Package ‘hergm’
May 25, 2019

Version 4.1-3
Date 2019-05-24
Title Hierarchical Exponential-Family Random Graph Models
Author Michael Schweinberger <michael.schweinberger@rice.edu> [aut, cre], Mark S. Handcock <handcock@ucla.edu> [aut], Sergii Babkin <babkin.sergii@gmail.com> [aut], Jonathan Stewart <jonathan.stewart@rice.edu> [aut], Duy Vu <duy.vu@unimelb.edu.au> [aut], Pamela Luna <pamela.luna@rice.edu>
Maintainer Michael Schweinberger <michael.schweinberger@rice.edu>
Depends ergm, latentnet, mcgibbsit, network, sna
Imports methods, mlergm, Rcpp (>= 0.12.7), Matrix, igraph, intergraph, parallel, stringr
Description Hierarchical exponential-family random graph models with local dependence.
License GPL-3
LinkingTo Rcpp
NeedsCompilation yes
BugReports http://www.stat.rice.edu/~msXX/hergm/issues.html
Repository CRAN
Date/Publication 2019-05-25 04:40:03 UTC

R topics documented:

bali .................................................. 2
bunt .................................................. 2
example ............................................ 3
goF.hergm ....................................... 4
hergm ............................................... 5
hergm-terms ..................................... 13
hergm.mcmc.diagnostics .......................... 14
hergm.postprocess ............................... 15
kapferer .......................................... 17
plot.hergm ....................................... 18
print.hergm ...................................... 19
simulate.hergm .................................. 20
summary.hergm ................................... 21
### bali

**Bali terrorist network**

**Description**

The network corresponds to the contacts between the 17 terrorists who carried out the bombing in Bali, Indonesia in 2002. The network is taken from Koschade (2006).

**Usage**

```r
data(bali)
```

**Value**

Undirected network.

**References**


**See Also**

network, hergm, ergm.terms, hergm.terms

**Examples**

```r
## Not run: data(bali)
object <- hergm(bali ~ edges_ij + triangle_ijk, max_number=3, method="bayes")
summary(object)

## End(Not run)
```

---

### bunt

**Van de Bunt friendship network**

**Description**

Van de Bunt (1999) and Van de Bunt et al. (1999) collected data on friendships between 32 freshmen at a European university at 7 time points. Here, the last time point is used. A directed edge from student i to j indicates that student i considers student j to be a “friend” or “best friend”.

```r
## Not run: data(bali)
object <- hergm(bali ~ edges_ij + triangle_ijk, max_number=3, method="bayes")
summary(object)

## End(Not run)
```
Example network

Description

Example data set: synthetic, undirected network with 15 nodes.

Usage

data(example)

Value

Undirected network.
See Also

network, hergm, ergm.terms, hergm.terms

Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_iL max_number=3)
summary(object)

## End(Not run)
```

---

gof.hergm Goodness-of-fit

Description

The function `gof.hergm` accepts an object of class `hergm` as argument and assesses the goodness-of-fit of the model estimated by function `hergm`.

Usage

```r
## S3 method for class 'hergm'
gof(object, sample_size = 1000, ...)
```

Arguments

- `object` object of class `hergm`; objects of class `hergm` can be generated by function `hergm`.
- `sample_size` number of samples to generate.
- `...` additional arguments, to be passed to lower-level functions in the future.

Value

The function `gof.hergm` returns a list with components:

- `component.number` number of components.
- `max.component.size` size of largest component.
- `distance` geodesic distance of pairs of nodes.
- `degree` degree of nodes.
- `edges` number of edges.
- `stars` number of 2-stars.
- `triangle` number of triangles.
References

See Also
hergm, simulate.hergm

Examples
```r
## not run: data(example)
object <- hergm(d ~ edges_i)
gof(object)
## End(Not run)
```

**hergm**  
Hierarchical exponential-family random graph models with local dependence

Description
The function `hergm` estimates and simulates three classes of hierarchical exponential-family random graph models:

1. The `p_1` model of Holland and Leinhardt (1981) in exponential-family form and extensions by Vu, Hunter, and Schweinberger (2013) and Schweinberger, Petrescu-Prahova, and Vu (2014) to both directed and undirected random graphs with additional model terms, with and without covariates, and with parametric and nonparametric priors (see `arcs_i`, `arcs_j`, `edges_i`, `edges_ij`, `mutual_i`, `mutual_ij`).

2. The stochastic block model of Snijders and Nowicki (1997) and Nowicki and Snijders (2001) in exponential-family form and extensions by Vu, Hunter, and Schweinberger (2013) and Schweinberger, Petrescu-Prahova, and Vu (2014) with additional model terms, with and without covariates, and with parametric and nonparametric priors (see `arcs_i`, `arcs_j`, `edges_i`, `edges_ij`, `mutual_i`, `mutual_ij`).

3. The exponential-family random graph models with local dependence of Schweinberger and Handcock (2015), with and without covariates, and with parametric and nonparametric priors (see `arcs_i`, `arcs_j`, `edges_i`, `edges_ij`, `mutual_i`, `mutual_ij`, `twostar_ijk`, `triangle_ijk`, `ttriple_ijk`, `ctriple_ijk`). The exponential-family random graph models with local dependence replace the long-range dependence of conventional exponential-family random graph models by short-range dependence. Therefore, exponential-family random graph models with local dependence replace the strong dependence of conventional exponential-family random graph models by weak dependence, reducing the problem of model degeneracy (Handcock, 2003; Schweinberger, 2011) and improving goodness-of-fit (Schweinberger and Handcock, 2015). In addition, exponential-family random graph models with local dependence satisfy a weak form of self-consistency in the sense that these models are self-consistent under neighborhood sampling (Schweinberger and Handcock, 2015), which enables consistent estimation of neighborhood-dependent parameters (Schweinberger and Stewart, 2017; Schweinberger, 2017).
Usage

hergm(formula,
       max_number = 2,
       hierarchical = TRUE,
       parametric = FALSE,
       parameterization = "offset",
       initialize = FALSE,
       initialization_method = 1,
       estimate_parameters = TRUE,
       initial_estimate = NULL,
       n_em_step_max = 100,
       max_iter = 4,
       perturb = FALSE,
       scaling = NULL,
       alpha = NULL,
       alpha_shape = NULL,
       alpha_rate = NULL,
       eta = NULL,
       eta_mean = NULL,
       eta_sd = NULL,
       eta_mean_mean = NULL,
       eta_mean_sd = NULL,
       eta_precision_shape = NULL,
       eta_precision_rate = NULL,
       mean_between = NULL,
       indicator = NULL,
       parallel = 1,
       simulate = FALSE,
       method = "ml",
       seeds = NULL,
       sample_size = NULL,
       sample_size_multiplier_blocks = 20,
       NR_max_iter = 200,
       NR_step_len = 1,
       NR_step_len_multiplier = 0.2,
       interval = 1024,
       burnin = 16*interval,
       mh.scale = 0.25,
       variational = FALSE,
       temperature = c(1,100),
       predictions = FALSE,
       posterior.burnin = 2000,
       posterior.thinning = 1,
       relabel = 1,
       number_runs = 1,
       verbose = 0,
       ...)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>formula</code></td>
<td>formula of the form ( \text{network} \sim \text{terms} ). <code>network</code> is an object of class <code>network</code> and can be created by calling the function <code>network</code>. Possible terms can be found in <code>ergm.terms</code> and <code>hergm.terms</code>.</td>
</tr>
<tr>
<td><code>max_number</code></td>
<td>maximum number of blocks.</td>
</tr>
<tr>
<td><code>hierarchical</code></td>
<td>hierarchical prior; if <code>hierarchical = \text{TRUE}</code>, prior is hierarchical (i.e., the means and variances of block parameters are governed by a hyper-prior), otherwise non-hierarchical (i.e., the means and variances of block parameters are fixed).</td>
</tr>
<tr>
<td><code>parametric</code></td>
<td>parametric prior; if <code>parametric = \text{FALSE}</code>, prior is truncated Dirichlet process prior, otherwise parametric Dirichlet prior.</td>
</tr>
<tr>
<td><code>parameterization</code></td>
<td>if <code>parameterization = \text{&quot;offset&quot;}</code>, size-dependent parameterizations of the form ( \text{parameter} - \log(\text{size of block}) ) along the lines of Krivitsky, Handcock, and Morris (2011) and Krivitsky and Kolaczyk (2015) are used, otherwise size-independent parameterizations are used; note that size-dependent parameterizations are useful when the network is sparse and the blocks do not have the same size.</td>
</tr>
<tr>
<td><code>initialize</code></td>
<td>if <code>initialize = \text{TRUE}</code>, initialize block memberships of nodes.</td>
</tr>
<tr>
<td><code>initialization_method</code></td>
<td>if <code>initialization_method = 1</code>, block memberships of nodes are initialized by walk trap; if <code>initialization_method = 2</code>, block memberships of nodes are initialized by spectral clustering.</td>
</tr>
<tr>
<td><code>estimate_parameters</code></td>
<td>if method = &quot;ml&quot; and <code>estimate_parameters = \text{TRUE}</code>, estimate parameters.</td>
</tr>
<tr>
<td><code>initial_estimate</code></td>
<td>if method = &quot;ml&quot; and <code>estimate_parameters = \text{TRUE}</code>, specifies starting point.</td>
</tr>
<tr>
<td><code>n_em_step_max</code></td>
<td>if method = &quot;ml&quot;, maximum number of iterations of Generalized Expectation Maximization algorithm estimating the block structure.</td>
</tr>
<tr>
<td><code>max_iter</code></td>
<td>if method = &quot;ml&quot;, maximum number of iterations of Monte Carlo maximization algorithm estimating parameters given block structure.</td>
</tr>
<tr>
<td><code>perturb</code></td>
<td>if <code>initialize = \text{TRUE}</code> and <code>perturb = \text{TRUE}</code>, initialize block memberships of nodes by spectral clustering and perturb.</td>
</tr>
<tr>
<td><code>scaling</code></td>
<td>if <code>scaling = \text{TRUE}</code>, use size-dependent parameterizations which ensure that the scaling of between- and within-block terms is consistent with sparse edge terms.</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>concentration parameter of truncated Dirichlet process prior of natural parameters of exponential-family model.</td>
</tr>
<tr>
<td><code>alpha_shape</code>, <code>alpha_rate</code></td>
<td>shape and rate parameter of Gamma prior of concentration parameter.</td>
</tr>
<tr>
<td><code>eta</code></td>
<td>the parameters of <code>ergm.terms</code> and <code>hergm.terms</code>; the parameters of <code>hergm.terms</code> must consist of <code>max_number</code> within-block parameters and one between-block parameter.</td>
</tr>
</tbody>
</table>
eta_mean, eta_sd
means and standard deviations of Gaussian baseline distribution of Dirichlet process prior of natural parameters.

eta_mean_meanL, eta_mean_sd
means and standard deviations of Gaussian prior of mean of Gaussian baseline distribution of Dirichlet process prior.

eta_precision_shapeL, eta_precision_rate
shape and rate (inverse scale) parameter of Gamma prior of precision parameter of Gaussian baseline distribution of Dirichlet process prior.

mean_between
if simulate = TRUE and eta = NULL, then mean_between specifies the mean-value parameter of edges between blocks.

indicator
if the indicators of block memberships of nodes are specified as integers between 1 and max_number, the specified indicators are fixed, which is useful when indicators of block memberships are observed (e.g., in multilevel networks).

parallel
number of computing nodes; if parallel > 1, hergm is run on parallel computing nodes.

simulate
if simulate = TRUE, simulate networks from model, otherwise estimate model given observed network.

method
if method = "bayes", Bayesian methods along the lines of Schweinberger and Handcock (2015) and Schweinberger and Luna (2018) are used; otherwise, if method = "ml", then approximate maximum likelihood methods along the lines of Babkin and Schweinberger (2019) are used; note that Bayesian methods are the gold standard but are too time-consuming to be applied to networks with more than 100 nodes, whereas the approximate maximum likelihood methods can be applied to networks with thousands of nodes.

seeds
seed of pseudo-random number generator; if parallel > 1, number of seeds must equal number of computing nodes.

sample_size
if simulate = TRUE, number of network draws, otherwise number of posterior draws; if parallel > 1, number of draws on each computing node.

sample_size_multiplier_blocks
if method = "ml", multiplier of the number of network draws from within-block subgraphs; the total number of network draws from within-block subgraphs is sample_size_multiplier_blocks * number of possible edges of largest within-block subgraph; if sample_size_multiplier_blocks = NULL, then total number of network draws from within-block subgraphs is sample_size.

NR_max_iter
if method = "ml", the maximum number of iterations to be used in the estimation of parameters.

NR_step_len
if method = "ml", the step-length to be used for increments in the estimation of parameters.

NR_step_len_multiplier
if method = "ml", multiplier for adjusting the step-length in the estimation procedure after a divergent increment.

interval
if simulate = TRUE, number of proposals between sampled networks.

burnin
if simulate = TRUE, number of burn-in iterations.
mh.scale if simulate = FALSE, scale factor of candidate-generating distribution of Metropolis-Hastings algorithm.

variational if simulate = FALSE and variational = TRUE, variational methods are used to construct the proposal distributions of block memberships of nodes; limited to selected models.

temperature if simulate = FALSE and variational = TRUE, minimum and maximum temperature; the temperature is used to melt down the proposal distributions of indicators, which are based on the full conditional distributions of indicators but can have low entropy, resulting in slow mixing of the Markov chain; the temperature is a function of the entropy of the full conditional distributions and is designed to increase the entropy of the proposal distributions, and the minimum and maximum temperature are user-defined lower and upper bounds on the temperature.

predictions if predictions = TRUE and simulate = FALSE, returns posterior predictions of statistics in the model.

posterior.burnin number of posterior burn-in iterations; if computing is parallel, posterior.burnin is applied to the sample generated by each processor; please note that hergm returns min(sample_size, 10000) sample points and the burn-in is applied to the sample of size min(sample_size, 10000), therefore posterior.burnin should be smaller than min(sample_size, 10000).

posterior.thinning if posterior.thinning > 1, every posterior.thinning-th sample point is used while all others discarded; if computing is parallel, posterior.thinning is applied to the sample generated by each processor; please note that hergm returns min(sample_size, 10000) sample points and the thinning is applied to the sample of size min(sample_size, 10000) - posterior.burnin, therefore posterior.thinning should be smaller than min(sample_size, 10000) - posterior.burnin.

relabel if relabel > 0, relabel MCMC sample by minimizing the posterior expected loss of Schweinberger and Handcock (2015) (relabel = 1) or Peng and Carvalho (2016) (relabel = 2).

number_runs if relabel = 1, number of runs of relabeling algorithm.

verbose if verbose = -1, no console output; if verbose = 0, short console output; if verbose = +1, long console output. If, e.g., simulate = FALSE and verbose = 1, then hergm reports the following console output:

Progress: 50.00% of 1000000

... means of block parameters: -0.2838 1.3323

precisions of block parameters: 0.9234 1.4682

block parameters:

-0.2544 -0.2560 -0.1176 -0.0310 -0.1915 -1.9626
0.4022 1.8887 1.9719 0.6499 1.7265 0.0000

block indicators: 1 3 1 1 1 1 3 1 1 2 2 2 2 2 1 1 1

block sizes: 10 5 2 0 0
block probabilities: 0.5396 0.2742 0.1419 0.0423 0.0020
block probabilities prior parameter: 0.4256
posterior prediction of statistics: 66 123
where ... indicates additional information about the Markov chain Monte Carlo algorithm that is omitted here. The console output corresponds to:
- "means of block parameters" correspond to the mean parameters of the Gaussian base distribution of parameters of hergm-terms.
- "precisions of block parameters" correspond to the precision parameters of the Gaussian base distribution of parameters of hergm-terms.
- "block parameters" correspond to the parameters of hergm-terms.
- "block indicators" correspond to the indicators of block memberships of nodes.
- "block sizes" correspond to the block sizes.
- "block probabilities" correspond to the prior probabilities of block memberships of nodes.
- "block probabilities prior parameter" corresponds to the concentration parameter of truncated Dirichlet process prior of parameters of hergm-terms.
- if predictions = TRUE, "posterior prediction of statistics" correspond to posterior predictions of sufficient statistics.

additional arguments, to be passed to lower-level functions in the future.

Value
The function hergm returns an object of class hergm with components:

- **network**
  - network is an object of class network and can be created by calling the function network.

- **formula**
  - formula of the form network ~ terms. network is an object of class network and can be created by calling the function network. Possible terms can be found in ergm.terms and hergm.terms.

- **n**
  - number of nodes.

- **hyper_prior**
  - indicator of whether hyper prior has been specified, i.e., whether the parameters alpha, eta_mean, and eta_precision are estimated.

- **alpha**
  - concentration parameter of truncated Dirichlet process prior of parameters of hergm-terms.

- **ergm_theta**
  - parameters of ergm-terms.

- **eta_mean**
  - mean parameters of Gaussian base distribution of parameters of hergm-terms.

- **eta_precision**
  - precision parameters of Gaussian base distribution of parameters of hergm-terms.

- **d1**
  - total number of parameters of ergm terms.

- **d2**
  - total number of parameters of hergm terms.

- **hergm_theta**
  - parameters of hergm-terms.

- **relabel.ergm_theta**
  - relabeled parameters of hergm-terms by using relabel = 1 or relabel = 2.

- **number_fixed**
  - number of fixed indicators of block memberships of nodes.
indicator indicators of block memberships of nodes.

relabel if relabel > 0, relabel MCMC sample by minimizing the posterior expected loss of Schweinberger and Handcock (2015) (relabel = 1) or Peng and Carvalho (2016) (relabel = 2).

relabel.indicator relabeled indicators of block memberships of nodes by using relabel = 1 or relabel = 2.

size the size of the blocks, i.e., the number of nodes of blocks.

parallel number of computing nodes; if parallel > 1, hergm is run on parallel computing nodes.

p_i_k posterior probabilities of block membership of nodes.

p_k probabilities of block memberships of nodes.

predictions if predictions = TRUE and simulate = FALSE, returns posterior predictions of statistics in the model.

simulate if simulate = TRUE, simulation of networks, otherwise Bayesian inference.

prediction posterior predictions of statistics.

edgelist edge list of simulated network.

sample_size if simulate = TRUE, number of network draws, otherwise number of posterior draws minus number of burn-in iterations; if parallel > 1, number of draws on each computing node.

extract indicator of whether function hergm.postprocess has postprocessed the object of class hergm generated by function hergm and thus whether the MCMC sample generated by function hergm has been extracted from the object of class hergm.

convergence.diagnostics MCMC diagnostics generated by function mcmc.diagnostics, which in turn relies on function mcgibbsit in R package mcgibbsit; see ?mcgibbsit.

verbose if verbose = -1, no console output; if verbose = 0, short console output; if verbose = +1, long console output.

References


See Also

network, ergm.terms, hergm.terms, hergm.postprocess, mcmc.diagnostics, summary, print, plot, gof, simulate

Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_ij, max_number=3)
summary(object)

data(sampson)
object <- hergm(samplike ~ edges_ij + transitivities_ijk, max_number=3)
summary(object)

## End(Not run)
```
Hierarchical exponential-family random graph models with local dependence can be specified by
calling the function `hergm(formula)`, where formula is a formula of the form `network ~ terms`.
By specifying suitable terms, it is possible to specify a wide range of models: see `hergm`. `hergm.terms`
can be found here. In addition, `ergm.terms` can be used to include covariates.

### Arguments

- **edges_i** (undirected network)
  
  adding the term `edges_i` to the model adds node-dependent edge terms to the model; please note: the term `edges_i` can be used with method = "bayes" but cannot be used with the default method = "ml".

- **arcs_i** (directed network)
  
  adding the term `arcs_i` to the model adds node-dependent outdegree terms to the model; please note: the term `arcs_i` can be used with method = "bayes" but cannot be used with the default method = "ml".

- **arcs_j** (directed network)
  
  adding the term `arcs_j` to the model adds node-dependent indegree terms to the model; please note: the term `arcs_j` can be used with method = "bayes" but cannot be used with the default method = "ml".

- **edges_ij** (undirected, directed network)
  
  adding the term `edges_ij` to the model adds block-dependent edge terms to the model.

- **mutual_i** (directed network)
  
  adding the term `mutual_i` to the model adds additive, block-dependent mutual edge terms to the model.

- **mutual_ij** (directed network)
  
  adding the term `mutual_ij` to the model adds block-dependent mutual edge terms to the model.

- **twostar_ijk** (undirected network)
  
  adding the term `twostar_ijk` to the model adds block-dependent two-star terms to the model.

- **transitiveties_ijk** (directed network)
  
  adding the term `transitiveties_ijk` to the model adds block-dependent transitive ties terms to the model.

- **triangle_ijk** (undirected, directed network)
  
  adding the term `triangle_ijk` to the model adds block-dependent triangle terms to the model.

- **ttriple_ijk** (directed network)
  
  adding the term `ttriple_ijk` to the model adds block-dependent transitive triple terms to the model; please note: the term `ttriple_ijk` can be used with method = "bayes" but cannot be used with the default method = "ml".
ctriple_ijk (directed network)
adding the term ctriple_ijk to the model adds block-dependent cyclic triple terms to the model; please note: the term ctriple_ijk can be used with method = "bayes" but cannot be used with the default method = "ml".

References

See Also
hergm, ergm.terms

Examples
## not run: data(sampson)
hergm(samplike ~ edges + transitivity_ijk)
## End(Not run)
Value

The function `mcmc.diagnostics` returns a list with the following components:

- **mcmc.alpha**: MCMC diagnostics for the concentration parameter of truncated Dirichlet process prior of parameters of `hergm-terms`.
- **mcmc.eta_mean**: MCMC diagnostics for the mean parameters of Gaussian base distribution of parameters of `hergm-terms`.
- **mcmc.eta_precision**: MCMC diagnostics for the precision parameters of Gaussian base distribution of parameters of `hergm-terms`.
- **mcmc.ergm_theta**: MCMC diagnostics for the parameters of `ergm-terms`.
- **mcmc.hergm_theta**: MCMC diagnostics for the parameters of `hergm-terms`.

References


See Also

`hergm`

Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_ij + triangle_ijk, method="bayes")
mcmc.diagnostics(object)

## End(Not run)
```

Description

The function `hergm.postprocess` postprocesses an object of class `hergm`. Please note that the function `hergm` calls the function `hergm.postprocess` with `relabel = 0` by default or with other values of `relabel` specified by the user, therefore users do not need to call the function `hergm.postprocess` unless it is desired to postprocess an object of class `hergm` with a value of `relabel` that was not used by function `hergm`.

If `hergm.postprocess` is called with `relabel > 0`, it solves the so-called label-switching problem. The label-switching problem is rooted in the invariance of the likelihood function to permutations of the labels of blocks, and implies that raw MCMC samples from the posterior cannot be used to infer to block-dependent entities. The label-switching problem can be solved in a Bayesian approach.
decision-theoretic framework: by choosing a loss function and minimizing the posterior expected loss. Two loss functions are implemented in hergm.postprocess, the loss function of Schweinberger and Handcock (2015) (relabel == 1) and the loss function of Peng and Carvalho (2016) (relabel == 2). The first loss function seems to be superior in terms of the reported clustering probabilities, but is more expensive in terms of computing time. A rule of thumb is to use the first loss function when \( \text{max\_number} < 15 \) and use the second loss function otherwise.

Usage

```r
hergm.postprocess(object,
    burnin = 2000,
    thinning = 1,
    relabel = 1,
    number_runs = 1,
    ...)```

Arguments

- **object**: object of class `hergm`; objects of class `hergm` can be generated by function `hergm`.
- **burnin**: number of posterior burn-in iterations; if computing is parallel, burnin is applied to the sample generated by each processor; please note that `hergm` returns \( \min(\text{sample\_size}, 10000) \) sample points and the burn-in is applied to the sample of size \( \min(\text{sample\_size}, 10000) \), therefore burnin should be smaller than \( \min(\text{sample\_size}, 10000) \).
- **thinning**: if thinning > 1, every thinning-th sample point is used while all others discarded; if computing is parallel, thinning is applied to the sample generated by each processor; please note that `hergm` returns \( \min(\text{sample\_size}, 10000) \) sample points and the thinning is applied to the sample of size \( \min(\text{sample\_size}, 10000) - \text{burnin} \), therefore thinning should be smaller than \( \min(\text{sample\_size}, 10000) - \text{burnin} \).
- **relabel**: if relabel > 0, relabel MCMC sample by minimizing the posterior expected loss of Schweinberger and Handcock (2015) (relabel == 1) or Peng and Carvalho (2016) (relabel == 2).
- **number_runs**: if relabel == 1, number of runs of relabeling algorithm.
- **...**: additional arguments, to be passed to lower-level functions in the future.

Value

- **ergm\_theta**: parameters of `ergm\_terms`.
- **alpha**: concentration parameter of truncated Dirichlet process prior of parameters of `hergm\_terms`.
- **eta\_mean**: mean parameters of Gaussian base distribution of parameters of `hergm\_terms`.
- **eta\_precision**: precision parameters of Gaussian base distribution of parameters of `hergm\_terms`.
- **hergm\_theta**: parameters of `hergm\_terms`. 
loss if relabel == TRUE, local minimum of loss function.

\( p_k \) probabilities of block memberships of nodes.

\( \text{indicator} \) indicators of block memberships of nodes.

\( p_{i,k} \) posterior probabilities of block memberships of nodes.

\( \text{prediction} \) posterior predictions of statistics.

References


See Also

hergm

Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_ij + triangle_ijk)
hergmPostprocess(object)
## End(Not run)
```

kapferer

Kapferer collaboration network

Description

The network corresponds to collaborations between 39 workers in a tailor shop in Africa: an undirected edge between workers \( i \) and \( j \) indicates that the workers collaborated. The network is taken from Kapferer (1972).

Usage

```r
data(kapferer)
```

Value

Undirected network.
References


See Also

network, hergm, ergm.terms, hergm.terms

Examples

```r
## not run: data(kapferer)
object <- hergm(kapferer ~ edges_ij + transitivities_ijk, max_number=3)
summary(object)

## End(not run)
```

plot.hergm

Plot summary of object of class hergm

Description

The function `plot.hergm` accepts an object of class `hergm` as argument and plots a summary of a sample of block memberships of nodes from the posterior. Please note that the function `hergm` should have been called with `relabel > 0` to solve the so-called label-switching problem, which is done by default. If the function `hergm` has not been called with option `relabel > 0`, call the function `hergmNpostprocess` with `relabel > 0`.

Usage

```r
## S3 method for class 'hergm'
plot(x, threshold = c(.7, .8, .9), ...)  
```

Arguments

- `x`: object of class `hergm`; objects of class `hergm` can be generated by function `hergm`
- `threshold`: if the component `relabel` of the object of class `hergm` is `relabel = 3`, then `threshold` is a vector of thresholds between 0 and 1, indicating the thresholds at which the same-block-membership posterior probabilities of nodes are to be thresholded to construct the same-block graphs.
- `...`: additional arguments, to be passed to lower-level functions in the future.

References

### Description

The function `print.hergm` accepts an object of class `hergm` as argument and prints a summary of parameters from the posterior. Please note that the function `hergm` should have been called with `relabel > 0` to solve the so-called label-switching problem, which is done by default. If the function `hergm` has not been called with option `relabel > 0`, call the function `hergm.postprocess` with `relabel > 0`.

### Usage

```r
## S3 method for class 'hergm'
print(x, ...)
```

### Arguments

- **x**
  - object of class `hergm`; objects of class `hergm` can be generated by function `hergm`.
- **...**
  - additional arguments, to be passed to lower-level functions in the future.

### References


### See Also

- `hergm`, `hergm.postprocess`, `plot.hergm`, `summary.hergm`
Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_ij)
print(object)

## End(Not run)
```

Description

The function `simulate.hergm` accepts an object of class `hergm` as argument and simulates networks.

Usage

```r
## S3 method for class 'hergm'
simulate(object, nsim = 1, seed = NULL, max_number = NULL, indicator = NULL, eta = NULL, sample_size = 1, verbose = 0, ...)
```

Arguments

- `object`: either object of class `hergm` or formula of the form `network ~ terms`; objects of class `hergm` can be generated by function `hergm`; `network` is an object of class network and can be created by calling the function `network`; possible terms can be found in `ergm.terms` and `hergm.terms`.
- `nsim`: redundant, but ensures that the simulate method is compatible with the simulate method of `R` package `stats`.
- `seed`: redundant, but ensures that the simulate method is compatible with the simulate method of `R` package `stats`.
- `max_number`: maximum number of blocks.
- `indicator`: indicators of block memberships of nodes.
- `eta`: `ergm.terms` and `hergm.terms` parameters.
- `sample_size`: number of networks to be simulated.
- `verbose`: if `verbose == -1`, no console output; if `verbose == 0`, short console output; if `verbose == +1`, long console output.
- `...`: additional arguments, to be passed to lower-level functions in the future.
Value

The function `simulate.hergm` returns the simulated networks in the form of edge lists.

References


See Also

`hergm`, `ergm.terms`, `hergm.terms`, `gof.hergm`

Examples

```r
## Not run: setNseed(1RSTU67X9)

# Simulate 100 networks from model:
d <- matrix(rbinom(n=45*45, size=1, prob=.1), 45, 45)
d <- as.network(d, directed=TRUE, loops = FALSE)
N <- 100
indicator <- c(rep.int(1, 10), rep.int(2, 15), rep.int(3, 20))
eta <- c(-1.5, -2, -2.5, -4, -.5, .5, .5, 0)
edgelists <- simulate.hergm(d ~ edges_ij + transitiveties_ijk,
                           max_number=3, indicator=indicator, eta=eta, sample_size=N)
d <- lapply(as.network, x=edgelists$edgelist)
coord <- plot(d[[1]], vertex.col=indicator, edge.col=same_clusters(indicator))
for (i in 1:N) plot(d[[i]], vertex.col=indicator, edge.col=same_clusters(indicator), coord=coord)

# Estimate model from first simulated network:
object <- hergm(network ~ edges_ij + transitiveties_ijk, max_number=3)
# Plot network with estimated block structure:
plot(object)

## End(not run)
```

---

**summary.hergm**

*Summary of object of class hergm*

**Description**

The function `summary.hergm` generates a summary of an object of class `hergm` by using the functions `print.hergm` and `plot.hergm`. The function `print.hergm` prints a summary of a sample of parameters from the posterior, whereas the function `plot.hergm` plots a summary of a sample of block memberships of nodes from the posterior.
Usage

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

Arguments

- `object` object of class `hergm`; objects of class `hergm` can be generated by function `hergm`.
- `...` additional arguments, to be passed to lower-level functions in the future.

References


See Also

`hergm`, `hergm.postprocess`, `print.hergm`, `plot.hergm`

Examples

```r
## Not run: data(example)
object <- hergm(d ~ edges_ij)
summary(object)

## End(Not run)```
Index

arcs_i (hergm-terms), 13
arcs_j (hergm-terms), 13
bali, 2
bunt, 2
ctriple_ijk (hergm-terms), 13
d (example), 3
edges_i (hergm-terms), 13
edges_ij (hergm-terms), 13
example, 3
gof.hergm, 4
hergm, 5
hergm-terms, 13
hergm.gof (gof.hergm), 4
hergm.mcmc.diagnostics, 14
hergm.plot (plot.hergm), 18
hergm.postprocess, 15
hergm.print (print.hergm), 19
hergm.simulate (simulate.hergm), 20
hergm.summary (summary.hergm), 21
hergm.terms (hergm-terms), 13
kapferer, 17
mcmc.diagnostics.hergm
   (hergm.mcmc.diagnostics), 14
mutual_i (hergm-terms), 13
mutual_ij (hergm-terms), 13
plot (plot.hergm), 18
plot.hergm, 18
postprocess.hergm (hergm.postprocess), 15
print.hergm, 19
simulate (simulate.hergm), 20

simulate.hergm, 20
summary.hergm, 21
terms-hergm (hergm-terms), 13
terms.hergm (hergm-terms), 13
transitivities_ijk (hergm-terms), 13
triangle_ijk (hergm-terms), 13
ttriple_ijk (hergm-terms), 13	wostar_ijk (hergm-terms), 13

23