Package 'Umpire'

August 4, 2017

Version 1.3.4

Date 2017-07-11

Title Simulating Realistic Gene Expression Data

Author Kevin R. Coombes, Jiexin Zhang

Maintainer Kevin R. Coombes <krc@silicovore.com>

Description The Ultimate Microrray Prediction, Reality and Inference Engine (UMPIRE) is a package to facilitate the simulation of realistic microarray data sets with link to associate outcomes. See Zhang and Coombes (2012) <doi:10.1186/1471-2105-13-S13-S1>.

Depends R (>= 3.0)

Imports methods, stats

Suggests mclust, survival

License Apache License (== 2.0)

URL http://oompa.r-forge.r-project.org/

NeedsCompilation no

R topics documented:

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

Description

A suite of 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, it incorporates a simple block correlation structure between genes.

Details

Package:	Umpire
Type:	Package
Version:	1.2.3
Date:	2011-12-01
License:	Artistic-2.0
LazyLoad:	yes

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

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

`alterMean-method` *Methods "alterMean" and "alterSD"*

Description

`alterMean` and `alterSD` are generic functions used to alter means or standard deviations respectively based on the input object. probability based on the input object. Each method invokes particular `methods` which depend on the `class` of the first argument.

Usage

```
## S4 method for signature 'ANY'
alterMean(object, TRANSFORM, ...)
## S4 method for signature 'ANY'
alterSD(object, TRANSFORM, ...)
```

Arguments

object	an object for which altering mean or standard deviation is desired
TRANSFORM	function that returns its transformed input
...	additional arguments affecting the specific transformation performed

Value

The form of the value returned by alterMean or alterSD depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

---

BlockHyperParameters-class

The "BlockHyperParameters" Class

Description

Provides tools to create a CancerEngine with block correlation structure. Also makes it possible to simulate paired clinical and gene expression data with this block structure.

Usage

```
BlockHyperParameters(nExtraBlocks=100,
                     meanBlockSize=100,
                     sigmaBlockSize=30,
                     minBlockSize=5,
                     mu0=6,
                     sigma0=1.5,
                     rate=28.11,
                     shape=44.25,
                     p.cor=0.6,
                     wt.cor=5)
makeBlockStructure(cm, hyperp, xform=normalOffset, ...)
```

Arguments

cm	object of class CancerModel
hyperp	object of class BlockHypeParameters
nExtraBlocks	integer scalar specifying number of blocks not involved in the "hit" structure defined by the CancerModel
meanBlockSize	numeric scalar specifying mean number of genes in a correlated block
sigmaBlockSize	numeric scalar specifying standard deviation of the number of genes in a correlated block
minBlockSize	integer scalar specifying minimal number of genes in a correlated block
mu0	numeric scalar specifying expected mean expression level of a gene on the log scale

sigma0	numeric scalar specifying standard deviation of the mean expression level of a gene on the log scale
rate	numeric scalar specifying one of the gamma parameters
shape	numeric scalar specifying one of the gamma parameters
p.cor	numeric scalar specifying expected correlation within each block
wt.cor	numeric scalar specifying weight given to the expected block correlation
xform	A function that will be passed to the alterMean method
...	extra arguments that wil be passed back to the xform function

Details

Our standard model for gene expression in a homogeneous sample assumes that the overall correlation matrix is block diagonal. Correlation between genes in different blocks is assumed to be zero. Correlation for genes in the same block is assumed to be a constant, but different correlation constants can be used in different blocks. The actual correlations are assumed to arise from a beta distribution of the form Beta(pw, (1-p)w), where p=p.cor and w=wt.cor are two of the hyperparameters.

The number of blocks is determined jointly by the CancerModel, cm, and the hyperparameter nExtraBlocks. The size of a block is assumed to arise from a normal distribution with mean given by meanBlockSize and standard deviaion given by sigmaBlockSize. To avoid accidentally assigning non-postive block sizes, this distribution is truncated below by minBlockSize.

The expression of each gene is assumed to come from a log-normal distribution with parameters describing the per-gene mean (μ_g) and standard deviation (σ_g) n the log scale. These parameters, in turn, are assumed to come from hyperdistributions. Specifically, we assume that μ_g comes from a normal distribution with mean mu0 and standard deviation sigma0. We also assume that σ_g comes from an inverse gamma distribution with parameters rate and shape.

The BlockHyperParameters class simply bundles the parameters for this model into a single structure. The default values are consistent with data we have seen from several Affymetrix microarray studies.

The makeBlockStructure function takes a CancerModel and a BlockHyperParameters object as arguments and produces a CancerEngine object. The rand method for this class can be used to generate matched clinical data (with the structure defined by the CancerModel object) and gene expression data with the specified block correlation structure.

Value

The BlockHyperParameters generator returns an object of class BlockHyperParameters.

The function makeBlockStructure returns an object of the CancerEngine class.

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the BlockHyperParameters generator function.

Slots

nExtraBlocks: An integer; the number of blocks not involved in the "hit" structure defined by the CancerModel.

meanBlockSize: A real number; the mean number of genes in a correlated block.

`sigmaBlockSize`: A real number; standard deviation of the number of genes in a correlated block.

`minBlockSize`: An integer; the minimal number of genes in a correlated block.

`mu0`: A real number; the expected mean expression level of a gene on the log scale.

`sigma0`: A real number; the standard deviation of the mean expression level of a gene on the log scale.

`rate`: Gamma parameter; see details.

`shape`: Gamma parameter; see details.

`p.cor`: A real number; the expected correlation within each block.

`wt.cor`: A real number; the weight given to the expected block correlation.

Methods

There are no special methods defind for this class.

Author(s)

Kevin R. Coombes <krc@silicovore.com>,

See Also

`CancerModel`, `CancerEngine`

Examples

```
showClass("BlockHyperParameters")
sm <- SurvivalModel(baseHazard=1/3, units=52, unitName="weeks")
cm <- CancerModel("myModel", nPossible=10, nPattern=5,
                  survivalModel=sm)
hyper <- BlockHyperParameters()
engine <- makeBlockStructure(cm, hyper)
outcome <- rand(engine, 100)
summary(outcome$clinical)
dim(outcome$data)
```

---

blur-method *Method "blur"*

Description

`blur` is a generic function used to add noise to a signal as defined by various objects. The method invokes particular `methods` which depend on the `class` of the first argument.

Usage

```
## S4 method for signature 'ANY'
blur(object, x, ...)
```

Arguments

`object`	an object from which adding noise to its signal is desired
`x`	matrix containing signal to be affected
`...`	additional arguments affecting the noise addition

Value

The form of the value returned by `blur` depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

Kevin R. Coombes <krc@silicovore.com>,

CancerEngine-class *The "CancerEngine" Class*

Description

A CancerEngine combines a CancerModel (which defines the combinatorics of hits that produce cancer subtypes) with a pair of gene expression Engines that can be used to simulate microarray data depending on the presence or absence of different hits.

Usage

```
CancerEngine(cm, base, altered)
## S4 method for signature 'CancerEngine'
summary(object, ...)
```

Arguments

`cm`	object of class `CancerModel`
`base`	character string giving the name of an `Engine` or `EngineWithActivity`, or an object of class `Engine`. Represents the expected gene expression in the absence of any hits.
`altered`	character string giving the name of an `Engine` or `EngineWithActivity`, or an object of class `Engine`. Represents the expected gene expression in the presence of hits.
`object`	object of class `CancerEngine`
`...`	extra arguments for generic routines

Objects from the Class

Although objects of the class can be created by a direct call to `new`, the preferred method is to use the `CancerEngine` generator function.

Slots

`cm`: A `CancerModel` object.

`base`: Object of class "character" giving the name of an `Engine` or `EngineWithActivity`. Represents the expected gene expression in the absence of any hits.

`altered`: Object of class "character" giving the name of an `Engine` or `EngineWithActivity`. Represents the expected gene expression in the presence of hits.

`localenv`: Object of class `"environment"`; used in the internal implementation.

Methods

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

summary `signature(object = "CancerEngine")`: :TODO:

Author(s)

Kevin R. Coombes <krc@silicovore.com>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

See Also

CancerModel

Examples

```
showClass("CancerEngine")
set.seed(391629)
## Set up survival outcome; baseline is exponential
sm <- SurvivalModel(baseHazard=1/5, accrual=5, followUp=1)
## Build a CancerModel with 6 subtypes
nBlocks <- 20    # number of possible hits
cm <- CancerModel(name="cansim",
                  nPossible=nBlocks,
                  nPattern=6,
                  OUT = function(n) rnorm(n, 0, 1),
                  SURV= function(n) rnorm(n, 0, 1),
                  survivalModel=sm)
## Include 100 blocks/pathways that are not hit by cancer
nTotalBlocks <- nBlocks + 100
## Assign values to hyperparameters
## block size
blockSize <- round(rnorm(nTotalBlocks, 100, 30))
## log normal mean hypers
mu0    <- 6
sigma0 <- 1.5
## log normal sigma hypers
rate   <- 28.11
shape  <- 44.25
## block corr
p <- 0.6
w <- 5
## Set up the baseline Engine
rho <- rbeta(nTotalBlocks, p*w, (1-p)*w)
base <- lapply(1:nTotalBlocks,
               function(i) {
                 bs <- blockSize[i]
                 co <- matrix(rho[i], nrow=bs, ncol=bs)
                 diag(co) <- 1
                 mu <- rnorm(bs, mu0, sigma0)
```

```
                sigma <- matrix(1/rgamma(bs, rate=rate, shape=shape), nrow=1)
                covo <- co *(t(sigma) %*% sigma)
                MVN(mu, covo)
              })
eng <- Engine(base)
## Alter the means if there is a hit
altered <- alterMean(eng, normalOffset, delta=0, sigma=1)
## Build the CancerEngine using character strings
object <- CancerEngine(cm, "eng", "altered")
## Or build it using the actual Engine components
ob <- CancerEngine(cm, eng, altered)
summary(object)
summary(ob)
## Simulate the data
dset <- rand(object, 20)
summary(dset$clinical)
summary(dset$data[, 1:3])
```

CancerModel-class *The "CancerModel" Class*

Description

A CancerModel object contains a number of pieces of information representing an abstract, heterogeneous collection of cancer patients. It can be used to simulate patient outcome data linked to hit classes.

Usage

```
CancerModel(name,
            nPossible,
            nPattern,
            HIT = function(n) 5,
            SURV = function(n) rnorm(n, 0, 2),
            OUT = function(n) rnorm(n, 0, 2),
            survivalModel=NULL,
            prevalence=NULL)
nPatterns(object)
nPossibleHits(object)
nHitsPerPattern(object)
outcomeCoefficients(object)
survivalCoefficients(object)
## S4 method for signature 'CancerModel'
ncol(x)
## S4 method for signature 'CancerModel'
nrow(x)
## S4 method for signature 'CancerModel'
rand(object, n, balance=FALSE, ...)
## S4 method for signature 'CancerModel'
summary(object, ...)
```

Arguments

name	character string specifying name given to this model
object, x	object of class CancerModel
nPossible	integer scalar specifying number of potential hits relevant to the kind of cancer being modeled
nPattern	integer scalar specifying number of different cancer subtypes
HIT	function that generates non-negative integers from a discrete distribution. Used to determine the number of hits actually present in each cancer subtype.
SURV	function that generates real numbers from a continuous distributions. Used in simulations to select the coefficients associated with each hit in Cox proportional hazards models.
OUT	function that generates real numbers from a continuous distributions. Used in simulations to select the coefficients associated with each hit in logistic models of a binary outcome.
survivalModel	object of class SurvivalModel used to simulate survival times for each simulated patient
prevalence	optional numeric vector of relative prevalences of cancer subtypes
n	numeric scalar specifying quantity of random numbers
balance	logical scalar specifying how patients should be simulated
...	extra arguments for generic routines

Details

The rand method is the most important method for objects of this class. It returns a data frame with four columns: the CancerSubType (as an integer that indexes into the hitPattern slot of the object), a binary Outcome that takes on values "Bad" or "Good", an LFU column with censored survival times, and a logical Event column that describes whether the simulated survival event has occurred.

The rand method for the CancerModel class adds an extra logical parameter, balanced, to the signature specified by the default method. If balanced=FALSE (the default), then patients are simulated based on the prevalence defined as apart of the model. If balanced=TRUE, then patients are simulated with equal numbers in each hit pattern class, ordered by the hit pattern class.

Value

The CancerModel function is used to contruct and return an object of the CancerModel class.

The ncol and nrow functions return integers with the size of the matrix of hit patterns.

The rand method returns data frame with four columns:

CancerSubType	integer index into object's 'hitPattern' slot
Outcome	outcomes with values "Bad" or "Good"
LFU	censored survival times
Event	has simulated survival event has occurred?

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the `CancerModel` generator function.

Slots

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

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

`survivalBeta`: Object of class `"numeric"` containing the coeffieicents associated with each hit in a Cox proportional hazards model of survival.

`outcomeBeta`: Object of class `"numeric"` contaning the coefficients associated with each hit in a logistic model to predict a binary outcome.

`prevalence`: Object of class `"numeric"` containing the prevalence of each cancer subtype.

`survivalModel`: Object of class `"survivalModel"` containing parameters used to simualte survival times.

`call`: object of class `"call"` recording the function call used to initialize the object.

Methods

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

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

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

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

Author(s)

Kevin R. Coombes <krc@silicovore.com>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

See Also

SurvivalModel

Examples

```
showClass("CancerModel")
set.seed(391629)
# set up survival outcome; baseline is exponential
sm <- SurvivalModel(baseHazard=1/5, accrual=5, followUp=1)
# now build a CancerModel with 6 subtypes
cm <- CancerModel(name="cansim",
                  nPossible=20,
                  nPattern=6,
                  OUT = function(n) rnorm(n, 0, 1),
                  SURV= function(n) rnorm(n, 0, 1),
```

```
                  survivalModel=sm)
# simulate 100 patients
clinical <- rand(cm, 100)
summary(clinical)
```

Engine-class *The "Engine" Class*

Description

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

Usage

```
Engine(components)
nComponents(object)
## 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, ...)
```

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	function takes as its input a vector of mean expression or standard deviation and returns a transformed vector that can be used to alter the appropriate slot of the object.
n	numeric scalar representing number of samples to be simulated
...	extra arguments for generic or plotting routines

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.

Value

The Engine generator returns an object of class Engine.

The alterMean method returns an object of class Engine with altered mean.

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

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

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

The summary method prints out the number of components included in the Engine object.

The nComponents method 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.

Methods

alterMean(object, TRANSFORM, ...) Takes an object of class Engine, loops over the components in the Engine, alters the mean as defined by TRANSFORM function, and returns a modified object of class Engine.

alterSD(object, TRANSFORM, ...) Takes an object of class Engine, loops over the components in the Engine, alters 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, ...) Prints out the number of components included in the object of Engine.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

See Also

EngineWithActivity

Examples

```
showClass("Engine")
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'
rand(object, n, ...)
## S4 method for signature 'EngineWithActivity'
summary(object, ...)
```

Arguments

active	logical vector with length equal to number of components specifying whether each component should be transcriptionally active, or a numeric scalar specifying the probability for a component to be active
components	list where each element contains the parameters for the underlying distribution that the gene expression follows
base	numeric scalar specifying the logarithmic scale to which the data should be transformed
object	object of class EngineWithActivity

n	number of samples to be simulated
...	extra arguments for generic routines

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.

Value

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

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

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

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the `EngineWithActivity` generator function.

Slots

`active`**:** logical vector specifying whether each component should be transcriptionally active or not

`base`**:** numeric scalar specifying the logarithmic scale

`components`**:** list specifying the parameters of the underlying distribution

Extends

Class `Engine`, directly.

Methods

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`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

Examples

```
showClass("EngineWithActivity")
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/nComponents, 44, 28))
}
myEngine <- EngineWithActivity(active, comp, 2)
summary(myEngine)
myData <- rand(myEngine, 5)
dim(myData)
```

IndependentLogNormal-class

The "IndependentLogNormal" 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	numeric vector specifying the mean expression values on the logarithmic scale.
logsigma	numeric vector specifying the standard deviation of the gene expression values on the logarithmic scale
object, x	object of class IndependentLogNormal
n	numeric scalar specifying number of samples to be simulated
...	extra arguments for generic or plotting routines

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the IndependentLogNormal generator function.

Slots

`logmu`: numeric vector containing the mean expression values on the logarithmic scale

`logsigma`: numeric vector containing the standard deviation of the gene expression values on the logarithmic scale

Methods

nrow(x) 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 <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

See Also

`Engine`, `IndependentNormal`, `MVN`

Examples

```
showClass("IndependentLogNormal")
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	numeric vector specifying the mean expression values
sigma	numeric vector specifying the standard deviation of the gene expression values
object, x	object of class IndependentNormal
TRANSFORM	function that takes a vector of mean expression or standard deviation and returns a transformed vector that can be used to alter the appropriate slot of the object.
n	numeric scalar specifying number of samples to be simulated
...	extra arguments for generic or plotting routines

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 construct the engine.

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: see corresponding argument above

sigma: see corresponding argument above

Methods

alterMean(object, TRANSFORM, ...) Takes an object of class `IndependentNormal`, loops over the `mu` slot, alters 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`, loops over the `sigma` slot, alters the standard deviation as defined by `TRANSFORM` function, and returns an object of class `IndependentNormal` with altered `sigma`.

nrow(x) 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 <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

See Also

`Engine`, `IndependentLogNormal`, `MVN`

Examples

```
showClass("IndependentNormal")
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 "MVN" 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)
covar(object)
correl(object)
## 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, ...)
```

Arguments

mu	numeric vector representing k-dimensional means
Sigma	numeric k-by-k covariance matrix containing the measurement of the linear coupling between every pair of random vectors
tol	numeric scalar roundoff error that will be tolerated when assessing the singularity of the covariance matrix
object, x	object of class MVN
TRANSFORM	function that takes a vector of mean expression or standard deviation and returns a transformed vector that can be used to alter the appropriate slot of the object.
n	numeric scalar representing number of samples to be simulated
...	extra arguments for generic or plotting routines

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** package 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.

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 $Sigma = Delta\%*\%R\%*\%Delta$. So, we can change the 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:

1. `covar` should return the same matrix that was used in the function call to construct the `MVN` object.
2. After applying an `alterMean` method, the covariance matrix is unchanged.
3. The diagonal of correlation matrix consists of all ones.
4. After applying an `alterMean` or an `alterSD` method, the correlation matrix is unchanged.

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the `MVN` generator function.

Slots

`mu`**:** numeric vector containing the k-dimentional means

`lambda`**:** numeric vector containing the square roots of eigenvalues of the covariance matrix

`half`**:** numeric matrix with $k*k$ dimensions whose columns contain the eigenvectors of the covariance matrix

Methods

alterMean(object, TRANSFORM, ...) Takes an object of class `MVN`, loops over the `mu` slot, alters 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, alters the standard deviation as defined by TRANSFORM function, and reconstructs 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 <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

See Also

`Engine`, `IndependentNormal`

Examples

```
showClass("MVN")
  ## 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 choose 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 NoiseModel 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 affecting blur applied

Details

We model both additive and multiplicative noise, so that the observed expression of gene *g* in sample *i* is given by: $Y_g i = S_g i exp(H_g i) + E_g i$, 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.

Methods

blur(object, x, ...) Adds and multiplies random noise to the data matrix `x` containing the true signal from the gene expression.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

Examples

```
showClass("NoiseModel")
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)
```

`NormalVsCancer` *Simulating Cancer Versus Normal Datasets*

Description

These functions are useful for simulating data that compares a homogeneous "cancer" group to a homogeneous "normal" group of samples.

Usage

```
NormalVsCancerModel(nBlocks, survivalModel=NULL, name="NormalVsCancer")
NormalVsCancerEngine(nBlocks, hyperp)
```

Arguments

nBlocks	scalar integer representing number of correlated blocks that are differentially expressed between cancer and normal
survivalModel	a SurvivalModel object
name	character string specifying name of the object being created
hyperp	object of class BlockHyperParameters that describes the block correlation structure.

Details

The simplest simulation model assumes that we are comparing two homogeneous groups.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

See Also

BlockHyperParameters, CancerEngine, CancerModel

Examples

```
nvc <- NormalVsCancerModel(10)
summary(nvc)
rand(nvc, 10)
rand(nvc, 10, balance=TRUE)
engine <- NormalVsCancerEngine(10)
dset <- rand(engine, 10, balance=TRUE)
```

---

rand-method *Method "rand"*

Description

rand is a generic function used to produce random numbers from the distribution defined by various objects. The method invokes particular methods which depend on the class of the first argument.

Usage

```
## S4 method for signature 'ANY'
rand(object, n, ...)
```

Arguments

object	an object from which random numbers from a distribution is desired
n	numeric scalar specifying quantity of random numbers
...	additional arguments affecting the random numbers produced

Value

The form of the value returned by rand depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

Author(s)

Kevin R. Coombes <krc@silicovore.com>,

SurvivalModel-class *The "SurvivalModel" Class*

Description

A SurvivalModel class represents the information for simulating survival times for each patient.

Usage

```
SurvivalModel(baseHazard=1/5,
              accrual=5,
              followUp=1,
              units=12,
              unitName="months")
## S4 method for signature 'SurvivalModel'
rand(object, n, beta=NULL, ...)
```

Arguments

baseHazard	numeric scalar describing the underlying hazard rate at baseline levels of covariates
accrual	numeric scalar representing number of patient accrual years
followUp	numeric scalar representing frequency of follow up in the unit of year
units	numeric scalar representing number of units per year where units are specified by unitName
unitName	character string representing the unit argument type
object	object of class SurvivalModel
n	numeric scalar specifying quantity of random numbers
beta	numeric vector specifying beta parameters for patients
...	extra arguments for generic routines

Value

The SurvivalModel generator returns an object of class SurvivalModel.

The rand method returns a data.frame with components:

LFU	time to event
Event	whether the data is censored

Objects from the Class

Although objects of the class can be created by a direct call to new, the preferred method is to use the `SurvivalModel` generator function.

Slots

`baseHazard`: see corresponding argument above

`accrual`: see corresponding argument above

`followUp`: see corresponding argument above

`units`: see corresponding argument above

`unitName`: see corresponding argument above

Methods

rand(object, n, beta, ...) Simulate survival data for `n` patients given `beta`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

References

Zhang J, Coombes KR.
Sources of variation in false discovery rate estimation include sample size, correlation, and inherent differences between groups.
BMC Bioinformatics. 2012; 13 Suppl 13:S1.

See Also

`CancerModel`

Examples

```
showClass("SurvivalModel")
sm <- SurvivalModel()
## Generate data from base model
outcome <- rand(sm, 100)
summary(outcome)
## Generate data from five classes with different beta parameters
beta <- rep(rnorm(5, 0, 2), each=20)
outcome <- rand(sm, 100, beta=beta)
summary(outcome)
```

transforms *Transform functions*

Description

normalOffset is a function that can be used as the TRANSFORM argument 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 function that can be used as the TRANSFORM argument in an alterSD operation, which multiplies each standard deviation by a positive value chosen from an inverse gamma distribution with parameters shape and scale.

Usage

```
normalOffset(x, delta, sigma)
invGammaMultiple(x, shape, rate)
```

Arguments

x	numeric vector of mean expression or standard deviation defined in the object
delta, sigma	numeric vector used as mean and/or sd parameters specifying the normal distribution
shape, rate	numeric vector used as shape and/or rate parameters specifying the gamma distribution. Must be positive.
...	additional arguments affecting the specific transformation performed

Value

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 <krc@silicovore.com>, Jiexin Zhang <jiexinzhang@mdanderson.org>,

See Also

alterMean, alterSD, RNGkind

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)
```

Index