Package 'MBASED'

November 12, 2026

Type Package

Title Package containing functions for ASE analysis using Meta-analysis Based Allele-Specific Expression Detection

Version 1.46.0

Date 2014-08-27

Author Oleg Mayba, Houston Gilbert

Maintainer Oleg Mayba <mayba.oleg@gene.com>

Description The package implements MBASED algorithm for detecting allele-specific gene expression from RNA count data, where allele counts at individual loci (SNVs) are integrated into a gene-specific measure of ASE, and utilizes simulations to appropriately assess the statistical significance of observed ASE.

biocViews Sequencing, GeneExpression, Transcription

Depends RUnit, BiocGenerics, BiocParallel, GenomicRanges, SummarizedExperiment

Suggests BiocStyle

License Artistic-2.0

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

git_branch RELEASE_3_23

git_last_commit ca73ae8

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

estimateMAF1s 2
estimateMAF2s 3
FT 5

getMuRho 6
getPFinal 7
getSimulationPvalue 8
logLikelihoodCalculator1s 9
logLikelihoodCalculator2s 10
maxLogLikelihoodCalculator1s 11
maxLogLikelihoodCalculator2s 12
MBASED 14
MBASEDMetaAnalysis 14
MBASEDMetaAnalysisGetMeansAndSEs 15
MBASEDVectorizedMetaprop 17
MBASEDVectorizedPropDiffTest 19
runMBASED 20
runMBASED1s 25
runMBASED1s1aseID 27
runMBASED2s 29
runMBASED2s1aseID 31
shiftAndAttenuateProportions 33
testNumericDiff 34
testQuantiles 35
vectorizedRbetabinomAB 36

Index **38**

`estimateMAF1s` *Function that given observed count data returns a maximum likelihood estimate of the underlying haplotype frequency. Both situations where the haplotype are known and unknown are handled. In the latter case, likelihood is further maximized over all possible assignments of alleles to haplotypes.*

Description

Function that given observed count data returns a maximum likelihood estimate of the underlying haplotype frequency. Both situations where the haplotype are known and unknown are handled. In the latter case, likelihood is further maximized over all possible assignments of alleles to haplotypes.

Usage

```
estimateMAF1s(lociAllele1Counts, lociTotalCounts, lociAllele1NoASEProbs,
  lociRhos, isPhased = FALSE, checkArgs = FALSE)
```

Arguments

`lociAllele1Counts`
: counts of allele1-supporting reads at individual loci. Must be a vector of non-negative integers.

`lociTotalCounts`
: total read counts of at individual loci. Must be a vector of positive integers.

`lociAllele1NoASEProbs`
: probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus). Must be a vector with entries >0 and <1.

`lociRhos`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Must be a numeric vector with entries >=0 and <1.

`isPhased`
: single boolean specifying whether the phasing has already been performed, in which case the lociAllele1Counts represent the same haplotype. If FALSE (DEFAULT), likelihood is further maximized over all possible assignments of alleles to haplotypes.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting allele1 at a collection of loci, the total read counts at those loci, the probablities of observing allele1-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a maximum likelihood estimate of the major haplotype frequency as well as corresponding assignment of alleles to haplotypes.

Value

a list with two elements: MAF (MLE of major allele frequency) and allele1IsMajor (whether allele1 is assigned to haplotype corresponding to maximum likleihood MAF).

Examples

```
MBASED:::estimateMAF1s(lociAllele1Counts=c(5, 24), lociTotalCounts=c(15, 36), lociAllele1NoASEProbs=c(0.5, 0.5),
MBASED:::estimateMAF1s(lociAllele1Counts=c(5, 24), lociTotalCounts=c(15, 36), lociAllele1NoASEProbs=c(0.5, 0.5),
```

`estimateMAF2s` *Function that given observed count data returns a maximum likelihood estimate of the underlying haplotype frequency. Both situations where the haplotype are known and unknown are handled. In the latter case, likelihood is further maximized over all possible assignments of alleles to haplotypes.*

Description

Function that given observed count data returns a maximum likelihood estimate of the underlying haplotype frequency. Both situations where the haplotype are known and unknown are handled. In the latter case, likelihood is further maximized over all possible assignments of alleles to haplotypes.

Usage

```
estimateMAF2s(lociAllele1CountsSample1, lociTotalCountsSample1,
  lociAllele1CountsSample2, lociTotalCountsSample2,
  lociAllele1NoASEProbsSample1, lociAllele1NoASEProbsSample2, lociRhosSample1,
  lociRhosSample2, isPhased = FALSE, checkArgs = FALSE)
```

Arguments

`lociAllele1CountsSample1`, `lociAllele1CountsSample2`
: counts of allele1-supporting reads at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociTotalCountsSample1`, `lociTotalCountsSample2`
: total read counts of at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociAllele1NoASEProbsSample1`, `lociAllele1NoASEProbsSample2`
: probabilities of observing haplotype A-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus) in sample1 and sample2, respectively. Both arguments must be vectors with entries >0 and <1.

`lociRhosSample1`, `lociRhosSample2`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial) in sample1 and sample2, respectively. Both arguments must be vectors with entries >=0 and <1.

`isPhased`
: single boolean specifying whether the phasing has already been performed, in which case the lociAllele1CountsSample1 (and, therefore, lociAllele1CountsSample2) represent the same haplotype. If FALSE (DEFAULT), likelihood is further maximized over all possible assignments of alleles to haplotypes.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting allele1 at a collection of loci in two samples, the total read counts at those loci, the probablities of observing allele1-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a maximum likelihood estimate of the major haplotype frequency as well as corresponding assignment of alleles to haplotypes.

Value

a list with two elements: MAF (MLE of major allele frequency) and allele1IsMajor (whether allele1 is assigned to haplotype corresponding to maximum likleihood MAF).

Examples

```
MBASED:::estimateMAF2s(lociAllele1CountsSample1=c(5, 24), lociTotalCountsSample1=c(15, 36), lociAllele1CountsSam
MBASED:::estimateMAF2s(lociAllele1CountsSample1=c(5, 12), lociTotalCountsSample1=c(15, 36), lociAllele1CountsSam
```

FT *Freeman-Tukey transformation functions.*

Description

Freeman-Tukey transformation functions.

Usage

```
FT(x, n, checkArgs = FALSE)

unFT(z, n, checkArgs = FALSE)

FTAdjust(x, n, p, checkArgs = FALSE)

isCountMajorFT(x, n, p, tieBreakRandom = FALSE, checkArgs = FALSE)
```

Arguments

n	number of trials, (vector/matrix of) positive number(s).
x	number of successes, (vector/matrix of) non-negative number(s) <=n.
p	probability of success on each trial, (vector/matrix of) value(s) between 0 and 1.
tieBreakRandom	if FALSE, a backransformed value of 0.5 in isCountMajorFT() will be called major; if TRUE, it will be called major with probability 0.5 and minor with probability 0.5. DEFAULT: FALSE
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE
z	(vector/matrix of) transformed proportion(s).

Details

FT takes integers x and n, where x is observed Bin(n, p) random variable, and performs Freeman-Tukey transformation. Arguments x and n are vectorized and must be of the same length (if vectors) or dimension (if matrices).

unFT takes transformed proportion and original total count and untransforms it, using the same approach as metaprop() function from R package "meta", with one correction: to avoid situations that arise in practice when z takes a value that cannot result from the supplied value of n (e.g. z corresponding to a count of < 0 out of n or > n out of n), we assign z to be the smallest/largest allowed value. Arguments z, and n are vectorized and must be of same length (if vectors) or dimension (if matrices).

FTAdjust takes integers x and n, and probability p, where x is observed Bin(n, p) random variable and performs Freeman-Tukey transformation, followed by shifting the transformed variable so that its mean is 2*arcsin(sqrt(0.5)) instead of 2*arcsin(sqrt(p)). Arguments x, n and p are vectorized and must be of the same length (if vectors) or dimension (if matrices).

isCountMajorFT takes original observed count and total count, transforms, adjusting for underlying probability of success and returns TRUE or FALSE depending on whether the count is major (back-transformed proportion >=0.5 or not). Arguments x, n and p are vectorized and must be of same length (if vectors) or dimension (if matrices).

Value

FT returns (vector of) transformed proportion(s) of successes.

unFT returns (vector/matrix of) backtransformed proportion(s).

FTAdjust returns (vector/matrix of) shifted transformed proportion(s).

isCountMajorFT returns (vector/matrix of) TRUE or FALSE, depending on whether count is judged to be from 'major' allele or not.

Examples

```
isTRUE(all.equal(MBASED:::FT(x=5,n=10), pi/2))
MBASED:::unFT(z=MBASED:::FT(x=5, n=10), n=10)
MBASED:::unFT(z=MBASED:::FT(x=7, n=10), n=10)
isTRUE(all.equal(MBASED:::unFT(z=MBASED:::FT(x=7, n=10), n=10), 0.7))
MBASED:::FT(x=50, n=100)
MBASED:::FTAdjust(x=50, n=100, p=0.5) ## transformation is trivial if underlying probability of success is 0.5
MBASED:::FT(x=80, n=100)
MBASED:::FTAdjust(x=80, n=100, p=0.8) ## if underlying probability of success is 0.8, the shift adjusts transformed
MBASED:::isCountMajorFT(x=6, n=10, p=0.5, tieBreakRandom=FALSE)
MBASED:::isCountMajorFT(x=6, n=10, p=0.8, tieBreakRandom=FALSE)
MBASED:::isCountMajorFT(x=4, n=10, p=0.2, tieBreakRandom=FALSE)
table(replicate(1000, MBASED:::isCountMajorFT(x=5, n=10, p=0.5, tieBreakRandom=FALSE)))
table(replicate(1000, MBASED:::isCountMajorFT(x=5, n=10, p=0.5, tieBreakRandom=TRUE)))
```

---

getMuRho *Functions to convert between shape parameters a and b for beta distribution and parameters mu (mean) and rho (dispersion).*

Description

Functions to convert between shape parameters a and b for beta distribution and parameters mu (mean) and rho (dispersion).

Usage

```
getMuRho(a, b, checkArgs = FALSE)

getAB(mu, rho, checkArgs = FALSE)
```

Arguments

a, b	shape parameters for beta distribution. Must be >0.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE
mu, rho	mean and dispersion parameters for beta distribution, respectively. Must be in (0,1) interval, although rho is allowed to take on value of 0 (binomial distribution).

Details

getMuRho takes in shape parameters a and b and returns list with parameters mu (a/(a+b)) and rho (1/(a+b+1)). The function is vectorized (both a and b can be vectors (of the same length) or matrices (of the same dimension)).

getAB takes in shape mean and dispersion parameters mu and rho and returns shape parameters a (mu*(1/rho-1)) and b ((1-mu)*(1/rho-1)). The function is vectorized (both mu and rho can be vectors (of the same length) or matrices (of the same dimension)).

Value

getMuRho returns a list with 2 elements: mu and rho (vectors, if the arguments a and b are vectors).

getAB returns a list with 2 elements: a and b (vectors, if arguments mu and rho are vectors). For values of rho=0, the resulting entries are NA.

See Also

Other bbFunctions: vectorizedRbetabinomAB, vectorizedRbetabinomMR, vectorizedRbetabinomMR

Other bbFunctions: vectorizedRbetabinomAB, vectorizedRbetabinomMR, vectorizedRbetabinomMR

Examples

```
MBASED:::getMuRho(a=1, b=1)
MBASED:::getAB(mu=1/2, rho=1/3)
MBASED:::getMuRho(MBASED:::getAB(mu=0.7, rho=0.0045)$a, MBASED:::getAB(mu=0.7, rho=0.0045)$b)
MBASED:::getAB(MBASED:::getMuRho(a=0.2, b=4)$mu, MBASED:::getMuRho(a=0.2, b=4)$rho)
```

---

getPFinal — *Function that adjusts true underlying allele frequency for pre-existing allelic bias to produce actual generating probability of observing allele-supporting read*

Description

Function that adjusts true underlying allele frequency for pre-existing allelic bias to produce actual generating probability of observing allele-supporting read

Usage

```
getPFinal(trueAF, noASEAF, checkArgs = FALSE)
```

Arguments

trueAF	true underlying allele frequency. Must be a single number >=0 and <=1.
noASEAF	probability of observing allele-supporting read under conditions of no ASE. Must be a vector of numbers >0 and <1.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given true underlying allele frequency AF and probability of observing reads supporting that allele under conditiosn of no ASE (P(allele, noASE)), it calculates the generating probability for observed allele-supporting reads as P(allele-supporting read)=AF*P(allele, noASE)/(AF*P(allele, noASE) + (1-AF)*(1-P(allele, noASE))).

Value

a vector of generating probabilities of the same length as noASEAF

Examples

```
MBASED:::getPFinal(trueAF=1, noASEAF=seq(0.1, 0.9, by=0.1)) ## is always 1
MBASED:::getPFinal(trueAF=0, noASEAF=seq(0.1, 0.9, by=0.1)) ## is always 0
MBASED:::getPFinal(trueAF=0.3, noASEAF=0.5) ## no pre-existing allelic bias
c(MBASED:::getPFinal(trueAF=0.3, noASEAF=0.9), MBASED:::getPFinal(trueAF=1-0.3, noASEAF=1-0.9)) ## strong pre-ex
```

---

getSimulationPvalue *Function to calculate simulations-based p-values*

Description

Function to calculate simulations-based p-values

Usage

```
getSimulationPvalue(observedVal, simulatedVals, direction = "greater",
  checkArgs = FALSE)
```

Arguments

observedVal	observed statistic (single number)
simulatedVals	statistics observed in simulations of the outcomes based on assumed null distribution.
direction	one of 'greater' or 'less', depending on the nature of statistic.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

this function calculates fraction of simulated values (statistics from null distribution) that are >= (direction='greater') or <= (direction='less') than the observed statistic. The choice of direction depends on the nature of the statistic (i.e., direction is 'greater' if large values of statistic indicate departure from null hypothesis, and direction is 'less' if the opposite is the case)

Value

a fraction of simulated statistics that are as or more extreme as the observed one

Examples

```
MBASED:::getSimulationPvalue(observedVal=2, simulatedVals=1:10, direction='greater')
MBASED:::getSimulationPvalue(observedVal=2, simulatedVals=1:10, direction='less')
```

---

logLikelihoodCalculator1s

Function that given observed count data along a known haplotype returns a function that can calculate the likelihood of observing that data for a supplied underlying haplotype frequency.

Description

Function that given observed count data along a known haplotype returns a function that can calculate the likelihood of observing that data for a supplied underlying haplotype frequency.

Usage

```
logLikelihoodCalculator1s(lociHapACounts, lociTotalCounts, lociHapANoASEProbs,
  lociRhos, checkArgs = FALSE)
```

Arguments

lociHapACounts	counts of haplotype A-supporting reads at individual loci. Must be a vector of non-negative integers.
lociTotalCounts	total read counts of at individual loci. Must be a vector of positive integers.
lociHapANoASEProbs	probabilities of observing haplotype A-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus). Must be a vector with entries >0 and <1.
lociRhos	dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Must be a numeric vector with entries >=0 and <1.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting hapltoype A at a collection of loci, the total read counts at those loci, the probablities of observing haplotype A-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a function of a single argument, pHapA, that calculates the likelihood of observing the given haplotype A-supporting counts under the assumption that the true underlying frequency of haplotype A is pHapA.

Value

a function of a single argument pHapA that calculates log likelihood of the observed data if the true underlying haplotype A frequency is pHapA.

Examples

```
LLC <- MBASED:::logLikelihoodCalculator1s(lociHapACounts=c(5, 12), lociTotalCounts=c(10, 24), lociHapANoASEProbs
LLC(0.5) ## the MLE estimate of hapA frequency
LLC(0.1) ## highly implausible value of pHapA
LLC (0.51)
```

---

logLikelihoodCalculator2s

Function that given observed count data along a known haplotype returns a function that can calculate the likelihood of observing that data for a supplied underlying haplotype frequency.

Description

Function that given observed count data along a known haplotype returns a function that can calculate the likelihood of observing that data for a supplied underlying haplotype frequency.

Usage

```
logLikelihoodCalculator2s(lociHapACountsSample1, lociTotalCountsSample1,
  lociHapACountsSample2, lociTotalCountsSample2, lociHapANoASEProbsSample1,
  lociHapANoASEProbsSample2, lociRhosSample1, lociRhosSample2,
  checkArgs = FALSE)
```

Arguments

`lociHapACountsSample1`, `lociHapACountsSample2`
: counts of haplotype A-supporting reads at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociTotalCountsSample1`, `lociTotalCountsSample2`
: total read counts of at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociHapANoASEProbsSample1`, `lociHapANoASEProbsSample2`
: probabilities of observing haplotype A-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus) in sample1 and sample2, respectively. Both arguments must be vectors with entries >0 and <1.

`lociRhosSample1`, `lociRhosSample2`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial) in sample1 and sample2, respectively. Both arguments must be vectors with entries >=0 and <1.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting hapltoype A at a collection of loci in two samples, the total read counts at those loci, the probablities of observing haplotype A-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a function of a single argument, pHapA, that calculates the likelihood of observing the given haplotype A-supporting counts under the assumption that the true underlying frequency of haplotype A is pHapA.

Value

a function of a single argument pHapA that calculates log likelihood of the observed data if the true underlying haplotype A frequency is pHapA.

Examples

```
LLC <- MBASED:::logLikelihoodCalculator2s(lociHapACountsSample1=c(5, 12), lociTotalCountsSample1=c(15, 36), loci
LLC(1/3) ## the MLE estimate of hapA frequency
LLC(0.9) ## highly implausible value of pHapA
LLC (0.334)
```

---

maxLogLikelihoodCalculator1s

Function that given observed count data along a known haplotype returns a maximum likelihood estimate of the underlying haplotype frequency.

Description

Function that given observed count data along a known haplotype returns a maximum likelihood estimate of the underlying haplotype frequency.

Usage

```
maxLogLikelihoodCalculator1s(lociHapACounts, lociTotalCounts,
  lociHapANoASEProbs, lociRhos, checkArgs = FALSE)
```

Arguments

lociHapACounts	counts of haplotype A-supporting reads at individual loci. Must be a vector of non-negative integers.
lociTotalCounts	total read counts of at individual loci. Must be a vector of positive integers.
lociHapANoASEProbs	probabilities of observing haplotype A-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus). Must be a vector with entries >0 and <1.
lociRhos	dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Must be a numeric vector with entries >=0 and <1.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting hapltoype A at a collection of loci, the total read counts at those loci, the probablities of observing haplotype A-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a maximum likelihood estimate of the true underlying frequency of haplotype A as well as corresponding value of log-likelihood.

Value

a list with two elements: maximum (MLE of haplotype A frequency) and objective (loglikelihood at MLE). These are the two elements that are output by the optimize() function, which is used internally by the maxLogLikelihoodCalculator1s.

Examples

```
MBASED:::maxLogLikelihoodCalculator1s(lociHapACounts=c(5, 12), lociTotalCounts=c(10, 24), lociHapANoASEProbs=c(0
```

---

maxLogLikelihoodCalculator2s

Function that given observed count data along a known haplotype returns a maximum likelihood estimate of the underlying haplotype frequency.

Description

Function that given observed count data along a known haplotype returns a maximum likelihood estimate of the underlying haplotype frequency.

Usage

```
maxLogLikelihoodCalculator2s(lociHapACountsSample1, lociTotalCountsSample1,
  lociHapACountsSample2, lociTotalCountsSample2, lociHapANoASEProbsSample1,
  lociHapANoASEProbsSample2, lociRhosSample1, lociRhosSample2,
  checkArgs = FALSE)
```

Arguments

`lociHapACountsSample1`, `lociHapACountsSample2`
: counts of haplotype A-supporting reads at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociTotalCountsSample1`, `lociTotalCountsSample2`
: total read counts of at individual loci in sample1 and sample2, respectively. Both arguments must be vectors of non-negative integers.

`lociHapANoASEProbsSample1`, `lociHapANoASEProbsSample2`
: probabilities of observing haplotype A-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus) in sample1 and sample2, respectively. Both arguments must be vectors with entries >0 and <1.

`lociRhosSample1`, `lociRhosSample2`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial) in sample1 and sample2, respectively. Both arguments must be vectors with entries >=0 and <1.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

Given observed read counts supporting hapltoype A at a collection of loci in two samples, the total read counts at those loci, the probablities of observing haplotype A-supporting reads under conditions of no ASE and the dispersion parameters, this function returns a maximum likelihood estimate of the true underlying frequency of haplotype A as well as corresponding value of log-likelihood.

Value

a list with two elements: maximum (MLE of haplotype A frequency) and objective (loglikelihood at MLE). These are the two elements that are output by the optimize() function, which is used internally by the maxLogLikelihoodCalculator2s.

Examples

```
MBASED:::maxLogLikelihoodCalculator2s(lociHapACountsSample1=c(5, 12), lociTotalCountsSample1=c(15, 36), lociHapA
```

MBASED *MBASED*

Description

Package that contains functions to process sets of SNVs and determine the genes that show allele-specific expression (ASE)

Details

The package implements MBASED method for detecting allele-specific gene expression. The main workhorse function is runMBASED which is used to run both 1-sample and 2-sample (allelic imbalance) analyses. Please consult the accompanying vignette and the runMBASED help page for more details.

Author(s)

Oleg Mayba <maybao@gene.com> Houston Gilbert <gilbert.houston@gene.com>

MBASEDMetaAnalysis *Generic function to perform standard meta analysis.*

Description

Generic function to perform standard meta analysis.

Usage

```
MBASEDMetaAnalysis(zValuesMat, zVariancesMat, alternative = "two.sided",
  checkArgs = FALSE)
```

Arguments

zValuesMat	matrix of z-values, on standard normal scale. Each row represents a specific genomic locus, while each column represents a set of observed values across loci (in practice, multiple columns represent different outcomes of simulations).
zVariancesMat	matrix of (estimated) variances of each z-value in zValuesMat. The interpretation of rows and columns is the same as for zValuesMat.
alternative	one of 'two.sided', 'greater', 'less'. DEFAULT: 'two.sided'.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

MBASEDMetaAnalysis performs meta analysis calculations in a vectorized fashion. Input matrices zValuesMat and zVariancesMat have one column for each set of loci ('independent studies') to be combined, with each row corresponding to an individual locus. MBASEDMetaAnalysis uses meta analysis approach to combine values in each column of zValuesMat into a single column-specific value of z (using corresponding supplied variances to appropriately weight contributions of each individual z). The function reports the resulting averaged z values, together with corresponding standard deviations (standard errors), for fixed-effects setting (note: random effects are not meaningful in the context of SNVs in ASE). If the supplied matrices have a single row (only one locus), no meta-analysis is possible, and the original value and corresponding standard deviations are returned.

Value

a list with 5 elements:

`hetPVal` a 1-row marix of heterogeneity p-values.

`hetQ` a 1-row matrix of heterogeneity statistics.

`fixedEffectsMeans` a 1-row matrix of column-specific fixed-effects meta analysis restults.

`fixedEffectsSEs` a 1-row matrix of estimated SEs of fixed-effects meta analysis results.

`pvalueFixed` a 1-row matrix of p-values for fixed-effects analysis.

Examples

```
set.seed(127000)
zVals1=rnorm(5, mean=rep(2,5), sd=sqrt(1:5))
zVals2=rnorm(5, mean=0, sd=1)+c(0,0,5,0,0) ## one outlier
MBASED:::MBASEDMetaAnalysis(zValuesMat=matrix(c(zVals1, zVals2), ncol=2), zVariancesMat=matrix(c(1:5, rep(1,5)),
```

MBASEDMetaAnalysisGetMeansAndSEs

Helper function to obtain estimate of underlying mean and the standard error of the estimate in meta analysis framework.

Description

Helper function to obtain estimate of underlying mean and the standard error of the estimate in meta analysis framework.

Usage

```
MBASEDMetaAnalysisGetMeansAndSEs(zValuesMat, zVariancesMat, checkArgs = FALSE)
```

Arguments

zValuesMat	matrix of z-values, on standard normal scale. Each row represents a specific genomic locus, while each column represents a set of observed values across loci (in practice, multiple columns represent different outcomes of simulations).
zVariancesMat	matrix of (estimated) variances of each z-value in zValuesMat. The interpretation of rows and columns is the same as for zValuesMat.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

MBASEDMetaAnalysisGetMeansAndSEs is a helper function employed by MBASEDMetaAnalysis(). For each column of input matrices, it calculates the inverse-variance weighted column average and provides an estimate of the standard error of this mean estimator. Input matrices zValuesMat and zVariancesMat have one column for each set of loci ('independent studies') to be combined, with each row corresponding to an individual locus.

Value

a list with 4 elements:

weightsMat	a matrix of same dimension as zValuesMat, giving the assigned weight for each observation
totalWeights	a vector of length equal to number of rows in zValuesMat, giving the column sum of assigned weights
hetQ	a vector of length equal to number of rows in zValuesMat, giving the estimated standard error for the corresponding entries in meanValues
meanValues	a vector of length equal to number of rows in zValuesMat, giving for each column the estimated average value.
hetQ	a vector of length equal to number of rows in zValuesMat, giving the estimated standard error for the corresponding entries in meanValues

Examples

```
set.seed(127000)
zVals1=rnorm(5, mean=rep(2,5), sd=sqrt(1:5))
zVals2=rnorm(5, mean=0, sd=1)+c(0,0,5,0,0) ## one outlier
MBASED:::MBASEDMetaAnalysisGetMeansAndSEs(zValuesMat=matrix(c(zVals1, zVals2), ncol=2), zVariancesMat=matrix(c(
```

MBASEDVectorizedMetaprop

Vectorized wrapper around metaprop() function from R package "meta" with some modifications and extensions to beta-binomial count models.

Description

Vectorized wrapper around metaprop() function from R package "meta" with some modifications and extensions to beta-binomial count models.

Usage

```
MBASEDVectorizedMetaprop(countsMat, totalsMat, probsMat, rhosMat,
  alternative = "two.sided", checkArgs = FALSE)
```

Arguments

countsMat	matrix of observed major allele counts. Each row represents a specific genomic locus, while each column represents a set of observed major allele counts across loci (in practice, multiple columns represent different outcomes of count simulations).
totalsMat	matrix of total read counts across both alleles. The interpretation of rows and columns is the same as for countsMat.
probsMat	matrix of probabilities of success (means of beta distributions in case of beta-binomial extensions). The interpretation of rows and columns is the same as for countsMat.
rhosMat	matrix of dispersion parameters of beta distribution in case of beta-binomial counts. The interpretation of rows and columns is the same as for countsMat.
alternative	one of 'two.sided', 'greater', 'less'. DEFAULT: 'two.sided'.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

MBASEDVectorizedMetaprop performs computations similar to metaprop() with default options (fixed-effects only), but with less overhang, in a vectorized fashion, and accomodating extensions to beta-binomial distribution. It also allows the input counts to come from loci with different underlying binomial probabilities (means of beta distribution, in cases of beta-binomial extensions). One technical difference is the way the value of 'n' is calculated for Freeman-Tukey back-transformation of average 'z' into a proportion. While metaprop() uses harmonic mean of n's at individual loci (which puts more weight toward loci with small read counts), we use the weighted mean of n's with weights proportional to n's, by analogy with how the value of average 'z' is calculated from individual z's. Input matrices countsMat and totalsMat have one column for each set of loci ('independent studies') to be combined, with each row corresponding to an individual locus. Matrix probsMat provides the underlying binomial probabilities (means of beta distributions, in case of

beta-binomial extensions), while matrix rhosMat gives the values of the dispersion at the loci. MBASEDVectorizedMetaprop uses meta analysis approach with Freeman-Tukey transformation to report for each set of loci (each column) its estimated overall proportion on both transformed and untransformed scale, corresponding standard errors (on transformed scale), z-values (based on expected value of 2*asin(sqrt(0.5)) on transformed scale under the null hypothesis of common underlying proportion (binomial probability or mean of beta distribution) of 0.5), and corresponding p-values based on imposing normal distribution assumption on z-values, where alternative hypothesis of 'two.sided', 'greater', and 'less' can be specified, with the latter two specified w.r.t. 2*asin(sqrt(0.5)). If some of the supplied entries in probsMat are different from 0.5, then the corresponding transformed proportions are shifted, so that the new mean for the resulting z-values is still approximately 2*asin(sqrt(0.5)). Extensions to beta-binomial counts are accomodated by increasing the variance of each individual z from 1/(n+0.5) to rho+1/(n+0.5), where rho is the dispersion parameter of the beta distribution. The function is used to cacluate p-values in ASE settings, where countsMat represents major allele counts, totalsMat represents total allele counts, probsMat represents the underlying binomial probabilities of observing major allele-supporting read (means of beta distributions in case of beta-binomial extensions), which may be different, e.g., for major allele counts coming from reference and alternative alleles in case of pre-existing allelic bias, and rhosMat provides values of dispersion parameter for beta-binomial counts (0, in case of binomial model) for individual loci.

Value

a list with 7 elements:

`hetPVal`	a 1-row marix of heterogeneity p-values.
`hetQ`	a 1-row matrix of heterogeneity statistics.
`TEFinal`	a 1-row matrix of estimated proportions on transformed scale.
`seTEFinal`	a 1-row matrix of estimated SEs of proportion estimates on transformed scale.
`propFinal`	a 1-row matrix of estimated proportions on 0-1 scale.
`pValue`	a 1-row matrix of corresponding p-values.
`propLoci`	a matrix of same dimension as original input matrices giving estimated proportions on transformed scale at each individual locus.

Examples

```
SNVCoverage=rep(sample(10:100,5),2) ## 2 genes with 5 loci each
SNVAllele1Counts=rbinom(length(SNVCoverage), SNVCoverage, 0.5)
SNVMajorAlleleCounts=pmax(SNVAllele1Counts, SNVCoverage-SNVAllele1Counts)
MBASED:::MBASEDVectorizedMetaprop(countsMat=matrix(SNVAllele1Counts, ncol=2), totalsMat=matrix(SNVCoverage, nco
MBASED:::MBASEDVectorizedMetaprop(countsMat=matrix(SNVMajorAlleleCounts, ncol=2), totalsMat=matrix(SNVCoverage,
```

`MBASEDVectorizedPropDiffTest` *Vectorized wrapper around a test for difference of 2 proportions.*

Description

Vectorized wrapper around a test for difference of 2 proportions.

Usage

```
MBASEDVectorizedPropDiffTest(countsMatSample1, totalsMatSample1,
  countsMatSample2, totalsMatSample2, probsMatSample1, probsMatSample2,
  rhosMatSample1, rhosMatSample2, alternative = "two.sided",
  checkArgs = FALSE)
```

Arguments

`countsMatSample1`, `countsMatSample2`
: matrices of observed major allele counts in sample1 and sample2, respectively. Each row represents a specific genomic locus, while each column represents a set of observed major allele counts across loci (in practice, multiple columns represent different outcomes of count simulations).

`totalsMatSample1`, `totalsMatSample2`
: matrices of total read counts across both alleles in sample1 and sample2, respectively. The interpretation of rows and columns is the same as for countsMatSample1.

`probsMatSample1`, `probsMatSample2`
: matrices of underlying probabilites of observing the major allele in sample1 and sample2, respectively. The interpretation of rows and columns is the same as for countsMatSample1.

`rhosMatSample1`, `rhosMatSample2`
: matrices of dispersion parameters of beta distributions for each locus in sample1 and sample2, respectively. The interpretation of rows and columns is the same as for countsMatSample1.

`alternative`
: one of 'two.sided', 'greater', 'less'. DEFAULT: 'two.sided'

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Details

MBASEDVectorizedPropDiffTest implements meta-analysis-like apporoach using proportion differences at each locus as variables to be aggregated. Input matrices countsMatSample1, totalsMatSample1, countsMatSample2, totalsMatSample2, probsMatSample1, probsMatSample2, rhosMatSample1, and rhosMatSample2 have 1 column for each set of loci ('independent studies') to be combined, with each row corresponding to an individual locus. MBASEDVectorizedPropDiffTest uses meta analysis approach by transforming counts at each locus into proportions and combininig

the proportion differences (between sample1 and sample2) using the inverse-variance weighted schema. The function reports proportion difference estimates, corresponding standard errors, z-values (based on expected value of 0 under the null hypothesis of overall difference of 0) , and corresponding p-values based on normal distribution assumption of z-values, where alternative hypothesis of 'two.sided', 'greater', and 'less' can be specified, with the latter two specified w.r.t. 0. Adjustment for pre-existing allelic bias is performed by taking observed proportion in each sample, transforming it with FT transformation, adjusting for allelic bias as in 1-sample case and back-transforming to get a shifted proportion. The shifted proportion is then used to estimate its variance. The function is used to cacluate p-values in ASE settings, where countsMatSample1 and countsMatSample2 represent major allele counts in sample1 and sample2, respectively, and totalsMatSample1 and totalsMatSample2 represent total allele counts. Matrices probsMatSample1 and probsMatSample2 capture the pre-existing allelic bias by supplying the underlying probabilities of observing alleles currently specified as major in absence of any allele-specific expression, and rhosMatSample1 and rhosMatSample2 provide values of dispersion parameter for beta-binomial counts (0, in case of binomial model) for individual loci within each sample.

Value

a list with 7 elements:

`hetPval` a 1-row marix of heterogeneity P-values

`hetQ` a 1-row matrix of heterogeneity statistics

`TEFinal` a 1-row matrix of estimated proportion differences

`seTEFinal` a 1-row matrix of estimated SEs of prop differences estimates

`propDifferenceFinal` a 1-row matrix of estimated proportion differences

`pValue` a 1-row matrix of corresponding p-values.

`propDifferenceLoci` a matrix of same dimension as original input matrices giving estimated proportion differences on transformed scale at each individual locus.

Examples

```
SNVCoverageTumor=sample(10:100,10) ## 2 genes with 5 loci each
SNVCoverageNormal=sample(10:100,10) ## 2 genes with 5 loci each
SNVAllele1CountsTumor=rbinom(length(SNVCoverageTumor), SNVCoverageTumor, 0.5)
SNVAllele1CountsNormal=rbinom(length(SNVCoverageNormal), SNVCoverageNormal, 0.5)
MBASED:::MBASEDVectorizedPropDiffTest(countsMatSample1=matrix(SNVAllele1CountsTumor, ncol=2), totalsMatSample1=
```

`runMBASED` *Main function that implements MBASED.*

Description

Main function that implements MBASED.

Usage

```
runMBASED(ASESummarizedExperiment, isPhased = FALSE, numSim = 0,
  BPPARAM = SerialParam())
```

Arguments

`ASESummarizedExperiment`

RangedSummarizedExperiment object containing information on read counts to be used for ASE detection. Rows represent individual heterozygous loci (SNVs), while columns represent individual samples. There should be either one or two columns, depending on whether one- or two-sample analysis is to be performed. Joint analysis of multiple samples or replicates is currently not supported, and one-sample analysis of multiple samples must be done through independent series of calls to runMBASED(). Note that for two-sample analysis, only loci which are heterozygous in both samples must be supplied (this excludes, e.g., tumor-specific mutations in cases of tumor/normal comparisons). For two-sample analysis, it is assumed that the first column corresponds to 'sample1' and the second column to 'sample2' in the sample1-vs-sample2 comparison. This is important, since differential ASE assessment is not symmetric and sample1-vs-sample2 comparison may yield different results from sample2-vs-sample1 comparison (the relationship is set up by assuming that only instances of ASE greater in sample1 than in sample2 are of interest). assays(ASESummarizedExperiment) must contain matrices lociAllele1Counts and lociAllele2Counts of non-negative integers, containing counts of allele1 (e.g. reference) and allele2 (e.g. alternative) at individual loci. All supplied loci must have total read count (across both alleles) greater than 0 (in each of the two samples, in the case of two-sample analysis). Allele counts are not necessarily phased (see 'isPhased' argument below), so allele1 counts may not represent the same haplotype. assays(ASESummarizedExperiment) may also contain matrix lociAllele1CountsNoASEProbs with entries >0 and <1, containing probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (which may differ for individual samples in the two-sample analysis). If this matrix is not provided, it is constructed such that every entry in the matrix is set to 0.5 (no pre-existing allelic bias at any locus in any sample). assays(ASESummarizedExperiment) may also contain matrix lociCountsDispersions with entries >=0 and <1, containing dispersion parameters of beta-binomial read count distribution at individual loci (which may differ for individual samples in the two-sample analysis). If this matrix is not provided, it is constructed such that every entry in the matrix is set to 0 (read count-generating distribution at each locus in each sample is binomial). Any other matrices in assays(ASESummarizedExperiment) are ignored by MBASED. rowRanges(ASESummarizedExperiment) must be supplied by the user, containing additional information about SNVs, including a required column 'aseID', specifying for each locus the unique unit of expression that it belongs to (e.g., gene; must be non-NA). MBASED uses names(rowRanges(ASESummarizedExperiment)), when specified, to give a unique identifier to each SNV; if no names are provided, the SNVs are labeled 'locus1', 'locus2', ..., in the row order.

`isPhased` specifies whether the true haplotypes are known, in which case the lociAl-

lele1Counts are assumed to represent allelic counts along the same haplotype (and the same is true of lociAllele2Counts). Must be either TRUE or FALSE (DEFAULT).

`numSim` number of simulations to perform to estimate statistical signficance of observed ASE. Must be a non-negative integer. If set to 0 (DEFAULT), no simulations are performed and nominal p-values are reported.

`BPPARAM` argument to be passed to function bplapply(), when parallel achitecture is used to speed up simulations (parallelization is done over aseIDs). DEFAULT: SerialParam() (no parallelization).

Value

RangedSummarizedExperiment object with rows representing individual aseIDs (genes) and a single column. assays(returnObject) includes single-column matrices 'majorAlleleFrequency' (1-sample analysis only), 'majorAlleleFrequencyDifference' (2-sample analysis only), 'pValueASE' (unadjusted ASE p-value), 'pValueHeterogeneity' (unadjusted inter-loci variability p-value, set to NA for aseIDs with only 1 locus). Note that p-values are not adjusted for multiple hypothesis testing, and the users should carry out such an adjustment themselves, e.g. by employing the utilities in the multtest package. In addition, metadata(returnObject) is a list containing a RangedSummarizedExperiment object names 'locusSpecificResults', with rows corresponding to individual loci (SNVs) and a single column, that provides information on locus-level MBASED analysis results. assays(metadata(returnObject)$locusSpecificResults) contains single-column matrices 'MAF' (estimate of allele frequency for gene-wide major allele at the locus, 1-sample analysis only), 'MAFDifference' (estimate of allele frequency difference for gene-wide major allele at the locus, 2-sample analysis only), and 'allele1IsMajor' (whether allele1 is assigned to major haplotype by MBASED).

Examples

```
mySNVs <- GRanges(
    seqnames=c('chr1', 'chr2', 'chr2', 'chr2'),
     ranges=IRanges(start=c(1000, 20020, 20285, 21114), width=1),
     aseID=c('gene1', rep('gene2', 3)),
    allele1=c('G', 'A', 'C', 'A'),
    allele2=c('T', 'C', 'T', 'G')
)
names(mySNVs) <- paste0('SNV', 1:4)
## RangedSummarizedExperiment object with data to run tumor vs. normal comparison
mySE_TumorVsNormal <- SummarizedExperiment(
     assays=list(
        lociAllele1Counts=matrix(
             c(
                c(25,10,22,14),
                c(18,17,14,28)
            ),
            ncol=2,
            dimnames=list(
                names(mySNVs),
                c('tumor', 'normal')
            )
         ),
```

```
            lociAllele2Counts=matrix(
                 c(
                    c(20,16,15,16),
                     c(23,9,24,17)
                ),
                ncol=2,
                dimnames=list(
                    names(mySNVs),
                     c('tumor', 'normal')
                )
             ),
             lociAllele1CountsNoASEProbs=matrix(
                 c(
                     c(0.48, 0.51, 0.55, 0.45),
                     c(0.52, 0.43, 0.52, 0.43)
                 ),
                 ncol=2,
                 dimnames=list(
                     names(mySNVs),
                     c('tumor', 'normal')
                 )
             ),
            lociCountsDispersions=matrix(
                c(
                     c(0.005, 0.007, 0.003, 0.01),
                     c(0.001, 0.004, 0.02, 0.006)
                 ),
                ncol=2,
                dimnames=list(
                     names(mySNVs),
                     c('tumor', 'normal')
                )
             )
        ),
         rowRanges=mySNVs
    )
    twoSampleAnalysisTumorVsNormal <- runMBASED(
        ASESummarizedExperiment=mySE_TumorVsNormal,
         numSim=10^6,
         BPPARAM=SerialParam(),
         isPhased=FALSE
    )
    rowRanges(twoSampleAnalysisTumorVsNormal)
    assays(twoSampleAnalysisTumorVsNormal)$majorAlleleFrequencyDifference
    assays(twoSampleAnalysisTumorVsNormal)$pValueASE
    assays(twoSampleAnalysisTumorVsNormal)$pValueHeterogeneity
    assays(metadata(twoSampleAnalysisTumorVsNormal)$locusSpecificResults)$MAFDifference
    assays(metadata(twoSampleAnalysisTumorVsNormal)$locusSpecificResults)$allele1IsMajor

    ## exchanging the order of the columns will allow us to run normal vs. tumor comparison
    ## Note that while results are the same for single-locus gene1, they differ for multi-locus gene2
    mySE_NormalVsTumor <- SummarizedExperiment(
        assays=lapply(names(assays(mySE_TumorVsNormal)), function(matName) {
```

```
        curMat <- assays(mySE_TumorVsNormal)[[matName]]
        modifiedMat <- curMat[,c('normal','tumor')]
        return(modifiedMat)
    }),
    colData=colData(mySE_TumorVsNormal)[2:1,],
    rowRanges=rowRanges(mySE_TumorVsNormal)
)
names(assays(mySE_NormalVsTumor )) <- names(assays(mySE_TumorVsNormal))
twoSampleAnalysisNormalVsTumor <- runMBASED(
    ASESummarizedExperiment=mySE_NormalVsTumor,
     numSim=10^6,
     BPPARAM=SerialParam(),
     isPhased=FALSE
)
rowRanges(twoSampleAnalysisNormalVsTumor)
assays(twoSampleAnalysisNormalVsTumor)$majorAlleleFrequencyDifference
assays(twoSampleAnalysisNormalVsTumor)$pValueASE
assays(twoSampleAnalysisNormalVsTumor)$pValueHeterogeneity
assays(metadata(twoSampleAnalysisNormalVsTumor)$locusSpecificResults)$MAFDifference
assays(metadata(twoSampleAnalysisNormalVsTumor)$locusSpecificResults)$allele1IsMajor

## we can also do separate one-sample analysis on tumor and normal samples
mySE_Tumor <- SummarizedExperiment(
    assays=lapply(names(assays(mySE_TumorVsNormal)), function(matName) {
        curMat <- assays(mySE_TumorVsNormal)[[matName]]
        modifiedMat <- curMat[,'tumor',drop=FALSE]
        return(modifiedMat)
    }),
    colData=colData(mySE_TumorVsNormal)[1,],
    rowRanges=rowRanges(mySE_TumorVsNormal)
)
names(assays(mySE_Tumor)) <- names(assays(mySE_TumorVsNormal))
oneSampleAnalysisTumor <- runMBASED(
    ASESummarizedExperiment=mySE_Tumor,
     numSim=10^6,
     BPPARAM=SerialParam(),
     isPhased=FALSE
)
rowRanges(oneSampleAnalysisTumor)
assays(oneSampleAnalysisTumor)$majorAlleleFrequency
assays(oneSampleAnalysisTumor)$pValueASE
assays(oneSampleAnalysisTumor)$pValueHeterogeneity
assays(metadata(oneSampleAnalysisTumor)$locusSpecificResults)$MAF
assays(metadata(oneSampleAnalysisTumor)$locusSpecificResults)$allele1IsMajor

mySE_Normal <- SummarizedExperiment(
    assays=lapply(names(assays(mySE_TumorVsNormal)), function(matName) {
        curMat <- assays(mySE_TumorVsNormal)[[matName]]
        modifiedMat <- curMat[,'normal',drop=FALSE]
        return(modifiedMat)
    }),
    colData=colData(mySE_TumorVsNormal)[1,],
    rowRanges=rowRanges(mySE_TumorVsNormal)
```

```
)
names(assays(mySE_Normal)) <- names(assays(mySE_TumorVsNormal))
oneSampleAnalysisNormal <- runMBASED(
    ASESummarizedExperiment=mySE_Normal,
     numSim=10^6,
     BPPARAM=SerialParam(),
     isPhased=FALSE
)
rowRanges(oneSampleAnalysisNormal)
assays(oneSampleAnalysisNormal)$majorAlleleFrequency
assays(oneSampleAnalysisNormal)$pValueASE
assays(oneSampleAnalysisNormal)$pValueHeterogeneity
assays(metadata(oneSampleAnalysisNormal)$locusSpecificResults)$MAF
assays(metadata(oneSampleAnalysisNormal)$locusSpecificResults)$allele1IsMajor
```

`runMBASED1s` *Function that runs single-sample ASE calling using data from individual loci (SNVs) within units of ASE (genes). Vector arguments 'lociAllele1Counts', 'lociAllele2Counts', 'lociAllele1NoASEProbs', 'lociRhos', and 'aseIDs' should all be of the same length. Letting i1, i2, .., iN denote the indices corresponding to entries within aseIDs equal to a given aseID, the entries at those indices in the other vector arguments provide information for the loci within that aseID. This information is then used by runMBASED1s1aseID. It is assumed that for any i, the i-th entries of all vector arguments correspond to the same locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype.*

Description

Function that runs single-sample ASE calling using data from individual loci (SNVs) within units of ASE (genes). Vector arguments 'lociAllele1Counts', 'lociAllele2Counts', 'lociAllele1NoASEProbs', 'lociRhos', and 'aseIDs' should all be of the same length. Letting i1, i2, .., iN denote the indices corresponding to entries within aseIDs equal to a given aseID, the entries at those indices in the other vector arguments provide information for the loci within that aseID. This information is then used by runMBASED1s1aseID. It is assumed that for any i, the i-th entries of all vector arguments correspond to the same locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype.

Usage

```
runMBASED1s(lociAllele1Counts, lociAllele2Counts, lociAllele1NoASEProbs,
  lociRhos, aseIDs, numSim = 0, BPPARAM = SerialParam(), isPhased = FALSE,
  tieBreakRandom = FALSE, checkArgs = FALSE)
```

Arguments

`lociAllele1Counts`, `lociAllele2Counts`
: vectors of counts of allele1 (e.g. reference) and allele2 (e.g., alternative) at individual loci. Allele counts are not necessarily phased (see argument 'isPhased'), so allele1 counts may not represent the same haplotype. Both arguments must be vectors of non-negative integers.

`lociAllele1NoASEProbs`
: probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus). Must be a vector with entries >0 and <1.

`lociRhos`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Must be a vector with entries >=0 and <1.

`aseIDs`
: the IDs of ASE units corresponding to the individual loci (e.g. gene names).

`numSim`
: number of simulations to perform. Must be a non-negative integer. If 0 (DEFAULT), no simulations are performed.

`BPPARAM`
: argument to be passed to bplapply(), when parallel achitecture is used to speed up simulations (parallelization is done over aseIDs). DEFAULT: SerialParam() (no parallelization).

`isPhased`
: single boolean specifying whether the phasing has already been performed, in which case the lociAllele1Counts represent the same haplotype. DEFAULT: FALSE.

`tieBreakRandom`
: single boolean specifying how ties should be broken during pseudo-phasing in cases of unphased data (isPhased=FALSE). If TRUE, each of the two allele will be assigned to major haplotype with probability=0.5. If FALSE (DEFAULT), allele1 will be assigned to major haplotype and allele2 to minor haplotype.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Value

list with 3 elements:

`ASEResults`
: Data frame with each row reporting MBASED results for a given aseID (aseIDs are provided as row names of this data frame). The columns of the data frame are: majorAlleleFrequency, pValueASE, heterogeneityQ, and pValueHeterogeneity.

`allele1IsMajor`
: Vector of TRUE/FALSE of length equal to the number of supplied SNVs, reporting for each SNV whether allele1 represents major (TRUE) or minor (FALSE) haplotype of the corresponding aseID.

`lociMAF`
: Vector of locus-specific estimates of the frequency of major allele, where 'major' refers to the haplotype of the gene found to be major by the ASE analysis. Note that since the determination of the major/minor status is done at the level of the gene, there may be loci with locus-specific MAF < 0.5.

Examples

```
SNVCoverage1 <- sample(10:100,5) ## gene with 5 loci
SNVAllele1Counts1 <- rbinom(length(SNVCoverage1), SNVCoverage1, 0.5)
SNVCoverage2 <- sample(10:100,5) ## gene with 5 loci
SNVAllele1Counts2 <- rbinom(length(SNVCoverage2), SNVCoverage2, 0.5)
MBASED:::runMBASED1s(lociAllele1Counts=c(SNVAllele1Counts1, SNVAllele1Counts2), lociAllele2Counts=c(SNVCoverage
```

`runMBASED1s1aseID` *Function that runs single-sample ASE calling using data from loci (SNVs) within a single unit of ASE (gene). The i-th entry of each of vector arguments 'lociAllele1Counts', 'lociAllele2Counts', 'lociAllele1NoASEProbs', 'lociRhos' should correspond to the i-th locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype. Note: for each locus, at least one allele should have >0 supporting reads.*

Description

Function that runs single-sample ASE calling using data from loci (SNVs) within a single unit of ASE (gene). The i-th entry of each of vector arguments 'lociAllele1Counts', 'lociAllele2Counts', 'lociAllele1NoASEProbs', 'lociRhos' should correspond to the i-th locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype. Note: for each locus, at least one allele should have >0 supporting reads.

Usage

```
runMBASED1s1aseID(lociAllele1Counts, lociAllele2Counts, lociAllele1NoASEProbs,
  lociRhos, numSim = 0, isPhased = FALSE, tieBreakRandom = FALSE,
  checkArgs = FALSE)
```

Arguments

`lociAllele1Counts`, `lociAllele2Counts`
: vectors of counts of allele1 (e.g. reference) and allele2 (e.g., alternative) at individual loci. Allele counts are not necessarily phased (see argument 'isPhased'), so allele1 counts may not represent the same haplotype. Both arguments must be vectors of non-negative integers.

`lociAllele1NoASEProbs`
: probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus). Must be a vector with entries >0 and <1.

`lociRhos`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Must be a vector with entries >=0 and <1.

`numSim`
: number of simulations to perform. Must be a non-negative integer. If 0 (DEFAULT), no simulations are performed.

isPhased — single boolean specifying whether the phasing has already been performed, in which case the lociAllele1Counts represent the same haplotype. DEFAULT: FALSE.

tieBreakRandom — single boolean specifying how ties should be broken during pseudo-phasing in cases of unphased data (isPhased=FALSE). If TRUE, each of the two allele will be assigned to major haplotype with probability=0.5. If FALSE (DEFAULT), allele1 will be assigned to major haplotype and allele2 to minor haplotype.

checkArgs — single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Value

list with 6 elements

majorAlleleFrequency
: Estimate of major allele frequency for this unit of ASE (gene).

pValueASE — Estimate of p-value for observed extent of ASE (nominal if no simulations are performed, simulations-based otherwise).

heterogeneityQ — Statistic summarizing variability of locus-specific estimates of major allele frequency if >1 locus is present. Set to NA for single-locus cases.

pValueHeterogeneity
: Estimate of p-value for observed extent of variability of locus-specific estimates of major allele frequency if >1 locus is present. Set to NA for single-locus cases.

lociAllele1IsMajor
: Vector of booleans, specifying for each locus whether allele1 is assigned to major (TRUE) or minor (FALSE) haplotype. If the data is phased (isPhased=TRUE), then all elements of the vector are TRUE if haplotype 1 is found to be major, and are all FALSE if haplotype 1 is found to be minor. In cases of unphased data (isPhased=FALSE), the assignment is provided by the pseudo-phasing procedure.

lociMAF — Estimate of major allele (haplotype) frequency at individual loci. Note that since 'major' and 'minor' distinction is made at the level of gene haplotype, there may be some loci where the frequency of the 'major' haplotype is <0.5.

Examples

```
SNVCoverage <- sample(10:100,5) ## gene with 5 loci
SNVAllele1Counts <- rbinom(length(SNVCoverage), SNVCoverage, 0.5)
MBASED:::runMBASED1s1aseID(lociAllele1Counts=SNVAllele1Counts, lociAllele2Counts=SNVCoverage-SNVAllele1Counts,
MBASED:::runMBASED1s1aseID(lociAllele1Counts=SNVAllele1Counts, lociAllele2Counts=SNVCoverage-SNVAllele1Counts,
MBASED:::runMBASED1s1aseID(lociAllele1Counts=SNVAllele1Counts, lociAllele2Counts=SNVCoverage-SNVAllele1Counts,
MBASED:::runMBASED1s1aseID(lociAllele1Counts=SNVAllele1Counts, lociAllele2Counts=SNVCoverage-SNVAllele1Counts,
```

runMBASED2s *Function that runs between-sample (differential) ASE calling using data from individual loci (SNVs) within units of ASE (genes). Vector arguments 'lociAllele1CountsSample1', 'lociAllele2CountsSample1', 'lociAllele1NoASEProbsSample1', 'lociRhosSample1', 'lociAllele1CountsSample2', 'lociAllele2CountsSample2', 'lociAllele1NoASEProbsSample2', 'lociRhosSample2', and 'aseIDs' should all be of the same length. Letting i1, i2, .., iN denote the indices corresponding to entries within aseIDs equal to a given aseID, the entries at those indices in the other vector arguments provide information for the loci within that aseID for the respective samples. This information is then used by runMBASED2s1aseID. It is assumed that for any i, the i-th entries of all vector arguments correspond to the same locus, and that the entries corresponding to allele1 in sample1 and sample2 provide information on the same allele. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype.*

Description

Function that runs between-sample (differential) ASE calling using data from individual loci (SNVs) within units of ASE (genes). Vector arguments 'lociAllele1CountsSample1', 'lociAllele2CountsSample1', 'lociAllele1NoASEProbsSample1', 'lociRhosSample1', 'lociAllele1CountsSample2', 'lociAllele2CountsSample2', 'lociAllele1NoASEProbsSample2', 'lociRhosSample2', and 'aseIDs' should all be of the same length. Letting i1, i2, .., iN denote the indices corresponding to entries within aseIDs equal to a given aseID, the entries at those indices in the other vector arguments provide information for the loci within that aseID for the respective samples. This information is then used by runMBASED2s1aseID. It is assumed that for any i, the i-th entries of all vector arguments correspond to the same locus, and that the entries corresponding to allele1 in sample1 and sample2 provide information on the same allele. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype.

Usage

```
runMBASED2s(lociAllele1CountsSample1, lociAllele2CountsSample1,
  lociAllele1CountsSample2, lociAllele2CountsSample2,
  lociAllele1NoASEProbsSample1, lociAllele1NoASEProbsSample2, lociRhosSample1,
  lociRhosSample2, aseIDs, numSim = 0, BPPARAM = SerialParam(),
  isPhased = FALSE, tieBreakRandom = FALSE, checkArgs = FALSE)
```

Arguments

`lociAllele1CountsSample1`, `lociAllele2CountsSample1`, `lociAllele1CountsSample2`, `lociAllele2CountsSample2`

vectors of counts of allele1 (e.g. reference) and allele2 (e.g. alternative) at indivdiual loci in sample1 and sample2. Allele counts are not necessarily phased

(see argument 'isPhased'), so allele1 counts may not represent the same haplotype. However, the two alleles (allele1 and allele2) must be defined identically for both samples at each locus. All 4 arguments must be vectors of non-negative integers.

`lociAllele1NoASEProbsSample1`, `lociAllele1NoASEProbsSample2`
: probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus) in sample1 and sample2, respectively. Note that these probabilities are allowed to be sample-specific. Each argument must be a vector with entries >0 and <1.

`lociRhosSample1`, `lociRhosSample2`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Note that the dispersions are allowed to be sample-specific. Each argument must be a vector with entries >=0 and <1.

`aseIDs`
: the IDs of ASE units corresponding to the individual loci (e.g. gene names).

`numSim`
: number of simulations to perform. Must be a non-negative integer. If 0 (DEFAULT), no simulations are performed.

`BPPARAM`
: argument to be passed to bplapply(), when parallel achitecture is used to speed up simulations (parallelization is done over aseIDs). DEFAULT: SerialParam() (no parallelization).

`isPhased`
: single boolean specifying whether the phasing has already been performed, in which case the lociAllele1CountsSample1 (and, therefore, lociAllele1CountsSample2) represent the same haplotype. DEFAULT: FALSE.

`tieBreakRandom`
: single boolean specifying how ties should be broken during pseudo-phasing in cases of unphased data (isPhased=FALSE). If TRUE, each of the two allele will be assigned to major haplotype with probability=0.5. If FALSE (DEFAULT), allele1 will be assigned to major haplotype and allele2 to minor haplotype.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Value

list with 3 elements:

`ASEResults`
: Data frame with each row reporting MBASED results for a given aseID (aseIDs are provided as row names of this data frame). The columns of the data frame are: majorAlleleFrequencyDifference, pValueASE, heterogeneityQ, and pValueHeterogeneity.

`allele1IsMajor`
: Vector of TRUE/FALSE of length equal to the number of supplied SNVs, reporting for each SNV whether allele1 represents major (TRUE) or minor (FALSE) haplotype of the corresponding aseID.

`lociMAFDifference`
: Vector of locus-specific estimates of the difference of major allele (haplotype) frequency between the two samples. Note that 'major' and 'minor' distinction is made at the level of gene haplotype in sample1.

Examples

```
SNVCoverageTumor=sample(10:100, 5)
SNVCoverageNormal=sample(10:100, 5)
SNVAllele1CountsTumor=rbinom(length(SNVCoverageTumor), SNVCoverageTumor, 0.5)
SNVAllele1CountsNormal=rbinom(length(SNVCoverageNormal), SNVCoverageNormal, 0.5)
MBASED:::runMBASED2s(lociAllele1CountsSample1=SNVAllele1CountsTumor, lociAllele2CountsSample1=SNVCoverageTumor-
```

`runMBASED2s1aseID` *Function that runs between-sample (differential) ASE calling using data from loci (SNVs) within a single unit of ASE (gene). The i-th entry of each of vector arguments 'lociAllele1CountsSample1', 'lociAllele2CountsSample1', 'lociAllele1NoASEProbsSample1', 'lociRhosSample1', 'lociAllele1CountsSample2', 'lociAllele2CountsSample2', 'lociAllele1NoASEProbsSample2', and 'lociRhosSample2' should correspond to the i-th locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype. Note: for each locus in each sample, at least one allele should have >0 supporting reads.*

Description

Function that runs between-sample (differential) ASE calling using data from loci (SNVs) within a single unit of ASE (gene). The i-th entry of each of vector arguments 'lociAllele1CountsSample1', 'lociAllele2CountsSample1', 'lociAllele1NoASEProbsSample1', 'lociRhosSample1', 'lociAllele1CountsSample2', 'lociAllele2CountsSample2', 'lociAllele1NoASEProbsSample2', and 'lociRhosSample2' should correspond to the i-th locus. If argument 'isPhased' (see below) is true, then entries corresponding to allele1 at each locus must represent the same haplotype. Note: for each locus in each sample, at least one allele should have >0 supporting reads.

Usage

```
runMBASED2s1aseID(lociAllele1CountsSample1, lociAllele2CountsSample1,
  lociAllele1CountsSample2, lociAllele2CountsSample2,
  lociAllele1NoASEProbsSample1, lociAllele1NoASEProbsSample2, lociRhosSample1,
  lociRhosSample2, numSim = 0, isPhased = FALSE, tieBreakRandom = FALSE,
  checkArgs = FALSE)
```

Arguments

`lociAllele1CountsSample1`, `lociAllele2CountsSample1`, `lociAllele1CountsSample2`, `lociAllele2CountsSample2`

vectors of counts of allele1 (e.g. reference) and allele2 (e.g. alternative) at individiual loci in sample1 and sample2. Allele counts are not necessarily phased (see argument 'isPhased'), so allele1 counts may not represent the same haplotype. However, the two alleles (allele1 and allele2) must be defined identically for both samples at each locus. All 4 arguments must be vectors of non-negative integers.

`lociAllele1NoASEProbsSample1`, `lociAllele1NoASEProbsSample2`
: probabilities of observing allele1-supporting reads at individual loci under conditions of no ASE (e.g., vector with all entries set to 0.5, if there is no pre-existing allelic bias at any locus) in sample1 and sample2, respectively. Note that these probabilities are allowed to be sample-specific. Each argument must be a vector with entries >0 and <1.

`lociRhosSample1`, `lociRhosSample2`
: dispersion parameters of beta distribution at individual loci (set to 0 if the read count-generating distribution at the locus is binomial). Note that the dispersions are allowed to be sample-specific. Each argument must be a vector with entries >=0 and <1.

`numSim`
: number of simulations to perform. Must be a non-negative integer. If 0 (DEFAULT), no simulations are performed.

`isPhased`
: single boolean specifying whether the phasing has already been performed, in which case the lociAllele1CountsSample1 (and, therefore, lociAllele1CountsSample2) represent the same haplotype. DEFAULT: FALSE.

`tieBreakRandom`
: single boolean specifying how ties should be broken during pseudo-phasing in cases of unphased data (isPhased=FALSE). If TRUE, each of the two allele will be assigned to major haplotype with probability=0.5. If FALSE (DEFAULT), allele1 will be assigned to major haplotype and allele2 to minor haplotype.

`checkArgs`
: single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Value

list with 7 elements

`majorAlleleFrequencyDifference`
: Estimate of major allele frequency difference for this unit of ASE (gene). 'Major' here refers to the allelic imbalance within sample1, and the difference is defined as Frequency(major, sample1)-Frequency(major, sample2).

`pValueASE`
: Estimate of p-value for observed extent of ASE (nominal if no simulations are performed, simulations-based otherwise).

`heterogeneityQ`
: Statistic summarizing variability of locus-specific estimates of major allele frequency difference if >1 locus is present. Set to NA for single-locus cases.

`pValueHeterogeneity`
: Estimate of p-value for observed extent of variability of locus-specific estimates of major allele frequency difference if >1 locus is present. Set to NA for single-locus cases.

`lociAllele1IsMajor`
: Vector of booleans, specifying for each locus whether allele1 is assigned to major (TRUE) or minor (FALSE) haplotype (where 'major' and 'minor' refer to abundances in sample1). If the data is phased (isPhased=TRUE), then all elements of the vector are TRUE if haplotype 1 is found to be major in sample1, and are all FALSE if haplotype 1 is found to be minor. In cases of unphased data (isPhased=FALSE), the assignment is provided by the pseudo-phasing procedure within sample1.

nullHypothesisMAF
: Estimate of major allele frequency under the null hypothesis that allelic frequencies are the same in both samples. This estimate is obtained by maximum likelihood, and, in case of unphased data (isPhased=FALSE), the likelihood is further maximized over all possible assignments of alleles to haplotypes.

lociMAFDifference
: Estimate of the difference of major allele (haplotype) frequency at individual loci. Note that 'major' and 'minor' distinction is made at the level of gene haplotype in sample1.

Examples

```
SNVCoverageTumor=sample(10:100, 5) ## gene with 5 loci
SNVCoverageNormal=sample(10:100, 5)
SNVAllele1CountsTumor=rbinom(length(SNVCoverageTumor), SNVCoverageTumor, 0.5)
SNVAllele1CountsNormal=rbinom(length(SNVCoverageNormal), SNVCoverageNormal, 0.5)
MBASED:::runMBASED2s1aseID(lociAllele1CountsSample1=SNVAllele1CountsTumor, lociAllele2CountsSample1=SNVCoverage
```

---

shiftAndAttenuateProportions

Helper function to adjust proportions for pre-existing allelic bias and also to obtain estimate of proportion variance based on attenuated read counts (adding pseudocount of 0.5 to each allele in each sample).

Description

Helper function to adjust proportions for pre-existing allelic bias and also to obtain estimate of proportion variance based on attenuated read counts (adding pseudocount of 0.5 to each allele in each sample).

Usage

```
shiftAndAttenuateProportions(countsMat, totalsMat, probsMat, rhosMat,
  checkArgs = FALSE)
```

Arguments

countsMat
: matrix of observed major allele counts. Each row represents a specific genomic locus, while each column represents a set of observed major allele counts across loci (in practice, multiple columns represent different outcomes of count simulations).

totalsMat
: matrix of total read counts across both alleles. The interpretation of rows and columns is the same as for countsMat.

probsMat
: matrix of underlying probabilites of observing the major allele. The interpretation of rows and columns is the same as for countsMat.

rhosMat	matrix of dispersion parameters of beta distributions for each locus. The interpretation of rows and columns is the same as for countsMat.
checkArgs	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE

Value

a list with 2 elements:

propsShifted	a 1-row marix of shifted major allele frequencies
propsShiftedVars	a 1-row matrix of estimated variances of obtained MAF estimates

Examples

```
SNVCoverageTumor=sample(10:100,10) ## 2 genes with 5 loci each
SNVAllele1CountsTumor=rbinom(length(SNVCoverageTumor), SNVCoverageTumor, 0.5)
MBASED:::shiftAndAttenuateProportions(countsMat=matrix(SNVAllele1CountsTumor, ncol=2), totalsMat=matrix(SNVCove
```

---

testNumericDiff *Function that checks to see if the difference between 2 number is small enough.*

Description

Function that checks to see if the difference between 2 number is small enough.

Usage

```
testNumericDiff(queryVals, targetVals, cutoffFraction)
```

Arguments

queryVals, targetVals	vectors of values to be compared (pairwise comparison will be performed)
cutoffFraction	the value of cutoff to be used to declare if the two numbers are close enough.

Details

for 2 numbers a and b, the function checks to see if |a-b|/min(a,b) <= cutoff.

Value

vector of same length as input vectors queryVals and targetVals, recording for each pair of numbers whether they pas the cutoff (TRUE) or not (FALSE).

See Also

Other unitTestsFunctions: testQuantiles

`testQuantiles` *Function to test quantile equality for theoretical and observed distributions*

Description

Function to test quantile equality for theoretical and observed distributions

Usage

```
testQuantiles(theoreticalCumDist, observedCumDist, numTotalCounts,
  numSEsToCheck, errorMessage)
```

Arguments

`theoreticalCumDist`	for (unspecified) value of x, P(X<=x)
`observedCumDist`	for (unspecified) value of x, observed Fraction(values<=x) = Num(values<=x)/Num(total values). Actual values of x must be the same as those for corresponding entries in theoreticalCumDist
`numTotalCounts`	Num(total values) (see argument observedCumDist)
`numSEsToCheck`	number of standard errors to go in each direction from theoretical quantity to see if the estimate falls into the confidence interval
`errorMessage`	error message to return if observed fraction falls outside of confidence interval

Details

For some random variable X, observed sample x1, x2, .., xN, and attainable value x, we compare theoretical P(X<=x) to observed Num(xi <= x)/N.

Value

TRUE (all tests were passed, otherwise exits with error message).

See Also

Other unitTestsFunctions: `testNumericDiff`

`vectorizedRbetabinomAB` *Functions to generate beta-binomial random variables.*

Description

Functions to generate beta-binomial random variables.

Usage

```
vectorizedRbetabinomAB(n, size, a, b, checkArgs = FALSE)

vectorizedRbetabinomMR(n, size, mu, rho, checkArgs = FALSE)
```

Arguments

`n`	sample size, must be a single positive integer
`size`	number of trials for each count to be generated in the sample, must be a vector of positive integers
`a, b`	vectors of shape parameters for beta distributions used to generate probability of success for each count to be generated in the sample, must be >0
`checkArgs`	single boolean specifying whether arguments should be checked for adherence to specifications. DEFAULT: FALSE
`mu, rho`	mean (a/(a+b)) and dispersion (1/(a+b+1)) parameters for beta distribution, must be in (0,1). Value of 0 is allowed for rho and implies binomial distribution.

Details

vectorizedRbetabinomAB is the same function as rbetabinom.ab from VGAM package but it avoids a lot of overhang and requires that arguments size, a (shape1), and b (shape2) be of length equal to argument n.

vectorizedRbetabinomMR is a wrapper around vectorizedRbetabinomAB using mu/rho parametrization. Requires that arguments size, mu, and rho be of length equal to argument n.

Value

a numeric vector of betabinomial random variables.

See Also

Other bbFunctions: `getAB`, `getAB`, `getMuRho`

Other bbFunctions: `getAB`, `getAB`, `getMuRho`

Examples

```
set.seed(111)
MBASED:::vectorizedRbetabinomAB(n=10, size=rep(50,10), a=rep(1,10), b=rep(1,10))
set.seed(111)
MBASED:::vectorizedRbetabinomMR(n=10, size=rep(50,10), mu=rep(1/2,10), rho=rep(1/3,10))
```

Index

estimateMAF1s, 2
estimateMAF2s, 3

FT, 5
FTAdjust *(*FT*)*, 5

getAB, *36*
getAB *(*getMuRho*)*, 6
getMuRho, 6, *36*
getPFinal, 7
getSimulationPvalue, 8

isCountMajorFT *(*FT*)*, 5

logLikelihoodCalculator1s, 9
logLikelihoodCalculator2s, 10

maxLogLikelihoodCalculator1s, 11
maxLogLikelihoodCalculator2s, 12
MBASED, 14
MBASED-package *(*MBASED*)*, 14
MBASEDMetaAnalysis, 14
MBASEDMetaAnalysisGetMeansAndSEs, 15
MBASEDVectorizedMetaprop, 17
MBASEDVectorizedPropDiffTest, 19

runMBASED, 20
runMBASED1s, 25
runMBASED1s1aseID, 27
runMBASED2s, 29
runMBASED2s1aseID, 31

shiftAndAttenuateProportions, 33

testNumericDiff, 34, *35*
testQuantiles, *34*, 35

unFT *(*FT*)*, 5

vectorizedRbetabinomAB, *7*, 36
vectorizedRbetabinomMR, *7*
vectorizedRbetabinomMR
*(*vectorizedRbetabinomAB*)*, 36