Package: bigergm (via r-universe)

October 23, 2026

Title Fit, Simulate, and Diagnose Hierarchical Exponential-Family Models for Big Networks

Version 1.2.6

Description A toolbox for analyzing and simulating large networks based on hierarchical exponential-family random graph models (HERGMs).'bigergm' implements the estimation for large networks efficiently building on the 'lighthergm' and 'hergm' packages. Moreover, the package contains tools for simulating networks with local dependence to assess the goodness-of-fit.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 3.5.0), ergm (>= 4.5.0), Rcpp

LinkingTo Rcpp, RcppArmadillo (>= 0.10.5)

Imports RcppArmadillo (>= 0.10.5), network (>= 1.16.0), Matrix, cachem, tidyr, statnet.common, methods, stringr, intergraph, igraph, parallel, magrittr, purrr, dplyr, glue, readr, foreach, rlang, memoise, reticulate, ergm.multi

Suggests rmarkdown, knitr, testthat, sna, tibble

VignetteBuilder knitr

NeedsCompilation yes

Author Cornelius Fritz [aut, cre], Michael Schweinberger [aut], Shota Komatsu [aut], Juan Nelson Martínez Dahbura [aut], Takanori Nishida [aut], Angelo Mele [aut]

Maintainer Cornelius Fritz <corneliusfritz2010@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-03-24 10:40:18 UTC

RemoteUrl https://github.com/cran/bigergm

RemoteRef HEAD

RemoteSha 36ebf996f7c0589c5dadcccfc600dd5fe15cea33

Contents

ari . . . 2
bali . . . 3
bigergm . . . 3
bunt . . . 9
est_between . . . 10
est_within . . . 11
get_between_networks . . . 13
get_within_networks . . . 13
gof.bigergm . . . 14
kapferer . . . 15
plot.bigergm . . . 16
py_dep . . . 16
reed . . . 17
rice . . . 17
simulate.bigergm . . . 18
simulate_bigergm . . . 19
state_twitter . . . 20
toyNet . . . 21
yule . . . 22

Index **23**

`ari` *Compute the adjusted rand index (ARI) between two clusterings*

Description

This function computes the adjusted rand index (ARI) of the true and estimated block membership (its definition can be found here https://en.wikipedia.org/wiki/Rand_index). The adjusted rand index is used as a measure of association between two group membership vectors. The more similar the two partitions z_star and z are, the closer the ARI is to 1.

Usage

```
ari(z_star, z)
```

Arguments

`z_star`	The true block membership
`z`	The estimated block membership

Value

The adjusted rand index

Examples

```
data(toyNet)
set.seed(123)
ari(z_star = toyNet%v% "block",
z = sample(c(1:4),size = 200,replace = TRUE))
```

`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).

Format

A `statnet`'s network class object. data(bali)

References

Koschade, S. (2006). A social network analysis of Jemaah Islamiyah: The applications to counterterrorism and intelligence. Studies in Conflict and Terrorism, 29, 559–575.

`bigergm` *bigergm: Exponential-family random graph models for large networks with local dependence*

Description

The function `bigergm` estimates and simulates three classes of exponential-family random graph models for large networks under local dependence:

1. The p_1 model of Holland and Leinhardt (1981) in exponential-family form and extensions by Vu, Hunter, and Schweinberger (2013), Schweinberger, Petrescu-Prahova, and Vu (2014), Dahbura et al. (2021), and Fritz et al. (2024) to both directed and undirected random graphs with additional model terms, with and without covariates.
2. The stochastic block model of Snijders and Nowicki (1997) and Nowicki and Snijders (2001) in exponential-family form.
3. The exponential-family random graph models with local dependence of Schweinberger and Handcock (2015), with and without covariates. 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

```
bigergm(
  object,
  add_intercepts = FALSE,
  n_blocks = NULL,
  n_cores = 1,
  blocks = NULL,
  estimate_parameters = TRUE,
  verbose = 0,
  n_MM_step_max = 100,
  tol_MM_step = 1e-04,
  initialization = "infomap",
  use_infomap_python = FALSE,
  virtualenv_python = "r-bigergm",
  seed_infomap = NULL,
  weight_for_initialization = 1000,
  seed = NULL,
  method_within = "MPLE",
  control_within = ergm::control.ergm(),
  clustering_with_features = TRUE,
  compute_pi = FALSE,
  check_alpha_update = FALSE,
  check_blocks = FALSE,
  cache = NULL,
  return_checkpoint = TRUE,
  only_use_preprocessed = FALSE,
  ...
)
```

Arguments

object	An R formula object or bigergm class object. If a formula is given, the function estimates a new model specified by it. It needs to be of the form y ~ <model terms>, where y is a network object. For the details on the possible <model terms>, see ergmTerm and Morris, Handcock and Hunter (2008). All terms that induce dependence are excluded from the between block model, while the within block model includes all terms. When you pass a bigergm class object to the function, you continue from the previous MM step. Note that the block allocation (which is either provided by parameter blocks or estimated in the first step) is saved as the vertex.attribute block of the network. This attribute can also be used in the specified formula. The L-ergmTerm is supported to

enable size-dependent coefficients for the within-blocks model. Note, however, that for size-dependent parameters of terms that are included in the between-blocks model, the intercept in the linear model provided to L-ergmTerm should not include the intercept. See the second example below for a demonstration.

add_intercepts Boolean value to indicate whether adequate intercepts should be added to the provided formula so that the model in the first stage of the estimation is a nested model of the estimated model in the second stage of the estimation.

n_blocks The number of blocks. This must be specified by the user. When you pass a bigergm class object to the function, you don't have to specify this argument.

n_cores The number of CPU cores to use.

blocks The pre-specified block memberships for each node. If NULL, the latent community structure is estimated, assuming that the number of communities is n_blocks.

estimate_parameters
If TRUE, both clustering and parameter estimation are implemented. If FALSE, only clustering is executed.

verbose A logical or an integer: if this is TRUE/1, the program will print out additional information about the progress of estimation and simulation. A higher value yields lower level information.

n_MM_step_max The maximum number of MM iterations. Currently, no early stopping criteria is introduced. Thus n_MM_step_max MM iterations are exactly implemented.

tol_MM_step Tolerance regarding the relative change of the lower bound of the likelihood used to decide on the convergence of the clustering step

initialization How the blocks should be initialized. If infomap (the default), igraph' or Python's infomap is implemented. If random, the initial clusters are randomly uniformally selected. If spectral, spectral clustering is conducted. If walktrap, the walktrap clustering algorithm as implemented in cluster_walktrap is conducted. If initialization is a vector of integers of the same length as the number of nodes in the provided network (in object), then the provided vector is used as the initial cluster assignment. If initialization is a string relating to a file path, bigergm will interpret it as block allocations saved in Python's infomap .clu format under that path.

use_infomap_python
If TRUE, the cluster initialization is implemented using Pythons' infomap.

virtualenv_python
Which virtual environment should be used for the infomap algorithm?

seed_infomap seed value (integer) for the infomap algorithm, which can be used to initialize the estimation of the blocks.

weight_for_initialization
weight value used for cluster initialization. The higher this value, the more weight is put on the initialized block allocation.

seed seed value (integer) for the random number generator.

method_within If "MPLE" (the default), then the maximum pseudolikelihood estimator is implemented when estimating the within-block network model. If "MLE", then an approximate maximum likelihood estimator is conducted. If "CD" (EXPERIMENTAL), the Monte-Carlo contrastive divergence estimate is returned.

`control_within` A list of control parameters for the `ergm` function used to estimate the parameters of the within model. See `control.ergm` for details.

`clustering_with_features`
If `TRUE`, clustering is implemented using the discrete covariates specified in the formula. At the moment, we only support the `nodematch` term and disregard diff = TRUE if provided.

`compute_pi` If `TRUE`, this function keeps track of pi matrices at each MM iteration. If the network is large, we strongly recommend to set to be `FALSE`.

`check_alpha_update`
If `TRUE`, this function keeps track of alpha matrices at each MM iteration. If the network is large, we strongly recommend to set to be `FALSE`.

`check_blocks` If TRUE, this function keeps track of estimated block memberships at each MM iteration.

`cache` a `cachem` cache object used to store intermediate calculations such as eigenvector decomposition results.

`return_checkpoint`
If `TRUE`, the function returns the checkpoint list. For most applications, this should be set to `TRUE` but if memory space needed by the output is an issue, set to `FALSE`.

`only_use_preprocessed`
If `TRUE`, the function only uses the preprocessed data from a previous fit but does not continue the estimation from its final iteration, instead the estimation is started again from the provided initialization.

`...` Additional arguments, to be passed to lower-level functions (mainly to the `ergm` function used for the estimation of within-block connections).

Value

An object of class 'bigergm' including the results of the fitted model. These include:

call: call of the mode

block: vector of the found block of the nodes into cluster

initial_block: vector of the initial block of the nodes into cluster

sbm_pi: Connection probabilities represented as a `n_blocks` x `n_blocks` matrix from the first stage of the estimation between all clusters

MM_list_z: list of cluster allocation for each node and each iteration

MM_list_alpha: list of posterior distributions of cluster allocations for all nodes for each iteration

MM_change_in_alpha: change in 'alpha' for each iteration

MM_lower_bound: vector of the evidence lower bounds from the MM algorithm

alpha: matrix representing the converged posterior distributions of cluster allocations for all nodes

counter_e_step: integer number indicating the number of iterations carried out

adjacency_matrix: sparse matrix representing the adjacency matrix used for the estimation

estimation_status: character stating the status of the estimation

est_within: ergm object of the model for within cluster connections

est_between: ergm object of the model for between cluster connections

checkpoint: list of information to continue the estimation (only returned if `return_checkpoint = TRUE`)

membership_before_kmeans: vector of the found blocks of the nodes into cluster before the final check for bad clusters

estimate_parameters: binary value if the parameters in the second step of the algorithm should be estimated or not

References

Babkin, S., Stewart, J., Long, X., and M. Schweinberger (2020). Large-scale estimation of random graph models with local dependence. Computational Statistics and Data Analysis, 152, 1–19.

Dahbura, J. N. M., Komatsu, S., Nishida, T. and Mele, A. (2021), 'A structural model of business cards exchange networks'. https://arxiv.org/abs/2105.12704

Fritz C., Georg C., Mele A., and Schweinberger M. (2024). A strategic model of software dependency networks. https://arxiv.org/abs/2402.13375

Handcock, M. S. (2003). Assessing degeneracy in statistical models of social networks. Technical report, Center for Statistics and the Social Sciences, University of Washington, Seattle. https://csss.uw.edu/Papers/wp39.pdf

Holland, P. W. and S. Leinhardt (1981). An exponential family of probability distributions for directed graphs. Journal of the American Statistical Association, Theory & Methods, 76, 33–65.

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. Journal of Statistical Software, 24.

Nowicki, K. and T. A. B. Snijders (2001). Estimation and prediction for stochastic blockstructures. Journal of the American Statistical Association, Theory & Methods, 96, 1077–1087.

Schweinberger, M. (2011). Instability, sensitivity, and degeneracy of discrete exponential families. Journal of the American Statistical Association, Theory & Methods, 106, 1361–1370.

Schweinberger, M. (2020). Consistent structure estimation of exponential-family random graph models with block structure. Bernoulli, 26, 1205–1233.

Schweinberger, M. and M. S. Handcock (2015). Local dependence in random graph models: characterization, properties, and statistical inference. Journal of the Royal Statistical Society, Series B (Statistical Methodology), 7, 647-676.

Schweinberger, M., Krivitsky, P. N., Butts, C.T. and J. Stewart (2020). Exponential-family models of random graphs: Inference in finite, super, and infinite population scenarios. Statistical Science, 35, 627-662.

Schweinberger, M. and P. Luna (2018). HERGM: Hierarchical exponential-family random graph models. Journal of Statistical Software, 85, 1–39.

Schweinberger, M., Petrescu-Prahova, M. and D. Q. Vu (2014). Disaster response on September 11, 2001 through the lens of statistical network analysis. Social Networks, 37, 42–55.

Schweinberger, M. and J. Stewart (2020). Concentration and consistency results for canonical and curved exponential-family random graphs. The Annals of Statistics, 48, 374–396.

Snijders, T. A. B. and K. Nowicki (1997). Estimation and prediction for stochastic blockmodels for graphs with latent block structure. Journal of Classification, 14, 75–100.

Stewart, J., Schweinberger, M., Bojanowski, M., and M. Morris (2019). Multilevel network data facilitate statistical inference for curved ERGMs with geometrically weighted terms. Social Networks, 59, 98–119.

Vu, D. Q., Hunter, D. R. and M. Schweinberger (2013). Model-based clustering of large networks. Annals of Applied Statistics, 7, 1010–1039.

Examples

```
# Load an embedded network object.
data(toyNet)

# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle
# Estimate the model
bigergm_res <- bigergm(
  object = model_formula,
  # The model you would like to estimate
  n_blocks = 4,
  # The number of blocks
  n_MM_step_max = 10,
  # The maximum number of MM algorithm steps
  estimate_parameters = TRUE,
  # Perform parameter estimation after the block recovery step
  clustering_with_features = TRUE,
  # Indicate that clustering must take into account nodematch on characteristics
  check_blocks = FALSE)

 # Example with N() operator

 ## Not run:
set.seed(1)
# Prepare ingredients for simulating a network
N <- 500
K <- 10

list_within_params <- c(1, 2, 2,-0.5)
list_between_params <- c(-8, 0.5, -0.5)
formula <- g ~ edges + nodematch("x") + nodematch("y")  + N(~edges,~log(n)-1)

memb <- sample(1:K,prob = c(0.1,0.2,0.05,0.05,0.10,0.1,0.1,0.1,0.1,0.1),
               size = N, replace = TRUE)
vertex_id <- as.character(11:(11 + N - 1))

x <- sample(1:2, size = N, replace = TRUE)
y <- sample(1:2, size = N, replace = TRUE)


df <- tibble::tibble(
  id = vertex_id,
  memb = memb,
```

```
  x = x,
  y = y
)
g <- network::network.initialize(n = N, directed = FALSE)
g %v% "vertex.names" <- df$id
g %v% "block" <- df$memb
g %v% "x" <- df$x
g %v% "y" <- df$y

# Simulate a network
g_sim <-
  simulate_bigergm(
    formula = formula,
    coef_within = list_within_params,
    coef_between = list_between_params,
    nsim = 1,
    control_within = control.simulate.formula(MCMC.burnin = 200000))

estimation <- bigergm(update(formula,new = g_sim~.), n_blocks = 10,
                      verbose = T)
summary(estimation)

## 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".

Format

A statnet's network class object. data(bunt)

References

Van de Bunt, G. G. (1999). Friends by choice. An Actor-Oriented Statistical Network Model for Friendship Networks through Time. Thesis Publishers, Amsterdam.

Van de Bunt, G. G., Van Duijn, M. A. J., and T. A. B. Snijders (1999). Friendship Networks Through Time: An Actor-Oriented Statistical Network Model. Computational and Mathematical Organization Theory, 5, 167–192.

`est_between` *Estimate between-block parameters*

Description

Function to estimate the between-block model by relying on the maximum likelihood estimator.

Usage

```
est_between(
  formula,
  network,
  add_intercepts = TRUE,
  clustering_with_features = FALSE
)
```

Arguments

`formula`	An R `formula` object of the form y ~ `<model terms>`, where y is a `network` object. The network object must contain block information as a vertex attribute with the name 'block'. For the details on the possible `<model terms>`, see `ergmTerm` and Morris, Handcock and Hunter (2008). All terms that induce dependence are excluded from the between block model.
`network`	a network object with one vertex attribute called 'block' representing which node belongs to which block
`add_intercepts`	Boolean value to indicate whether adequate intercepts should be added to the provided formula so that the model in the first stage of the estimation is a nested model of the estimated model in the second stage of the estimation
`clustering_with_features`	Boolean value to indicate if the clustering was carried out making use of the covariates or not (only important if `add_intercepts = TRUE`)

Value

'ergm' object of the estimated model.

References

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. Journal of Statistical Software, 24.

Examples

```
adj <- c(
c(0, 1, 0, 0, 1, 0),
c(1, 0, 1, 0, 0, 1),
c(0, 1, 0, 1, 1, 0),
```

```
c(0, 0, 1, 0, 1, 1),
c(1, 0, 1, 1, 0, 1),
c(0, 1, 0, 1, 1, 0)
)
adj <- matrix(data = adj, nrow = 6, ncol = 6)
rownames(adj) <- as.character(1001:1006)
colnames(adj) <- as.character(1001:1006)

# Use non-consecutive block names
block <- c(50, 70, 95, 50, 95, 70)
g <- network::network(adj, matrix.type = "adjacency")
g %v% "block" <- block
est <- est_between(
  formula = g ~ edges,network = g,
  add_intercepts = FALSE, clustering_with_features = FALSE
)
```

`est_within` *Estimate a within-block network model.*

Description

Function to estimate the within-block model. Both pseudo-maximum likelihood and monte carlo approximate maximum likelihood estimators are implemented.

Usage

```
est_within(
  formula,
  network,
  seed = NULL,
  method = "MPLE",
  add_intercepts = TRUE,
  clustering_with_features = FALSE,
  return_network = FALSE,
  ...
)
```

Arguments

`formula`	An R `formula` object of the form y ~ <model terms>, where y is a `network` object. The network object must contain block information as a vertex attribute with the name 'block'. For the details on the possible <model terms>, see `ergmTerm` and Morris, Handcock and Hunter (2008). The `L-ergmTerm` is supported to enable size-dependent coefficients.
`network`	a network object with one vertex attribute called 'block' representing which node belongs to which block
`seed`	seed value (integer) for the random number generator

method	If "MPLE" (the default), then the maximum pseudolikelihood estimator is returned. If "MLE", then an approximate maximum likelihood estimator is returned.
add_intercepts	Boolean value to indicate whether adequate intercepts should be added to the provided formula so that the model in the first stage of the estimation is a nested model of the estimated model in the second stage of the estimation
clustering_with_features	Boolean value to indicate if the clustering was carried out making use of the covariates or not (only important if add_intercepts = TRUE)
return_network	Boolean value to indicate if the network object should be returned in the output. This is needed if the user wants to use, e.g., the gof function as opposed to the gof.bigergm function.
...	Additional arguments, to be passed to the ergm function

Value

'ergm' object of the estimated model.

References

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. Journal of Statistical Software, 24.

Examples

```
adj <- c(
c(0, 1, 0, 0, 1, 0),
c(1, 0, 1, 0, 0, 1),
c(0, 1, 0, 1, 1, 0),
c(0, 0, 1, 0, 1, 1),
c(1, 0, 1, 1, 0, 1),
c(0, 1, 0, 1, 1, 0)
)
adj <- matrix(data = adj, nrow = 6, ncol = 6)
rownames(adj) <- as.character(1001:1006)
colnames(adj) <- as.character(1001:1006)

# Use non-consecutive block names
block <- c(70, 70, 70, 70, 95, 95)
g <- network::network(adj, matrix.type = "adjacency", directed = FALSE)
g %v% "block" <- block
g %v% "vertex.names" <- 1:length(g %v% "vertex.names")
est <- est_within(
formula = g ~ edges,
  network = g,
  parallel = FALSE,
  verbose = 0,
  initial_estimate = NULL,
  seed = NULL,
  method = "MPLE",
```

```
add_intercepts = FALSE,
clustering_with_features = FALSE
)
```

`get_between_networks` *Obtain the between-block networks defined by the block attribute.*

Description

Function to return a list of networks, each network representing the within-block network of a block.

Usage

```
get_between_networks(network, block)
```

Arguments

`network`	a network object
`block`	a vector of integers representing the block of each node

Value

a list of networks

Examples

```
# Load an embedded network object.
data(toyNet)
get_within_networks(toyNet, toyNet %v% "block")
```

`get_within_networks` *Obtain the within-block networks defined by the block attribute.*

Description

Function to return a list of networks, each network representing the within-block network of a block.

Usage

```
get_within_networks(network, block, combined_networks = TRUE)
```

Arguments

`network`	a network object
`block`	a vector of integers representing the block of each node
`combined_networks`	a boolean indicating whether the between-block networks should be returned as a `combined_networks` object or not (default is TRUE)

Value

a list of networks

Examples

```
# Load an embedded network object.
data(toyNet)
get_within_networks(toyNet, toyNet %v% "block")
```

`gof.bigergm` *Conduct Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model for big networks*

Description

A sample of graphs is randomly drawn from the specified model. The first argument is typically the output of a call to `bigergm` and the model used for that call is the one fit.

By default, the sample consists of 100 simulated networks, but this sample size (and many other settings) can be changed using the `ergm_control` argument described above.

Usage

```
## S3 method for class 'bigergm'
gof(
  object,
  ...,
  type = "full",
  control_within = ergm::control.simulate.formula(),
  seed = NULL,
  nsim = 100,
  compute_geodesic_distance = TRUE,
  start_from_observed = TRUE,
  simulate_sbm = FALSE
)
```

Arguments

`object`	An `bigergm` object.
`...`	Additional arguments, to be passed to `simulate_bigergm`, which, in turn, passes the information to `simulate_formula`. See documentation for `bigergm`.
`type`	the type of evaluation to perform. Can take the values `full` or `within`. `full` performs the evaluation on all edges, and `within` only considers within-block edges.
`control_within`	MCMC parameters as an instance of `control.simulate.formula` to be used for the within-block simulations.
`seed`	the seed to be passed to simulate_bigergm. If `NULL`, a random seed is used.

nsim the number of simulations to employ for calculating goodness of fit, default is 100.

compute_geodesic_distance
if TRUE, the distribution of geodesic distances is also computed (considerably increases computation time on large networks. FALSE by default.)

start_from_observed
if TRUE, MCMC uses the observed network as a starting point. If FALSE, MCMC starts from a random network.

simulate_sbm if TRUE, the between-block connections are simulated from the estimated stochastic block model from the first stage not the estimated ERGM.

Value

gof.bigergm returns a list with two entries. The first entry 'original' is another list of the network stats, degree distribution, edgewise-shared partner distribution, and geodesic distance distribution (if compute_geodesic_distance = TRUE) of the observed network. The second entry is called 'simulated' is also list compiling the network stats, degree distribution, edgewise-shared partner distribution, and geodesic distance distribution (if compute_geodesic_distance = TRUE) of all simulated networks.

Examples

```
data(toyNet)

# Specify the model that you would like to estimate.
data(toyNet)
# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle
estimate <- bigergm(model_formula,n_blocks = 4)
gof_res <- gof(estimate,
nsim = 100
)
plot(gof_res)
```

---

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).

Format

A statnet's network class object. data(kapferer)

References

Kapferer, B. (1972). Strategy and Transaction in an African Factory. Manchester University Press, Manchester, U.K.

`plot.bigergm` *Plot the network with the found clusters*

Description

This function plots the network with the found clusters. The nodes are colored according to the found clusters. Note that the function uses the `network` package for plotting the network and should therefore not be used for large networks with more than 1-2 K vertices

Usage

```
## S3 method for class 'bigergm'
plot(x, ...)
```

Arguments

`x`	The output of the bigergm function
`...`	Additional arguments, to be passed to lower-level functions

---

`py_dep` *Install optional Python dependencies for bigergm*

Description

Install Python dependencies needed for using the Python implementation of infomap. The code uses the `reticulate` package to install the Python packages `infomap` and `numpy`. These packages are needed for the `bigergm` function when `use_infomap_python = TRUE` else the Python implementation is not needed.

Usage

```
py_dep(envname = "r-bigergm", method = "auto", ...)
```

Arguments

`envname`	The name, or full path, of the environment in which Python packages are to be installed. When NULL (the default), the active environment as set by the RETICULATE_PYTHON_ENV variable will be used; if that is unset, then the r-reticulate environment will be used.
`method`	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
`...`	Additional arguments, to be passed to lower-level functions

Value

No return value, called for installing the Python dependencies 'infomap' and 'numpy'

`reed` *A network of friendships between students at Reed College.*

Description

The data was collected by Facebook and provided as part of Traud et al. (2012)

Format

A `statnet`'s network class object. It has three nodal features.

doorm anonymized dorm in which each node lives.

gender gender of each node.

high.school anonymized highschool to which each node went to.

year year of graduation of each node. ...

data(reed)

References

Traud, Mucha, Porter (2012). Social Structure of Facebook Network. Physica A: Statistical Mechanics and its Applications, 391, 4165-4180

`rice` *A network of friendships between students at Rice University.*

Description

The data was collected by Facebook and provided as part of Traud et al. (2012)

Format

A `statnet`'s network class object. It has three nodal features.

doorm anonymized dorm in which each node lives.

gender gender of each node.

high.school anonymized highschool to which each node went to.

year year of graduation of each node.

data(rice)

References

Traud, Mucha, Porter (2012). Social Structure of Facebook Network. Physica A: Statistical Mechanics and its Applications, 391, 4165-4180

`simulate.bigergm` *Simulate networks under Exponential Random Graph Models (ERGMs) under local dependence*

Description

This function simulates networks under the Exponential Random Graph Model (ERGM) with local dependence with all parameters set according to the estimated model (`object`). See `simulate_bigergm` for details of the simulation process

Usage

```
## S3 method for class 'bigergm'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  ...,
  output = "network",
  control_within = ergm::control.simulate.formula(),
  only_within = FALSE,
  verbose = 0
)
```

Arguments

`object`	an object of class `bigergm`
`nsim`	number of networks to be randomly drawn from the given distribution on the set of all networks.
`seed`	seed value (integer) for network simulation.
`...`	Additional arguments, passed to `simulate_formula`.
`output`	Normally character, one of "network" (default), "stats", "edgelist", to determine the output of the function.
`control_within`	`control.simulate.formula` object for fine-tuning ERGM simulation of within-block networks.
`only_within`	If this is TRUE, only within-block networks are simulated.
`verbose`	If this is TRUE/1, the program will print out additional information about the progress of simulation.

Value

Simulated networks, the output form depends on the parameter `output` (default is a list of networks).

simulate_bigergm *Simulate networks under Exponential Random Graph Models (ERGMs) under local dependence*

Description

This function simulates networks under Exponential Random Graph Models (ERGMs) with local dependence. There is also an option to simulate only within-block networks and a S3 method for the class `bigergm`.

Usage

```
simulate_bigergm(
  formula,
  coef_within,
  coef_between,
  network = ergm.getnetwork(formula),
  control_within = ergm::control.simulate.formula(),
  only_within = FALSE,
  seed = NULL,
  nsim = 1,
  output = "network",
  verbose = 0,
  ...
)
```

Arguments

`formula`	An R `formula` object of the form `y ~ <model terms>`, where `y` is a `network` object. The network object must contain block information as a vertex attribute with the name 'block'. For the details on the possible `<model terms>`, see `ergmTerm` and Morris, Handcock and Hunter (2008). All terms that induce dependence are excluded from the between block model, while the within block model includes all terms. The `L-ergmTerm` is supported to enable size-dependent coefficients for the within-blocks model. Note, however, that for size-dependent parameters of terms that are included in the between-blocks model, the intercept in the linear model provided to `L-ergmTerm` should not include the intercept. See the second example of `bigergm` for a demonstration.
`coef_within`	a vector of within-block parameters. The order of the parameters should match that of the formula.
`coef_between`	a vector of between-block parameters. The order of the parameters should match that of the formula without externality terms.
`network`	a network object to be used as a seed network for the simulation (if none is provided, the network on the lhs of the `formula` is used).
`control_within`	auxiliary function as user interface for fine-tuning ERGM simulation for within-block networks.

only_within	If this is TRUE, only within-block networks are simulated.
seed	seed value (integer) for network simulation.
nsim	number of networks generated.
output	Normally character, one of "network" (default), "stats", "edgelist", to determine the output format.
verbose	If this is TRUE/1, the program will print out additional information about the progress of simulation.
...	Additional arguments, passed to simulate_formula.

Value

Simulated networks, the output form depends on the parameter output (default is a list of networks).

References

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. Journal of Statistical Software, 24.

Examples

```
data(toyNet)
# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle
# Simulate network stats
sim_stats <- bigergm::simulate_bigergm(
formula = model_formula,
  # Formula for the model
coef_between = c(-4.5,0.8, 0.4),
# The coefficients for the between connections
coef_within = c(-1.7,0.5,0.6,0.15),
# The coefficients for the within connections
nsim = 10,
# Number of simulations to return
output = "stats",
# Type of output
)
```

---

state_twitter *Twitter (X) network of U.S. state legislators*

Description

The network includes the Twitter (X) following interactions between U.S. state legislators. The data was collection by Gopal et al. (2022) and Kim et al. (2022). For this network, we only include the largest connected component of state legislators that were active on Twitter in the six months leading up to and including the insurrection at the United States Capitol on January 6, 2021. All state senate and state representatives for states with a bicameral system are included and all state legislators for state (Nebraska) with a unicameral system are included.

Usage

```
data(state_twitter)
```

Format

A `statnet`'s network class object. It has the following categorical attributes for each state legislator.

gender factor stating whether the legislator is 'female' or 'male'.

party party affiliation of the legislator, which is 'Democratic', 'Independent' or 'Republican'.

race race with the following levels: 'Asian or Pacific Islander', 'Black', 'Latino', 'MENA(Middle East and North Africa)','Multiracial', 'Native American', and 'White'.

state character of the state that the legislator represents.

References

Gopal, Kim, Nakka, Boehmke, Harden, Desmarais. The National Network of U.S. State Legislators on Twitter. Political Science Research & Methods, Forthcoming.

Kim, Nakka, Gopal, Desmarais,Mancinelli, Harden, Ko, and Boehmke (2022). Attention to the COVID-19 pandemic on Twitter: Partisan differences among U.S. state legislators. Legislative Studies Quarterly 47, 1023–1041.

`toyNet` *A toy network to play* `bigergm` *with.*

Description

This network has a clear cluster structure. The number of clusters is four, and which cluster each node belongs to is defined in the variable "block".

Usage

```
data(toyNet)
```

Format

A statnet's network class object. It has three nodal features.

block block membership of each node

x a covariate. It has 10 labels.

y a covariate. It has 10 labels. ...

1 and 2 are not variables with any particular meaning.

---

yule *Compute Yule's Phi-coefficient*

Description

This function computes Yule's Phi-coefficient between the true and estimated block membership (its definition can be found here https://en.wikipedia.org/wiki/Phi_coefficient). In this context, the Phi Coefficient is a measure of association between two group membership vectors.

Usage

```
yule(z_star, z)
```

Arguments

z_star	a true block membership
z	an estimated block membership

Value

Real value of Yule's Phi-coefficient between the true and estimated block membership is returned.

Examples

```
data(toyNet)
yule(z_star = toyNet%v% "block",
                 z = sample(c(1:4),size = 200,replace = TRUE))
```

Index

ari, 2

bali, 3
bigergm, 3, *4, 5*, *14*, *19*
bunt, 9

cluster_walktrap, *5*
control.ergm, *6*
control.simulate.formula, *18*

ergm, *6, 7*, *12*
ergmTerm, *4*, *10, 11*, *19*
est_between, 10
est_within, 11

formula, *4*, *10, 11*, *19*

get_between_networks, 13
get_within_networks, 13
gof, *12*
gof.bigergm, *12*, 14, *15*

kapferer, 15

network, *4*, *10, 11*, *19*

plot.bigergm, 16
py_dep, 16

reed, 17
rice, 17

simulate.bigergm, 18
simulate_bigergm, *14*, *18*, 19
simulate_formula, *14*, *18*, *20*
state_twitter, 20

toyNet, 21

yule, 22