created by glm_gp(...)) with the corresponding estimate of the standard error. If newdata is NULL, mu is returned for the original input data.

Usage

## S3 method for class 'glmGamPoi'
predict(
  object,
  newdata = NULL,
  type = c("link", "response"),
  se.fit = FALSE,
  offset = mean(object$Offset),
  on_disk = NULL,
  verbose = FALSE,
  ...)

Arguments

object a glmGamPoi fit object (produced by glm_gp()).
newdata a specification of the new data for which the expression for each gene is predicted. newdata should be a

  data.frame if the original fit was specified with a formula, provide a data.frame with one column for each variable in the formula. For example, if glm_gp(se, design = ~ age + batch + treatment), then the data.frame needs a age, batch, and treatment column that contain the same data types as the original fit.

  vector if the original fit was specified using a vector, you need to again provide a vector with the same format.

  matrix if newdata is a matrix, it is applied directly as Mu <- exp(object$Beta %*% t(newdata) + object$offset_matrix). So make sure, that it is constructed correctly.

  NULL if newdata is NULL, the predicted values for the original input data are returned.
type: either 'link' or 'response'. The default is 'link', which returns the predicted values before the link function (exp()) is applied. Thus, the values can be positive and negative numbers. However, often the predicted values are easier to interpret after the link function is applied (i.e., type = "response"), because then the values are on the same scale as the original counts.

se.fit: boolean that indicates if in addition to the mean the standard error of the mean is returned.

offset: count models (in particular for sequencing experiments) usually have a sample specific size factor (offset = log(size factor)). It defines how big we expect the predicted results are. If newdata is NULL, the offset is ignored, because the predict() returns a result based on the pre-calculated object$Mu. If newdata is not NULL, by default the offset is mean(object$Offset), which puts the in the same size as the average sample.

on_disk: a boolean that indicates if the results are HDF5Matrix’s from the HDF5Array package. If newdata is NULL, on_disk is ignored. Otherwise, if on_disk = NULL, the result is calculated on disk depending if offset is stored on disk.

verbose: a boolean that indicates if information about the individual steps are printed while predicting. Default: FALSE.

... currently ignored.

Details

For se.fit = TRUE, the function sticks very close to the behavior of stats::predict.glm() for fits from MASS::glm.nb().

Value

If se.fit == FALSE, a matrix with dimensions nrow(object$data) x nrow(newdata).
If se.fit == TRUE, a list with three entries

fit: the predicted values as a matrix with dimensions nrow(object$data) x nrow(newdata). This is what would be returned if se.fit == FALSE.

se.fit: the associated standard errors for each fit. Also a matrix with dimensions nrow(object$data) x nrow(newdata).

residual.scale: Currently fixed to 1. In the future, this might become the values from object$overdispersion_shrinkage_list$ql_disp_shrunken.

See Also

stats::predict.lm() and stats::predict.glm()

Examples

set.seed(1)
# The simplest example
y <- rnbinom(n = 10, mu = 3, size = 1/2.4)
fit <- glm_gp(y, size_factors = FALSE)
predict(fit, type = "response")
predict(fit, type = "link", se.fit = TRUE)
# Fitting a whole matrix

```r
model_matrix <- cbind(1, rnorm(5))
true_Beta <- cbind(rnorm(n = 30), rnorm(n = 30, mean = 3))
sf <- exp(rnorm(n = 5, mean = 0.7))
model_matrix
Y <- matrix(rnbinom(n = 30 * 5, mu = sf * exp(true_Beta %*% t(model_matrix)), size = 1/2.4),
nrow = 30, ncol = 5)
fit <- glm_gp(Y, design = model_matrix, size_factors = sf, verbose = TRUE)
head(predict(fit, type = "response"))
pred <- predict(fit, type = "link", se.fit = TRUE, verbose = TRUE)
head(pred$fit)
head(pred$se.fit)
```

# Fitting a model with covariates

```r
data <- data.frame(fav_food = sample(c("apple", "banana", "cherry"), size = 50, replace = TRUE),
                  city = sample(c("heidelberg", "paris", "new york"), size = 50, replace = TRUE),
                  age = rnorm(n = 50, mean = 40, sd = 15))
Y <- matrix(rnbinom(n = 4 * 50, mu = 3, size = 1/3.1), nrow = 4, ncol = 50)
fit <- glm_gp(Y, design = ~ fav_food + city + age, col_data = data)
predict(fit)[, 1:3]
```

```r
nd <- data.frame(fav_food = "banana", city = "paris", age = 29)
predict(fit, newdata = nd)
```

```r
nd <- data.frame(fav_food = "banana", city = "paris", age = 29:40)
predict(fit, newdata = nd, se.fit = TRUE, type = "response")
```

---

**print.glmGamPoi**  
Pretty print the result from `glm_gp()`

**Description**

Pretty print the result from `glm_gp()`

**Usage**

```r
## S3 method for class 'glmGamPoi'
print(x, ...)

## S3 method for class 'glmGamPoi'
format(x, ...)

## S3 method for class 'glmGamPoi'
summary(object, ...)```
## S3 method for class 'summary.glmGamPoi'
print(x, ...)

## S3 method for class 'summary.glmGamPoi'
format(x, ...)

### Arguments

- `x` the glmGamPoi object
- `...` additional parameters, currently ignored
- `object` the glmGamPoi object that is summarized

### Value

The `print()` methods return the object `x`. The `format()` method returns a string. The `summary()` method returns an object of class `summary.glmGamPoi`.

---

### pseudobulk

Create a `SingleCellExperiment` containing pseudo-bulk samples

### Description

Create a `SingleCellExperiment` containing pseudo-bulk samples

### Usage

```r
pseudobulk(
  data,
  group_by,
  ...,
  aggregation_functions = list(counts = "rowSums2", .default = "rowMeans2"),
  col_data = NULL,
  make_colnames = TRUE,
  verbose = TRUE
)
```

### Arguments

- `data` a `SingleCellExperiment` or an object of a related class
- `group_by` an unquoted expression that can refer to columns in the `colData()`. All observations with the same factor level are aggregated. The argument follows the same logic as `dplyr::group_by()`. The argument must wrapped using `vars()`.
- `...` named expressions that summarize columns in `colData()`. Each expression must produce a value of length 1. The arguments follow the same logic as `dplyr::summarize()`.
residuals.glmGamPoi

aggregation_functions

a named list with functions that are used to aggregate the assays in the data.

col_data

additional data with ncol(data) rows. The group_by and named arguments can refer to the columns of the col_data in addition to the columns in colData(data) (assuming data is a SummarizedExperiment).

make_colnames

a boolean that decides if the column names are the concatenated values of group_by. Default: TRUE.

verbose

a boolean that indicates if information about the process are printed. Default: TRUE.

Value

a SingleCellExperiment object

Examples

library(SingleCellExperiment)
data <- data.frame(sample = sample(c("samp1", "samp2", "samp3"), size = 50, replace = TRUE),
                   celltype = sample(c("T cells", "B cells", "Macrophages"), size = 50, replace = TRUE),
                   size = rnorm(n = 50, mean = 40, sd = 15))
Y <- matrix(rnbinom(n = 100 * 50, mu = 3, size = 1/3.1), nrow = 100, ncol = 50)
sce <- SingleCellExperiment(Y, colData = data)
aggr_sce <- pseudobulk(sce, group_by = vars(sample, celltype), size = mean(size))
aggr_sce
colData(aggr_sce)

residuals.glmGamPoi  Extract Residuals of Gamma Poisson Model

Description

Extract Residuals of Gamma Poisson Model

Usage

## S3 method for class 'glmGamPoi'
residuals(
  object,
  type = c("deviance", "pearson", "randomized_quantile", "working", "response"),
  ...
)

Arguments

object

a fit of type glmGamPoi. It is usually produced with a call to glm_gp().

type

the type of residual that is calculated. See details for more information. Default: "deviance".

... currently ignored.
solve_lm_for_A

Details

This method can calculate a range of different residuals:

**deviance** The deviance for the Gamma-Poisson model is

\[
dev = 2\times (1/\theta \log((1+m\theta)/(1+y\theta)) - y \log((m+y\theta)/(y+m\theta)))
\]

and the residual accordingly is

\[
res = \text{sign}(y-m)\sqrt{dev}.
\]

**pearson** The Pearson residual is \(res = (y - m)/\sqrt{m + m^2 \theta}\)

**randomized_quantile** The randomized quantile residual was originally developed by Dunn & Smyth, 1995. Please see that publication or `statmod::qresiduals()` for more information.

**working** The working residuals are \(res = (y - m)/m\).

**response** The response residuals are \(res = y - m\).

Value

a matrix with the same size as `fit$data`. If `fit$data` contains a `DelayedArray` than the result will be a `DelayedArray` as well.

See Also

`glm_gp()` and `stats::residuals.glm()`

---

**solve_lm_for_A**  
*Solve the equation \(Y = A \times B\) for \(A\) or \(B\)*

Description

Solve the equation \(Y = A \times B\) for \(A\) or \(B\)

Usage

```r
solve_lm_for_A(Y, B, w = NULL)
```

```r
solve_lm_for_B(Y, A, w = NULL)
```

Arguments

- **Y**  
  the left side of the equation

- **w**  
  a vector with weights. If NULL it is ignored, otherwise it must be of length 1 or have the same length as columns in \(Y\). Default: NULL

- **A, B**  
  the known matrix on the right side of the equation
Description

Conduct a quasi-likelihood ratio test for a Gamma-Poisson fit.

Usage

test_de(
  fit,
  contrast,
  reduced_design = NULL,
  full_design = fit$model_matrix,
  subset_to = NULL,
  pseudobulk_by = NULL,
  pval_adjust_method = "BH",
  sort_by = NULL,
  decreasing = FALSE,
  n_max = Inf,
  max_lfc = 10,
  verbose = FALSE
)

Arguments

fit object of class glmGamPoi. Usually the result of calling glm_gp(data, ...)
contrast The contrast to test. Can be a single column name (quoted or as a string) that is
removed from the full model matrix of fit. Or a complex contrast comparing
two or more columns: e.g. \( A - B, \) \( "A - 3 \times B", \) \( (A + B) / 2 - C \) etc. For com-
plicated experimental design that involved nested conditions, you specify the
condition level to compare using the cond() helper function.
Only one of contrast or reduced_design must be specified.
reduced_design a specification of the reduced design used as a comparison to see what how much
better fit describes the data. Analogous to the design parameter in glm_gp(),
it can be either a formula, a model_matrix(), or a vector.
Only one of contrast or reduced_design must be specified.
full_design option to specify an alternative full_design that can differ from fit$model_matrix.
Can be a formula or a matrix. Default: fit$model_matrix
subset_to a vector with the same length as ncol(fit$data) or an expression that evalu-
ates to such a vector. The expression can reference columns from colData(fit$data).
A typical use case in single cell analysis would be to subset to a specific cell type
(e.g. subset_to = cell_type == "T-cells"). Note that if this argument is set
a new the model for the full_design is re-fit.
Default: NULL which means that the data is not subset.
pseudobulk_by  
**DEPRECATED**, please use the `pseudobulk` function instead.

A vector with the same length as `ncol(fit$data)` that is used to split the columns into different groups (calls `split()`). `pseudobulk_by` can also be an expression that evaluates to a vector. The expression can reference columns from `colData(fit$data)`.

The counts are summed across the groups to create "pseudobulk" samples. This is typically used in single cell analysis if the cells come from different samples to get a proper estimate of the differences. This is particularly powerful in combination with the `subset_to` parameter to analyze differences between samples for subgroups of cells. Note that this does a fresh fit for both the full and the reduced design. Default: `NULL` which means that the data is not aggregated.

**pval_adjust_method**

One of the p-value adjustment methods from `p.adjust.methods`. Default: "BH".

**sort_by**

The name of the column or an expression used to sort the result. If `sort_by` is `NULL` the table is not sorted. Default: `NULL`.

**decreasing**

Boolean to decide if the result is sorted increasing or decreasing order. Default: `FALSE`.

**n_max**

The maximum number of rows to return. Default: `Inf` which means that all rows are returned.

**max_lfc**

Set the maximum absolute log fold change that is returned. Large log fold changes occur for lowly expressed genes because the ratio of two small numbers can be impractically large. For example, limiting the range of log fold changes can clarify the patterns in a volcano plot. Default: `10` which corresponds to a thousand-fold ($2^{10}$) increase in expression.

**verbose**

A boolean that indicates if information about the individual steps are printed while fitting the GLM. Default: `FALSE`.

**Details**

The `cond()` helper function simplifies the specification of a contrast for complex experimental designs. Instead of working out which combination of coefficients corresponds to a research question, you can simply specify the two conditions that you want to compare.

You can only call the `cond` function inside the `contrast` argument. The arguments are the selected factor levels for each covariate. To compare two conditions, simply subtract the two `cond` calls. Internally, the package calls `model.matrix` using the provided values and the original formula from the fit to produce a vector. Subtracting two of these vectors produces a contrast vector. Missing covariates are filled with the first factor level or zero for numerical covariates.

**Value**

A data.frame with the following columns

- **name** the rownames of the input data
- **pval** the p-values of the quasi-likelihood ratio test
- **adj_pval** the adjusted p-values returned from `p.adjust()`
- **f_statistic** the F-statistic: $F = (\text{Dev}_{\text{full}} - \text{Dev}_{\text{red}})/(df_1 \times \text{disp}_q - \text{shrunken})$
df1 the degrees of freedom of the test: `ncol(design) - ncol(reduced_design)`

df2 the degrees of freedom of the fit: `ncol(data) - ncol(design) + df_0`

lfc the log2-fold change. If the alternative model is specified by `reduced_design` will be NA.

References


See Also

glm_gp()

Examples

```r
# Make Data
Y <- matrix(rnbinom(n = 30 * 100, mu = 4, size = 0.3), nrow = 30, ncol = 100)
annot <- data.frame(mouse = sample(LETTERS[1:6], size = 100, replace = TRUE),
           celltype = sample(c("Tcell", "Bcell", "Macrophages"), size = 100, replace = TRUE),
           cont1 = rnorm(100), cont2 = rnorm(100, mean = 30))
annot$condition <- ifelse(annot$mouse %in% c("A", "B", "C"), "ctrl", "treated")
head(annot)
se <- SummarizedExperiment::SummarizedExperiment(Y, colData = annot)

# Fit model
fit <- glm_gp(se, design = ~ condition + celltype + cont1 + cont2)
# Test with reduced design
res <- test_de(fit, reduced_design = ~ celltype + cont1 + cont2)
head(res)
# Test with contrast argument, the results are identical
res2 <- test_de(fit, contrast = conditiontreated)
head(res2)
# Test with explicit specification of the conditions
# The results are still identical
res3 <- test_de(fit, contrast = cond(condition = "treated", celltype = "Bcell") -
               cond(condition = "ctrl", celltype = "Bcell"))
head(res3)

# The column names of fit$Beta are valid variables in the contrast argument
colnames(fit$Beta)
# You can also have more complex contrasts:
# the following compares cont1 vs cont2:
test_de(fit, cont1 - cont2, n_max = 4)
# You can also sort the output
# test_de(fit, cont1 - cont2, n_max = 4, sort_by = "pval")
test_de(fit, cont1 - cont2, n_max = 4, sort_by = - abs(f_statistic))
```
# If the data has multiple samples, it is a good
# idea to aggregate the cell counts by samples.
# This is called forming a "pseudobulk".
se_reduced <- pseudobulk(se, group_by = vars(mouse, condition, celltype),
  cont1 = mean(cont1), cont2 = min(cont2))
fit_reduced <- glm_gp(se_reduced, design = ~ condition + celltype)

# Description

This function implements Smyth’s 2004 variance shrinkage. It also supports covariates that are
fitted to log(s2) with natural splines. This is based on the 2012 Lund et al. quasi-likelihood paper.

# Usage

variance_prior(s2, df, covariate = NULL, abundance_trend = NULL)

# Arguments

- **s2**: vector of observed variances. Must not contain 0’s.
- **df**: vector or single number with the degrees of freedom
- **covariate**: a vector with the same length as s2. covariate is used to regress out the trend
  in s2. If covariate = NULL, it is ignored.
- **abundance_trend**: logical that decides if the additional abundance trend is fit to the data. If NULL
  the abundance trend is fitted if there are more than 10 observations and the
  covariate is not NULL. Default: NULL

# Value

a list with three elements:

- **variance0**: estimate of the scale of the inverse Chi-squared distribution. If covariate is NULL a single
  number, otherwise a vector of length(covariate)
- **df0**: estimate of the degrees of freedom of the inverse Chi-squared distribution
- **var_pos**: the shrunken variance estimates: a combination of s2 and variance0
See Also

limma::squeezeVar()

vars
Quote grouping variables

Description

Quote grouping variables

Usage

vars(...)

Arguments

... the quoted expression

See Also

ggplot2::vars, dplyr::vars
## Index

**internals**
- `solve_lm_for_A`, 24

**internal**
- `estimate_betas_fisher_scoring`, 3
- `estimate_betas_group_wise`, 4
- `estimate_betas_roughly`, 4
- `estimate_betas_roughly_group_wise`, 5
- `estimate_size_factors`, 5
- `format_matrix`, 6
- `glm_gp_impl`, 11
- `variance_prior`, 28

`as.list.glmGamPoi`, 2

`DelayedArray`, 7, 12

`estimate_betas_fisher_scoring`, 3
- `estimate_betas_group_wise`, 4
- `estimate_betas_roughly`, 4
- `estimate_betas_roughly_group_wise`, 5
- `estimate_size_factors`, 5

`format.glmGamPoi (print.glmGamPoi)`, 21
- `format.summary.glmGamPoi (print.glmGamPoi)`, 21
- `format_matrix`, 6

`glm_gp`, 6
- `glm_gp()`, 14, 16, 24, 27
- `glm_gp_impl`, 11

`HDF5Matrix`, 7, 12

`loc_median_fit`, 14

`matrix`, 7
- `model.matrix`, 26
- `model.matrix()`, 7
- `overdispersion_mle`, 15

`overdispersion_mle()`, 9, 11, 14
`overdispersion_shrinkage`, 17
- `overdispersion_shrinkage()`, 10, 11

`p.adjust()`, 26
- `p.adjust.methods`, 26
- `predict.glmGamPoi`, 19
- `print.glmGamPoi`, 21
- `print.summary.glmGamPoi (print.glmGamPoi)`, 21
- `pseudobulk`, 22, 26

`residuals.glmGamPoi`, 23

`solve_lm_for_A`, 24
- `solve_lm_for_B (solve_lm_for_A)`, 24
- `split()`, 26
- `statmod::qresiduals()`, 24
- `stats::predict.glm()`, 20
- `stats::predict.lm()`, 20
- `SummarizedExperiment`, 7
- `summary.glmGamPoi (print.glmGamPoi)`, 21

`test_de`, 25
- `test_de()`, 11

`variance_prior`, 28
- `vars`, 29