# Package 'missSBM'

February 1, 2022

**Type** Package

**Title** Handling Missing Data in Stochastic Block Models

**Version** 1.0.2

**Maintainer** Julien Chiquet <julien.chiquet@inrae.fr>

**Description**
When a network is partially observed (here, NAs in the adjacency matrix rather than 1 or 0
due to missing information between node pairs), it is possible to account for the underlying process
that generates those NAs. 'missSBM', presented in 'Barbillon, Chi-
quet and Tabouy' (2021) <doi:10.18637/jss.v101.i12>,
adjusts the popular stochastic block model from network data sampled under various miss-
ing data conditions,
as described in 'Tabouy, Barbillon and Chiquet' (2019) <doi:10.1080/01621459.2018.1562934>.

**URL** https://grosssbm.github.io/missSBM/

**BugReports** https://github.com/grossSBM/missSBM/issues

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.2

**Depends** R (>= 3.4.0)

**Imports** Rcpp, methods, igraph, nloptr, ggplot2, future.apply, R6,
rlang, sbm, magrittr, Matrix, RSpectra

**LinkingTo** Rcpp, RcppArmadillo, nloptr

**Collate** 'utils_missSBM.R' 'R6Class-networkSampling.R'
'R6Class-networkSampling_fit.R' 'R6Class-simpleSBM_fit.R'
'R6Class-missSBM_fit.R' 'R6Class-missSBM_collection.R'
'R6Class-networkSampler.R' 'R6Class-partlyObservedNetwork.R'
'RcppExports.R' 'er_network.R' 'estimateMissSBM.R'
'frenchblog2007.R' 'kmeans.R' 'missSBM-package.R'
'observeNetwork.R' 'war.R'

**Suggests** aricode, blockmodels, corrplot, future, testthat (>= 2.1.0),
covr, knitr, rmarkdown, spelling

**VignetteBuilder** knitr

**Language** en-US

**NeedsCompilation** yes

**Author** Julien Chiquet [aut, cre] (<<https://orcid.org/0000-0002-3629-3429>>),
  Pierre Barbillon [aut] (<<https://orcid.org/0000-0002-7766-7693>>),
  Timothée Tabouy [aut],
  Jean-Benoist Léger [ctb] (provided C++ implementaion of K-means),
  François Gindraud [ctb] (provided C++ interface to NLopt),
  großBM team [ctb]

**Repository** CRAN

**Date/Publication** 2022-02-01 16:00:20 UTC

# R **topics documented:**

blockDyadSampler *Class for defining a block dyad sampler*

## Description

Class for defining a block dyad sampler

Class for defining a block dyad sampler

## Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::dyadSampler](#) -> blockDyadSampler

## Active bindings

df the number of parameters of this sampling

## Methods

### Public methods:

- [blockDyadSampler$new()](#)
- [blockDyadSampler$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

```
blockDyadSampler$new(
  parameters = NA,
  nbNodes = NA,
  directed = FALSE,
  clusters = NA
)
```

*Arguments:*

parameters the vector of parameters associated to the sampling at play

nbNodes number of nodes in the network

directed logical, directed network of not

clusters a vector of class memberships

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
blockDyadSampler$clone(deep = FALSE)
```

*Arguments:*

deep  Whether to make a deep clone.

---

blockDyadSampling_fit   *Class for fitting a block-dyad sampling*

---

## Description

Class for fitting a block-dyad sampling

Class for fitting a block-dyad sampling

## Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingDyads_fit](#) -> blockDyadSampling_fit

## Active bindings

vExpec  variational expectation of the sampling

log_lambda  matrix, term for adjusting the imputation step which depends on the type of sampling

## Methods

### Public methods:

- [blockDyadSampling_fit$new()](#)
- [blockDyadSampling_fit$update_parameters()](#)
- [blockDyadSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

```
blockDyadSampling_fit$new(partlyObservedNetwork, blockInit)
```

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed data with possibly missing entries

blockInit  n x Q matrix of initial block indicators

**Method** update_parameters(): a method to update the estimation of the parameters. By default, nothing to do (corresponds to MAR sampling)

*Usage:*

```
blockDyadSampling_fit$update_parameters(nu, Z)
```

*Arguments:*

nu  the matrix of (uncorrected) imputation for missing entries

Z probabilities of block memberships

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*
blockDyadSampling_fit$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

blockNodeSampler  *Class for defining a block node sampler*

---

### Description

Class for defining a block node sampler

Class for defining a block node sampler

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::nodeSampler](#) -> blockNodeSampler

### Methods

#### Public methods:

- [blockNodeSampler$new()](#)
- [blockNodeSampler$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*
```
blockNodeSampler$new(
  parameters = NA,
  nbNodes = NA,
  directed = FALSE,
  clusters = NA
)
```

*Arguments:*

parameters the vector of parameters associated to the sampling at play

nbNodes number of nodes in the network

directed logical, directed network of not

clusters a vector of class memberships

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*
blockNodeSampler$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

blockNodeSampling_fit    *Class for fitting a block-node sampling*

---

### Description

Class for fitting a block-node sampling

Class for fitting a block-node sampling

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingNodes_fit](#) -> blockNodeSampling_fit

### Active bindings

vExpec  variational expectation of the sampling

log_lambda  double, term for adjusting the imputation step which depends on the type of sampling

### Methods

#### Public methods:

- [blockNodeSampling_fit$new()](#)
- [blockNodeSampling_fit$update_parameters()](#)
- [blockNodeSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

blockNodeSampling_fit$new(partlyObservedNetwork, blockInit)

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed data with possibly missing entries

blockInit  n x Q matrix of initial block indicators

**Method** update_parameters(): a method to update the estimation of the parameters. By default, nothing to do (corresponds to MAR sampling)

*Usage:*

blockNodeSampling_fit$update_parameters(imputedNet, Z)

*Arguments:*

imputedNet  an adjacency matrix where missing values have been imputed

Z  indicator of blocks

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

blockNodeSampling_fit$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

coef.missSBM_fit            *Extract model coefficients*

---

### Description

Extracts model coefficients from objects `missSBM_fit` returned by `estimateMissSBM()`

### Usage

```
## S3 method for class 'missSBM_fit'
coef(
  object,
  type = c("mixture", "connectivity", "covariates", "sampling"),
  ...
)
```

### Arguments

| | |
|---|---|
| object | an R6 object with class `missSBM_fit` |
| type | type of parameter that should be extracted. Either "mixture" (default), "connectivity", "covariates" or "sampling" |
| ... | additional parameters for S3 compatibility. Not used |

### Value

A vector or matrix of coefficients extracted from the missSBM_fit model.

---

covarDyadSampling_fit  *Class for fitting a dyad sampling with covariates*

---

### Description

Class for fitting a dyad sampling with covariates

Class for fitting a dyad sampling with covariates

### Super classes

`missSBM::networkSampling` -> `missSBM::networkSamplingDyads_fit` -> `covarDyadSampling_fit`

### Active bindings

vExpec  variational expectation of the sampling

## Methods

### Public methods:

- covarDyadSampling_fit$new()
- covarDyadSampling_fit$clone()

**Method** new(): constructor

*Usage:*

covarDyadSampling_fit$new(partialNet, ...)

*Arguments:*

partialNet a object with class partlyObservedNetwork representing the observed data with possibly missing entries

... used for compatibility

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

covarDyadSampling_fit$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

covarNodeSampling_fit *Class for fitting a node-centered sampling with covariate*

---

## Description

Class for fitting a node-centered sampling with covariate

Class for fitting a node-centered sampling with covariate

## Super classes

missSBM::networkSampling -> missSBM::networkSamplingNodes_fit -> covarNodeSampling_fit

## Active bindings

vExpec variational expectation of the sampling

## Methods

### Public methods:

- covarNodeSampling_fit$new()
- covarNodeSampling_fit$clone()

**Method** new(): constructor

*Usage:*

```
covarNodeSampling_fit$new(partlyObservedNetwork, ...)
```

*Arguments:*

partlyObservedNetwork a object with class partlyObservedNetwork representing the observed
data with possibly missing entries

`...` used for compatibility

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
covarNodeSampling_fit$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

degreeSampler *Class for defining a degree sampler*

---

### Description

Class for defining a degree sampler

Class for defining a degree sampler

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::nodeSampler](#) -> degreeSampler

### Methods

#### Public methods:

- [degreeSampler$new()](#)
- [degreeSampler$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

```
degreeSampler$new(parameters = NA, degrees = NA, directed = FALSE)
```

*Arguments:*

parameters the vector of parameters associated to the sampling at play

degrees vector of nodes' degrees

directed logical, directed network of not

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
degreeSampler$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

degreeSampling_fit *Class for fitting a degree sampling*

---

### Description

Class for fitting a degree sampling

Class for fitting a degree sampling

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingNodes_fit](#) -> degreeSampling_fit

### Active bindings

vExpec variational expectation of the sampling

### Methods

#### Public methods:

- [degreeSampling_fit$new()](#)
- [degreeSampling_fit$update_parameters()](#)
- [degreeSampling_fit$update_imputation()](#)
- [degreeSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

degreeSampling_fit$new(partlyObservedNetwork, blockInit, connectInit)

*Arguments:*

partlyObservedNetwork a object with class partlyObservedNetwork representing the observed data with possibly missing entries

blockInit n x Q matrix of initial block indicators

connectInit Q x Q matrix of initial block probabilities of connection

**Method** update_parameters(): a method to update the estimation of the parameters. By default, nothing to do (corresponds to MAR sampling)

*Usage:*

degreeSampling_fit$update_parameters(imputedNet, ...)

*Arguments:*

imputedNet an adjacency matrix where missing values have been imputed

... used for compatibility

**Method** update_imputation(): a method to update the imputation of the missing entries.

*Usage:*

degreeSampling_fit$update_imputation(PI, ...)

*Arguments:*

PI  the matrix of inter/intra class probability of connection

...  use for compatibility

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

degreeSampling_fit$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

doubleStandardSampler  *Class for defining a double-standard sampler*

---

**Description**

Class for defining a double-standard sampler

Class for defining a double-standard sampler

**Super classes**

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::dyadSampler](#) -> doubleStandardSampler

**Methods**

**Public methods:**

- [doubleStandardSampler$new()](#)
- [doubleStandardSampler$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

doubleStandardSampler$new(parameters = NA, adjMatrix = NA, directed = FALSE)

*Arguments:*

parameters  the vector of parameters associated to the sampling at play

adjMatrix  matrix of adjacency

directed  logical, directed network of not

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

doubleStandardSampler$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

doubleStandardSampling_fit

*Class for fitting a double-standard sampling*

---

### Description

Class for fitting a double-standard sampling

Class for fitting a double-standard sampling

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingDyads_fit](#) -> doubleStandardSampling_fit

### Active bindings

vExpec  variational expectation of the sampling

### Methods

#### Public methods:

- [doubleStandardSampling_fit$new()](#)
- [doubleStandardSampling_fit$update_parameters()](#)
- [doubleStandardSampling_fit$update_imputation()](#)
- [doubleStandardSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

doubleStandardSampling_fit$new(partlyObservedNetwork, ...)

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed
    data with possibly missing entries

...  used for compatibility

**Method** update_parameters():  a method to update the estimation of the parameters. By
default, nothing to do (corresponds to MAR sampling)

*Usage:*

doubleStandardSampling_fit$update_parameters(nu, ...)

*Arguments:*

nu  an adjacency matrix with imputed values (only)

...  use for compatibility

**Method** update_imputation(): a method to update the imputation of the missing entries.

*Usage:*

doubleStandardSampling_fit$update_imputation(nu)

*Arguments:*

nu  the matrix of (uncorrected) imputation for missing entries

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`doubleStandardSampling_fit$clone(deep = FALSE)`

*Arguments:*

deep  Whether to make a deep clone.

---

dyadSampler                    *Virtual class for all dyad-centered samplers*

---

**Description**

Virtual class for all dyad-centered samplers

Virtual class for all dyad-centered samplers

**Super classes**

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> dyadSampler

**Methods**

**Public methods:**

- [dyadSampler$new()](#)
- [dyadSampler$clone()](#)

**Method** `new()`: constructor for networkSampling

*Usage:*

`dyadSampler$new(type = NA, parameters = NA, nbNodes = NA, directed = FALSE)`

*Arguments:*

type  character for the type of sampling.  must be in ("dyad", "covar-dyad", "node", "covar-node", "block-node", "block-dyad", "double-standard", "degree")

parameters  the vector of parameters associated to the sampling at play

nbNodes  number of nodes in the network

directed  logical, directed network of not

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`dyadSampler$clone(deep = FALSE)`

*Arguments:*

deep  Whether to make a deep clone.

---

dyadSampling_fit     *Class for fitting a dyad sampling*

---

## Description

Class for fitting a dyad sampling

Class for fitting a dyad sampling

## Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingDyads_fit](#) -> dyadSampling_fit

## Active bindings

vExpec  variational expectation of the sampling

## Methods

### Public methods:

- [dyadSampling_fit$new()](#)
- [dyadSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

dyadSampling_fit$new(partlyObservedNetwork, ...)

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed data with possibly missing entries

...  used for compatibility

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

dyadSampling_fit$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

er_network                          *ER ego centered network*

---

## Description

A dataset containing the weighted PPI network centered around the ESR1 (ER) protein

## Usage

```
er_network
```

## Format

A sparse symmetric matrix with 741 rows and 741 columns ESR1

## Source

<https://string-db.org/>

## Examples

```
data("er_network")
class(er_network)
```

---

estimateMissSBM                     *Estimation of simple SBMs with missing data*

---

## Description

Variational EM inference of Stochastic Block Models indexed by block number from a partially observed network.

## Usage

```
estimateMissSBM(
  adjacencyMatrix,
  vBlocks,
  sampling,
  covariates = list(),
  control = list()
)
```

## Arguments

adjacencyMatrix

        The N x N adjacency matrix of the network data. If `adjacencyMatrix` is symmetric, we assume an undirected network with no loop; otherwise the network is assumed to be directed.

vBlocks        The vector of number of blocks considered in the collection.

sampling       The model used to described the process that originates the missing data: MAR designs ("dyad", "node","covar-dyad","covar-node","snowball") and MNAR designs ("double-standard", "block-dyad", "block-node" , "degree") are available. See details.

covariates     An optional list with M entries (the M covariates). If the covariates are node-centered, each entry of `covariates` must be a size-N vector; if the covariates are dyad-centered, each entry of `covariates` must be N x N matrix.

control        a list of parameters controlling advanced features. See details.

## Details

Internal functions use `future_lapply`, so set your plan to 'multisession' or 'multicore' to use several cores/workers. The list of parameters `control` tunes more advanced features, such as the initialization, how covariates are handled in the model, and the variational EM algorithm:

- "useCov": logical. If `covariates` is not null, should they be used for the for the SBM inference (or just for the sampling)? Default is TRUE.

- "clusterInit": Initial method for clustering: either a character ("spectral") or a list with `length(vBlocks)` vectors, each with size `ncol(adjacencyMatrix)`, providing a user-defined clustering. Default is "spectral".

- "similarity": An R x R -> R function to compute similarities between node covariates. Default is `missSBM:::l1_similarity`, that is, -abs(x-y). Only relevant when the covariates are node-centered (i.e. `covariates` is a list of size-N vectors).

- "threshold": V-EM algorithm stops stop when an optimization step changes the objective function or the parameters by less than threshold. Default is 1e-2.

- "maxIter": V-EM algorithm stops when the number of iteration exceeds maxIter. Default is 50.

- "fixPointIter": number of fix-point iterations in the V-E step. Default is 3.

- "exploration": character indicating the kind of exploration used among "forward", "backward", "both" or "none". Default is "both".

- "iterates": integer for the number of iterations during exploration. Only relevant when `exploration` is different from "none". Default is 1.

- "trace": logical for verbosity. Default is TRUE.

The different sampling designs are split into two families in which we find dyad-centered and node-centered samplings. See doi: [10.1080/01621459.2018.1562934](10.1080/01621459.2018.1562934) for a complete description.

- Missing at Random (MAR)

    – "dyad": parameter = p = Prob(Dyad(i,j) is observed)

– "node": parameter = p = Prob(Node i is observed)

– "covar-dyad": parameter = beta in R^M, such that Prob(Dyad (i,j) is observed) = logistic(parameter' covarArray (i,j, .))

– "covar-node": parameter = nu in R^M such that Prob(Node i is observed) = logistic(parameter' covarMatrix (i,)

– "snowball": parameter = number of waves with Prob(Node i is observed in the 1st wave)

• Missing Not At Random (MNAR)

– "double-standard": parameter = (p0,p1) with p0 = Prob(Dyad (i,j) is observed | the dyad is equal to 0), p1 = Prob(Dyad (i,j) is observed | the dyad is equal to 1)

– "block-node": parameter = c(p(1),...,p(Q)) and p(q) = Prob(Node i is observed | node i is in cluster q)

– "block-dyad": parameter = c(p(1,1),...,p(Q,Q)) and p(q,l) = Prob(Edge (i,j) is observed | node i is in cluster q and node j is in cluster l)

– "degree": parameter = c(a,b) and logit(a+b*degree(i)) = Prob(Node i is observed | Degree(i))

## Value

Returns an R6 object with class `missSBM_collection`.

## See Also

`observeNetwork`, `missSBM_collection` and `missSBM_fit`.

## Examples

```
## SBM parameters
N <- 100 # number of nodes
Q <- 3   # number of clusters
pi <- rep(1,Q)/Q     # block proportion
theta <- list(mean = diag(.45,Q) + .05 ) # connectivity matrix

## Sampling parameters
samplingParameters <- .75 # the sampling rate
sampling  <- "dyad"       # the sampling design

## generate a undirected binary SBM with no covariate
sbm <- sbm::sampleSimpleSBM(N, pi, theta)

## Uncomment to set parallel computing with future
## future::plan("multicore", workers = 2)

## Sample some dyads data + Infer SBM with missing data
collection <-
   observeNetwork(sbm$networkData, sampling, samplingParameters) %>%
   estimateMissSBM(vBlocks = 1:5, sampling = sampling)
plot(collection, "monitoring")
plot(collection, "icl")

collection$ICL
```

```
coef(collection$bestModel$fittedSBM, "connectivity")

myModel <- collection$bestModel
plot(myModel, "expected")
plot(myModel, "imputed")
plot(myModel, "meso")
coef(myModel, "sampling")
coef(myModel, "connectivity")
predict(myModel)[1:5, 1:5]
```

---

fitted.missSBM_fit          *Extract model fitted values from object* missSBM_fit, *return by* estimateMissSBM()

---

### Description

Extract model fitted values from object missSBM_fit, return by estimateMissSBM()

### Usage

```
## S3 method for class 'missSBM_fit'
fitted(object, ...)
```

### Arguments

| | |
|---|---|
| object | an R6 object with class missSBM_fit |
| ... | additional parameters for S3 compatibility. |

### Value

A matrix of estimated probabilities of connection

---

frenchblog2007          *Political Blogosphere network prior to 2007 French presidential election*

---

### Description

French Political Blogosphere network dataset consists of a single day snapshot of over 200 political blogs automatically extracted the 14 October 2006 and manually classified by the "Observatoire Présidentielle" project. Originally part of the 'mixer' package

### Usage

```
frenchblog2007
```

## Format

An igraph object with 196 nodes. The vertex attribute "party" provides a possible clustering of the nodes.

## Source

<https://www.linkfluence.com/>

## Examples

```
data(frenchblog2007)
igraph::V(frenchblog2007)$party
igraph::plot.igraph(frenchblog2007,
  vertex.color = factor(igraph::V(frenchblog2007)$party),
  vertex.label = NA
 )
```

---

| missSBM | *Adjusting Stochastic Block Models under various missing data conditions* |

---

## Description

The missSBM package provides the following top-level functions functions:

- observeNetworka function to draw a partially observe network from an existing, fully observed network according to a variety of sampling designs
- estimateMissSBMa function to perform inference of SBM from a partially observed under various sampling designs.

## Details

These function leads to the manipulation of a variety of R objects instantiated from some R6 classes, with their respective fields and methods. They are all generated by the top-level functions itemized above, so that the user should generally not use their constructor or internal methods directly. The user should only have a basic understanding of the fields of each object to manipulate the output in R. The main objects are the following:

- missSBM_fitan object that put together an SBM fit and and network sampling fit - the main point of the missSBM package !
- missSBM_collectionan object to store a collection of missSBM_fit, ordered by number of block
- SimpleSBM_fit_MNARan object to define and store an SBM fit with MNAR values
- SimpleSBM_fit_noCovan object to define and store an SBM fit without covariate, MAR values
- SimpleSBM_fit_withCovan object to define and store an SBM fit with covariates, MAR values

- `networkSampling`an object to define and store a network sampling fit

missSBM extends some functionality of the package sbm, by inheriting from classes and methods associated to simple stochastic block models.

## Author(s)

Pierre Barbillon <pierre.barbillon@agroparistech.fr>

Julien Chiquet <julien.chiquet@inrae.fr>

Timothée Tabouy <timothee.tabouy@gmail.com>

## References

Pierre Barbillon, Julien Chiquet & Timothée Tabouy (2022) "missSBM: An R Package for Handling Missing Values in the Stochastic Block Model", Journal of Statistical Software, doi: 10.18637/jss.v101.i12

Timothée Tabouy, Pierre Barbillon & Julien Chiquet (2019) "Variational Inference for Stochastic Block Models from Sampled Data", Journal of the American Statistical Association, doi: 10.1080/01621459.2018.1562934

---

missSBM_collection            *An R6 class to represent a collection of SBM fits with missing data*

---

## Description

The function `estimateMissSBM()` fits a collection of SBM with missing data for a varying number of block. These models with class `missSBM_fit` are stored in an instance of an object with class `missSBM_collection`, described here.

Fields are accessed via active binding and cannot be changed by the user.

This class comes with a set of R6 methods, some of them being useful for the user and exported as S3 methods. See the documentation for `show()` and `print()`

## Active bindings

models  a list of models

ICL  the vector of Integrated Classification Criterion (ICL) associated to the models in the collection (the smaller, the better)

bestModel  the best model according to the ICL

vBlocks  a vector with the number of blocks

optimizationStatus  a data.frame summarizing the optimization process for all models

**Methods**

**Public methods:**

- `missSBM_collection$new()`
- `missSBM_collection$estimate()`
- `missSBM_collection$explore()`
- `missSBM_collection$plot()`
- `missSBM_collection$show()`
- `missSBM_collection$print()`
- `missSBM_collection$clone()`

**Method** new(): constructor for networkSampling

*Usage:*

`missSBM_collection$new(partlyObservedNet, sampling, clusterInit, control)`

*Arguments:*

partlyObservedNet An object with class `partlyObservedNetwork`.

sampling The sampling design for the modelling of missing data: MAR designs ("dyad", "node") and MNAR designs ("double-standard", "block-dyad", "block-node" ,"degree")

clusterInit Initial clustering: a list of vectors, each with size ncol(adjacencyMatrix).

control a list of parameters controlling advanced features. Only 'trace' and 'useCov' are relevant here. See `estimateMissSBM()` for details.

**Method** estimate(): method to launch the estimation of the collection of models

*Usage:*

`missSBM_collection$estimate(control)`

*Arguments:*

control a list of parameters controlling the variational EM algorithm. See details of function `estimateMissSBM()`

**Method** explore(): method for performing exploration of the ICL

*Usage:*

`missSBM_collection$explore(control)`

*Arguments:*

control a list of parameters controlling the exploration, similar to those found in the regular function `estimateMissSBM()`

**Method** plot(): plot method for missSBM_collection

*Usage:*

`missSBM_collection$plot(type = c("icl", "elbo", "monitoring"))`

*Arguments:*

type the type specifies the field to plot, either "icl", "elbo" or "monitoring". Default is "icl"

**Method** show(): show method for missSBM_collection

*Usage:*

```
   missSBM_collection$show()
```

**Method** `print()`: User friendly print method

   *Usage:*
```
   missSBM_collection$print()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

   *Usage:*
```
   missSBM_collection$clone(deep = FALSE)
```

   *Arguments:*

   `deep` Whether to make a deep clone.

## Examples

```
## Uncomment to set parallel computing with future
## future::plan("multicore", workers = 2)

## Sample 75% of dyads in  French political Blogosphere's network data
adjacencyMatrix <- missSBM::frenchblog2007 %>%
  igraph::delete.vertices(1:100) %>%
  igraph::as_adj () %>%
  missSBM::observeNetwork(sampling = "dyad", parameters = 0.75)
collection <- estimateMissSBM(adjacencyMatrix, 1:5, sampling = "dyad")
class(collection)
```

---

missSBM_fit                          *An R6 class to represent an SBM fit with missing data*

---

## Description

The function `estimateMissSBM()` fits a collection of SBM for varying number of block. Each fitted
SBM is an instance of an R6 object with class `missSBM_fit`, described here.

Fields are accessed via active binding and cannot be changed by the user.

This class comes with a set of R6 methods, some of them being useful for the user and exported as
S3 methods. See the documentation for `show()`, `print()`, `fitted()`, `predict()`, `plot()`.

## Active bindings

`fittedSBM` the fitted SBM with class `SimpleSBM_fit_noCov`, `SimpleSBM_fit_withCov` or `SimpleSBM_fit_MNAR`
   inheriting from class `sbm::SimpleSBM_fit`

`fittedSampling` the fitted sampling, inheriting from class `networkSampling` and corresponding
   fits

`imputedNetwork` The network data as a matrix with NAs values imputed with the current model

`monitoring` a list carrying information about the optimization process

entropyImputed the entropy of the distribution of the imputed dyads

entropy the entropy due to the distribution of the imputed dyads and of the clustering

vExpec double: variational expectation of the complete log-likelihood

penalty double, value of the penalty term in ICL

loglik double: approximation of the log-likelihood (variational lower bound) reached

ICL double: value of the integrated classification log-likelihood

## Methods

### Public methods:

- [missSBM_fit$new()](#)
- [missSBM_fit$doVEM()](#)
- [missSBM_fit$show()](#)
- [missSBM_fit$print()](#)
- [missSBM_fit$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

missSBM_fit$new(partlyObservedNet, netSampling, clusterInit, useCov = TRUE)

*Arguments:*

partlyObservedNet An object with class [partlyObservedNetwork](#).

netSampling The sampling design for the modelling of missing data: MAR designs ("dyad", "node") and MNAR designs ("double-standard", "block-dyad", "block-node" ,"degree")

clusterInit Initial clustering: a vector with size ncol(adjacencyMatrix), providing a user-defined clustering. The number of blocks is deduced from the number of levels in with clusterInit.

useCov logical. If covariates are present in partlyObservedNet, should they be used for the inference or of the network sampling design, or just for the SBM inference? default is TRUE.

**Method** doVEM(): a method to perform inference of the current missSBM fit with variational EM

*Usage:*

```
missSBM_fit$doVEM(
  control = list(threshold = 0.01, maxIter = 100, fixPointIter = 3, trace = TRUE)
)
```

*Arguments:*

control a list of parameters controlling the variational EM algorithm. See details of function [estimateMissSBM()](#)

**Method** show(): show method for missSBM_fit

*Usage:*

missSBM_fit$show()

**Method** `print()`: User friendly print method

*Usage:*
`missSBM_fit$print()`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*
`missSBM_fit$clone(deep = FALSE)`

*Arguments:*

deep  Whether to make a deep clone.

## Examples

```
## Sample 75% of dyads in  French political Blogosphere's network data
adjMatrix <- missSBM::frenchblog2007 %>%
  igraph::as_adj (sparse = FALSE) %>%
  missSBM::observeNetwork(sampling = "dyad", parameters = 0.75)
collection <- estimateMissSBM(adjMatrix, 3:5, sampling = "dyad")
my_missSBM_fit <- collection$bestModel
class(my_missSBM_fit)
plot(my_missSBM_fit, "imputed")
```

---

networkSampler                *Definition of R6 Class 'networkSampling_sampler'*

---

## Description

Definition of R6 Class 'networkSampling_sampler'

Definition of R6 Class 'networkSampling_sampler'

## Details

This class is use to define a sampling model for a network. Inherits from 'networkSampling'. Owns a rSampling method which takes an adjacency matrix as an input and send back an object with class partlyObservedNetwork.

## Super class

[missSBM::networkSampling](missSBM::networkSampling) -> networkSampler

## Active bindings

`samplingMatrix` a matrix of logical indicating observed entries

## Methods

### Public methods:

- [networkSampler$new()](networkSampler$new())
- [networkSampler$rSamplingMatrix()](networkSampler$rSamplingMatrix())
- [networkSampler$clone()](networkSampler$clone())

**Method** new(): constructor for networkSampling

*Usage:*

networkSampler$new(type = NA, parameters = NA, nbNodes = NA, directed = FALSE)

*Arguments:*

type character for the type of sampling. must be in ("dyad", "covar-dyad", "node", "covar-node", "block-node", "block-dyad", "double-standard", "degree")

parameters the vector of parameters associated to the sampling at play

nbNodes number of nodes in the network

directed logical, directed network of not

**Method** rSamplingMatrix(): a method for drawing a sampling matrix according to the current sampling design

*Usage:*

networkSampler$rSamplingMatrix()

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

networkSampler$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

## See Also

[partlyObservedNetwork](partlyObservedNetwork)

---

networkSampling    *Definition of R6 Class 'networkSampling'*

---

## Description

Definition of R6 Class 'networkSampling'

Definition of R6 Class 'networkSampling'

## Details

this virtual class is the mother of all subtypes of networkSampling (either sampler or fit) It is used to define a sampling model for a network. It has a rSampling method which takes an adjacency matrix as an input and send back an object with class partlyObservedNetwork.

**Active bindings**

type a character for the type of sampling

parameters the vector of parameters associated with the sampling at play

df the number of entries in the vector of parameters

**Methods**

**Public methods:**

- [networkSampling$new()](#)
- [networkSampling$show()](#)
- [networkSampling$print()](#)
- [networkSampling$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

```
networkSampling$new(type = NA, parameters = NA)
```

*Arguments:*

type character for the type of sampling. must be in ("dyad", "covar-dyad", "node", "covar-node", "block-node", "block-dyad", "double-standard", "degree")

parameters the vector of parameters associated to the sampling at play

**Method** show(): show method

*Usage:*

```
networkSampling$show(
  type = paste0(private$name, "-model for network sampling\n")
)
```

*Arguments:*

type character used to specify the type of sampling

**Method** print(): User friendly print method

*Usage:*

```
networkSampling$print()
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
networkSampling$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

networkSamplingDyads_fit

*Virtual class used to define a family of networkSamplingDyads_fit*

## Description

Virtual class used to define a family of networkSamplingDyads_fit

Virtual class used to define a family of networkSamplingDyads_fit

## Super class

[missSBM::networkSampling](#) -> networkSamplingDyads_fit

## Active bindings

penalty double, value of the penalty term in ICL

log_lambda double, term for adjusting the imputation step which depends on the type of sampling

## Methods

### Public methods:

- [networkSamplingDyads_fit$new()](#)
- [networkSamplingDyads_fit$show()](#)
- [networkSamplingDyads_fit$update_parameters()](#)
- [networkSamplingDyads_fit$update_imputation()](#)
- [networkSamplingDyads_fit$clone()](#)

**Method** new(): constructor for networkSampling_fit

*Usage:*
networkSamplingDyads_fit$new(partlyObservedNetwork, name)

*Arguments:*
partlyObservedNetwork a object with class partlyObservedNetwork representing the observed data with possibly missing entries

name a character for the name of sampling to fit on the partlyObservedNetwork

**Method** show(): show method

*Usage:*
networkSamplingDyads_fit$show()

**Method** update_parameters(): a method to update the estimation of the parameters. By default, nothing to do (corresponds to MAR sampling)

*Usage:*
networkSamplingDyads_fit$update_parameters(...)

*Arguments:*

`...` use for compatibility

**Method** `update_imputation()`: a method to update the imputation of the missing entries.

*Usage:*

`networkSamplingDyads_fit$update_imputation(nu)`

*Arguments:*

`nu` the matrix of (uncorrected) imputation for missing entries

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`networkSamplingDyads_fit$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

---

networkSamplingNodes_fit

*Virtual class used to define a family of networkSamplingNodes_fit*

---

### Description

Virtual class used to define a family of networkSamplingNodes_fit

Virtual class used to define a family of networkSamplingNodes_fit

### Super class

[missSBM::networkSampling](#) `-> networkSamplingNodes_fit`

### Active bindings

`penalty` double, value of the penalty term in ICL

`log_lambda` double, term for adjusting the imputation step which depends on the type of sampling

### Methods

#### Public methods:

- [networkSamplingNodes_fit$new()](#)
- [networkSamplingNodes_fit$show()](#)
- [networkSamplingNodes_fit$update_parameters()](#)
- [networkSamplingNodes_fit$update_imputation()](#)
- [networkSamplingNodes_fit$clone()](#)

**Method** `new()`: constructor

*Usage:*

`networkSamplingNodes_fit$new(partlyObservedNetwork, name)`

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed data with possibly missing entries

name  a character for the name of sampling to fit on the partlyObservedNetwork

**Method** show(): show method

*Usage:*

networkSamplingNodes_fit$show()

**Method** update_parameters():  a method to update the estimation of the parameters. By default, nothing to do (corresponds to MAR sampling)

*Usage:*

networkSamplingNodes_fit$update_parameters(...)

*Arguments:*

...  use for compatibility

**Method** update_imputation(): a method to update the imputation of the missing entries.

*Usage:*

networkSamplingNodes_fit$update_imputation(nu)

*Arguments:*

nu  the matrix of (uncorrected) imputation for missing entries

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

networkSamplingNodes_fit$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

nodeSampler                *Virtual class for all node-centered samplers*

---

### Description

Virtual class for all node-centered samplers

Virtual class for all node-centered samplers

### Super classes

missSBM::networkSampling -> missSBM::networkSampler -> nodeSampler

## Methods

### Public methods:

- [nodeSampler$clone()](#)

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

nodeSampler$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

| nodeSampling_fit | *Class for fitting a node sampling* |
|---|---|

---

## Description

Class for fitting a node sampling

Class for fitting a node sampling

## Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSamplingNodes_fit](#) -> nodeSampling_fit

## Active bindings

vExpec  variational expectation of the sampling

## Methods

### Public methods:

- [nodeSampling_fit$new()](#)
- [nodeSampling_fit$clone()](#)

**Method** new(): constructor

*Usage:*

nodeSampling_fit$new(partlyObservedNetwork, ...)

*Arguments:*

partlyObservedNetwork  a object with class partlyObservedNetwork representing the observed data with possibly missing entries

...  used for compatibility

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

nodeSampling_fit$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

---

observeNetwork *Observe a network partially according to a given sampling design*

---

## Description

This function draws observations in an adjacency matrix according to a given network sampling design.

## Usage

```
observeNetwork(
  adjacencyMatrix,
  sampling,
  parameters,
  clusters = NULL,
  covariates = list(),
  similarity = missSBM:::l1_similarity,
  intercept = 0
)
```

## Arguments

adjacencyMatrix

The N x N adjacency matrix of the network to sample.

sampling        The sampling design used to observe the adjacency matrix, see details.

parameters      The sampling parameters (adapted to each sampling, see details).

clusters        An optional clustering membership vector of the nodes. Only necessary for block samplings.

covariates      An optional list with M entries (the M covariates). If the covariates are node-centered, each entry of covariates. must be a size-N vector; if the covariates are dyad-centered, each entry of covariates must be N x N matrix.

similarity      An optional function to compute similarities between node covariates. Default is missSBM:::l1_similarity, that is, -abs(x-y). Only relevant when the covariates are node-centered.

intercept       An optional intercept term to be added in case of the presence of covariates. Default is 0.

## Details

Internal functions use future_lapply, so set your plan to 'multisession' or 'multicore' to use several cores/workers. The list of parameters control tunes more advanced features, such as the initialization, how covariates are handled in the model, and the variational EM algorithm:

- "useCov": logical. If covariates is not null, should they be used for the for the SBM inference (or just for the sampling)? Default is TRUE.

- "clusterInit": Initial method for clustering: either a character ("spectral") or a list with `length(vBlocks)` vectors, each with size `ncol(adjacencyMatrix)`, providing a user-defined clustering. Default is "spectral".

- "similarity": An R x R -> R function to compute similarities between node covariates. Default is `missSBM:::l1_similarity`, that is, -abs(x-y). Only relevant when the covariates are node-centered (i.e. `covariates` is a list of size-N vectors).

- "threshold": V-EM algorithm stops stop when an optimization step changes the objective function or the parameters by less than threshold. Default is 1e-2.

- "maxIter": V-EM algorithm stops when the number of iteration exceeds maxIter. Default is 50.

- "fixPointIter": number of fix-point iterations in the V-E step. Default is 3.

- "exploration": character indicating the kind of exploration used among "forward", "backward", "both" or "none". Default is "both".

- "iterates": integer for the number of iterations during exploration. Only relevant when `exploration` is different from "none". Default is 1.

- "trace": logical for verbosity. Default is TRUE.

The different sampling designs are split into two families in which we find dyad-centered and node-centered samplings. See doi: 10.1080/01621459.2018.1562934 for a complete description.

- Missing at Random (MAR)

  - "dyad": parameter = p = Prob(Dyad(i,j) is observed)

  - "node": parameter = p = Prob(Node i is observed)

  - "covar-dyad": parameter = beta in R^M, such that Prob(Dyad (i,j) is observed) = logistic(parameter' covarArray (i,j, .))

  - "covar-node": parameter = nu in R^M such that Prob(Node i is observed) = logistic(parameter' covarMatrix (i,)

  - "snowball": parameter = number of waves with Prob(Node i is observed in the 1st wave)

- Missing Not At Random (MNAR)

  - "double-standard": parameter = (p0,p1) with p0 = Prob(Dyad (i,j) is observed | the dyad is equal to 0), p1 = Prob(Dyad (i,j) is observed | the dyad is equal to 1)

  - "block-node": parameter = c(p(1),...,p(Q)) and p(q) = Prob(Node i is observed | node i is in cluster q)

  - "block-dyad": parameter = c(p(1,1),...,p(Q,Q)) and p(q,l) = Prob(Edge (i,j) is observed | node i is in cluster q and node j is in cluster l)

  - "degree": parameter = c(a,b) and logit(a+b*degree(i)) = Prob(Node i is observed | Degree(i))

**Value**

an adjacency matrix with the same dimension as the input, yet with additional NAs.

## Examples

```
## SBM parameters
N <- 300 # number of nodes
Q <- 3   # number of clusters
pi <- rep(1,Q)/Q     # block proportion
theta <- list(mean = diag(.45,Q) + .05 ) # connectivity matrix

## simulate an unidrected binary SBM without covariate
sbm <- sbm::sampleSimpleSBM(N, pi, theta)

## Sample network data

# some sampling design and their associated parameters
sampling_parameters <- list(
   "dyad" = .3,
   "node" = .3,
   "double-standard" = c(0.4, 0.8),
   "block-node" = c(.3, .8, .5),
   "block-dyad" = theta$mean,
   "degree" = c(.01, .01),
   "snowball" = c(2,.1)
 )

observed_networks <- list()

for (sampling in names(sampling_parameters)) {
  observed_networks[[sampling]] <-
     missSBM::observeNetwork(
        adjacencyMatrix = sbm$networkData,
        sampling        = sampling,
        parameters      = sampling_parameters[[sampling]],
        cluster         = sbm$memberships
     )
}
```

---

partlyObservedNetwork  *An R6 Class used for internal representation of a partially observed*
*network*

---

## Description

An R6 Class used for internal representation of a partially observed network

An R6 Class used for internal representation of a partially observed network

## Details

This class is not exported to the user

**Active bindings**

samplingRate  The percentage of observed dyads

nbNodes  The number of nodes

nbDyads  The number of dyads

is_directed  logical indicating if the network is directed or not

networkData  The adjacency matrix of the network

covarArray  the array of covariates

covarMatrix  the matrix of covariates

samplingMatrix  matrix of observed and non-observed edges

samplingMatrixBar  matrix of observed and non-observed edges

observedNodes  a vector of observed and non-observed nodes (observed means at least one non NA value)

**Methods**

**Public methods:**

- partlyObservedNetwork$new()
- partlyObservedNetwork$clustering()
- partlyObservedNetwork$imputation()
- partlyObservedNetwork$clone()

**Method** new()**:** constructor

*Usage:*
```
partlyObservedNetwork$new(
  adjacencyMatrix,
  covariates = list(),
  similarity = missSBM:::l1_similarity
)
```
*Arguments:*

adjacencyMatrix  The adjacency matrix of the network

covariates  A list with M entries (the M covariates), each of whom being either a size-N vector or N x N matrix.

similarity  An R x R -> R function to compute similarities between node covariates. Default is l1_similarity, that is, -abs(x-y).

**Method** clustering()**:** method to cluster network data with missing value

*Usage:*
```
partlyObservedNetwork$clustering(
  vBlocks,
  imputation = ifelse(is.null(private$phi), "median", "average")
)
```
*Arguments:*

vBlocks  The vector of number of blocks considered in the collection.

    imputation  character indicating the type of imputation among "median", "average"

  **Method** `imputation()`: basic imputation from existing clustering

  *Usage:*

  `partlyObservedNetwork$imputation(type = c("median", "average", "zero"))`

  *Arguments:*

  type  a character, the type of imputation. Either "median" or "average"

  **Method** `clone()`: The objects of this class are cloneable with this method.

  *Usage:*

  `partlyObservedNetwork$clone(deep = FALSE)`

  *Arguments:*

  deep  Whether to make a deep clone.

---

  plot.missSBM_fit          *Visualization for an object* `missSBM_fit`

---

### Description

Plot function for the various fields of a `missSBM_fit`: the fitted SBM (network or connectivity), and a plot monitoring the optimization.

### Usage

```
## S3 method for class 'missSBM_fit'
plot(
  x,
  type = c("imputed", "expected", "meso", "monitoring"),
  dimLabels = list(row = "node", col = "node"),
  ...
)
```

### Arguments

| | |
|---|---|
| x | an object with class `missSBM_fit` |
| type | the type specifies the field to plot, either "imputed", "expected", "meso", or "monitoring" |
| dimLabels | : a list of two characters specifying the labels of the nodes. Default to `list(row= 'node',col = 'node'))` |
| ... | additional parameters for S3 compatibility. Not used |

### Value

a ggplot object

---

predicted.missSBM_fit   *Prediction of a* missSBM_fit *(i.e.  network with imputed missing dyads)*

---

### Description

Prediction of a missSBM_fit (i.e. network with imputed missing dyads)

### Usage

```
## S3 method for class 'missSBM_fit'
predict(object, ...)
```

### Arguments

object            an R6 object with class missSBM_fit

...               additional parameters for S3 compatibility.

### Value

an adjacency matrix between pairs of nodes. Missing dyads are imputed with their expected values, i.e. by there estimated probabilities of connection under the missing SBM.

---

simpleDyadSampler         *Class for defining a simple dyad sampler*

---

### Description

Class for defining a simple dyad sampler

Class for defining a simple dyad sampler

### Super classes

missSBM::networkSampling -> missSBM::networkSampler -> missSBM::dyadSampler -> simpleDyadSampler

### Methods

#### Public methods:

- simpleDyadSampler$new()
- simpleDyadSampler$clone()

**Method** new(): constructor for networkSampling

*Usage:*

```
simpleDyadSampler$new(
  parameters = NA,
  nbNodes = NA,
  directed = FALSE,
  covarArray = NULL,
  intercept = 0
)
```

*Arguments:*

parameters  the vector of parameters associated to the sampling at play

nbNodes  number of nodes in the network

directed  logical, directed network of not

covarArray  an array of covariates used

intercept  double, intercept term used to compute the probability of sampling in the presence of covariates. Default 0.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
simpleDyadSampler$clone(deep = FALSE)
```

*Arguments:*

deep  Whether to make a deep clone.

---

simpleNodeSampler          *Class for defining a simple node sampler*

---

### Description

Class for defining a simple node sampler

Class for defining a simple node sampler

### Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::nodeSampler](#) -> simpleNodeSampler

### Methods

#### Public methods:

- [simpleNodeSampler$new()](#)
- [simpleNodeSampler$clone()](#)

**Method** new(): constructor for networkSampling

*Usage:*

```
simpleNodeSampler$new(
  parameters = NA,
  nbNodes = NA,
  directed = FALSE,
  covarMatrix = NULL,
  intercept = 0
)
```

*Arguments:*

parameters the vector of parameters associated to the sampling at play

nbNodes number of nodes in the network

directed logical, directed network of not

covarMatrix a matrix of covariates used

intercept double, intercept term used to compute the probability of sampling in the presence of covariates. Default 0.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

simpleNodeSampler$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

SimpleSBM_fit  *This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.*

---

### Description

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

### Details

It is not designed not be call by the user

### Super classes

[sbm::SBM](#) -> [sbm::SimpleSBM](#) -> SimpleSBM_fit

### Active bindings

type the type of SBM (distribution of edges values, network type, presence of covariates)

penalty double, value of the penalty term in ICL

entropy double, value of the entropy due to the clustering distribution

loglik double: approximation of the log-likelihood (variational lower bound) reached

ICL double: value of the integrated classification log-likelihood

## Methods

### Public methods:

- [SimpleSBM_fit$new()](#)
- [SimpleSBM_fit$doVEM()](#)
- [SimpleSBM_fit$reorder()](#)
- [SimpleSBM_fit$clone()](#)

**Method** new(): constructor for simpleSBM_fit for missSBM purpose

*Usage:*

```
SimpleSBM_fit$new(networkData, clusterInit, covarList = list())
```

*Arguments:*

networkData a structure to store network under missing data condition: either a matrix possibly with NA, or a missSBM:::partlyObservedNetwork

clusterInit Initial clustering: a vector with size ncol(adjacencyMatrix), providing a user-defined clustering with nbBlocks levels.

covarList An optional list with M entries (the M covariates).

**Method** doVEM(): method to perform estimation via variational EM

*Usage:*

```
SimpleSBM_fit$doVEM(
  threshold = 0.01,
  maxIter = 100,
  fixPointIter = 3,
  trace = FALSE
)
```

*Arguments:*

threshold stop when an optimization step changes the objective function by less than threshold. Default is 1e-4.

maxIter V-EM algorithm stops when the number of iteration exceeds maxIter. Default is 10

fixPointIter number of fix-point iterations in the Variational E step. Default is 5.

trace logical for verbosity. Default is FALSE.

**Method** reorder(): permute group labels by order of decreasing probability

*Usage:*

```
SimpleSBM_fit$reorder()
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
SimpleSBM_fit$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

SimpleSBM_fit_MNAR        *This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.*

### Description

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

### Details

It is not designed not be call by the user

### Super classes

[sbm::SBM](#) -> [sbm::SimpleSBM](#) -> [missSBM::SimpleSBM_fit](#) -> [missSBM::SimpleSBM_fit_noCov](#) -> SimpleSBM_MNAR_noCov

### Active bindings

imputation  the matrix of imputed values

vExpec  double: variational approximation of the expectation complete log-likelihood

### Methods

#### Public methods:

- [SimpleSBM_fit_MNAR$new()](#)
- [SimpleSBM_fit_MNAR$update_parameters()](#)
- [SimpleSBM_fit_MNAR$update_blocks()](#)
- [SimpleSBM_fit_MNAR$clone()](#)

**Method** new(): constructor for simpleSBM_fit for missSBM purpose

*Usage:*

SimpleSBM_fit_MNAR$new(networkData, clusterInit)

*Arguments:*

networkData  a structure to store network under missing data condition: either a matrix possibly with NA, or a missSBM:::partlyObservedNetwork

clusterInit  Initial clustering: a vector with size ncol(adjacencyMatrix), providing a user-defined clustering with nbBlocks levels.

**Method** update_parameters(): update parameters estimation (M-step)

*Usage:*

SimpleSBM_fit_MNAR$update_parameters(nu = NULL)

*Arguments:*

nu  currently imputed values

**Method** `update_blocks()`: update variational estimation of blocks (VE-step)

*Usage:*

`SimpleSBM_fit_MNAR$update_blocks(log_lambda = 0)`

*Arguments:*

`log_lambda` additional term sampling dependent used to de-bias estimation of tau

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`SimpleSBM_fit_MNAR$clone(deep = FALSE)`

*Arguments:*

deep  Whether to make a deep clone.

---

| | |
|---|---|
| `SimpleSBM_fit_noCov` | *This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.* |

---

### Description

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

### Details

It is not designed not be call by the user

### Super classes

[sbm::SBM](#) -> [sbm::SimpleSBM](#) -> [missSBM::SimpleSBM_fit](#) -> SimpleSBM_fit_noCov

### Active bindings

imputation  the matrix of imputed values

vExpec  double: variational approximation of the expectation complete log-likelihood

vExpec_corrected  double: variational approximation of the expectation complete log-likelihood with correction to be comparable with MNAR criteria

## Methods

### Public methods:

- [SimpleSBM_fit_noCov$update_parameters()](#)
- [SimpleSBM_fit_noCov$update_blocks()](#)
- [SimpleSBM_fit_noCov$clone()](#)

**Method** update_parameters(): update parameters estimation (M-step)

*Usage:*

SimpleSBM_fit_noCov$update_parameters(...)

*Arguments:*

... additional arguments, only required for MNAR cases

**Method** update_blocks(): update variational estimation of blocks (VE-step)

*Usage:*

SimpleSBM_fit_noCov$update_blocks(...)

*Arguments:*

... additional arguments, only required for MNAR cases

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

SimpleSBM_fit_noCov$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

SimpleSBM_fit_withCov   *This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.*

---

## Description

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

This internal class is designed to adjust a binary Stochastic Block Model in the context of missSBM.

## Details

It is not designed not be call by the user

## Super classes

[sbm::SBM](#) -> [sbm::SimpleSBM](#) -> [missSBM::SimpleSBM_fit](#) -> SimpleSBM_fit_withCov

## Active bindings

`imputation` the matrix of imputed values

`vExpec` double: variational approximation of the expectation complete log-likelihood

`vExpec_corrected` double: variational approximation of the expectation complete log-likelihood with correction to be comparable with MNAR criteria

## Methods

### Public methods:

- [SimpleSBM_fit_withCov$update_parameters()](#)
- [SimpleSBM_fit_withCov$update_blocks()](#)
- [SimpleSBM_fit_withCov$clone()](#)

**Method** `update_parameters()`: update parameters estimation (M-step)

*Usage:*

`SimpleSBM_fit_withCov$update_parameters(...)`

*Arguments:*

`...` use for compatibility

`control` a list to tune nlopt for optimization, see documentation of nloptr

**Method** `update_blocks()`: update variational estimation of blocks (VE-step)

*Usage:*

`SimpleSBM_fit_withCov$update_blocks(...)`

*Arguments:*

`...` use for compatibility

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`SimpleSBM_fit_withCov$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

---

| snowballSampler | *Class for defining a snowball sampler* |
| --- | --- |

---

## Description

Class for defining a snowball sampler

Class for defining a snowball sampler

## Super classes

[missSBM::networkSampling](#) -> [missSBM::networkSampler](#) -> [missSBM::nodeSampler](#) -> snowballSampler

## Methods

### Public methods:

- snowballSampler$new()
- snowballSampler$clone()

**Method** new(): constructor for networkSampling

*Usage:*

snowballSampler$new(parameters = NA, adjacencyMatrix = NA, directed = FALSE)

*Arguments:*

parameters the vector of parameters associated to the sampling at play

adjacencyMatrix the adjacency matrix of the network

directed logical, directed network of not

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

snowballSampler$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

summary.missSBM_fit    *Summary method for a* missSBM_fit

---

## Description

Summary method for a missSBM_fit

## Usage

```
## S3 method for class 'missSBM_fit'
summary(object, ...)
```

## Arguments

object        an R6 object with class missSBM_fit

...           additional parameters for S3 compatibility.

## Value

a basic printing output

---

war | *War data set*

---

### Description

This dataset contains two networks where the nodes are countries and an edge in network "belligerent" means that the two countries have been at least once at war between years 1816 to 2007 while an edge in network "alliance" means that the two countries have had a formal alliance between years 1816 to 2012. The network `belligerent` have less nodes since countries which have not been at war are not considered.

### Usage

```
war
```

### Format

A list with 2 two igraph objects, `alliance` and `belligerent`. Each graph have three attributes: 'name' (the country name), 'power' (a score related to military power: the higher, the better) and 'trade' (a score related to the trade effort between pairs of countries).

### Source

networks were extracted from <https://www.correlatesofwar.org/>

### References

Sarkees, Meredith Reid and Frank Wayman (2010). Resort to War: 1816 - 2007. Washington DC: CQ Press.

Gibler, Douglas M. 2009. International military alliances, 1648-2008. CQ Press

### Examples

```
data(war)
class(war$belligerent)
igraph::gorder(war$alliance)
igraph::gorder(war$belligerent)
igraph::edges(war$alliance)
igraph::get.graph.attribute(war$alliance)
```

# Index