Package ‘denoiSeq’

January 31, 2018

Type Package

Title Differential Expression Analysis Using a Bottom-Up Model

Version 0.1.1

Description Given count data from two conditions, it determines which transcripts are differentially expressed across the two conditions using Bayesian inference of the parameters of a bottom-up model for PCR amplification. This model is developed in Ndifon Wilfred, Hilah Gal, Eric Shifrut, Rina Aharoni, Nissan Yissachar, Nir Waysbort, Shlomit Reich Zeliger, Ruth Arnon, and Nir Friedman (2012), <http://www.pnas.org/content/109/39/15865.full>, and results in a distribution for the counts that is a superposition of the binomial and negative binomial distribution.

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Imports methods, stats, utils

Suggests knitr, rmarkdown

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Gershom Buri [aut, cre],
Wilfred Ndifon [aut]

Maintainer Gershom Buri <buri@aims.edu.gh>

Repository CRAN

Date/Publication 2018-01-31 20:26:31 UTC

R topics documented:

`denoiseq` *Differential expression analysis using a bottom-up model*

Description

The denoiseq function perfoms default analysis by first normalising the counts and then estimating the model parameters using Bayesian inference. Size factors are estimated from count matrix and used for the normalisation. The Gibb's sampling algorithm is then used to sample from the joint posterior distribution of the model parameters.

Usage

```
denoiseq(RDobject, steps, tuningSteps = floor(steps/3))
```

Arguments

`RDobject`	A readsData object.
`steps`	An integer representing the number of iterations.
`tuningSteps`	An integer representing the number of iterations to be used for tuning the step sizes. Defaulted to a third of steps.

Details

The denoiSeq package is based on a bottom-up model for PCR sequencing developed by Ndifon et al. (2012). The model generates, in a bottom-up manner, a probability distribution for the final copy number of a gene, that is a superposition of the negative binomial and the binomial distributions. The derived distribution has three main parameters, i.e *N, p* and *f*, which represent the initial gene amount before amplification, the amplification efficiency and the dilution rate, respectively.

Bayesian inference is used to estimate the model parameters. The counts in each column are used to estimate the size factors (Anders and Huber, 2010) which are in turn used to normalise the counts. For an m by n matrix, inference aims at estimating the three sets of parameters, i.e p, f and N_i 's (2m in total because we are considering 2 conditions with the same *m* genes in each). denoiseq uses the rows in each condition to estimate parameter N_i for each gene in that condition, and uses the entire dataset, combined from both conditions, to estimate p and f.

For differential expression analysis, the primary parameters of interest are N_{iA} and N_{iB} (from conditions A and B respectively), for each gene i.

Value

The same readsData object but with a filled output slot. The output slot now contains 2 lists, i.e **samples** which contains posterior samples for each of the parameters N_i, p and f, and **stepsize** which contains the tuned step sizes.

Examples

```
#pre -filtering to remove lowly expressed genes
ERCC <- ERCC[rowSums(ERCC)>0, ]
RD <- new('readsData', counts = ERCC)
steps <- 30
#30 steps are used for illustration here. Atleast 5000 steps are adequate.
BI <- denoiseq(RD, steps)
```

ERCC *ERCC dataset*

Description

RNA-seq data from biological replicates of 3 cell lines. This dataset contains a mixture of spike-in synthetic oligonucleotides that are mixed into samples A and B at four mixing ratios: 1/2, 2/3, 1 and 4.

Usage

```
ERCC
```

Format

A matrix with 92 rows and 10 columns:

Conditions There are 5 columns for each of the conditions A and B.

Transcripts There are 92 distinct transcripts distinguishable by their names.

Source

https://bitbucket.org/soccin/seqc/src/ccd0502ef25422e83b3f208f50f8e252f62f17a3/data/?at=master

`getSamplesOf` *Get posterior samples of a parameter*

Description

Extracts posterior samples of individual parameters contained in the output slot of the readsData object returned by denoiSeq.

Usage

```
getSamplesOf(RDobject, parm, steps, condition = "A")
```

Arguments

`RDobject`	A readsData object with a filled output slot.
`parm`	A parameter name string i.e p, f or gene name.
`steps`	An integer representing number of iterations used while calling denoiseq.
`condition`	A character (either A or B) representing the two experimental conditions.

Value

A vector of parameter samples, of length equal to steps.

Examples

```
#pre-filtering to remove lowly expressed genes
ERCC <- ERCC[rowSums(ERCC)>0, ]
RD <- new('readsData', counts = ERCC)
steps <- 30
#30 steps are just for illustration here. Atleast 5000 steps are adequate.
BI <- denoiseq(RD, steps)
samples <- getSamplesOf(BI, 'ERCC-00051', steps)
plot(samples, type='l', main = 'History plot of ERCC-00051')
```

`readsData` *An S4 class to represent summarised counts and the output of Bayesian inference.*

Description

An S4 class to represent summarised counts and the output of Bayesian inference.

Slots

`counts` A positive integer matrix containing summarised counts for each genomic event (genes, exons, transcripts, etc) in the two conditions, A and B.

`replicates` A list containing the indices of the columns that belong to each of the two experimental conditions, A and B. It is defaulted to A = 1:(n/2), B = (n/2+1):n for an m by n matrix.

`geneNames` A character vector containing the names of the genomic event. It is appropriately defaulted to names of the matrix.

`initValues` A list containing initial values for each parameter. Defaulted to `N_A` = rep(1, nrow(counts)), `N_B` = rep(1, nrow(counts)), `p`= 0.0001, `f` = 0.01.

`stepSizes` A list containing step sizes for sampling each parameter. Defaulted to `stepsizeN_A` = rep(1, nrow(counts)), `stepsizeN_B` = rep(1, nrow(counts)), `stepsize_p`= 1e3, `stepsize_f` = 5e7

`output` A list containing the samples for each parameter which are generated by Bayesian inference. It can only be filled inside the results function.

`results` *Compute the test statistic*

Description

Extracts posterior samples of the parameters which are returned by denoiseq function and computes the summary and test statistics.

Usage

```
results(RDobject, steps, burnin = floor(steps/3), rope_limit = 0.5)
```

Arguments

`RDobject` A readsData object with a filled output slot.

`steps` An integer representing the number of iterations.

`burnin` An integer for the number of iterations to be considered as burn in values. A default value equivalent to a third of steps is used.

`rope_limit` A float that delimits the range of the region of practical equivalence, ROPE. A default value of 0.5 is used.

Details

To calculate the test statistic, this function first log2 transforms the posterior samples of the two relevant parameters i.e N_{iA} and N_{iB}. It then randomly subtracts posterior samples of one of the parameters from the other and determines the proportion of this distribution of differences that lies in the region of practical equivalence (ROPE) (Kruschke, 2011). The genes can then be arranged in an ascending order of the ROPE_propn column and we can select the most differentially expressed genes as those whose ROPE_propn is less than a particular threshold value.

Using both real and simulated data, optimal values between 0.0007 and 0.4 were obtained for the threshold.

Value

A dataframe with 3 columns namely; the log2 fold change (log2FC), the standard error of the log2 fold change (lgfcSE) and the test static (ROPE_propn).

Examples

```
#pre -filtering to remove lowly expressed genes
ERCC <- ERCC[rowSums(ERCC) > 0, ]
RD <- new('readsData', counts = ERCC)
steps <- 30
#30 steps are just for illustration here. At least 5000 steps are adequate.
BI <- denoiseq(RD, steps)
rez <- results(BI, steps)
head(rez)

#Re-ordering according to most differentially expressed
rez <- rez[with(rez, order( ROPE_propn)), ]
head(rez, 10)

#Determine significant genes using a threshold of 0.38.
sgf <- rez[rez$ROPE_propn<0.38, ]
head(sgf)
dim(sgf)
```

`setInitValues` *Generic for altering the initValues slot*

Description

Updates the value of the initValues slot for the readsData object supplied.

Usage

```
setInitValues(object, initval)

## S4 method for signature 'readsData'
setInitValues(object, initval)
```

Arguments

`object`	a readsData object
`initval`	A list of initial values for each of the parameters.

Value

The same readsData object with the initValues slot updated.

Methods (by class)

- `readsData`: Alters the value of the initValues slot of a readsData object.

Examples

```
RD <- new("readsData", counts = ERCC)
initvals <- list(N_A = rep(2, 92), N_B = rep(1.5, 92), p = 0.0005, f = 0.03)
RD <- setInitValues(RD, initvals)
RD@initValues
```

`setReplicates` *Generic for the altering setReplicates slot.*

Description

Updates the value of the replicates slot for the readsData object supplied.

Usage

```
setReplicates(object, repsval)

## S4 method for signature 'readsData'
setReplicates(object, repsval)
```

Arguments

`object`	a readsData object
`repsval`	A list of column indices for the samples in each condition.

Value

The same readsData object with the replicates slot updated.

Methods (by class)

- `readsData`: Alters the value of the replicates slot of a readsData object.

Examples

```
RD <- new("readsData", counts = ERCC)
reps <- list(A = c(2,4,5,3,10),B = c(9,7,1,8,6))
RD <- setReplicates(RD, reps)
RD@replicates
```

`setStepSizes` *Generic for altering the stepSizes slot.*

Description

Updates the value of the stepSizes slot for the readsData object supplied.

Usage

```
setStepSizes(object, stepSizesval)

## S4 method for signature 'readsData'
setStepSizes(object, stepSizesval)
```

Arguments

`object`	a readsData object
`stepSizesval`	A list of step sizes for each of the parameters.

Value

The same readsData object with the stepSizes slot updated.

Methods (by class)

- `readsData`: Alters the value of the stepSizes slot of a readsData object.

Examples

```
RD <- new("readsData", counts = ERCC)
ss <- list(N_A = rep(2,  92), N_B = rep(1.5, 92), p = 3e5, f = 3.5e7)
RD <- setStepSizes(RD, ss)
RD@stepSizes
```

`simdat` *simulated data*

Description

A dataset containing simulated data based on parameter values N = 1,2,...,50 , p = 0.0017 and f =0.1,0.2,...,0.5. The values of N were repeated 15 times to generate 750 genes. This dataset contains 750 observational genes with 5 experimental samples for each condition, summarised as a 750 by 10 integer matrix. The first 428 genes are not differentially expressed between the two conditions whereas the last 322 genes are. The gene counts were generated in accordance to the probability distribution derived in Ndifon et al.

Usage

```
simdat
```

Format

A matrix with 750 rows and 10 columns:

Conditions There are 5 columns for each of the conditions A and B.

Transcripts There are 750 distinct genes without names.

`tunedStepSize` *Get values of the tuned step sizes.*

Description

Extracts the tuned step sizes for sampling each parameter from the return value of denoiseq.

Usage

```
tunedStepSize(RDobject)
```

Arguments

`RDobject`	A readsData object with a filled output slot.

Value

A list of the tuned step sizes for sampling each of the parameters.

Examples

```
#pre -filtering to remove lowly expressed genes
ERCC <- ERCC[rowSums(ERCC)>0, ]
RD <- new('readsData', counts = ERCC)
steps <- 30
#30 steps are just for illustration here. Atleast 5000 steps are adequate.
BI <- denoiseq(RD, steps)
tunedStepSize(BI)
```

Index