Package 'Umpire'

January 6, 2011

Version 1.0.0

Date 2010-09-23

Title Umpire Package

Author Kevin R. Coombes, Jiexin Zhang

Maintainer Kevin R. Coombes <krcoombes@mdanderson.org>

Description The Ultimate Microrray Prediction, Reality and Inference Engine (UMPIRE) is a package to facilite the simulation of realistic microarray dataset.

Depends R (>= 2.10), methods, stats

License file LICENSE

LazyLoad yes

R topics documented:

`Umpire-package` *UMPIRE: Ultimate Microarray Prediction, Inference, and Reality Engine*

Description

A suite microarray simulation software which includes additive and multiplicative noise, mixture of expressed and unexpressed genes, and uses statistical distributions to capture differences in mean expression and in standard deviation both within groups and between groups of samples. Finally, we incorporate a simple block correlation structure between genes.

Details

Package:	Umpire
Type:	Package
Version:	1.0
Date:	2009-07-13
License:	Perl Artistic License (see LICENSE file)
LazyLoad:	yes

For a complete list of functions, use `library(help = 'Umpire')`.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

`alterObjectComponents-method`
Functions for alter components in the list("Engine") object

Description

`alterMean` and `alterSD` are functions that alter means or standard deviations in the `Engine` object. `normalOffset` is a possible `TRANSFORM` to be used in an `alterMean` operation, which adds an offset to each value in the mean where the offset is chosen from a normal distribution. `invGammaMultiple` is a possible `TRANSFORM` to be used in an `alterSD` operation, which multiplies each standard deviation by a positive value chosen from an inverse gamma distribution with paramters `shape` and `scale`.

Usage

```
alterMean(object, TRANSFORM, ...)
alterSD(object,TRANSFORM,...)
normalOffset(x, delta, sigma)
invGammaMultiple(x, shape, rate)
```

Arguments

object	object of class Engine
TRANSFORM	the TRANSFORM function for each object should take as its input a vector of mean expression or standard deviation and return a transformed vector that can be used to alter the appropriate slot of the object.
x	numeric vector of mean expression or standard deviation defined in the Engine object
delta, sigma	mean and sd parameters specifying the normal distribution
shape, rate	shape and rate parameters specifiying the gamma distribution. Must be positive.
...	extra arguments for generic or plotting routines

Value

alterMean and alterSD return a modified object of class Engine with corresponding components altered.

normalOffset returns a new vector, each element of which is added by a offset chosen from a normal distribution with parameters mean and sd invGammaMultiple returns a new vector, each element of which is multiplied by a positive value chosen from an inverse gamma distribution with parameters shape and scale

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

Examples

```
nComp <- 10
nGenes <- 100
comp <- list()
for (i in 1:nComp){
  comp[[i]] <- IndependentNormal(rnorm(nGenes/nComp,6,1.5),1/rgamma(nGenes/nComp,44,28))
}
myEngine <- Engine(comp)
nrow(myEngine)
nComponents(myEngine)
summary(myEngine)
myData <- rand(myEngine,5)
dim(myData)
summary(myData)
MEAN <- 2
SD <- 2
myEngine.alterMean <- alterMean(myEngine,function(x)normalOffset(x,
MEAN, SD))
myData.alterMean <- rand(myEngine.alterMean,5)
summary(myData.alterMean)
RATE <- 1
SHAPE <- 2
myEngine.alterSD <- alterSD(myEngine,function(x)invGammaMultiple(x,SHAPE,RATE))
myData.alterSD <- rand(myEngine.alterSD,5)
summary(myData.alterSD)
```

blur-method *Method "blur"*

Description

This method could be thought of as a ...

Usage

```
blur(object, x, ...)
```

Arguments

`object`	object of implementing class
`x`	TBD
`...`	extra arguments for generic or plotting routines

Author(s)

P. Roebuck <plroebuck@mdanderson.org>

CancerEngine-class *Class "CancerEngine" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("CancerEngine", ...)`. ~~ describe objects here ~~

Slots

`base`**:** Object of class `"character"` ~~

`altered`**:** Object of class `"character"` ~~

Methods

rand `signature(object = "CancerEngine")`: ...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

CancerModel

Examples

```
showClass("CancerEngine")
```

CancerModel-class *The "CancerModel" Class*

Description

A CancerModel object contains a number of pieces of information representing an abstract, heterogeneous collection of cancer patients

Usage

```
CancerModel(name, nPossible, nPattern,
                HIT = function(n) 5,
                SURV = function(n) rnorm(n, 0, 2),
                OUT = function(n) rnorm(n, 0, 2),
                prevalence=NULL)
nPatterns(object)
nPossibleHits(object)
nHitsPerPattern(object)
outcomeCoefficients(object)
survivalCoefficients(object)
```

Arguments

name	~~Describe name here~~
object	~~Describe object here~~
NULL	~~Describe NULL here~~
nPossible	~~Describe nPossible here~~
nPattern	~~Describe nPattern here~~
HIT	~~Describe HIT here~~
SURV	~~Describe SURV here~~
OUT	~~Describe OUT here~~
prevalence	~~Describe prevalence here~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Objects from the Class

Objects can be created by calls of the form `new("CancerModel", ...)`. ~~ describe objects here ~~

Slots

`name`: Object of class `"character"` ~~

`hitPattern`: Object of class `"matrix"` ~~

`survivalBeta`: Object of class `"numeric"` ~~

`outcomeBeta`: Object of class `"numeric"` ~~

`prevalence`: Object of class `"numeric"` ~~

`call`: Object of class `"call"` ~~

Methods

ncol `signature(x = "CancerModel")`: ...

nrow `signature(x = "CancerModel")`: ...

rand `signature(object = "CancerModel")`: ...

summary `signature(object = "CancerModel")`: ...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

`SurvivalModel`

Examples

```
showClass("CancerModel")
```

CancerPatientSet-class
Class "CancerPatientSet" ~~~

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Usage

```
CancerPatientSet(object, n)
## S4 method for signature 'CancerPatientSet':
as.data.frame(x, row.names = NULL, optional = FALSE)
```

Arguments

`object`	~~Describe `object` here~~
`n`	~~Describe `n` here~~
`x`	~~Describe `x` here~~
`row.names`	~~Describe `row.names` here~~
`optional`	~~Describe `optional` here~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Objects from the Class

Objects can be created by calls of the form `new("CancerPatientSet", ...)`. ~~ describe objects here ~~

Slots

`parent`: Object of class `"CancerModel"` ~~

`hitClass`: Object of class `"numeric"` ~~

Methods

as.data.frame `signature(x = "CancerPatientSet")`: ...

summary `signature(object = "CancerPatientSet")`: ...

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

CancerModel

Examples

```
showClass("CancerPatientSet")
```

---

covar-method *Methods covar and correl*

Description

Functions to extract covariance and correlation matrix specified in the MVN object.

Usage

```
correl(object)
covar(object)
```

Arguments

object	object of class MVN

Details

covar and correl functions calculate the covariance matrix and correlation matrix underlies the covariance matrix for the objects of class MVN, respectively. We have four assertions as shown below, and will be tested in the examples section: Assertion 1: covar should return the same matrix that was used in the function call to construct the MVN object. Assertion 2: After applying an "alterMean" function (below), the covariance matrix is unchanged. Assertion 3: The diagonal of correlation matrix consists of all 1's. Assertion 4: After applying an "alterMean" or an "alterSD" function (below), the correlation matrix is unchanged.

Methods

covar(object) returns the covariance matrix of the object of class MVN

correl(object) returns the correlation matrix of the object of class MVN

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

MVN

`Engine-class` *The Engine Class*

Description

The `Engine` class is a tool (ie. an algorithm) used to simulate vectors of gene expression from some underlying distribution.

Usage

```
Engine(components)
## S4 method for signature 'Engine':
alterMean(object, TRANSFORM, ...)
## S4 method for signature 'Engine':
alterSD(object, TRANSFORM, ...)
## S4 method for signature 'Engine':
nrow(x)
## S4 method for signature 'Engine':
rand(object, n, ...)
## S4 method for signature 'Engine':
summary(object, ...)
nComponents(object)
```

Arguments

`components`	object of class `list`, each element of which contains the parameters for the underlying distribution that the gene expression follows. A component can be viewed as a special case of an engine that only has one component.
`object, x`	object of class `Engine`
`TRANSFORM`	the `TRANSFORM` function for each object should take as its input a vector of mean expression or standard deviation and return a transformed vector that can be used to alter the appropriate slot of the object.
`n`	Number of samples to be simulated
`...`	extra arguments for generic or plotting routines

Value

The `Engine` generator returns an object of class `Engine`.

The `alterMean` returns an object of class `Engine` with altered mean

The `alterSD` returns an object of class `Engine` with altered standard deviation

The `nrow` returns the number of genes (i.e, the length of the vector) the `Engine` object will generate.

The `rand` returns nrow(Engine)*n matrix representing the expressions of nrow(Engine) genes and n samples

The `summary` simply prints out the number of components included in the `Engine` object

The `nComponents` returns the number of components in the `Engine` object

Objects from the Class

Objects can be created by calls of the form `new("Engine", components=components)`, or use the `Engine` generator function. Every engine is an ordered list of components, which generates a contiguous subvector of the total vector of gene expression.

Details

In most cases, an engine object is an instantiation of a more general family or class that we call an ABSTRACT ENGINE. Every abstract engine is an ordered list of components, which can also be thought of as an engine with parameters. We instantiate an engine by binding all the free parameters of an abstract engine to actual values. Note that partial binding (of a subset of the free parameters) produces another abstract engine.

For all practical purposes, a COMPONENT should be viewed as an irreducible abstract engine. Every component must have an IDENTIFIER that is unique within the context of its enclosing abstract engine. The identifer may be implicitly taken to be the position of the component in the ordered list.

We interpret an `Engine` as the gene expression generator for a homogenous population; effects of cancer on gene expression are modeled at a higher level.

Methods

alterMean(object, TRANSFORM,...) takes an object of class `Engine`,loop over the components in the `Engine`, alter the mean as defined by TRANSFORM function, and returns a modified object of class `Engine`

alterSD(object, TRANSFORM,...) takes an object of class `Engine`,loop over the components in the `Engine`, alter the standard deviation as defined by TRANSFORM function, and returns a modified object of class `Engine`

nrow(x) counts the total number of genes (i.e, the length of the vector the `Engine` will generate)

rand(object, n, ...) generates nrow(Engine)*n matrix representing gene expressions of n samples following the underlying distribution captured in the object of `Engine`

summary(object, ...) simply prints out the number of components included in the object of `Engine`

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

`EngineWithActivity`

Examples

```
nComp <- 10
nGenes <- 100
comp <- list()
for (i in 1:nComp){
  comp[[i]] <- IndependentNormal(rnorm(nGenes/nComp,6,1.5),1/rgamma(nGenes/nComp,44,28))
}
```

```
myEngine <- Engine(comp)
nrow(myEngine)
nComponents(myEngine)
summary(myEngine)
myData <- rand(myEngine,5)
dim(myData)
summary(myData)
OFFSET <- 2
myEngine.alterMean <- alterMean(myEngine,function(x){x+OFFSET})
myData.alterMean <- rand(myEngine.alterMean,5)
summary(myData.alterMean)
SCALE <- 2
myEngine.alterSD <- alterSD(myEngine,function(x){x*SCALE})
myData.alterSD <- rand(myEngine.alterSD,5)
summary(myData.alterSD)
```

`EngineWithActivity-class`
The EngineWithActivity Class

Description

The `EngineWithActivity` is used to set some components in the object of class `Engine` to be transcriptionally inactive and transform the expression data to appropriate logarithmic scale.

Usage

```
EngineWithActivity(active, components, base = 2)
## S4 method for signature 'EngineWithActivity':
alterMean(object, TRANSFORM, ...)
## S4 method for signature 'EngineWithActivity':
alterSD(object, TRANSFORM, ...)
## S4 method for signature 'EngineWithActivity':
rand(object,n,...)
## S4 method for signature 'EngineWithActivity':
summary(object, ...)
## S4 method for signature 'EngineWithActivity':
nrow(x)
```

Arguments

`active`	Object of class `logical` with length equaling to the number of components specifying whether each component should be transcriptionally active or not. Or, `active` can be a numeric within the range of 0 to 1 specifying the probability for a component to be active.
`components`	Object of class `list`, each element of which contains the parameters for the underlying distribution that the gene expression follows.
`base`	A numeric specifying the logarithmic scale to which the data should be transformed.
`object, x`	Object of class `EngineWithActivity`

TRANSFORM	the TRANSFORM function for each object should take as its input a vector of mean expression or standard deviation and return a transformed vector that can be used to alter the appropriate slot of the object.
n	Number of samples to be simulated
...	extra arguments for generic or plotting routines

Value

The EngineWithActivity generator returns an object of class EngineWithActivity with slots described in Slots section.

The alterMean returns an object of class EngineWithActivity with altered mean

The alterSD returns an object of class EngineWithActivity with altered standard deviation

The rand returns nrow(EngineWithActivity)*n gene expression matrix with the inactive components being masked by 0.

The summary prints out the total number of components and the number of active components in the object of EngineWithActivity

The nrow returns the number of genes (i.e, the length of the vector) the EngineWithActivity object will generate.

Objects from the Class

Objects can be created by the use of the EngineWithActivity generator function.

Slots

active: Either an object of class logical specifying whether each component should be transcriptionally active or not, or a numeric specifying the probability for a component to be transcriptionally active

base: The logarithmic scale.

components: Object of class "list" specifying the parameters of the underlying distribution.

Details

An ENGINE WITH ACTIVITY allows for the possibility that some components (or genes) in an expression engine (or tissue) might be transcriptionally inactive. Thus, the true biological signal S_gi should really be viewed as a mixture:

S_gi = z_g * delta_0 + (1 - z_g) * T_gi

where delta_0 = a point mass at zero; T_gi = a random variable supported on the positive real line; z_g ~ Binom(pi) defines the activity state (1 = on, 0 = off)

The 'rand' method for an EngineWithActivity is a little bit tricky, since we do two things at once. First, we use the 'base' slot to exponentiate the random variables generated by the underlying Engine on the log scale. We treat base = 0 as a special case, which means that we should continue to work on the scale of the Engine. Second, we mask any inactive component by replacing the generated values with 0.

Note that this is terribly inefficient if we only have a single homogeneous population, since we generate a certain amount of data only to throw it away. The power comes when we allow cancer disregulation to turn a block on or off, when the underlying data reappears.

Methods

alterMean(object, TRANSFORM,...) takes an object of class `EngineWithActivity`,loop over the components and alter the mean as defined by TRANSFORM function, and returns a object of class `EngineWithActivity` with modified components and the same active and base slots

alterSD(object, TRANSFORM,...) takes an object of class `EngineWithActivity`,loop over the components and alter the standard deviation as defined by TRANSFORM function, and returns a modified object of class `EngineWithActivity` with modified components and the same active and base slots

rand(object,n,...) generates nrow(EngineWithActivity)*n matrix representing gene expressions of n samples, and the transcriptionally inactive components are masked by 0

summary(object,...) prints out the total number of components and the number of active components in the object of `EngineWithActivity`

nrow(x) counts the total number of genes (i.e, the length of the vector the `EngineWithActivity` will generate)

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

`Engine`

Examples

```
nComponents <- 10
nGenes <- 100
active <- 0.7
comp <- list()
for (i in 1:nComponents){
  comp[[i]] <- IndependentNormal(rnorm(nGenes/nComponents,6,1.5),1/rgamma(nGenes/nCompone
}
myEngine <- EngineWithActivity(active,comp,2)
summary(myEngine)
myData <- rand(myEngine,5)
dim(myData)
```

IndependentLogNormal-class

The IndependentNormal Class

Description

The `IndependentLogNormal` class is a tool used to generate gene expressions that follow log normal distribution, because the true expression value follows log normal in our model

Usage

```
IndependentLogNormal(logmu,logsigma)
## S4 method for signature 'IndependentLogNormal':
nrow(x)
## S4 method for signature 'IndependentLogNormal':
rand(object, n, ...)
## S4 method for signature 'IndependentLogNormal':
summary(object, ...)
```

Arguments

`logmu`	Object of class `numeric` specifying the mean expression values on the logarithmic scale.
`logsigma`	Object of class `numeric` specifying the standard deviation of the gene expression values on the logarithmic scale
`object, x`	Object of class `IndependentLogNormal`
`n`	Number of samples to be simulated
`...`	extra arguments for generic or plotting routines

Objects from the Class

Objects can be created by using the `IndependentLogNormal` generator function. The object of class `IndependentLogNormal` contains the mean and standard deviation on logarithmic scale for the log normal distribution.

Slots

`logmu`**:** Object of class `"numeric"` that contains the mean expression values on the logarithmic scale

`logsigma`**:** Object of class `"numeric"` that contains the standard deviation of the gene expression values on the logarithmic scale.

Methods

nrow(x) `signature(object = "IndependentLogNormal")`: Returns the number of genes (i.e, the length of the `logmu` vector)

rand(object, n, ...) generates nrow(IndependentLogNormal)*n matrix representing gene expressions of n samples following log normal distribution captured in the object of `IndependentLogNormal`

summary(object, ...) prints out the number of independent log normal random variables in the object of `IndependentLogNormal`

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

Engine,IndependentNormal, MVN

Examples

```
nGenes <- 20
logmu <- rnorm(nGenes, 6, 1)
logsigma <- 1/rgamma(nGenes, rate=14, shape=6)
ln <- IndependentLogNormal(logmu, logsigma)
nrow(ln)
summary(ln)
if(any(logmu - ln@logmu)) {
  print('means do not match')
} else {
  print('means verified')
}
if(any(logsigma - ln@logsigma)) {
  print('standard deviations do not match')
} else {
  print('sd verified')
}
x <- rand(ln, 1000)
print(dim(x))

print(paste("'ln' should be valid:", validObject(ln)))
ln@logsigma <- 1:3 # now we break it
print(paste("'ln' should not be valid:", validObject(ln, test=TRUE)))
tmp.sd<-sqrt(apply(log(x),1,var))
plot(tmp.sd,logsigma)
tmp.mu<-apply(log(x),1,mean)
plot(tmp.mu,logmu)
rm(nGenes, logmu, logsigma, ln, x, tmp.mu, tmp.sd)
```

`IndependentNormal-class`

The IndependentNormal Class

Description

The `IndependentNormal` class is a tool used to generate gene expressions that follow independent normal distribution

Usage

```
IndependentNormal(mu,sigma)
## S4 method for signature 'IndependentNormal':
alterMean(object, TRANSFORM, ...)
## S4 method for signature 'IndependentNormal':
alterSD(object, TRANSFORM, ...)
## S4 method for signature 'IndependentNormal':
nrow(x)
## S4 method for signature 'IndependentNormal':
rand(object, n, ...)
## S4 method for signature 'IndependentNormal':
summary(object, ...)
```

Arguments

mu	Object of class numeric specifying the mean expression values.
sigma	Object of class numeric specifying the standard deviation of the gene expression values
object, x	Object of class IndependentNormal
TRANSFORM	the TRANSFORM function for each object should take as its input a vector of mean expression or standard deviation and return a transformed vector that can be used to alter the appropriate slot of the object.
n	Number of samples to be simulated
...	extra arguments for generic or plotting routines

Objects from the Class

Objects can be created by using the IndependentNormal generator function. The object of class IndependentNormal contains the mean and standard deviation for the normal distribution

Slots

mu: Object of class "numeric" that contains the mean expression values

sigma: Object of class "numeric" that contains the standard deviation of the gene expression values.

Details

Note that we typically work on expression value with its logarithm to some appropriate base. That is, the independent normal should be used on the logarithmic scale in order to contruct engine.

Methods

alterMean(object, TRANSFORM,...) takes an object of class IndependentNormal,loop over the mu slot, alter the mean as defined by TRANSFORM function, and returns an object of class IndependentNormal with altered mu

alterSD(object, TRANSFORM,...) takes an object of class IndependentNormal,loop over the sigma slot, alter the standard deviation as defined by TRANSFORM function, and returns an object of class IndependentNormal with altered sigma

nrow(x) signature(object = "IndependentLogNormal"): Returns the number of genes (i.e, the length of the mu vector)

rand(object, n, ...) generates nrow(IndependentNormal)*n matrix representing gene expressions of n samples following the normal distribution captured in the object of IndependentNormal

summary(object, ...) prints out the number of independent normal random variables in the object of IndependentNormal

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

Engine, IndependentLogNormal, MVN

Examples

```
nGenes <- 20
  mu <- rnorm(nGenes, 6, 1)
  sigma <- 1/rgamma(nGenes, rate=14, shape=6)
  ind <- IndependentNormal(mu, sigma)
  nrow(ind)
  summary(ind)
  if(any(mu - ind@mu)) {
    print('means do not match')
  } else {
    print('means verified')
  }
  if(any(sigma - ind@sigma)) {
    print('standard deviations do not match')
  } else {
    print('sd verified')
  }
  x <- rand(ind, 3)
  print(dim(x))
  print(summary(x))
  print(paste("'ind' should be valid:", validObject(ind)))
  ind@sigma <- 1:3 # now we break it
  print(paste("'ind' should not be valid:", validObject(ind, test=TRUE)))
  rm(nGenes, mu, sigma, ind, x)
```

MVN-class *The MV Class*

Description

The MVN class is a tool used to generate gene expressions that follow multivariate normal distribution

Usage

```
MVN(mu, Sigma, tol = 1e-06)
## S4 method for signature 'MVN':
alterMean(object, TRANSFORM, ...)
## S4 method for signature 'MVN':
alterSD(object, TRANSFORM, ...)
## S4 method for signature 'MVN':
nrow(x)
## S4 method for signature 'MVN':
rand(object, n, ...)
## S4 method for signature 'MVN':
summary(object, ...)
## S4 method for signature 'MVN':
covar(object)
```

```
## S4 method for signature 'MVN':
correl(object)
```

Arguments

mu	k-dimentional mean vector
Sigma	k*k covariance matrix containing the measurement of the linear coupling between every pair of randome vectors.
tol	Roundoff error that will be tolerated when assessing the singularity of the covariance matrix
object, x	Object of class MVN
TRANSFORM	the TRANSFORM function for each object should take as its input a vector of mean expression or standard deviation and return a transformed vector that can be used to alter the appropriate slot of the object.
n	Number of samples to be simulated
...	extra arguments for generic or plotting routines

Objects from the Class

Objects can be created by using the MNV generator function.

Slots

mu: Object of class "numeric" containing the k-dimentional mean vector

lambda: Object of class "numeric" containing the square roots of eigenvalues of the covariance matrix

half: a k*k matrix whose columns containing the eigenvectors of the covariance matrix

Details

The implementation of MVN class is designed for efficiency when generating new samples, since we expect to do this several times. Basically, this class separates the 'mvrnorm' function from the 'MASS' library into several steps. The computationally expensive step (when the dimension is large) is the eigenvector decomposition of the covariance matrix. This step is performed at construction and the pieces are stored in the object. The rand method for MVN objects contains the second half of the 'mvrnorm' function from the 'MASS' library.

Note that we typically work on expression value with its logarithm to some appropriate base. That is, the multivariate normal should be used on the logarithmic scale in order to contruct engine.

alterMean for an MVN simply replaces the appropriate slot by the transformed vector. alterSD for an MVN is trickier, because of the way the data is stored. In order to have some hope of getting this correct, we work in the space of the covariance matrix, Sigma. If we let R denote the correlation matrix and let Delta be the diagonal matrix whose entries are the individual standard deviations, then Sigma = Delta standard deviations by replacing Delta in this product. We then construct a new 'MVN' object with the old mean vector and the new covariance matrix.

covar and correl functions calculate the covariance matrix and correlation matrix underlies the covariance matrix for the objects of class MVN, respectively. We have four assertions as shown below, and will be tested in the examples section: Assertion 1: covar should return the same matrix that was used in the function call to construct the MVN object. Assertion 2: After applying an "alterMean" function (below), the covariance matrix is unchanged. Assertion 3: The diagonal of correlation matrix consists of all 1's. Assertion 4: After applying an "alterMean" or an "alterSD" function (below), the correlation matrix is unchanged.

Methods

alterMean(object, TRANSFORM,...) takes an object of class `MVN`,loop over the `mu` slot, alter the mean as defined by TRANSFORM function, and returns an object of class `MVN` with altered `mu`

alterSD(object, TRANSFORM,...) takes an object of class `MVN`,works on the diagonal matrix of the covariance matrix, alter the standard deviation as defined by TRANSFORM function, and recontructs an object of class `MVN` with the old `mu` and reconstructed covariance matrix

nrow(x) returns the number of genes (i.e, the length of the `mu` vector)

rand(object, n, ...) generates nrow(MVN)*n matrix representing gene expressions of n samples following the multivariate normal distribution captured in the object of `MVN`

summary(object, ...) prints out the number of multivariate normal random variables in the object of `MVN`

covar(object) returns the covariance matrix of the object of class `MVN`

correl(object) returns the correlation matrix of the object of class `MVN`

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

See Also

Engine, IndependentNormal

Examples

```
## Not run:
tolerance <- 1e-10
  # create a random orthogonal 2x2 matrix
  a <- runif(1)
  b <- sqrt(1-a^2)
  X <- matrix(c(a, b, -b, a), 2, 2)
  # now choos random positive squared-eigenvalues
  Lambda2 <- diag(rev(sort(rexp(2))), 2)
  # construct a covariance matrix
  Y <- t(X)
  # Use the MVN constructor
  marvin <- MVN(c(0,0), Y)
  # check the four assertions
  print(paste('Tolerance for assertion checking:', tolerance))
  print(paste('Covar  assertion 1:',
              all(abs(covar(marvin) - Y) < tolerance)
              ))
  mar2 <- alterMean(marvin, normalOffset, delta=3)
  print(paste('Covar  assertion 2:',
              all(abs(covar(marvin) - covar(mar2)) < tolerance)
              ))
  print(paste('Correl assertion 1:',
              all(abs(diag(correl(marvin)) - 1) < tolerance)
              ))
```

```
  mar3 <- alterSD(marvin, function(x) 2*x)
  print(paste('Correl assertion 1:',
              all(abs(correl(marvin) - correl(mar2)) < tolerance)
              ))
  rm(a, b, X, Lambda2, Y, marvin, mar2, mar3)

## End(Not run)
```

`NoiseModel-class` *The "NoiseModel" class*

Description

A `NOISE MODEL` represents the additional machine noise that is layered on top of any biological variabilty when measuring the gene expression in a set of samples

Usage

```
NoiseModel(nu, tau, phi)
## S4 method for signature 'NoiseModel':
blur(object, x, ...)
```

Arguments

`nu`	The mean value for the additive noise
`tau`	The standard deviation for the additive noise
`phi`	The standard deviation for the multiplicative noise. Note the mean of multiplicative noise is set to 0
`object`	object of class `NoiseModel`
`x`	The data matrix containing true signal from the gene expression
`...`	extra arguments for generic or plotting routines

Details

We model both additive and multiplicative noise, so that the observed expression of gene g in sample i is given by: Y_gi = S_gi exp(H_gi) + E_gi, where Y_gi = observed expression, S_gi = true bilogical signal, H_gi ~ N(0, phi) defines the multiplicative noise, and E_gi ~ N(nu,tau) defines the additive noise. Note that we allow a systematic offset/bias in the additive noise model.

`blur` is the main method associated with a noise model. The main operation is given by blur(object, x), which adds and multiplies random noise to the data matrix "x" containing the true signal.

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

Examples

```
nComp <- 10
nGenes <- 100
comp <- list()
for (i in 1:nComp){
  comp[[i]] <- IndependentLogNormal(rnorm(nGenes/nComp,6,1.5),1/rgamma(nGenes/nComp,44,28
}
myEngine <- Engine(comp)
myData <- rand(myEngine,5)
summary(myData)

nu <- 10
tau <- 20
phi <- 0.1
nm <- NoiseModel(nu,tau,phi)
realData <- blur(nm, myData)
summary(realData)
```

---

rand-method *Method "rand"*

Description

Generate a matrix representing gene expressions following the distribution defined in the object.

Usage

```
rand(object, n, ...)
```

Arguments

`object`	The `Engine` object defining the distributions of the gene expression
`n`	The number of samples being simulated
`...`	extra arguments for generic or plotting routines

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

summary-method *Method "summary"*

Description

Print out summary information for object. Content being printed out depends on the object passed to the method.

Usage

```
summary(object, ...)
```

Arguments

object	object of class Engine
...	extra arguments for generic or plotting routines

Author(s)

Kevin R. Coombes <kcoombes@mdanderson.org>, Jiexin Zhang <jiexinzhang@mdanderson.org>

References

KRC

SurvivalModel-class *Class "SurvivalModel" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Usage

```
SurvivalModel(baseHazard = 1/5, accrual = 5, followUp = 1, units = 12, unitName
```

Arguments

baseHazard	~~Describe baseHazard here~~
accrual	~~Describe accrual here~~
followUp	~~Describe followUp here~~
units	~~Describe units here~~
unitName	~~Describe unitName here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Objects from the Class

Objects can be created by calls of the form `new("SurvivalModel", ...)`. ~~ describe objects here ~~

Slots

`baseHazard`**:** Object of class `"numeric"` ~~

`accrual`**:** Object of class `"numeric"` ~~

`followUp`**:** Object of class `"numeric"` ~~

`units`**:** Object of class `"numeric"` ~~

`unitName`**:** Object of class `"character"` ~~

Methods

No methods defined with class "SurvivalModel" in the signature.

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

See Also

~~objects to See Also as `help`, ~~~

Examples

```
showClass("SurvivalModel")
```

Index