Package ‘BinQuasi’

July 27, 2018

Version 0.1-6

Date 2018-07-26

Title Analyzing Replicated ChIP Sequencing Data Using Quasi-Likelihood

Imports edgeR, mgcv, pracma, quadprog, Rsamtools, GenomicAlignments, GenomicRanges, IRanges, csaw(>= 1.12.0), SummarizedExperiment, BiocGenerics, S4Vectors, RMySQL

Suggests nleqslv, knitr, rmarkdown

Description Identify peaks in ChIP-seq data with biological replicates using a one-sided quasi-likelihood ratio test in quasi-Poisson or quasi-negative binomial models.

License GPL (>= 2)

NeedsCompilation yes

Maintainer Emily Goren <emily.goren@gmail.com>

URL https://github.com/emilygoren/BinQuasi

BugReports https://github.com/emilygoren/BinQuasi/issues

Author Emily Goren [aut, cre],
Steve Lund [aut] (The author of the QuasiSeq package, from which all functions were modified to produce this package.),
Long Qu [aut] (The author of the QuasiSeq package, from which all functions were modified to produce this package.),
Ian Marschner [aut] (The author of glm2::glm.fit2, which was modified slightly leading to glm.fit3 in this package.),
Daniel Gerhard [aut] (The author of mcprofile::orglm.fit, which was modified slightly and used under the same name in this package.),
R Core Team [aut] (The author of stats::glm.fit, which was modified slightly leading to glm.fit3 in this package.)

RoxygenNote 6.0.1

VignetteBuilder knitr

Repository CRAN

Date/Publication 2018-07-27 08:00:03 UTC

R topics documented:

BinQuasi *Analyzing Replicated ChIP Sequencing Data Using Quasi-Likelihood*

Description

Identify peaks in ChIP-seq data with biological replicates.

Details

Identify peaks in ChIP-seq data with biological replicates using a one-sided quasi-likelihood ratio test in quasi-Poisson or quasi-negative binomial models.

BQ *Call peaks in replicated ChIP-seq data using BinQuasi*

Description

Use the BinQuasi algorithm to call peaks using ChIP-seq data with biological replicates.

Usage

```
BQ(dir, ChIP.files, control.files, alpha = 0.05, bin.size = NULL,
  frag.length = NULL, minimum.count = 20, Model = "NegBin",
  print.progress = TRUE, method = "QLShrink", p.window.adjust = "BY",
  Dispersion = "Deviance", log.offset = NULL, NBdisp = "trend",
  bias.fold.tolerance = 1.1)
```

Arguments

`dir`	Directory where the sorted bam files (and their corresponding bam indices) are saved.
`ChIP.files`	File names (with file extensions) of the ChIP sample files in sorted bam format.
`control.files`	File names (with file extensions) of the control/input sample files in sorted bam format.
`alpha`	The desired significance threshold used to call peaks. Must be in (0, 0.5).
`bin.size`	Window size (constant across all samples) used to generate a partition for counts. If `NULL`, it will be estimated based on Shimazaki and Shinomoto (2007).
`frag.length`	Average length of the ChIP fragments in each sample provided. Reads are extended to this length in the 5'-to-3' direction. If `NULL`, cross correlation will be used to estimate the fragment
`minimum.count`	The count threshold used for filtering out windows with sparse counts. Any genomic window with a total count, across all samples, less than this value will be removed.
`Model`	Must be one of `"Poisson"` or `"NegBin"`, specifying use of a quasi-Poisson or quasi-negative binomial model, respectively.
`print.progress`	logical. If `TRUE`, updates are provided regarding which window (row number) is being analyzed. Updates occur frequently to start then eventually occur every 5000 windows.
`method`	Must be one of `"QL"`, `"QLShrink"`, or `"QLSpline"`, specifying which method of Lund, Nettleton, McCarthy and Smyth (2012) should be used to compute p-values.
`p.window.adjust`	FDR control method applied to the windows. Must be either `"BH"` or `"BY"` to specify the procedure of Benjamini-Hochberg or Benjamini-Yekutieli, respectively.
`Dispersion`	Must be one of `"Deviance"` or `"Pearson"`, specifying which type of estimator should be used for estimating the quasi-likelihood dispersion parameters.
`log.offset`	A vector of log-scale, additive factors used to adjust estimated log-scale means for differences in library sizes across samples. Commonly used offsets include `log.offset=log(colSums(counts))` and `log.offset=log(apply(counts[rowSums(counts)!=0,],` If `NULL`, the later offset is used.
`NBdisp`	Used only when `Model="NegBin"`. Must be one of `"trend"`, `"common"`, or a vector of non-negative real numbers with length equal to `nrow(counts)`. Specifying `NBdisp="trend"` or `NBdisp="common"` will use estimateGLMTrendedDisp or `estimateGLMCommonDisp`, respectively, from the package `edgeR` to estimate negative binomial dispersion parameters for each window. Estimates obtained from other sources can be used by entering `NBdisp` as a vector containing the negative binomial dispersion value to use for each window when fitting the quasi-likelihood model.
`bias.fold.tolerance`	A numerical value no smaller than 1. If the bias reduction of maximum likelihood estimates of (log) fold change is likely to result in a ratio of fold changes

greater than this value, then bias reduction will be performed on such windows. Setting `bias.fold.tolerance=Inf` will completely disable bias reduction; setting `bias.fold.tolerance=1` will always perform bias reduction. See `NBDev` or `PoisDev` for details.

Details

This function calls peaks in replicated ChIP-seq data using the BinQuasi algorithm of Goren, Liu, Wang, and Wang.

Value

A list containing:

`peaks`	Dataframe of the called peaks with columns for the start and end location, width, chromosome, p-value, and q-value computed using the Benjamini and Hochberg method.
`bin.size`	The window width used to create the counts dataframe.
`fragment.length`	Vector of the fragment lengths used to extend the reads in each sample.
`filter`	The count threshold used to create the counts dataframe. Windows with counts below this value were removed.

Author(s)

Emily Goren (`<emily.goren@gmail.com>`)

References

Goren, Liu, Wang and Wang (2018) "BinQuasi: a peak detection method for ChIP-sequencing data with biological replicates" *Bioinformatics*.

Shimazaki and Shinomoto (2007) "A method for selecting the bin size of a time histogram" *Neural computation*, **19**(6), 1503-27.

Ramachandran, Palidwor, Porter, and Perkins (2013) "MaSC: mappability-sensitive cross-correlation for estimating mean fragment length of single-end short-read sequencing data" *Bioinformatics* **29**(4), 444-50.

Benjamini and Hochberg (1995) "Controlling the false discovery rate: a practical and powerful approach to multiple testing" *Journal of the Royal Statistical Society Series B*, **57**: 289-300.

Benjamini and Yekutieli (2001) "The control of the false discovery rate in multiple testing under dependency" *Annals of Statistics*. **29**: 1165-1188.

Lund, Nettleton, McCarthy and Smyth (2012) "Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates" *SAGMB*, **11**(5).

Examples

```
## Not run:
# Fit a quasi-negative binomial model using all default settings.
fpath <- paste0(system.file(package = 'BinQuasi'), '/extdata/')
fpath
results <- BQ(fpath, ChIP.files = c('C1.bam', 'C2.bam'), control.files = c('I1.bam', 'I2.bam'))
head(results$peaks)

## End(Not run)
```

call.peaks *Call peaks from a list of window-level p-values*

Description

Call peaks from a list of p-values corresponding to window-level tests on a genomic partition of ChIP-seq counts. Used within the main peak calling function, BQ.

Usage

```
call.peaks(window.pvals, method = c("BY", "BH", "none"), start, end,
  chromosomes, alpha = 0.05)
```

Arguments

window.pvals	Vector of p-values, with each element corresponding to a window of a genomic partition. Typically obtained from the QL.fit and QL.results functions.
method	Correction method applied to window.pvals. Must be one of "BH", "BY", or "none" to specify Benjamini-Hochberg, Benjamini-Yekutieli, or no adjustment, respectively.
start	Vector of the genomic start locations corresponding to the supplied p-values.
end	Vector of the genomic end locations corresponding to the supplied p-values.
chromosomes	Vector of the chromosome names corresponding to the supplied p-values.
alpha	The desired significance threshold in (0, 0.5).

Details

After correcting for multiple testing using the adjustment specified by method, windows that are significant according to the threshold alpha are merged if adjacent and retained as candidate regions. Simes' procedure is used to combine the window-level p-values in each candidate region into a region-level p-value. The Benjamini-Hochberg procedure is applied to the resulting candidate regions and those that exceed the significance threshold alpha are returned as peaks.

Value

The called peaks as a dataframe with variables:

`start`	Genomic start locations of the called peaks.
`end`	Genomic end locations of the called peaks.
`width`	Width of the called peaks.
`chr`	Chromosomes of the called peaks.
`P.val`	p-values of the called peaks (aggregated from the windows comprising the peak using Simes' procedure).
`Q.val`	q-values of the called peaks (computing using the Benjamini-Hochberg procedure).

Author(s)

Emily Goren (<`emily.goren@gmail.com`>).

References

Benjamini and Hochberg (1995) "Controlling the false discovery rate: a practical and powerful approach to multiple testing" *Journal of the Royal Statistical Society Series B*, **57**: 289-300.

Benjamini and Yekutieli (2001) "The control of the false discovery rate in multiple testing under dependency" *Annals of Statistics*. **29**: 1165-1188.

Simes (1986) "An improved Bonferroni procedure for multiple tests of significance" *Biometrika*, **73**(3): 751-754.

Examples

```
# Example for a single chromosome.
start <- seq(1, 1e6, by = 200)
end <- start + 200 - 1
chromosomes <- rep('chr1', length(start))
p <- c(runif(length(start) - 10), rep(1e-12, 10))
called <- call.peaks(p, "BH", start, end, chromosomes)
called
```

`coef.glm` *Extract model coefficients*

Description

Extract model coefficients.

Usage

```
## S3 method for class 'glm'
coef(object, type = c("raw", "bias", "corrected"), ...)
```

Arguments

`object`	The fitted model.
`type`	Must be one of `raw`, `bias`, or `corrected`.
`...`	Additional arguments.

`count.table` *Create a matrix of ChIP-seq count data*

Description

Create a matrix of ChIP-seq count data from sorted bam files using a non-overlapping genomic partition. Used within the main peak calling function, `BQ`.

Usage

```
count.table(dir, ChIP.files, control.files, bin.size = NULL,
  frag.length = NULL, minimum.count = 20)
```

Arguments

`dir`	Directory where the sorted bam files (and their corresponding bam indices) are saved.
`ChIP.files`	File names (with file extensions) of the ChIP sample files in sorted bam format.
`control.files`	File names (with file extensions) of the input/control sample files in sorted bam format.
`bin.size`	Window size, constant across all samples, used to generate a non-overlapping partition for counts. If `NULL`, an estimate will be used (see details).
`frag.length`	Average length of the ChIP fragments in each sample provided. Reads are extended to this length from their 3' ends. If `NULL`, cross correlation will be used to estimate the fragment length of each sample (see details).
`minimum.count`	The count threshold used for filtering out windows with sparse counts. Any genomic window with counts less than this value across all samples will be removed.

Details

This function creates a count table of ChIP sequencing data (supplied as sorted bam files) using a non-overlapping partition across the genome.

The fragment length (if not provided) is estimated using the cross-correlation method of Ramachandran et al (2013). A fragment length is estimated for each sample, after removing duplicate reads, by taking the average over all chromosomes in the sample. Estimation is performed at 5 bp resolution and restricted to a minimum fragment length of 50 bp and maximum of 600 bp.

The bin size (if not provided) is selected using a procedure by Shimazaki and Shinomoto (2007) based on minimizing the mean-integrated squared error for a time-dependent Poisson point process.

This procedure is applied to each ChIP sample (at 5 bp resolution, restricted to a minimum of 50 bp and maximum of 1000 bp), and the minimum across all ChIP samples is returned as the bin size.

For a given sample and window, the count is determined as the number of fragments overlapping the window.

Value

A list containing:

`counts`	Data frame with rows corresponding to genomic windows and columns for the chromosomes, start and end locations, as well as a column for the counts of each sample.
`bin.size`	The bin size used to create the genomic partition.
`fragment.length`	Vector of the fragment lengths used to extend the reads in each sample.
`filter`	Count threshold used to create the counts data frame. Windows with counts summed across all samples that fall below this value were removed.

Author(s)

Emily Goren (<`emily.goren@gmail.com`>).

References

Shimazaki and Shinomoto (2007) "A method for selecting the bin size of a time histogram" *Neural computation*, **19**(6), 1503-27.

Ramachandran, Palidwor, Porter, and Perkins (2013) "MaSC: mappability-sensitive cross-correlation for estimating mean fragment length of single-end short-read sequencing data" *Bioinformatics* **29**(4), 444-50.

Examples

```
## Not run:
fpath <- paste0(system.file(package = 'BinQuasi'), '/extdata/')
d <- count.table(dir = fpath,
                 ChIP.files = c('C1.bam', 'C2.bam'),
                 control.files = c('I1.bam', 'I2.bam'),
                 bin.size = 60, frag.length = c(101, 300, 150, 10),
                 minimum.count = 20)
                 head(d$counts)

## End(Not run)
```

NBDev *Fit a negative binomial GLM for a given design matrix*

Description

A function called within `QL.fit` to fit a negative binomial GLM to each window for a given design matrix.

Usage

```
NBDev(counts, design, log.offset, nb.disp, print.progress = TRUE,
  bias.fold.tolerance = 1.1, chip.col.indicator)
```

Arguments

`counts`	A matrix of integers obtained by counting reads across a genomic partition. Each row contains observations from a single window of the genomic partition. Each column contains observations from a single sample (either ChIP or control/input).
`design`	A design matrix for the full model, including a column that indicates whether the observation is a ChIP sample (1) or control/input (0). The number of rows must be `ncol(counts)`. Means are modeled with a log link function.
`log.offset`	A vector of log-scale, additive factors used to adjust estimated log-scale means for differences in library sizes across samples. Commonly used offsets include `log.offset=log(colSums(counts))` and `log.offset=log(apply(counts[rowSums(counts)!=0,],` If `NULL`, the later offset is used.
`nb.disp`	Estimated negative binomial dispersion parameters obtained from either `estimateGLMTrendedDisp` or `estimateGLMCommonDisp` in package `edgeR`. These estimates are treated as known and are used to compute deviances.
`print.progress`	logical. If TRUE, the function will provide an update on what window (row number) is being analyzed. Updates occur frequently to start then eventually occur every 5000 windows.
`bias.fold.tolerance`	A numerical value no smaller than 1. If the bias reduction of maximum likelihood estimates of (log) fold change is likely to result in a ratio of fold changes greater than this value, then bias reduction will be performed on such windows. Setting `bias.fold.tolerance=Inf` will completely disable bias reduction; setting `bias.fold.tolerance=1` will always perform bias reduction (see details). If the constrained estimate differs from the unconstrained estimate, bias reduction is not performed.
`chip.col.indicator`	A binary vector of length `ncol(design.matrix)` that indicates which column of the full design matrix corresponds to the ChIP indicator.

Details

This functions fits, for each row of `counts`, a negative binomial log-linear model through the GLM framework with the over-dispersion parameter fixed.

Asymptotic biases of regression coefficients (i.e., log fold changes) are then estimated by a plug-in estimate [eqn. (15.4) of McCullagh and Nelder, 1989] from the last iteration of iteratively reweighted least squares (IWLS) procedure. The fitted response values are then compared with or without such a bias term. If the ratio of fitted response values are larger than `bias.fold.tolerance` for any observation and the unconstrained estimate equals the constrained estimate, then the bias-reduction (not bias-correction) procedure according to Firth (1993) and Kosmidis & Firth (2009) is applied to such rows of `counts`, by adjusting the score equation with a term based on the observed information. Such bias-reduced estimates are more advantageous than directly subtracting the estimated bias from the maximum likelihood estimates as the latter may not exist (e.g., when all counts in the control/input group are zero).

Value

A list containing:

`dev`	Vector containing the deviance for each window under a negative binomial model fit to design matrix specified by `design`. This vector is passed along within the `QL.fit` function.
`means`	Matrix of fitted mean values for each window.
`parms`	Matrix of estimated coefficients for each window. Note that these are given on the log scale. (i.e., intercept coefficient reports log(average count) and non-intercept coefficients report estimated (and bias reduced, when appropriate) log fold-changes.)
`dev.constrained`	Same as `dev`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.
`means.constrained`	Same as `means`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.
`parms.constrained`	Same as `parms`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.

Author(s)

Emily Goren (<`emily.goren@gmail.com`>) based on modifications of code by Steve Lund and Long Qu.

References

Firth (1993) "Bias reduction of maximum likelihood estimates" *Biometrika*, **80**, 27–38.

Kosmidis and Firth (2009) "Bias reduction in exponential family nonlinear models" *Biometrika*, **96**, 793–804.

Lund, Nettleton, McCarthy and Smyth (2012) "Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates" emphSAGMB, **11**(5).

McCullagh and Nelder (1989) "Generalized Linear Models", 2nd edition. London: Chapman and Hall.

`PoisDev` *Compute Poisson deviances for a given design matrix*

Description

A function called within `QL.fit` to compute Poisson deviances of each window for a given design matrix.

Usage

```
PoisDev(counts, design, log.offset, print.progress = TRUE,
  bias.fold.tolerance = 1.1, chip.col.indicator)
```

Arguments

`counts`	A matrix of integer counts obtained by counting reads across a genomic partition. Each row contains observations from a single window of the genomic partition. Each column contains observations from a single sample (either ChIP or control/input).
`design`	A design matrix for the full model, including a column that indicates whether the observation is a ChIP sample (1) or control/input (0). The number of rows must be `ncol(counts)`. Means are modeled with a log link function.
`log.offset`	A vector of log-scale, additive factors used to adjust estimated log-scale means for differences in library sizes across samples. Commonly used offsets include `log.offset=log(colSums(counts))` and `log.offset=log(apply(counts[rowSums(counts)!=0,],` If `NULL`, the later offset is used.
`print.progress`	logical. If TRUE, the function will provide an update on which window (row number) is being analyzed. Updates occur frequently to start then eventually occur every 5000 windows.
`bias.fold.tolerance`	A numerical value no smaller than 1. If the bias reduction of maximum likelihood estimates of (log) fold change is likely to result in a ratio of fold changes greater than this value, then bias reduction will be performed on such windows. Setting `bias.fold.tolerance=Inf` will completely disable bias reduction; setting `bias.fold.tolerance=1` will always perform bias reduction (see details). If the constrained estimate differs from the unconstrained estimate, bias reduction is not performed.
`chip.col.indicator`	A binary vector of length `ncol(design.matrix)` that indicates which column of the full design matrix corresponds to the ChIP indicator.

Details

This functions fits, for each row of `counts`, a Poisson log-linear model.

Asymptotic biases of regression coefficients (i.e., log fold changes) are then estimated by a plug-in estimate [eqn. (15.4) of McCullagh and Nelder, 1989] from the last iteration of iteratively reweighted least squares (IWLS) procedure. The fitted response values are then compared with or without such a bias term. If the ratio of fitted response values are larger than `bias.fold.tolerance` for any observation and the unconstrained estimate equals the constrained estimate, then the bias-reduction (not bias-correction) procedure according to Firth (1993) and Kosmidis & Firth (2009) is applied to such rows of `counts`, by adjusting the score equation with a term based on the observed information. Such bias-reduced estimates are more advantageous than directly subtracting the estimated bias from the maximum likelihood estimates as the latter may not exist (e.g., when all counts in one treatment group are zeros).

When the ChIP coefficient is constrained to be non-negative, quadratic programming is applied during IWLS using `solve.QP`. Note that if the constrained estimate of the regression coefficient differs from the unconstrained estimate for a given window, bias reduction is not performed for that window.

Value

A list containing:

`dev`	Vector containing the deviance for each window under a Poisson model fit to design matrix specified by `design`. This vector is passed along within the `QL.fit` function.
`means`	Matrix of fitted mean values for each window.
`parms`	Matrix of estimated coefficients for each window.
`dev.constrained`	Same as `dev`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.
`means.constrained`	Same as `means`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.
`parms.constrained`	Same as `parms`, subject to the constraint that the ChIP coefficient is non-negative. If fitting a reduced model with no ChIP coefficient, this will be `NA`.

Author(s)

Emily Goren (<`emily.goren@gmail.com`>) based on modifications of code by Steve Lund.

References

Firth (1993) "Bias reduction of maximum likelihood estimates" *Biometrika*, **80**, 27–38.

Kosmidis and Firth (2009) "Bias reduction in exponential family nonlinear models" *Biometrika*, **96**, 793–804.

Lund, Nettleton, McCarthy and Smyth (2012) "Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates" emphSAGMB, **11**(5).

McCullagh and Nelder (1989) "Generalized Linear Models", 2nd edition. London: Chapman and Hall.

`QL.fit` *Fit quasi-likelihood models to replicated ChIP-seq data partitioned into a count matrix*

Description

Fit constrained quasi-likelihood models to ChIP-seq data partitioned into a count matrix.

Usage

```
QL.fit(counts, design.matrix, chip.col.indicator, log.offset = NULL,
  Model = "NegBin", print.progress = TRUE, NBdisp = "trend",
  bias.fold.tolerance = 1.1, ...)
```

Arguments

`counts`	A matrix of integers obtained by counting reads across a genomic partition. Each row contains observations from a single window of the genomic partition. Each column contains observations from a single sample (either ChIP or control/input).
`design.matrix`	A design matrix for the full model, including a column that indicates whether the observation is a ChIP sample. The number of rows must be `ncol(counts)`. The number of columns must be at least two, usually an intercept and an indicator whether the sample is ChIP (1) or input/control (0). Means are modeled with a log link function.
`chip.col.indicator`	A binary vector of length `ncol(design.matrix)` that indicates which column of `design.matrix` corresponds to the ChIP indicator.
`log.offset`	A vector of log-scale, additive factors used to adjust estimated log-scale means for differences in library sizes across samples. Commonly used offsets include `log.offset=log(colSums(counts))` and `log.offset=log(apply(counts[rowSums(counts)!=0,],` If `NULL`, the later offset is used.
`Model`	Must be one of `"Poisson"` or `"NegBin"`, specifying use of a quasi-Poisson or quasi-negative binomial model, respectively.
`print.progress`	logical. If `TRUE`, updates are provided regarding which window (row number) is being analyzed. Updates occur frequently to start then eventually occur every 5000 windows.
`NBdisp`	Used only when `Model="NegBin"`. Must be one of `"trend"`, `"common"` or a vector of non-negative real numbers with length equal to `nrow(counts)`. Specifying `NBdisp="trend"` or `NBdisp="common"` will use estimateGLMTrendedDisp or `estimateGLMCommonDisp`, respectively, from the package `edgeR` to estimate negative binomial dispersion parameters for each window. Estimates obtained from other sources can be used by entering `NBdisp` as a vector containing

	the negative binomial dispersion value to use for each window when fitting the quasi-likelihood model.
`bias.fold.tolerance`	A numerical value no smaller than 1. If the bias reduction of maximum likelihood estimates of (log) fold change is likely to result in a ratio of fold changes greater than this value, then bias reduction will be performed on such windows. Setting `bias.fold.tolerance=Inf` will completely disable bias reduction; setting `bias.fold.tolerance=1` will always perform bias reduction. Estimates that are projected into the constrained space are not bias-reduced.
`...`	Arguments to be passed to the function `estimateGLMTrendedDisp` or `estimateGLMCommonDisp` from the package `edgeR`.

Details

A wrapper for `PoisDev` or `NBDev`, depending on whether quasi-Poisson or quasi-negative binomial models are requested. See the respective functions for details. Used within the main `BQ` peak calling function.

Value

A list containing:

`LRT`	Matrix providing unadjusted two-sided likelihood ratio test statistics. Each column contains statistics from a single hypothesis test that the ChIP coefficient is equal to zero versus not equal to zero, applied separately to each window.
`phi.hat.dev`	Vector providing unshrunken, deviance-based estimates of the quasi-dispersion parameter for each window.
`phi.hat.pearson`	Vector providing unshrunken, Pearson estimates of the quasi-dispersion parameter for each window.
`mn.cnt`	Vector of the average count for each window.
`den.df`	Denominator degrees of freedom. Equal to the number of samples minus the number of fitted parameters in the full model.
`num.df`	Vector of numerator degrees of freedom for each test, computed as the difference in the number of fitted parameters between the full and reduced models.
`Model`	Either "Poisson" or "NegBin", specifying which model (quasi-Poisson or quasi-negative binomial, respectively) was used.
`nb.disp`	Only appears when `Model="NegBin"`. Vector providing negative binomial dispersion parameter estimates used during fitting of quasi-negative binomial model for each window.
`fitted.values`	Matrix of fitted mean values without constraints.
`coefficients`	Matrix of estimated coefficients for each window. Note that these are given on the log scale. (i.e., intercept coefficients report log(average count) and non-intercept coefficients report estimated (and bias reduced, when appropriate) log fold-changes.)

`LRT.constrained`
: Same as `LRT`, but uses the constrained MLE in the full model. Each column contains statistics from a single hypothesis test that the ChIP coefficient is equal to zero versus greater than zero, applied separately to each window.

`phi.hat.dev.constrained`
: Same as `phi.hat.dev`, but subject to the constraint that the ChIP coefficient is non-negative.

`phi.hat.pearson.constrained`
: Same as `phi.hat.pearson`, but subject to the constraint that the ChIP coefficient is non-negative.

`fitted.values.constrained`
: Same as `fitted.values`, but subject to the constraint that the ChIP coefficient is non-negative.

`coefficients.constrained`
: Same as `coefficients`, but subject to the constraint that the ChIP coefficient is non-negative.

Author(s)

Emily Goren (<`emily.goren@gmail.com`>) based on modifications of code by Steve Lund.

References

Goren, Liu, Wang and Wang (2018) "BinQuasi: a peak detection method for ChIP-sequencing data with biological replicates" *Bioinformatics*.

Kosmidis and Firth (2009) "Bias reduction in exponential family nonlinear models" *Biometrika*, **96**, 793–804.

Lund, Nettleton, McCarthy and Smyth (2012) "Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates" *SAGMB*, **11**(5).

McCarthy, Chen and Smyth (2012) "Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation" *Nucleic Acids Res.* **40**(10), 4288–97.

Examples

```
set.seed(5)
####################################################
# Simulate data three replicates with one chromosome
####################################################
reps <- 3
chr.length <- 1e5
window.width <- 200
K <- chr.length / window.width
start <- seq(1, chr.length, by = window.width)
end <- start + window.width - 1
n.peaks <- 100 # No. of true peaks
peak.idx <- sample.int(K, n.peaks)
samples <- c(paste0('C', 1:reps), paste0('I', 1:reps))
# Set parameters
beta0 <- runif(K, log(10), log(100))
```

```
beta1 <- rep(0, K); beta1[peak.idx] <- log(5) / runif(n.peaks)^(1/5)
# Set means
mu.ChIP <- exp(beta0 + beta1)
mu.input <- exp(beta0)
# Negative binomial dispersion parameter
phi <- 1/rchisq(K, df = 5)
# Draw read counts using a negative binomial distribution
C <- lapply(1:reps, function(r) rpois(K, (mu.ChIP * rgamma(K, 1/phi))/(1/phi)))
I <- lapply(1:reps, function(r) rpois(K, (mu.input * rgamma(K, 1/phi))/(1/phi)))
counts <- do.call('cbind', append(C, I))
colnames(counts) <- samples
rownames(counts) <- start
head(counts)

####################################################
# Fit quasi-negative binomial model to each window.
####################################################
design.matrix  <- cbind(rep(1, reps*2), # Intercept
                        rep(c(1,0), each = reps)) # Indicates ChIP sample
chip.col.indicator <- c(0,1) # Second column of design matrix indicates ChIP sample
fit <- QL.fit(counts, design.matrix, chip.col.indicator,
              log.offset = rep(1, ncol(counts)), Model = 'NegBin')
# Look at fitted values
counts.fitted <- fit$fitted.values.constrained
head(round(counts.fitted, 2))
```

`QL.results` *Obtain p- and q-values using results from* `QL.fit`

Description

Obtain significance results for quasi-likelihood models fit to ChIP-seq data partitioned into counts.

Usage

```
QL.results(fit, Dispersion = "Deviance", one.sided = TRUE,
  spline.df = NULL, Plot = FALSE, padj = TRUE)
```

Arguments

`fit`	The list returned by the function `QL.fit`.
`Dispersion`	Must be one of `"Deviance"` or `"Pearson"`, specifying which type of estimator should be used for estimating the quasi-likelihood dispersion parameter.
`one.sided`	logical. If TRUE, a one-sided test for the ChIP coefficient is reported. Otherwise, if FALSE, a two-sided test is reported.
`spline.df`	Optional. User may specify the degrees of freedom to use when fitting a cubic spline to log-scale(estimated dispersion) versus the log(average count). Default uses cross-validation in `sreg` function to pick appropriate degrees of freedom.

`Plot`	logical. If TRUE, the estimated dispersion versus the average count are plotted on a log-scale with the corresponding cubic spline fit overlaid.
`padj`	logical. If TRUE, Benjamini & Hochberg's adjustment for multiple comparisons is applied.

Details

Obtain significance results from an object fitted using `QL.fit`. Used within the main peak calling function, `BQ`.

Value

A list containing:

`P.values`	List of matrices providing p-values for the QL, QLShrink and QLSpline methods, respectively. The i-th column of each element of `pvals` corresponds to the hypothesis test assigned in the i-th window.
`Q.values`	List of matrices providing q-values for the QL, QLShrink and QLSpline methods, respectively. The i-th column of each element of `qvals` corresponds to the hypothesis test assigned in the i-th window.
`F.stat`	List of matrices providing F-statistics for the QL, QLShrink and QLSpline methods, respectively. The i-th column of each element of `F.stat` corresponds to the hypothesis test assigned in the i-th window.
`d0`	Vector containing estimated additional denominator degrees of freedom gained from shrinking dispersion estimates in the QLShrink and QLSpline procedures, respectively.

Author(s)

Emily Goren (<`emily.goren@gmail.com`>) based on modifications of code by Steve Lund.

References

Goren, Liu, Wang and Wang (2018) "BinQuasi: a peak detection method for ChIP-sequencing data with biological replicates" *Bioinformatics*.

Benjamini and Hochberg (1995) "Controlling the false discovery rate: a practical and powerful approach to multiple testing" *Journal of the Royal Statistical Society Series B*, **57**: 289-300.

Lund, Nettleton, McCarthy and Smyth (2012) "Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates" *SAGMB*, **11**(5).

Examples

```
set.seed(5)
#####################################################
# Simulate data three replicates with one chromosome
#####################################################
reps <- 3
chr.length <- 1e5
window.width <- 200
```

```
 K <- chr.length / window.width
 start <- seq(1, chr.length, by = window.width)
 end <- start + window.width - 1
 n.peaks <- 100 # No. of true peaks
 peak.idx <- sample.int(K, n.peaks)
 samples <- c(paste0('C', 1:reps), paste0('I', 1:reps))
 # Set parameters
 beta0 <- runif(K, log(10), log(100))
 beta1 <- rep(0, K); beta1[peak.idx] <- log(5) / runif(n.peaks)^(1/5)
 # Set means
 mu.ChIP <- exp(beta0 + beta1)
 mu.input <- exp(beta0)
 # Negative binomial dispersion parameter
 phi <- 1/rchisq(K, df = 5)
 # Draw read counts using a negative binomial distribution
 C <- lapply(1:reps, function(r) rpois(K, (mu.ChIP * rgamma(K, 1/phi))/(1/phi)))
 I <- lapply(1:reps, function(r) rpois(K, (mu.input * rgamma(K, 1/phi))/(1/phi)))
 counts <- do.call('cbind', append(C, I))
 colnames(counts) <- samples
 rownames(counts) <- start

 ####################################################
 # Fit quasi-negative binomial model to each window.
 ####################################################
 design.matrix  <- cbind(rep(1, reps*2), # Intercept
                        rep(c(1,0), each = reps)) # Indicates ChIP sample
 chip.col.indicator <- c(0,1) # Second column of design matrix indicates ChIP sample
 fit <- QL.fit(counts, design.matrix, chip.col.indicator,
              log.offset = rep(1, ncol(counts)), Model = 'NegBin')
 window.results <- QL.results(fit)

# Number of significant windows.
sum(window.results$Q.values$QLShrink < 0.05)

# Compare significant windows to truth.
res <- as.numeric(window.results$Q.values$QLShrink < 0.05)
# Number of true positives
TP <- sum(res[peak.idx] == 1)
TP
# Number of false negatives
FN <- n.peaks - TP
FN
# Number of false positives
FP <- sum(res) - TP
FP
# Number of true negatives
TN <- (K - sum(res)) - FN
TN
```

Index