Package 'bigergm'

February 16, 2024

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

Version 1.1.0

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

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

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, tibble, glue, readr, foreach, rlang, doParallel, memoise, reticulate

Suggests rmarkdown, knitr, testthat, sna

VignetteBuilder knitr

NeedsCompilation yes

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

Maintainer Cornelius Fritz <corneliusfritz2010@gmail.com>

Repository CRAN

Date/Publication 2024-02-16 15:10:10 UTC

R topics documented:

`compute_multiplied_feature_matrices`
Get a list of sparse feature adjacency matrix from a formula.

Description

These matrices can be given to the `hergm` function as parameters. Generally, this function should only be used if users are working with large networks and are planning to continually estimate the model.

Usage

```
compute_multiplied_feature_matrices(net, list_feature_matrices)
```

Arguments

`net`	a network object from which nodal covariates are extracted.
`list_feature_matrices`	a list of feature adjacency matrices generated by `get_list_sparse_feature_adjmat()`.

Value

A list of sparse matrices of multiplied feature matrices that are needed for carrying our the first step of the estimation if the covariates should be used.

Examples

```
data(toyNet)

model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y")
list_feature_matrices <- get_list_sparse_feature_adjmat(toyNet, model_formula)
multiplied_feature_matrices <-
  compute_multiplied_feature_matrices(net = toyNet,
  list_feature_matrices = list_feature_matrices)
```

---

compute_yule_coef *Compute Yule's Phi-coefficient*

Description

Compute Yule's Phi-coefficient

Usage

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

`draw_between_block_connection` *Draw between-block connections.*

Description

Draw between-block connections. There may be some edges that appear both in within- and between-block links. The overlapped edges will be removed after this step.

Usage

```
draw_between_block_connection(
  formula_for_simulation,
  sorted_dataframe,
  coef_between_block,
  seed_edgelist_between = NULL,
  use_fast_between_simulation = FALSE,
  list_feature_matrices = NULL,
  seed = NULL,
  n_sim = 1,
  prevent_duplicate = TRUE,
  verbose = 0,
  ergm_control = ergm::control.simulate.formula(),
  output = "edgelist",
  ...
)
```

Arguments

`formula_for_simulation`
: formula for simulating a between-block network

`sorted_dataframe`
: a data frame with the covariate information. The order must match the nodes in the network and it must contain the coliumn 'vertex_id' matching the network.

`coef_between_block`
: a vector of between-block parameters. The order of the parameters should match that of the formula.

`seed_edgelist_between`
: a seed edgelist from which a between-block network is simulated.

`use_fast_between_simulation`
: If `TRUE`, this function uses an effcient way to simulate a between-block network. If the network is very large, you should consider using this option. Note that when you use this, the first element of `coef_between_block` must be the edges parameter.

`list_feature_matrices`
: a list of feature adjacency matrices. This is used when `use_fast_between_simulation`.

seed	seed value (integer) for the random number generator.
n_sim	number of networks generated.
prevent_duplicate	If TRUE, the coefficient on nodematch("block") is set to be a very large negative number in drawing between-block links, so that there will be (almost) no within-block links.
verbose	If this is TRUE/1, the program will print out additionalinformation about the progress of simulation.
ergm_control	auxiliary function as user interface for fine-tuning ERGM simulation
output	Normally character, one of "network" (default), "stats", "edgelist", to determine the output format.
...	Additional arguments, to be passed to lower-level functions

Value

A network.list object of the n_sim networks.

Examples

```
data(toyNet)

# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y")
# Estimate the model
nodes_data <- data.frame(
  vertex_id = 1:toyNet$gal$n,
  x = toyNet %v% "x",
  y = toyNet %v% "y",
  block = toyNet %v% "block"
)
list_feature_matrices <-
   get_list_sparse_feature_adjmat(toyNet, model_formula)
toyNet <- network::as.edgelist(toyNet)

draw_between_block_connection(formula_for_simulation = model_formula,
                              sorted_dataframe = nodes_data,
                              coef_between_block = c(-2,0.1,0.2),
                              n_sim = 10)
```

---

draw_within_block_connection

Draw within-block connections

Description

Draw within-block connections

Usage

```
draw_within_block_connection(
  seed_network,
  formula_for_simulation,
  coef_within_block,
  ergm_control,
  output = "network",
  seed,
  n_sim,
  verbose,
  ...
)
```

Arguments

seed_network	a seed network from which a network will be simulated.
formula_for_simulation	formula for simulating a network
coef_within_block	a vector of within-block parameters. The order of the parameters should match that of the formula.
ergm_control	auxiliary function as user interface for fine-tuning ERGM simulation
output	Normally character, one of "network" (default), "stats", "edgelist", to determine the output format.
seed	seed value (integer) for the random number generator.
n_sim	Number of networks to be randomly drawn from the given distribution on the set of all networks.
verbose	If this is TRUE/1, the program will print out additionalinformation about the progress of simulation.
...	Additional arguments, to be passed to lower-level functions

Value

Simulated within-block connections, the output form depends on the parameter output.

Examples

```
data(toyNet)

model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y")
draw_within_block_connection(formula_for_simulation = model_formula,
                             coef_within_block = c(-2,0.1,0.2),
                             ergm_control = control.simulate(),
                             seed_network =toyNet,
                             verbose = TRUE,
                             output = "edgelist",
                             seed = 123,
```

```
                                n_sim = 1)
```

`estimate_between_param` *Estimate between-block parameters by logit*

Description

Estimate between-block parameters by logit

Usage

```
estimate_between_param(formula, network, block)
```

Arguments

`formula`	formula for estimating between-block parameters
`network`	network object
`block`	a vector that represents which node belongs to which node

Value

'ergm' object of the estimated model.

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")

est <- estimate_between_param(
  formula = g ~ edges,network = g, block = block
)
```

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

Description

Estimate a within-block network model.

Usage

```
estimate_within_params(
  formula,
  network,
  z_memb,
  number_cores = 1,
  verbose = 1,
  seeds = NULL,
  method_second_step = c("MPLE", "MLE"),
  offset_coef = NULL,
  ...
)
```

Arguments

`formula`	a within network formula
`network`	a network object
`z_memb`	block memberships for each node
`number_cores`	The number of CPU cores to use.
`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.
`seeds`	seed value (integer) for the random number generator
`method_second_step`	If "MPLE" (the default), then the maximum pseudolikelihood estimator is returned. If "MLE", then an approximate maximum likelihood estimator is returned.
`offset_coef`	a vector of model parameters to be fixed when estimation.(i.e., not estimated).
`...`	Additional arguments, to be passed to lower-level functions

Value

'ergm' object of the estimated model.

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")

est <- estimate_within_params(
  formula = g ~ edges,
  network = g,
  z_memb = block,
  parallel = FALSE,
  verbose = 0,
  initial_estimate = NULL,
  seeds = NULL,
  method_second_step = "MPLE"
)
```

---

get_list_sparse_feature_adjmat

Get a list of sparse feature adjacency matrix from a formula

Description

Get a list of sparse feature adjacency matrix from a formula

Usage

```
get_list_sparse_feature_adjmat(network, formula)
```

Arguments

network	a network object from which nodal covariates are extracted.
formula	a network model to be considered

Value

The list of sparse matrices of feature matrices that are used for the first step of the estimation.

Examples

```
data(toyNet)
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y")
list_feature_matrices <-
  get_list_sparse_feature_adjmat(toyNet, model_formula)
```

`gof_bigergm` *Goodness of fit statistics for HERGM*

Description

Goodness of fit statistics for HERGM

Usage

```
gof_bigergm(
  net,
  data_for_simulation,
  list_feature_matrices,
  colname_vertex_id,
  colname_block_membership,
  bigergm_results,
  type = "full",
  ergm_control = ergm::control.simulate.formula(),
  seed = NULL,
  n_sim = 1,
  prevent_duplicate = TRUE,
  compute_geodesic_distance = FALSE,
  start_from_observed = FALSE,
  ...
)
```

Arguments

`net`	the target network
`data_for_simulation`	a dataframe with node-level covariates
`list_feature_matrices`	a list of feature adjacency matrices
`colname_vertex_id`	the name of the column that contains the node id
`colname_block_membership`	the name o the column that contains the block affiliation of each node
`bigergm_results`	a bigergm results object

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.
ergm_control	MCMC parameters as an instance of ergm.control
seed	the seed to be passed to simulate_hergm
n_sim	the number of simulations to employ for calculating goodness of fit
prevent_duplicate	see simulate_hergm
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
...	Additional arguments, to be passed to lower-level functions

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.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle
# Estimate the model
nodes_data <- data.frame(
  node_id = 1:toyNet$gal$n,
  x = toyNet %v% "x",
  y = toyNet %v% "y",
  block = toyNet %v% "block"
)
list_feature_matrices <- bigergm::get_list_sparse_feature_adjmat(toyNet, model_formula)
estimate <- hergm(model_formula,n_clusters = 4)
gof_res <- bigergm::gof_bigergm(
  toyNet,
  list_feature_matrices = list_feature_matrices,
  data_for_simulation = nodes_data,
  colname_vertex_id = "node_id",
  colname_block_membership = "block",
  bigergm_results = estimate,
  n_sim = 100
)
```

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

Description

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

Usage

```
hergm(
  object,
  n_clusters,
  n_cores = 1,
  block_membership = NULL,
  estimate_parameters = TRUE,
  verbose = 0,
  n_MM_step_max = 100,
  tol_MM_step = 1e-04,
  initialization_method = 1,
  use_infomap_python = FALSE,
  virtualenv_python = "r-bigergm",
  seed_infomap = NULL,
  weight_for_initialization = 1000,
  seeds = NULL,
  initialized_cluster_data = NULL,
  method_second_step = "MPLE",
  clustering_with_features = TRUE,
  list_multiplied_feature_matrices = NULL,
  fix_covariate_parameter = FALSE,
  compute_pi = FALSE,
  check_alpha_update = FALSE,
  check_block_membership = FALSE,
  cache = NULL,
  ...
)
```

Arguments

object	A formula or bigergm class object. A bigergm is returned by hergm(). When you pass a bigergm class object to hergm(), you can restart the MM step.
n_clusters	The number of blocks. This must be specified by the user. When you pass a "bigergm" class object to hergm(), you don't have to specify this argument.
n_cores	The number of CPU cores to use.

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

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_method
: Cluster initialization method. If 1 (the default), igraph's infomap is implemented. If 2, the initial clusters are randomly uniformally selected. If 3, spectral clustering is conducted.

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 alpha.

seeds
: seed value (integer) for the random number generator

initialized_cluster_data
: initialized cluster data from which the MM iterations begin. This can be either a vector of block affiliations of each node or initialized cluster data by Python's infomap (given by .clu format).

method_second_step
: 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.

clustering_with_features
: If TRUE, clustering is implemented using the discrete covariates specified in the formula.

list_multiplied_feature_matrices
: a list of multiplied feature adjacency matrices necessary for MM step. If NULL, hergm() automatically calculates. Or you can calculate by compute_multiplied_feature_matrices()

fix_covariate_parameter
: If TRUE, when estimating the within-block network model, parameters for covariates are fixed at the estimated of the between-block network model.

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_block_membership`
: 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.

`...`
: Additional arguments, to be passed to lower-level functions

Value

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

call: call of the mode

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

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

sbm_pi: Connection probabilities represented as a `n_clusters` x `n_clusters` 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

membership_before_kmeans: vector of the found partition 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

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
hergm_res <- bigergm::hergm(
```

```
  object = model_formula,
  # The model you would like to estimate
  n_clusters = 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_block_membership = FALSE)
```

`install_python_dependencies` *Install optional Python dependencies*

Description

Install Python dependencies needed for using the Python implementation of infomap

Usage

```
install_python_dependencies(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'

simulate_hergm *Simulate a network*

Description

Simulate a network

Usage

```
simulate_hergm(
  formula_for_simulation,
  data_for_simulation,
  colname_vertex_id,
  colname_block_membership,
  seed_edgelist = NULL,
  coef_within_block,
  coef_between_block,
  ergm_control = ergm::control.simulate.formula(),
  seed = NULL,
  directed = FALSE,
  n_sim = 1,
  output = "network",
  prevent_duplicate = TRUE,
  use_fast_between_simulation = FALSE,
  list_feature_matrices = NULL,
  verbose = 0,
  ...
)
```

Arguments

formula_for_simulation	formula for simulating a network
data_for_simulation	a data frame that contains vertex id, block membership, and vertex features.
colname_vertex_id	a column name in the data frame for the vertex id
colname_block_membership	a column name in the data frame for the block membership
seed_edgelist	an edgelist used for creating a seed network. It should have the "edgelist" class
coef_within_block	a vector of within-block parameters. The order of the parameters should match that of the formula.
coef_between_block	a vector of between-block parameters. The order of the parameters should match that of the formula without externality terms.

ergm_control	auxiliary function as user interface for fine-tuning ERGM simulation
seed	seed value (integer) for network simulation.
directed	whether the simulated network is directed
n_sim	number of networks generated
output	Normally character, one of "network" (default), "stats", "edgelist", to determine the output format.
prevent_duplicate	If TRUE, the coefficient on nodematch("block") is set to be a very large negative number in drawing between-block links, so that there will be (almost) no within-block links.
use_fast_between_simulation	If TRUE, this function uses an effcient way to simulate a between-block network. If the network is very large, you should consider using this option. Note that when you use this, the first element of coef_between_block must be the edges parameter.
list_feature_matrices	a list of feature adjacency matrices. If use_fast_between_simulation, this must be given.
verbose	If this is TRUE/1, the program will print out additional information about the progress of simulation.
...	Additional arguments, to be passed to lower-level functions

Value

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

Examples

```
data(toyNet)

# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y") + triangle

# Prepare a data frame that contains nodal id and covariates.
nodes_data <-
  data.frame(
    node_id = network::network.vertex.names(toyNet),
    block = network::get.vertex.attribute(toyNet, "block"),
    x = network::get.vertex.attribute(toyNet, "x"),
    y = network::get.vertex.attribute(toyNet, "y")
  )
# The feature adjacency matrices
list_feature_matrices <- bigergm::get_list_sparse_feature_adjmat(toyNet, model_formula)

# Simulate network stats
sim_stats <- bigergm::simulate_hergm(
  formula_for_simulation = model_formula,
```

```
  data_for_simulation = nodes_data,
  # Nodal data
  colname_vertex_id = "node_id",
  # Name of the column containing node IDs
  colname_block_membership = "block",
  # Name of the column containing block IDs
  coef_between_block = c(-4.5,0.8, 0.4),
  # The coefficients for the between connections
  coef_within_block = c(-1.7,0.5,0.6,0.15),
  n_sim = 10,
  # Number of simulations to return
  output = "stats",
  # Type of output
  list_feature_matrices = list_feature_matrices
  # Information on the covariates
)
```

`simulate_hergm_within` *Sample within cluster networks*

Description

Obtains network statistics based on MCMC simulations including only the within-blocks connections.

Usage

```
simulate_hergm_within(
  formula_for_simulation,
  data_for_simulation,
  colname_vertex_id,
  colname_block_membership,
  coef_within_block,
  seed_edgelist = NULL,
  output = "stats",
  ergm_control = ergm::control.simulate.formula(),
  seed = NULL,
  n_sim = 1,
  verbose = 0,
  ...
)
```

Arguments

`formula_for_simulation`
: formula for simulating a network

`data_for_simulation`
: a data frame that contains vertex id, block membership, and vertex features.

colname_vertex_id
: a column name in the data frame for the vertex ids

colname_block_membership
: a column name in the data frame for the block membership

coef_within_block
: a vector of within-block parameters. The order of the parameters should match that of the formula.

seed_edgelist
: an edgelist used for creating a seed network. It should have the "edgelist" class

output
: The desired output of the simulation (any of `stats`, `network` or `edgelist`). Defaults to `stats`

ergm_control
: auxiliary function as user interface for fine-tuning ERGM simulation

seed
: seed value (integer) for network simulation.

n_sim
: number of networks generated

verbose
: If this is TRUE/1, the program will print out additional information about the progress of simulation.

...
: arguments to be passed to low level functions

Value

A 'data.frame' object where the columns relate to the sufficient statistics specified in `formula_for_simulation` and each row relates to one of the `n_sim` simulations.

Examples

```
data(toyNet)
# Specify the model that you would like to estimate.
model_formula <- toyNet ~ edges + nodematch("x") + nodematch("y")
# Estimate the model
nodes_data <- data.frame(
  node_id = 1:toyNet$gal$n,
  x = toyNet %v% "x",
  y = toyNet %v% "y",
  block = toyNet %v% "block"
)
list_feature_matrices <-
   get_list_sparse_feature_adjmat(toyNet, model_formula)
toyNet <- network::as.edgelist(toyNet)

simulate_hergm_within(formula_for_simulation = model_formula,
                      data_for_simulation = nodes_data,
                      colname_vertex_id = "node_id",
                      colname_block_membership = "block",
                      coef_within_block = c(-2,0.1,0.2),
                      n_sim = 10)
```

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

```
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. ...

x and y are not variables with any particular meaning.

Index