Package 'ReliabilityTheory'

May 7, 2026

Title Structural Reliability Analysis

Description Perform structural reliability analysis, including computation and simulation with system signatures, Samaniego (2007) <doi:10.1007/978-0-387-71797-5>, and survival signatures, Coolen and Coolen-Maturi (2013) <doi:10.1007/978-3-642-30662-4_8>. Additionally supports parametric and topological inference given system lifetime data, Aslett (2012) <https://www.louisaslett.com/PhD_Thesis.pdf>.

URL https://www.louisaslett.com/,
https://github.com/louisaslett/ReliabilityTheory

BugReports https://github.com/louisaslett/ReliabilityTheory/issues

Version 0.3.1

Maintainer Louis Aslett <louis.aslett@durham.ac.uk>

Depends R (>= 3.5.0)

Imports actuar, combinat, FRACTION, igraph (>= 1.0.1), mcmc, PhaseType (>= 0.2.0), sfsmisc, utils

Suggests testthat, reshape2, ggplot2, xtable

License GPL-2 | GPL-3

ByteCompile yes

LazyLoad yes

NeedsCompilation no

Author Louis Aslett [aut, cre, cph]

Repository CRAN

Date/Publication 2024-09-25 13:20:08 UTC

Contents

`ReliabilityTheory-package`

Structural Reliability Theory Toolbox

Description

A collection of tools for working structural reliability problems, such as catalogues of system signatures and Bayesian inferential functions.

Details

Package:	ReliabilityTheory
Type:	Package
Version:	0.2.0
Date:	2023-05-03
License:	GPL-2 | GPL-3
LazyLoad:	yes

Author(s)

Louis J. M. Aslett, <louis.aslett@durham.ac.uk> (https://www.louisaslett.com)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

Samaniego, F. J. (2007), *System Signatures and Their Applications in Engineering Reliability*, Springer.

---

cn02 *Catalogue of Coherent Networks of Order 2*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all coherent networks of node order 2.

Usage

```
data(cn02)
```

Format

A list object, one item for each such network. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

---

cn03 *Catalogue of Coherent Networks of Order 3*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all coherent networks of node order 3.

Usage

```
data(cn03)
```

Format

A list object, one item for each such network. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

`computeSystemSignature`

Compute the signature of a system

Description

The system signature (Samaniego, 2007) is an alternative to the structure function as a starting point for a structural reliability analysis. This automatically computes the signature of the specified system or network. Here, system implies components are unreliable whereas network implies links are unreliable.

Usage

```
computeSystemSignature(sys, cutsets=NULL, frac=FALSE)
computeNetworkSignature(sys, cutsets=NULL, frac=FALSE)
```

Arguments

`sys`	a `system` object representing the system whose types are to be set. This should have been created by a call to `createSystem`.
`cutsets`	if the cut-sets of the system or network are already known they may be passed in as a list of numeric vectors. This can save time because cut-set computation is the slowest part of the algorithm. Leaving as NULL causes the function to find the cut sets itself.
`frac`	if TRUE then the function prints out signature elements as fractions rather than returning a decimal signature vector.

Details

The signature of a system is the probability vector $\mathbf{s} = (s_1, \ldots, s_n)$ with elements:

$$s_i = P(T = T_{i:n})$$

where T is the failure time of the system and $T_{i:n}$ is the ith order statistic of the n component failure times. Likewise the network signature is the same but where components are reliable and it is links which fail. See Samaniego (2007) for details.

The system or network is specified by means of a `system` object, whereby each end of the system is denoted by nodes named `s` and `t` which are taken to be perfectly reliable. It is easy to construct the appropriate reliability block diagram representation using the function `createSystem`. Note that each physically distinct component should be separately numbered when constructing this object.

Value

`computeSystemSignature` returns a numeric probability vector which is the system/network signature.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Samaniego, F. J. (2007), *System Signatures and Their Applications in Engineering Reliability*, Springer.

See Also

`computeSystemSurvivalSignature`

Examples

```
# Find the signature of two component series system (which is just s=(1, 0))
computeSystemSignature(createSystem(s -- 1 -- 2 -- t))

# Find the signature of two component parallel system (which is just s=(0, 1))
computeSystemSignature(createSystem(s -- 1:2 -- t))

# Find the signature of the five component 'bridge' system (which
# is s=(0, 0.2, 0.6, 0.2, 0))
computeSystemSignature(createSystem(s -- 1 -- 2 -- t, s -- 3 -- 4 -- t, 1:2 -- 5 -- 3:4))
```

`computeSystemSurvivalSignature` *Compute the survival signature of a system*

Description

The system survival signature (Coolen and Coolen-Maturi, 2012) is a generalisation of the signature to systems with multiple component types. This function automatically computes the survival signature of the specified system. Here, system implies components (as opposed to links) are unreliable.

Usage

```
computeSystemSurvivalSignature(sys, cutsets=NULL, frac=FALSE)
```

Arguments

`sys`	a `system` object representing the system whose types are to be set. This should have been created by a call to `createSystem`.
`cutsets`	if the cut-sets of the system or network are already known they may be passed in as a list of numeric vectors. This can save time because cut-set computation is the slowest part of the algorithm. Leaving as NULL causes the function to find the cut sets itself.
`frac`	if TRUE then the function prints out survival signature probabilities as fractions rather than decimals.

Details

The survival signature of a system with K types of component is the functional $\Phi(l_1, \ldots, l_K)$ giving the probability that the system works given exactly l_k of the components of type k are working. See Coolen and Coolen-Maturi (2012) for details. Thus, the survival signature can be represented by a table with $K + 1$ columns, the first K being the number of each type of component which is working and the final column being the probability the system works.

The system or network is specified by means of a `system` object, whereby each end of the system is denoted by nodes named `s` and `t` which are taken to be perfectly reliable. It is easy to construct the appropriate reliability block diagram representation using the function `createSystem`. Note that each physically distinct component should be separately numbered when constructing this object.

Once the topology of the system has been defined (or at definition time), one must indicate the type of each component (if not done when initially calling `createSystem` it can later be modified using `setCompTypes`). The Examples section below features the full computation of the survival signature for Figure 1 in Coolen and Coolen-Maturi (2012) and Figure 2 in Coolen *et al* (2013) to make this clear.

Value

computeSystemSurvivalSignature returns a data frame with $K+1$ columns. The first K columns represent the function inputs, $l_1, \ldots, l_K$ and the final column is the probability that the system works given the corresponding numbers of each component which are working.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J. M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Coolen, F. P. A. and Coolen-Maturi, T. (2012), Generalizing the signature to systems with multiple types of components, *in* 'Complex Systems and Dependability', Springer, pp. 115-130.

Coolen, F. P. A., Coolen-Maturi, T., Al-nefaiee, A. H. and Aboalkhair, A. M. (2013), 'Recent advances in system reliability using the survival signature', *Proceedings of Advances in Risk and Reliability Technology Symposium, Loughborough*.

See Also

computeSystemSignature

Examples

```
## EXAMPLE 1
## Figure 1 in Coolen and Coolen-Maturi (2012)

# First, define the structure, ensuring that each physically separate component
# is separately numbered
fig1 <- createSystem(s -- 1 -- 2:3 -- 4 -- 5:6 -- t, 2 -- 5, 3 -- 6)

# Second, specify the type of each of those numbered components
# (leaving s,t with no type)
fig1 <- setCompTypes(fig1,
                     list("Type 1" = c("1","2","5"),
                          "Type 2" = c("3","4","6")))

# Third, compute the survival signature (getting fractions rather than decimals)
computeSystemSurvivalSignature(fig1, frac = TRUE)



## EXAMPLE 2
## Figure 3 in Coolen et al (2013)

# First, define the structure, ensuring that each physically separate component
# is separately numbered.
```

```
# For this example, we demonstrate how to define the component types at system
# creation time
fig3 <- createSystem(s -- 1:4 -- 2:5 -- 3:6 -- t, s -- 7:8, 8 -- 9, 7:9 -- t,
                     types = list("Type 1" = "1",
                                  "Type 2" = c("2","3","4","7"),
                                  "Type 3" = c("5","6","8","9")))

# Third, compute the survival signature (getting fractions rather than decimals)
computeSystemSurvivalSignature(fig3, frac=TRUE)
```

`createSystem` *Create a system specification*

Description

Creates a system design specification based on passing a textual representation of design.

Usage

```
createSystem(..., types = NULL)
```

Arguments

`...`	multiple expressions which together define an undirected graph representation of the reliability block diagram for the system design. There should be two terminal 'dummy' nodes to represent either end of the system structure, which must be labelled `s` and `t` (assumed perfectly reliable). All reliability assessment is of the connectivity of these nodes. See details and examples.
`types`	(optional) named list of vectors. The names correspond to component types, whilst each vector indicates which components are of that type. When it is not specified then all components are assumed to be of the same type. This can be updated later using the function `setCompTypes`.

Details

This function enables specification of a system design by textual representation of the reliability block diagram, for use in many other functions in this package. The method of representing the system is as for an undirected graph in the **igraph** package.

There should be two terminal 'dummy' nodes to represent either end of the system structure, which must be labelled `s` and `t` (assumed perfectly reliable). Dashes `--` are then used to connect numbered nodes together. The full specification can be spread over multiple arguments. Colon notation can denote an edge to multiple nodes, but is *not* a range specifier (eg `1:5` means components 1 and 5, not components 1 to 5). Following are some concrete examples:

1. a series system of 3 components:
   `createSystem(s -- 1 -- 2 -- 3 -- t)`

2. a parallel system of 3 components:
   `createSystem(s -- 1 -- t, s -- 2 -- t, s -- 3 -- t)`
   Or, more succinctly:
   `createSystem(s -- 1:2:3 -- t)`
3. a classic 'bridge' system consisting of 5 components:
   `createSystem(s -- 1:2 -- 5 -- 3:4 -- t, 1 -- 3, 2 -- 4)`
   Exactly equivalently:
   `createSystem(s -- 1 -- 3 -- t, s -- 2 -- 4 -- t, 1:2 -- 5 -- 3:4)`

Value

Returns a system of the design specified.

Internally, this is an **igraph** object, with some additional attributes relevant to system specification.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

See Also

`setCompTypes` to specify component types after system creation, rather than in the same command.

Examples

```
# Create a bridge system, with all components of the same type (or with type to
# be defined later)
bridge <- createSystem(s -- 1:2 -- 5 -- 3:4 -- t, 1 -- 3, 2 -- 4)

# Create a bridge system, with two types of component
bridge <- createSystem(s -- 1:2 -- 5 -- 3:4 -- t, 1 -- 3, 2 -- 4,
                       types = list(T1 = 1:4, T2 = 5))
```

---

expectedSystemLifetimeExp

Compute the expected lifetime of a given system

Description

Computes the expected lifetime of a system/network specified by its signature or graph structure when the components have Exponential lifetime distribution with specified rate. Useful for ordering systems/networks by expected lifetime.

Usage

```
expectedSystemLifetimeExp(sys, rate=1)
expectedNetworkLifetimeExp(sys, rate=1)
expectedSignatureLifetimeExp(s, rate=1)
```

Arguments

sys	a system object representing the system whose types are to be set. This should have been created by a call to createSystem.
s	the signature vector of the system/network whose expected lifetime is to be computed.
rate	the rate parameter of the Exponential distribution.

Details

The system or network is specified by means of a system object, whereby each end of the system is denoted by nodes named s and t which are taken to be perfectly reliable. It is easy to construct the appropriate reliability block diagram representation using the function createSystem. Note that each physically distinct component should be separately numbered when constructing this object.

Alternatively, the signature may be provided instead (the other functions simply use the graph object to compute the signature).

Value

All the functions return a single scalar value which is the expected lifetime.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Samaniego, F. J. (2007), *System Signatures and Their Applications in Engineering Reliability*, Springer.

See Also

computeSystemSignature

Examples

```
# Find the expected lifetime of two component series system
expectedSystemLifetimeExp(createSystem(s -- 1 -- 2 -- t))

# Find the expected lifetime of two component series system using it's signature
# directly
expectedSignatureLifetimeExp(c(1,0))

# Find the expected lifetime of two component parallel system
expectedSystemLifetimeExp(createSystem(s -- 1:2 -- t))
```

```
# Find the expected lifetime of two component parallel system using it's
# signature directly
expectedSignatureLifetimeExp(c(0,1))
```

`maskedInferenceEXCHCustom` *Inference for Masked Exchangeable System Lifetimes, Custom Distribution*

Description

Performs Bayesian inference via a signature based data augmentation MCMC scheme for masked system lifetime data for any custom component lifetime distribution. The underlying assumption is of exchangeability at the system level (iid components within each exchangeable system).

Usage

```
maskedInferenceEXCHCustom(t, signature, cdfComp, pdfComp, rParmGivenData,
                          rCompGivenParm, startCompParm, startHypParm, iter, ...)
```

Arguments

`t`	a vector of masked system lifetimes.
`signature`	the signature vector of the system/network for which inference is performed. It may be a list of signatures which results in topological inference on the system design being jointly performed over the collection of signatures provided.
`cdfComp`	user-defined vectorised cumulative distribution function of component lifetime $F_Y()$ with prototype: `function(y, parametersm, ...)`
`pdfComp`	user-defined vectorised probability distribution function of component lifetime $f_Y()$ with prototype: `function(y, parameters, ...)`
`rParmGivenData`	user-defined function which should produce random draws from $f_{\Xi \mid Y}$ with prototype: `function(y, ...)` This must return the new parameters as a 2 item list: the first item being the hyperprior parameters drawn in the same named vector format and order as `startHypPriorParm`; the second item being a list of component lifetime parameters drawn in the same named vector format and order as `startCompParm`.
`rCompGivenParm`	user-defined function which should produce random draws from $f_{Y \mid \Psi}$ with prototype: `function(parameters, t, censoring, ...)` where censoring is -1 for left censoring, 0 for exact observations and 1 for right censoring.

`startCompParm`	list consisting of a vectors of starting values per system (in the same order as `t`) of named parameters for the component lifetime distribution. The order within each named vector should match the order expected for the `parameters` argument in the user defined functions above.
`startHypParm`	vector of starting values of named hyper-parameters for the hyperprior.
`iter`	number of MCMC iterations to perform.
`...`	additional arguments which are passed through to the user-defined functions above.

Details

This is a low level implementation of the signature based data augmented MCMC scheme described in Aslett (2012) for exchangeable systems. This function need only be used if the component lifetime distribution of interest has not already been implemented within this package.

The arguments of the function are the prerequisites described in Algorithm 6.2 of Aslett (2012). The interested user is advised to inspect the source code of this package at the file `MaskedLifetimeInference_Exponential.R` for an example of its usage, which may be seen in the function `maskedInferenceEXCHExponential` defined there, together with the associated user-definied functions above it.

Value

If a single signature vector is provided above, then a data frame of MCMC samples with columns named the same as the `startParm` argument is returned.

If a list of signature vectors is provided above, then a list is returned containing three items:

`topology`	A vector of posterior samples from the discrete marginal posterior distribution of topologies provided in the signature list.
`parameters`	A list of data frames of MCMC samples with columns named the same as the `startCompParm` argument.
`hyperparameters`	A data frame of MCMC samples with columns named the same as the `startHypPriorParm` argument.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

See Also

computeSystemSignature

Examples

```
# Please inspect the source of this package, file MaskedLifetimeInference_Exponential.R
# for example usage (see details section)
```

---

maskedInferenceEXCHExponential

Inference for Masked Exchangeable System Lifetimes, Exponential Components

---

Description

Performs Bayesian inference via a signature based data augmentation MCMC scheme for masked system lifetime data for Exponentially distributed component lifetimes. The underlying assumption is exchangeability at the system level.

Usage

```
maskedInferenceEXCHExponential(t, signature, iter, priorMu_Mu, priorSigma_Mu,
                               priorMu_Sigma, priorSigma_Sigma)
```

Arguments

t	a vector of masked system lifetimes.
signature	the signature vector of the system/network for which inference is performed. It may be a list of signatures which results in topological inference on the system design being jointly performed over the collection of signatures provided.
iter	number of MCMC iterations to perform.
priorMu_Mu, priorSigma_Mu	μ and σ parameters for Log-Normal hyperprior on the mean of the exchangeable Gamma population distribution for the rate.
priorMu_Sigma, priorSigma_Sigma	μ and σ parameters for Log-Normal hyperprior on the variance of the exchangeable Gamma population distribution for the rate.

Details

This is a full implementation of the signature based data augmented MCMC scheme described in Aslett (2012) for exchangeable systems with Exponential component lifetimes.

Thus, components are taken to have Exponential lifetimes where the rate of the components in any given system is a realisation from an exchangeable population distribution. However, only the failure time of the system is observed, not those of the components or indeed which components were failed at the system failure time. By specifying a Log-Normal hyper-prior on the exchangeable

Gamma rate parameter population distribution, this function then produces MCMC samples from the posterior of the rate parameter and hyperparameters.

The model is as follows:

$$Y \mid \lambda \sim \text{Exponential}(\lambda)$$

$$\lambda \mid \nu, \zeta \sim \text{Gamma}(\text{shape} = \nu, \text{scale} = \zeta)$$

$$\mu \sim \text{Log-Normal}(\mu_\nu, \sigma_\nu)$$

$$\sigma^2 \sim \text{Log-Normal}(\mu_\zeta, \sigma_\zeta)$$

Additionally, if one does not know the system design, then it is possible to pass a list of many system signatures in the `signature` argument, in which case the topology of the system is jointly inferred with the parameters.

Value

If a single signature vector is provided above, then a list is returned containing two items:

`parameters` A list of data frames, each with a single column of MCMC samples from the posterior samples from the exhcangeable rate parameter for the given system.

`hyperparameters` A data frame with a column of posterior MCMC samples for each of the Log-Normal hyperprior parameters.

If a list of signature vectors is provided above, then a list is returned containing three items – the two items above, plus:

`topology` A vector of posterior samples from the discrete marginal posterior distribution of topologies provided in the signature list.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

See Also

`computeSystemSignature`

Examples

```
# Some masked system lifetime data for an exchangeable collection of systems with
# Exponential component lifetime, rate drawn from the population distribution
# Gamma(shape=9, scale=0.5)
t <- c(0.2265, 0.0795, 0.1178, 0.2463, 0.1053, 0.0982, 0.0349, 0.0363,
0.1546, 0.1357, 0.1239, 0.0354, 0.0124, 0.1003, 0.0827, 0.2446,
0.1214, 0.1272, 0.5438, 0.2738, 0.0378, 0.2293, 0.1706, 0.0146,
0.1506, 0.3665, 0.046, 0.1196, 0.2724, 0.2593, 0.0438, 0.1493,
0.0697, 0.1774, 0.1157, 0.0996, 0.2815, 0.1411, 0.0921, 0.2088,
0.1164, 0.149, 0.048, 0.1019, 0.2349, 0.2211, 0.0475, 0.0721,
0.0371, 0.611, 0.1959, 0.0666, 0.0956, 0.1416, 0.2126, 0.0104,
0.088, 0.0159, 0.078, 0.1747, 0.1921, 0.3558, 0.4956, 0.0436,
0.2292, 0.1159, 0.1201, 0.1299, 0.043, 0.0584, 0.0347, 0.2084,
0.1334, 0.1491, 0.0384, 0.0589, 0.2962, 0.1023, 0.0506, 0.0501,
0.1859, 0.0714, 0.1424, 0.0027, 0.2812, 0.0318, 0.4147, 0.1088,
0.2894, 0.0734, 0.1405, 0.0367, 0.0323, 0.517, 0.1034, 0.026,
0.0485, 0.0512, 0.0116, 0.1629)

# Load the signatures of order 4 simply connected coherent systems -- the data
# above correspond to simulations from system number 3
data(sccsO4)

# Perform inference on the rate parameter:
# NB this will take some time to run
samps <- maskedInferenceEXCHExponential(t, sccsO4[[3]]$signature,
2000, priorMu_Mu=1, priorSigma_Mu=0.5, priorMu_Sigma=1, priorSigma_Sigma=0.7)

# Or perform inference on rate parameter and topology jointly, taking as candidate
# set all possible simply connected coherent systems of order 4:
# NB this will take some time to run
samps <- maskedInferenceEXCHExponential(t, sccsO4, 2000, priorMu_Mu=1,
priorSigma_Mu=0.5, priorMu_Sigma=1, priorSigma_Sigma=0.7)
```

`maskedInferenceIIDCustom` *Inference for Masked iid System Lifetimes, Custom Distribution*

Description

Performs Bayesian inference via a signature based data augmentation MCMC scheme for masked system lifetime data for any custom component lifetime distribution. The underlying assumption is iid components and iid systems.

Usage

```
maskedInferenceIIDCustom(t, signature, cdfComp, pdfComp, rParmGivenData,
                         rCompGivenParm, startParm, iter, ...)
```

Arguments

t	a vector of masked system lifetimes.
signature	the signature vector of the system/network for which inference is performed. It may be a list of signatures which results in topological inference on the system design being jointly performed over the collection of signatures provided.
cdfComp	user-defined vectorised cumulative distribution function of component lifetime $F_Y()$ with prototype: `function(y, parametersm, ...)`
pdfComp	user-defined vectorised probability distribution function of component lifetime $f_Y()$ with prototype: `function(y, parameters, ...)`
rParmGivenData	user-defined function which should produce random draws from $f_{\Psi \mid Y}$ with prototype: `function(y, ...)` This must return the parameters in the same order named vector format as used for `startParm`.
rCompGivenParm	user-defined function which should produce random draws from $f_{Y \mid \Psi}$ with prototype: `function(parameters, t, censoring, ...)` where censoring is -1 for left censoring, 0 for exact observations and 1 for right censoring.
startParm	vector of starting values of named parameters in the correct order for the `parameters` argument in the user defined functions above.
iter	number of MCMC iterations to perform.
...	additional arguments which are passed through to the user-defined functions above.

Details

This is a low level implementation of the signature based data augmented MCMC scheme described in Aslett (2012) for iid systems. This function need only be used if the component lifetime distribution of interest has not already been implemented within this package.

The arguments of the function are the prerequisites described in Algorithm 6.2 of Aslett (2012). The interested user is advised to inspect the source code of this package at the file `MaskedLifetimeInference_Exponential.R` for an example of its usage, which may be seen in the function `maskedInferenceIIDExponential` defined there, together with the associated user-definied functions above it.

Value

If a single signature vector is provided above, then a data frame of MCMC samples with columns named the same as the `startParm` argument is returned.

If a list of signature vectors is provided above, then a list is returned containing two items:

topology	A vector of posterior samples from the discrete marginal posterior distribution of topologies provided in the signature list.
parameters	A data frame of MCMC samples with columns named the same as the `startParm` argument.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

See Also

computeSystemSignature

Examples

```
# Please inspect the source of this package, file MaskedLifetimeInference_Exponential.R
# for example usage (see details section)
```

---

maskedInferenceIIDExponential

Inference for Masked iid System Lifetimes, Exponential Components

---

Description

Performs Bayesian inference via a signature based data augmentation MCMC scheme for masked system lifetime data for Exponentially distributed component lifetimes. The underlying assumption is iid components and iid systems.

Usage

```
maskedInferenceIIDExponential(t, signature, iter, priorShape, priorScale)
```

Arguments

t	a vector of masked system lifetimes.
signature	the signature vector of the system/network for which inference is performed. It may be a list of signatures which results in topological inference on the system design being jointly performed over the collection of signatures provided.
iter	number of MCMC iterations to perform.
priorShape	the shape parameter of the Gamma prior of the Exponential rate.
priorScale	the scale parameter of the Gamma prior of the Exponential rate.

Details

This is a full implementation of the signature based data augmented MCMC scheme described in Aslett (2012) for iid systems with Exponential component lifetimes.

Thus, components are taken to have Exponential lifetimes and be arranged into some system. However, only the failure time of the system is observed, not those of the components or indeed which components were failed at the system failure time. By specifying a Gamma prior distribution on the component lifetime Exponential rate parameter via `priorShape` and `priorScale`, this function then produces MCMC samples from the posterior of the rate parameter.

Additionally, if one does not know the system design, then it is possible to pass a list of many system signatures in the `signature` argument, in which case the topology of the system is jointly inferred with the parameters.

Value

If a single signature vector is provided above, then a data frame with a single column of MCMC samples from the posterior of the rate parameter are returned.

If a list of signature vectors is provided above, then a list is returned containing two items:

`topology`	A vector of posterior samples from the discrete marginal posterior distribution of topologies provided in the signature list.
`parameters`	A data frame with a single column of MCMC samples from the posterior of the rate parameter.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

See Also

`computeSystemSignature`

Examples

```
# Some masked system lifetime data for a system with Exponential component
# lifetime, rate=3.14
t <- c(0.2696, 0.3613, 0.0256, 0.1287, 0.2305, 0.1565, 0.2484, 0.7482,
0.1748, 0.1805, 0.1985, 0.0799, 0.2843, 0.2392, 0.2151, 0.1177,
0.1278, 0.4189, 0.4374, 0.0931, 0.2846, 0.0357, 0.1809, 0.2077,
0.5211, 0.4935, 0.1464, 0.0297, 0.5429, 0.1294, 0.7089, 0.5534,
```

```
0.1183, 0.2628, 0.0481, 0.0518, 0.0533, 0.3595, 0.0767, 0.2606,
0.1005, 0.227, 0.01, 0.0947, 0.1248, 0.2288, 0.1422, 0.233, 0.1428,
0.2043)

# Load the signatures of order 4 simply connected coherent systems -- the data
# above correspond to simulations from system number 3
data(sccsO4)

# Perform inference on the rate parameter:
# NB this will take some time to run
samps <- maskedInferenceIIDExponential(t, sccsO4[[3]]$signature, 2000,
priorShape=9, priorScale=0.5)

# Or perform inference on rate parameter and topology jointly, taking as candidate
# set all possible simply connected coherent systems of order 4:
# NB this will take some time to run
samps <- maskedInferenceIIDExponential(t, sccsO4, 2000, priorShape=9,
priorScale=0.5)
```

`nonParBayesSystemInference` *Non-parametric Bayesian posterior predictive system survival inference*

Description

Computes a non-parametric Bayesian posterior predictive survival probability given the survival signature of a system and test data on each of the components as described in Aslett *et al* (2015).

Usage

```
nonParBayesSystemInference(at.times, survival.signature, test.data, alpha=1, beta=1)
```

Arguments

`at.times`	a vector of times at which the posterior predictive estimate of survival probability should be computed.
`survival.signature`	the survival signature matrix of the system/network for which inference is performed. This should be in the same format as returned by `computeSystemSurvivalSignature`.
`test.data`	a list of vectors containing the component test data. The elements of the list should be named identically to the component columns in the `survival.signature` argument.
`alpha, beta`	the Beta prior shape parameters. Each must match in type and can be:

- a single scalar for a fixed prior across time and component types;

- a vector of parameters of the same length as the `at.times` argument, which indicates the time-varying prior parameter at the corresponding time in `at.times`. This is therefore time-varying, but indicates the same time-varying prior for all component types;
- a data frame where each column is named using the same names as for the `survival.signature` argument and each row corresponds to the time-varying prior parameter at the corresponding time in `at.times`.

By default the 'uninformative' prior with `alpha=1` and `beta=1` is used for all components at all times.

Details

This function implements the technique described in detail in Section 4 of Aslett *et al* (2015).

In brief, at any fixed time t, the functioning of a single component of type k is Bernoulli(p_t^k) distributed for suitable p_t^k, irrespective of the lifetime distribution of the component. Correspondingly, the distribution of the number of components still functioning at time t in a collection of n_k iid components of type k is Binomial(n_k, p_t^k).

Taking the priors $p_t^k \sim$ Beta(α_t^k, β_t^k), Aslett *et al* (2015) show that this leads to a posterior predictive survival distribution with a nice closed form (see equations 9 and 10 in Section 4).

Value

A vector of the same length as the `at.times` argument, where each element is the posterior predictive probability of a new system surviving to the corresponding time in `at.times`.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J. M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M., Coolen, F. P. A. and Wilson, S. P. (2015), 'Bayesian Inference for Reliability of Systems and Networks using the Survival Signature', *Risk Analysis* **39**(9), 1640–1651. Download paper

See Also

`computeSystemSurvivalSignature`

Examples

```
## Exactly reproduce the example in Section 4.1 of Aslett et al (2015), including Figure 5
# First specify the system layout, numbered as per Figure 4
sys <- createSystem(s -- 1 -- 2:4:5, 2 -- 3 -- t, 4:5 -- 6 -- t,
                    s -- 7 -- 8 -- t, s -- 9 -- 10 -- 11 -- t, 7 -- 10 -- 8,
                    types = list(T1 = c(1, 6, 11),
                                 T2 = c(2, 3, 9),
                                 T3 = c(4, 5, 10),
                                 T4 = c(7, 8)))

# Compute the survival signature table from Appendix
sig <- computeSystemSurvivalSignature(sys)

# Simulate the test data (same seed as used in the paper)
set.seed(233)
t1 <- rexp(100, rate=0.55)
t2 <- rweibull(100, scale=1.8, shape=2.2)
t3 <- rlnorm(100, 0.4, 0.9)
t4 <- rgamma(100, scale=0.9, shape=3.2)

# Compile into a list as required by this function
test.data <- list("T1"=t1, "T2"=t2, "T3"=t3, "T4"=t4)

# Create a vector of times at which to evaluate the posterior predictive
# survival probability and compute using this function
t <- seq(0, 5, length.out=300)
yS <- nonParBayesSystemInference(t, sig, test.data)

# Compute the survival curves for the individual components (just to match
# Figure 5)
y1 <- sapply(t, pexp, rate=0.55, lower.tail=FALSE)
y2 <- sapply(t, pweibull, scale=1.8, shape=2.2, lower.tail=FALSE)
y3 <- sapply(t, plnorm, meanlog=0.4, sdlog=0.9, lower.tail=FALSE)
y4 <- sapply(t, pgamma, scale=0.9, shape=3.2, lower.tail=FALSE)

# Plot

library(ggplot2)
p <- ggplot(data.frame(Time=rep(t,5), Probability=c(yS,y1,y2,y3,y4),
            Item=c(rep(c("System", "T1", "T2", "T3", "T4"), each=300))))
p <- p + geom_line(aes(x=Time, y=Probability, linetype=Item))
p <- p + xlab("Time") + ylab("Survival Probability")
p
```

`nonParBayesSystemInferencePriorSets` *Non-parametric Bayesian posterior predictive system survival inference using sets of priors*

Description

Computes a non-parametric Bayesian posterior predictive survival probability given the survival signature of a system, test data on each of the components and a set of priors. This is the methodology described in Walter *et al* (2017), which extends the method in `nonParBayesSystemInference` (Aslett *et al*, 2015) to allow modelling imperfect prior knowledge.

Usage

```
nonParBayesSystemInferencePriorSets(at.times, survival.signature, test.data,
                            nLower=2, nUpper=2, yLower=0.5, yUpper=0.5, cores=NA)
```

Arguments

`at.times` a vector of times at which the posterior predictive estimate of survival probability should be computed.

`survival.signature` the survival signature matrix of the system/network for which inference is performed. This should be in the same format as returned by `computeSystemSurvivalSignature`.

`test.data` a list of vectors containing the component test data. The elements of the list should be named identically to the component columns in the `survival.signature` argument.

`nLower`, `nUpper` the reparameterised lower/upper prior parameter n for the Beta distribution, where $n = \alpha + \beta$. Each must match in type and can be:

- a single scalar for a fixed prior across time and component types;
- a vector of parameters of the same length as the `at.times` argument, which indicates the time-varying prior parameter at the corresponding time in `at.times`. This is therefore time-varying, but indicates the same time-varying prior for all component types;
- or a data frame where each column is named using the same names as for the `survival.signature` argument and each row corresponds to the time-varying prior parameter at the corresponding time in `at.times`.

In all cases, `nUpper` but be elementwise greater than or equal to `nLower`.

By default the 'uninformative' (but certain) prior with `nLower=2` and `nUpper=1` is used for all components at all times.

`yLower`, `yUpper` the reparameterised lower/upper prior parameter y for the Beta distribution, where $y = \alpha/(\alpha + \beta)$. Each must match in type and can be:

- a single scalar for a fixed prior across time and component types;
- a vector of parameters of the same length as the `at.times` argument, which indicates the time-varying prior parameter at the corresponding time in `at.times`. This is therefore time-varying, but indicates the same time-varying prior for all component types;
- or a data frame where each column is named using the same names as for the `survival.signature` argument and each row corresponds to the time-varying prior parameter at the corresponding time in `at.times`.

In all cases, `yUpper` but be elementwise greater than or equal to `yLower`.

By default the 'uninformative' (but certain) prior with `yLower=0.5` and `yUpper=0.5` is used for all components at all times.

`cores` a scalar indicating how many CPU cores on which to execute parallel parts of the algorithm (uses the `parallel` library internally).

Details

This function implements the technique described in Walter *et al* (2017), which extends the methodology of Aslett *et al* (2015) to allow modelling partial or imperfect prior knowledge on component failure distributions.

In brief Aslett *et al* (2015) consider, at any fixed time t, the functioning of a single component of type k to be Bernoulli(p_t^k) distributed for suitable p_t^k, irrespective of the lifetime distribution of the component. Correspondingly, the distribution of the number of components still functioning at time t in a collection of n_k iid components of type k is Binomial(n_k, p_t^k).

Taking the priors $p_t^k \sim$ Beta(α_t^k, β_t^k), Aslett *et al* (2015) show that this leads to a posterior predictive survival distribution with a nice closed form (see equations 9 and 10 in Section 4).

Walter *et al* (2017) use the standard reparameterisation (dropping sub/super-scripts for readability) $n = \alpha + \beta$ and $y = \alpha/(\alpha + \beta)$. This allows a more natural interpretation, where n represents the prior strength (it represents a pseudo-count for number of failures informing the prior specification) and y represents the prior expectation for the probability a component functions.

In particular, Walter *et al* (2017) then enable imprecise prior specification by allowing lower and upper bounds on n and y, which may optionally be time varying. This is then propagated to construct bounds on the posterior predictive distribution for the functioning of a new system containing components exchangeable with those provided in the testing set and used in a system with design specified by the survival signature provided.

Value

A list containing two slots, `lower` and `upper`, each of which is a vector of the same length as the `at.times` argument, where each element is the lower/upper posterior predictive probability of a new system surviving to the corresponding time in `at.times`.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J. M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Walter, G., Aslett, L. J. M. and Coolen, F. P. A. (2017), 'Bayesian nonparametric system reliability using sets of priors', *International Journal of Approximate Reasoning*, **80**, 67–88. Download paper

Aslett, L. J. M., Coolen, F. P. A. and Wilson, S. P. (2015), 'Bayesian Inference for Reliability of Systems and Networks using the Survival Signature', *Risk Analysis* **39**(9), 1640–1651. Download paper

See Also

computeSystemSurvivalSignature, and also nonParBayesSystemInference which is the precise counterpart to this method.

Examples

```
## Exactly reproduce the toy bridge system example in Section 7.2.1 of Walter et al (2017)

# Produces survival signature matrix for one component of type "name", for use
# in nonParBayesSystemInference()
oneCompSurvSign <- function(name){
  res <- data.frame(name=c(0,1), Probability=c(0,1))
  names(res)[1] <- name
  res
}

# Produces data frame with prior and posterior lower & upper component survival
# function for component of type "name" based on
# nonParBayesSystemInferencePriorSets() inputs for all components except
# survival signature; nLower, nUpper, yLower, yUpper must be data frames where
# each column corresponds to the component type, so there must be a match
oneCompPriorPostSet <- function(name, at.times, test.data, nLower, nUpper, yLower, yUpper){
  sig <- oneCompSurvSign(name)
  nodata <- list(name=NULL)
  names(nodata) <- name
  nL <- nLower[, match(name, names(nLower))]
  nU <- nUpper[, match(name, names(nUpper))]
  yL <- yLower[, match(name, names(yLower))]
  yU <- yUpper[, match(name, names(yUpper))]
  data <- test.data[match(name, names(test.data))]
  # NB limit to 1 core on CRAN due to Windows -- make larger to speed up locally!
 prio <- nonParBayesSystemInferencePriorSets(at.times, sig, nodata, nL, nU, yL, yU, cores = 1)
 post <- nonParBayesSystemInferencePriorSets(at.times, sig,  data, nL, nU, yL, yU, cores = 1)
  data.frame(Time=rep(at.times,2),
             Lower=c(prio$lower,post$lower),
             Upper=c(prio$upper,post$upper),
             Item=rep(c("Prior", "Posterior"), each=length(at.times)))
}

# ------------------------------------------------

# System
b3 <- createSystem(s -- 2:3 -- 4 -- 5:6 -- 1 -- t, 2 -- 5, 3 -- 6,
                   types = list(T1 = c(2,3,5,6), T2 = c(4), T3 = c(1)))

# Data
b3nulldata <- list("T1"=NULL, "T2"=NULL, "T3"=NULL)
```

```
b3testdata <- list("T1"=c(2.2, 2.4, 2.6, 2.8),
                   "T2"=c(3.2, 3.4, 3.6, 3.8),
                   "T3"=(1:4)/10+4) # T3 late failures
b3testdata <- list("T1"=c(2.2, 2.4, 2.6, 2.8),
                   "T2"=c(3.2, 3.4, 3.6, 3.8),
                   "T3"=(1:4)/10+0.5) # T3 early failures
b3testdata <- list("T1"=c(2.2, 2.4, 2.6, 2.8),
                   "T2"=c(3.2, 3.4, 3.6, 3.8),
                   "T3"=(1:4)-0.5) # T3 fitting failures
b3dat <- reshape2::melt(b3testdata); names(b3dat) <- c("x", "Part")
b3dat$Part <- ordered(b3dat$Part, levels=c("T1", "T2", "T3", "System"))

# Setup to run
b3sig <- computeSystemSurvivalSignature(b3)
b3t <- seq(0, 5, length.out=301)
b3nL <- data.frame(T1=rep(1,301), T2=rep(1,301), T3=rep(1,301))
b3nU <- data.frame(T1=rep(2,301), T2=rep(2,301), T3=rep(4,301))
b3yL <- data.frame(T1=rep(0.001, 301),
                   T2=rep(0.001, 301),
                   T3=c(rep(c(0.625,0.375,0.250,0.125,0.010), each=60), 0.01))
b3yU <- data.frame(T1=rep(0.999, 301),
                   T2=rep(0.999, 301),
                   T3=c(rep(c(0.999,0.875,0.500,0.375,0.250), each=60), 0.25))

b3T1 <- oneCompPriorPostSet("T1", b3t, b3testdata, b3nL, b3nU, b3yL, b3yU)
b3T2 <- oneCompPriorPostSet("T2", b3t, b3testdata, b3nL, b3nU, b3yL, b3yU)
b3T3 <- oneCompPriorPostSet("T3", b3t, b3testdata, b3nL, b3nU, b3yL, b3yU)

# Compute prior and posterior sets!!
# NB limit to 1 core on CRAN due to Windows -- make larger to speed up locally!
b3prio <- nonParBayesSystemInferencePriorSets(b3t, b3sig, b3nulldata,
                                              b3nL, b3nU, b3yL, b3yU, cores = 1)
b3post <- nonParBayesSystemInferencePriorSets(b3t, b3sig, b3testdata,
                                              b3nL, b3nU, b3yL, b3yU, cores = 1)

b3df <- rbind(data.frame(b3T1, Part="T1"),
              data.frame(b3T2, Part="T2"),
              data.frame(b3T3, Part="T3"),
              data.frame(Time=rep(b3t,2),
                         Lower=c(b3prio$lower,b3post$lower),
                         Upper=c(b3prio$upper,b3post$upper),
                    Item=rep(c("Prior", "Posterior"), each=length(b3t)), Part="System"))
b3df$Item <- ordered(b3df$Item, levels=c("Prior", "Posterior"))
b3df$Part <- ordered(b3df$Part, levels=c("T1", "T2", "T3", "System"))


library("ggplot2")
library("xtable")

ggplot(b3df, aes(x=Time)) +
  scale_fill_manual(values = c("#b2df8a", "#1f78b4")) +
  scale_colour_manual(values = c("#b2df8a", "#1f78b4")) +
  geom_line(aes(y=Upper, group=Item, colour=Item)) +
```

```
  geom_line(aes(y=Lower, group=Item, colour=Item)) +
 geom_ribbon(aes(ymin=Lower, ymax=Upper, group=Item, colour=Item, fill=Item), alpha=0.5) +
  facet_wrap(~Part, nrow=2) +
  geom_rug(aes(x=x), data=b3dat) +
  xlab("Time") +
  ylab("Survival Probability") +
  theme_bw() +
  theme(legend.title = element_blank())

b3sigtable <- b3sig[b3sig$T3 == 1,]
b3sigtable$T1 <- as.factor(b3sigtable$T1)
b3sigtable$T2 <- as.factor(b3sigtable$T2)
b3sigtable$T3 <- as.factor(b3sigtable$T3)
xtable(b3sigtable)
```

`sccsO2` *Catalogue of Simply Connected Coherent Systems of Order 2*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all simply connected coherent systems of order 2.

Usage

```
data(sccsO2)
```

Format

A list object, one item for each such systems. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

sccs03 *Catalogue of Simply Connected Coherent Systems of Order 3*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all simply connected coherent systems of order 3.

Usage

```
data(sccs03)
```

Format

A list object, one item for each such systems. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

sccs04 *Catalogue of Simply Connected Coherent Systems of Order 4*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all simply connected coherent systems of order 4.

Usage

```
data(sccs04)
```

Format

A list object, one item for each such systems. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

`sccs05` *Catalogue of Simply Connected Coherent Systems of Order 5*

Description

This data set provides a catalogue of the network graph, signature and minimal cut-sets of all simply connected coherent systems of order 5.

Usage

```
data(sccs05)
```

Format

A list object, one item for each such systems. Each item is itself a list, with the elements `$graph`, `$cutsets` and `$signature`.

Source

Derived in the thesis Aslett (2012).

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

`setCompTypes` *Set component types in a system*

Description

A system created with `createSystem` by default assumes components to be of the same type. This function enables assigning particular types (or modified from a previous types assignment) to each component after system creation.

Usage

```
setCompTypes(sys, types)
```

Arguments

sys	a system object representing the system whose types are to be set. This should have been created by a call to createSystem.
types	a named list of vectors. The names correspond to component types, whilst each vector indicates which components are of that type.

Details

This function enables specifying (or modifying) the types of the components in a system. The types can be specified when the system is initially defined using createSystem, but if none are specified at that time then it is assumed all components are of the same type.

The types argument should be a named list of vectors, for example list("a" = 1:3, "b" = c(4,6), "c" = 5) would specify that component numbers 1 through 3 are of type a, 4 and 6 are type b, with the remaining component 5 as type c. The numbering should match numbering used when creating the system. The start and terminal nodes should not be given a type (as they are assumed perfectly reliable).

Value

The system which was passed in the sys argument is returned with the component types updated.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

See Also

createSystem

Examples

```
## EXAMPLE 1
## Figure 1 in Coolen and Coolen-Maturi (2012)

# First, define the structure, ensuring that each physically separate component
# is separately numbered
fig1 <- createSystem(s -- 1 -- 2:3 -- 4 -- 5:6 -- t, 2 -- 5, 3 -- 6)

# Second, assign types to the components with this function
setCompTypes(fig1, list("Type 1" = c(1, 2, 5), "Type 2" = c(3, 4, 6)))

# Note that one can create the same system and avoid using setCompTypes by
# specifying the types in the initial call to createSystem if desired.
# The following code results in exactly the same system specification as fig1:
fig1b <- createSystem(s -- 1 -- 2:3 -- 4 -- 5:6 -- t, 2 -- 5, 3 -- 6,
                      types = list("Type 1" = c(1, 2, 5), "Type 2" = c(3, 4, 6)))
```

`simulateSystem` *Simulate Masked Lifetime Data for a System*

Description

This function enables easy simulation of iid masked lifetime observations from a system or network.

Usage

```
simulateSystem(system, n, rdens, ...)
```

Arguments

`system`	may be: a `system` object (made with `createSystem` representing the system; the collection of cutsets of the system; or the system signature.
`n`	how many simulations to produce.
`rdens`	a user defined function which generates random realisations of the component lifetimes.
`...`	parameters passed to the user defined function `rdens`.

Details

The system or network is specified by means of a `system` object, whereby each end of the system is denoted by nodes named `s` and `t` which are taken to be perfectly reliable. It is easy to construct the appropriate reliability block diagram representation using the function `createSystem`. Note that each physically distinct component should be separately numbered when constructing this object.

This function then generates iid realisations of masked lifetimes.

Value

a numeric vector of length `n` containing the masked lifetime data.

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

Examples

```
# Simulate 20 masked lifetimes of a two component series system with Exponential(2)
# component lifetimes
# Using igraph object ...
simulateSystem(createSystem(s -- 1 -- 2 -- t), 20, rexp, rate=2)

# ... and using signature
simulateSystem(c(1,0), 20, rexp, rate=2)
```

`systemGraphToGenerator` *Construct a Continuous-time Markov Chain Generator*

Description

This function enables easy construction of an absorbing continuous-time Markov chain generator matrix representation for a system when components are treated as having Exponential failure and repair times.

Usage

```
systemGraphToGenerator(g, failRate, repairRate)
```

Arguments

`g`	an `igraph` object representing the system or network whose generator matrix representation is to be computed. There should be two terminal 'dummy' nodes to represent either end of the structure which must be labelled "s" and "t". They are assumed perfectly reliable. See details and examples.
`failRate`	the rate parameter of the Exponentially distributed lifetime distribution of the components.
`repairRate`	the rate parameter of the Exponentially distributed repair time distribution of the components.

Details

When the system or network is specified by means of an `igraph` object, each end of the system must be denoted by nodes named "s" and "t" which are taken to be perfectly reliable. It is easy to construct the appropriate graph representation using the function `graph.formula`.

This function then creates the generator matrix for an absorbing continuous-time Markov chain representation of such a system where components are repairable. All system states which in which the system is inoperative are collapsed into the absorbing state.

The returned values are in the format required by the `phtMCMC2`.

Full details are in Aslett (2012).

Value

A list is returned with both a numeric generator matrix (in `$G` with the failure rate, `failRate`, and repair rate, `repairRate`) and a symbolic matrix (in `$structure$G`), along with a matrix of the constant multiples of generator entries (in `$structure$C`).

Note

Please feel free to email <louis.aslett@durham.ac.uk> with any queries or if you encounter errors when running this function.

Author(s)

Louis J.M. Aslett <louis.aslett@durham.ac.uk> (https://www.louisaslett.com/)

References

Aslett, L. J. M. (2012), *MCMC for Inference on Phase-type and Masked System Lifetime Models*, PhD Thesis, Trinity College Dublin.

Examples

```
# Get the generator representing a repairable 5 component 'bridge' system with
# failure rate 1 and repair rate 365.
data(sccs05)
G <- systemGraphToGenerator(sccs05[[18]]$graph, 1, 365)
```

Index