Package ‘ergm.multi’

May 29, 2023

Version 0.2.0
Date 2023-05-29
Title Fit, Simulate and Diagnose Exponential-Family Models for Multiple or Multilayer Networks
Depends R (>= 3.5.0), ergm (>= 4.5.0), network (>= 1.18.0)
Imports statnet.common (>= 4.9.0), rlang (>= 1.0.6), purrr (>= 0.3.5), tibble (>= 3.1.8), glue (>= 1.6.2), rle (>= 0.9.2), Rdpack (>= 2.4), Matrix, methods, parallel
LinkingTo ergm
Suggests rmarkdown (>= 2.18), knitr (>= 1.41), dplyr (>= 1.0.10), testthat (>= 3.1.6), ggplot2, ggrepel, generics


License GPL-3 + file LICENSE
URL https://statnet.org
VignetteBuilder rmarkdown, knitr
RoxygenNote 7.2.3
Roxygen list(markdown = TRUE)
Config/testthat/parallel true
Config/testthat/edition 3
Encoding UTF-8

R topics documented:

   ergm.multi-package .................................................. 2
   as_tibble.combined_networks ....................................... 4
   b1degreeL-ergmTerm ................................................. 5
ergm.multi-package

ergm.multi-package: Fit, Simulate and Diagnose Exponential-Family Models for Multiple or Multilayer Networks

Description

ergm.multi is a collection of extensions and utilities for package ergm to facilitate modeling of multilayer and multi-network models. Some experimental support for multimode networks is also implemented.
Multilayer network models

Also known as multiplex, multirelational, or multivariate networks, in a multilayer network a pair of actors can have multiple simultaneous relations of different types. For example, in the Lazega lawyer data set included with this package, each pair of lawyers in the firm can have an advice relationship, a coworking relationship, a friendship relationship, or any combination thereof. Application of ERGMs to multilayer networks has a long history (Pattison and Wasserman 1999; Lazega and Pattison 1999), and a number of R packages exist for analysing and estimating them.

ergm.multi implements the general approach of Krivitsky et al. (2020) for specifying multilayer ERGMs, including Layer Logic and the various cross-layer specifications. Its features include:

- **Seamless integration with `ergm()`**: Multilayer specification is contained entirely in an `ergm()`-style formula and can be nested with any other `ergm()` terms, including dynamic and multilayer.
- **Unlimited layers**: The number of layers in the modeled network is limited only by computing power.
- **Flexibility and simplicity**: Any valid binary ERGM can be specified for any later or a logical combination of layers using simple term operators.
- **Heterogeneous layers**: A network can have directed and undirected layers, which can be modeled jointly.
- **Multimode/multilevel support (experimental)**: With some care, it is possible to specify models for unipartite and bipartite layers over different subsets of actors, which can be used to specify multimode models.

See `Layer()` and `ergmTerm?L` for examples.

Multi-network models

Joint modeling of independent samples of networks on disjoint sets of actors have a long history as well (Zijlstra et al. 2006, Slaughter and Koehly 2016, Stewart et al. 2019, and Vega Yon et al. 2021, for example). ergm.multi facilitates fixed-effect models for samples of networks (possibly heterogeneous in size and composition), using a multivariate linear model for each network’s ERGM parameters, with network-level attributes serving as predictors, as formulated by Slaughter and Koehly (2016) and Krivitsky et al. (2022).

Its features include:

- **Seamless integration with `ergm()`**: Multi-network model specification is contained entirely in an `ergm()`-style formula and can be nested with any other `ergm()` terms, including dynamic and multilayer.
- **Flexibility and simplicity**: Any valid binary or valued ERGM can be specified for the networks, using simple term operators and the network-level specification with an `lm()`-style formula.

See `Networks()`, `ergmTerm?N` for specification, `gofN()` for diagnostic facilities, and `vignette("Goeyvaerts_reproduction")` for a demonstration.

**Author(s)**

Pavel N. Krivitsky <pavel@statnet.org>
References


---

**as_tibble.combined_networks**

An as_tibble method for combined networks.

---

**Description**

A method to obtain a network attribute table from a combined_networks object, falling back to the network::as_tibble.network() if vertex or edge attributes are required.

**Usage**

```r
## S3 method for class 'combined_networks'
as_tibble(
  x,
  attrnames = (match.arg(unit) %in% c("vertices", "networks")),
  ..., 
  unit = c("edges", "vertices", "networks"),
  .NetworkID = ".NetworkID",
  .NetworkName = ".NetworkName"
)
```
Arguments

- **x**: a `combined_networks` (inheriting from `network::network`).
- **attrnames**: a list (or a selection index) for attributes to obtain; for combined networks, defaults to all.
- **...**: additional arguments, currently passed to `unlist()`.
- **unit**: whether to obtain edge, vertex, or network attributes.
- **.NetworkID, .NetworkName**: Optional strings indicating the vertex attributes used to distinguish and name the networks; intended to be used by term developers.

See Also

- `network::as_tibble.network()`

---

**b1degreeL-ergmTerm**  
*Degree for the first mode in a bipartite (aka two-mode) network*

Description

This term adds one network statistic to the model for each element in `d`; the `i`th such statistic equals the number of nodes of degree `d[i]` in the first mode of a bipartite network, i.e. with exactly `d[i]` edges. The first mode of a bipartite network object is sometimes known as the "actor" mode.

This term can only be used with undirected bipartite networks.

Usage

```
# binary: b1degreeL(d, by=NULL, levels=NULL, Ls=NULL)
```

Arguments

- **d**: a vector of distinct integers.
- **by**: a character string giving the name of an attribute in the network’s vertex attribute list. If this is specified then each node’s degree is tabulated only with other nodes having the same value of the by attribute.
- **levels**: if by is specified, which levels to consider.
- **Ls**: either a Layer Logic specification formula (c.f. Layer Logic section in the `Layer()` documentation) or a list thereof (constructed by `list()` or `c()`). If given, degree of a node

See Also

- `ergmTerm` for index of model terms currently visible to the package.

**Keywords**: bipartite, categorical nodal attribute, frequently-used, undirected, binary
### b2degree-ergmTerm

*Degree for the second mode in a bipartite (aka two-mode) network*

**Description**

This term adds one network statistic to the model for each element in \( d \); the \( i \)th such statistic equals the number of nodes of degree \( d[i] \) in the second mode of a bipartite network, i.e. with exactly \( d[i] \) edges. The second mode of a bipartite network object is sometimes known as the "event" mode. The optional term by is a character string giving the name of an attribute in the network’s vertex attribute list. This term can only be used with undirected bipartite networks.

**Usage**

```r
# binary: b2degree(d, by=NULL)
```

**Arguments**

- `d`: a vector of distinct integers
- `by`: a character string giving the name of an attribute in the network’s vertex attribute list. If this is specified then each node’s degree is tabulated only with other nodes having the same value of the by attribute.
- `levels`: if by is specified, which levels to consider.
- `Ls`: either a Layer Logic specification formula (c.f. Layer Logic section in the `Layer()` documentation) or a list thereof (constructed by `list()` or `c()`). If given, degree of a node

**See Also**

`ergmTerm` for index of model terms currently visible to the package.

**Keywords:** bipartite, categorical nodal attribute, frequently-used, undirected, binary

### CMBL-ergmTerm

*Conway–Maxwell-Binomial dependence among layers*

**Description**

Models marginal dependence layers within each dyad by imposing a Conway–Maxwell-Binomial (CMB) distribution on the number of layers in each dyad that have a tie.

The term adds one statistic to the model, equalling the sum over all the dyads in the network of \( \log\{E!(R-E)!/R!\} \), where \( E \) is the number of layers in \( Ls \) with an edge in that dyad and \( R \) being the total number of layers in \( Ls \).
**Usage**

# binary: CMBL(Ls=~.)

**Arguments**

Ls a list (constructed by `list()` or `c()` of at least two Layer Logic specifications (c.f. Layer Logic section in the `Layer()` documentation).

**Details**

A positive coefficient induces positive dependence and a negative one induces negative dependence.

**See Also**

`ergmTerm` for index of model terms currently visible to the package.

**Keywords:** directed, layer-aware, undirected, binary

**Description**

Given a list of compatible networks, the `combine_networks()` returns a single block-diagonal network, preserving attributes that can be preserved.

**Usage**

```r
combine_networks(
  nw1,
  ignore.nattr = c("mnext"),
  ignore.vattr = c(),
  ignore.eattr = c(),
  blockID.vattr = ".NetworkID",
  blockName.vattr = NULL,
  detect.edgecov = FALSE,
  keep.unshared.attr = FALSE,
  subnet.cache = FALSE
)
```

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

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

```r
## S3 method for class 'summary.combined_networks'
print(x, ...)
```
combine_networks

Arguments

nw1: a list of network::networks to be combined; they must have similar fundamental properties: directedness and bipartedness, though their sizes (and the size of each bipartite group) can vary.

ignore.nattr, ignore.vattr, ignore.eattr: network, vertex, and edge attributes not to be processed as described below.

blockID.vattr: name of the vertex attribute into which to store the index of the network to which that vertex originally belonged.

blockName.vattr: if not NULL, the name of the vertex attribute into which to store the name of the network to which that vertex originally belonged.

detect.edgecov: if TRUE, combine network attributes that look like dyadic covariate (ergm::edgecov) matrices into a block-diagonal matrix.

keep.unshared.attr: whether to keep those network, vertex, and edge attributes not shared by all networks in the list; if TRUE, positions corresponding to networks lacking the attribute are replaced with NA, NULL, or some other placeholder; incompatible with detect.edgecov==TRUE.

subnet.cache: whether to save the input network list as an attribute of the combined network, so that if the network is resplit using on the same attribute (e.g. using uncombine_network()), an expensive call to split.network() can be avoided, at the cost of storage.

x, object: a combined network.

...: additional arguments to methods.

Value

an object of class combined_networks inheriting from network::network with a block-diagonal structure (or its bipartite equivalent) comprising the networks passed in nw1. In particular,

• the returned network’s size is the sum of the input networks’;
• its basic properties (directedness and bipartedness) are the same;
• the input networks’ sociomatrices (both edge presence and edge attributes) are the blocks in the sociomatrix of the returned network;
• vertex attributes are concatenated;
• edge attributes are assigned to their respective edges in the returned network;
• network attributes are stored in a list; but if detect.edgecov==TRUE, those network attributes that have the same dimension as the sociomatrices of the constituent networks, they are combined into a single block-diagonal matrix that is then stored as that attribute.

In addition, two new vertex attributes, specified by blockID.vattr and (optionally) blockName.vattr contain, respectively, the index in nw1 of the network from which that vertex came and its name, determined as follows:

1. If nw1 is a named list, names from the list are used.
2. If not 1, but the network has an attribute `title`, it is used.

3. Otherwise, a numerical index is used.

If `blockID.vattr` already exists on the constituent networks, the index is *prepended* to the attribute.

The values of `blockID.vattr` and `blockName.vattr` are stored in network attributes ".blockID.vattr" and ".blockName.vattr".

Functions

- `print(combined_networks)`: A wrapper around `network::print.network()` to print constituent network information and omit some internal variables.
- `summary(combined_networks)`: A wrapper around `network::summary.network()` to print constituent network information and omit some internal variables.
- `print(summary.combined_networks)`: A wrapper around `network::print.summary.network()` to print constituent network information and omit some internal variables.

Examples

```r
data(samp1k)
o1 <- combine_networks(list(samp1k1, samp1k2, samp1k3))
image(as.matrix(o1))
head(get.vertex.attribute(o1, ".NetworkID"))
o2 <- combine_networks(list(o1, o1))
image(as.matrix(o2))
head(get.vertex.attribute(o2, ".NetworkID", unlist=FALSE))

data(florentine)
f1 <- combine_networks(list(business=flobusiness, marriage=flomarriage),
                        blockName.vattr=\.NetworkName")
image(as.matrix(f1))
head(get.vertex.attribute(f1, ".NetworkID"))
head(get.vertex.attribute(f1, ".NetworkName"))
```

control.gofN.ergm

**Auxiliary for Controlling Multinetwork ERGM Linear Goodness-of-Fit Evaluation**

**Description**

`control.gofN.ergm` (or its alias, `control.gofN`) is intended to be used with `gofN()` specifically and will "inherit" as many control parameters from `ergm` fit as possible.
Usage

```r
control.gofN.ergm(
  nsim = 100,
  obs.twostage = nsim/2,
  array.max = 128,
  simulate = control.simulate.ergm(),
  obs.simulate = control.simulate.ergm(),
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)
```

```r
control.gofN(
  nsim = 100,
  obs.twostage = nsim/2,
  array.max = 128,
  simulate = control.simulate.ergm(),
  obs.simulate = control.simulate.ergm(),
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)
```

Arguments

- **nsim**
  - Number of networks to be randomly drawn using Markov chain Monte Carlo. This sample of networks provides the basis for comparing the model to the observed network.

- **obs.twostage**
  - Either FALSE or an integer. This parameter only has an effect if the network has missing data or observational process. For such networks, evaluating the Pearson residual requires simulating the expected value of the conditional variance under the observation process. If FALSE, the simulation is performed conditional on the observed network. However, a more accurate estimate can be obtained via a two-stage process:

  1. Sample networks from the model without the observational constraint.
  2. Conditional on each of those networks, sample with the observational constraint, estimating the variance within each sample and then averaging over the first-stage sample.

  Then, obs.twostage specifies the number of unconstrained networks to simulate from, which should divide the `control.gofN.ergm()`’s nsim argument evenly.

- **array.max**
  - Try to avoid creating arrays larger in size (in megabytes) than this. Is ignored if save_stats is passed.
simulate, obs.simulate
Control lists produced by control.simulate.ergm() or equivalent for unconstrained and constrained simulation, respectively. Parameters are inherited from the model fit and can be overridden here.

parallel
Number of threads in which to run the sampling. Defaults to 0 (no parallelism). See the entry on parallel processing for details and troubleshooting.

parallel.type
API to use for parallel processing. Supported values are "MPI" and "PSOCK". Defaults to using the parallel package with PSOCK clusters. See ergm-parallel

parallel.version.check
Logical: If TRUE, check that the version of ergm running on the slave nodes is the same as that running on the master node.

parallel.inherit.MT
Logical: If TRUE, slave nodes and processes inherit the set.MT_terms() setting.

Details
Auxiliary function as user interface for fine-tuning ERGM Goodness-of-Fit Evaluation.

== dyadwise shared partners on layers (ddspL) ==

Description
This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of dyads in the network with exactly d[i] shared partners. For a directed network, multiple shared partner definitions are possible. dspl and dspl are aliases for consistency with ergm.

Usage
# binary: ddspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)
# binary: dspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)

Arguments
Ls.path, L.in_order
a vector of one or two formulas Ls.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order=TRUE, the first tie of the two-path must be the first element of Ls.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)
Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the type argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the relevant package):

- **Outgoing Two-path ("OTP")**: vertex $k$ is an OTP shared partner of ordered pair $(i, j)$ iff $i \rightarrow k \rightarrow j$. Also known as "transitive shared partner".
- **Incoming Two-path ("ITP")**: vertex $k$ is an ITP shared partner of ordered pair $(i, j)$ iff $j \rightarrow k \rightarrow i$. Also known as "cyclical shared partner"
- **Reciprocated Two-path ("RTP")**: vertex $k$ is an RTP shared partner of ordered pair $(i, j)$ iff $i \leftrightarrow k \leftrightarrow j$.
- **Outgoing Shared Partner ("OSP")**: vertex $k$ is an OSP shared partner of ordered pair $(i, j)$ iff $i \rightarrow k, j \rightarrow k$.
- **Incoming Shared Partner ("ISP")**: vertex $k$ is an ISP shared partner of ordered pair $(i, j)$ iff $k \rightarrow i, k \rightarrow j$. By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

**Note**

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

**See Also**

ergmTerm for index of model terms currently visible to the package.

**Keywords:** directed, layer-aware, undirected, binary

---

**Description**

This term adds one network statistic to the model for each element in $d$; the $i$th such statistic equals the number of nodes in the network of degree $d[i]$, i.e. with exactly $d[i]$ edges.

This term can only be used with undirected networks; for directed networks see idegree and odegree.

**Usage**

# binary: degreeL(d, by=NULL, homophily=FALSE, levels=NULL, Ls=NULL)
Arguments

d  a vector of distinct integers
by  a character string giving the name of an attribute in the network’s vertex attribute list.
homophily  If this is specified and homophily is TRUE, then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree statistics are calculated for nodes having each separate value of the attribute.
Ls  a list (constructed by list() or c()) of one or more Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation). If specified, degree of a node i is considered to be the number of edges in all layers, combined.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords:  categorical nodal attribute, directed, frequently-used, undirected, binary

---

despL-ergmTerm  Edgewise shared partners on layers

Description

This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of edges in the network with exactly d[i] shared partners. For a directed network, multiple shared partner definitions are possible.

espl and despL are aliases for consistency with ergm.

Usage

# binary: despL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)
# binary: espL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)

Arguments

d  a vector of distinct integers

A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.

L.base  the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specification for the base
\textbf{Ls.path, L.in_order}

a vector of one or two formulas \texttt{L.s.path} provides the Layer Logic (c.f. Layer Logic section in the \texttt{Layer()} documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If \texttt{L.in.order==TRUE}, the first tie of the two-path must be the first element of \texttt{L.s.path} and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)

\textbf{Shared partner types}

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the \texttt{type} argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the \texttt{relevent} package):

- Outgoing Two-path ("OTP"): vertex \(k\) is an OTP shared partner of ordered pair \((i, j)\) iff \(i \rightarrow k \rightarrow j\). Also known as "transitive shared partner".
- Incoming Two-path ("ITP"): vertex \(k\) is an ITP shared partner of ordered pair \((i, j)\) iff \(j \rightarrow k \rightarrow i\). Also known as "cyclical shared partner"
- Reciprocated Two-path ("RTP"): vertex \(k\) is an RTP shared partner of ordered pair \((i, j)\) iff \(i \leftrightarrow k \leftrightarrow j\).
- Outgoing Shared Partner ("OSP"): vertex \(k\) is an OSP shared partner of ordered pair \((i, j)\) iff \(i \rightarrow k, j \rightarrow k\).
- Incoming Shared Partner ("ISP"): vertex \(k\) is an ISP shared partner of ordered pair \((i, j)\) iff \(k \rightarrow i, k \rightarrow j\). By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

\textbf{Note}

This term takes an additional term option (see \texttt{options?ergm}), \texttt{cache.sp}, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

\textbf{See Also}

\texttt{ergmTerm} for index of model terms currently visible to the package.

\textbf{Keywords:} directed, layer-aware, undirected, binary
**dgwdspL-ergmTerm**

*Geometrically weighted dyadwise shared partner distribution on layers*

**Description**

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter. Note that the GWDSP statistic is equal to the sum of GWNSP plus GWESP. For a directed network, multiple shared partner definitions are possible. `gdwdspL` and `dgwdspL` are aliases for consistency with `ergm`.

**Usage**

```
# binary: dgwdspL(decay, fixed=FALSE, cutoff=30, type="OTP",
#  Ls.path=NULL, L.in_order=FALSE)
# binary: gwdspL(decay, fixed=FALSE, cutoff=30, type="OTP",
#  Ls.path=NULL, L.in_order=FALSE)
```

**Arguments**

- **decay**
  - nonnegative decay parameter for the shared partner or selected directed analogue count; required if `fixed=TRUE` and ignored with a warning otherwise.

- **fixed**
  - optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a curved exponential family.

- **cutoff**
  - This optional argument sets the number of underlying DSP terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See `?control.ergm`.)

- **type**
  - A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.

- **Ls.path, L.in_order**
  - a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE`, the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)
Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the type argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the relevant package):

- **Outgoing Two-path ("OTP")**: vertex \( k \) is an OTP shared partner of ordered pair \((i, j)\) iff \( i \to k \to j \). Also known as "transitive shared partner".

- **Incoming Two-path ("ITP")**: vertex \( k \) is an ITP shared partner of ordered pair \((i, j)\) iff \( j \to k \to i \). Also known as "cyclical shared partner"

- **Reciprocated Two-path ("RTP")**: vertex \( k \) is an RTP shared partner of ordered pair \((i, j)\) iff \( i \leftrightarrow k \leftrightarrow j \).

- **Outgoing Shared Partner ("OSP")**: vertex \( k \) is an OSP shared partner of ordered pair \((i, j)\) iff \( i \to k, j \to k \).

- **Incoming Shared Partner ("ISP")**: vertex \( k \) is an ISP shared partner of ordered pair \((i, j)\) iff \( k \to i, k \to j \). By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

**Note**

This term takes an additional term option (see `options?ergm`), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

**See Also**

`ergmTerm` for index of model terms currently visible to the package.

**Keywords**: directed, layer-aware, undirected, binary
Usuage

```r
# binary: dgwespL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#   L.s.path=NULL, L.in_order=FALSE)

# binary: gwespL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#   L.s.path=NULL, L.in_order=FALSE)
```

Arguments

- **decay**: nonnegative decay parameter for the shared partner or selected directed analogue count; required if `fixed=TRUE` and ignored with a warning otherwise.
- **fixed**: optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a curved exponential family.
- **cutoff**: This optional argument sets the number of underlying ESP terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See ?control.ergm.)
- **type**: A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.
- **L.base**: the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specification for the base
- **L.s.path, L.in_order**: a vector of one or two formulas. `L.s.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE`, the first tie of the two-path must be the first element of `L.s.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- **Outgoing Two-path** ("OTP"): vertex `k` is an OTP shared partner of ordered pair `(i, j)` iff `i \rightarrow k \rightarrow j`. Also known as "transitive shared partner".
- **Incoming Two-path** ("ITP"): vertex `k` is an ITP shared partner of ordered pair `(i, j)` iff `j \rightarrow k \rightarrow i`. Also known as "cyclical shared partner"
• Reciprocated Two-path ("RTP"): vertex $k$ is an RTP shared partner of ordered pair $(i, j)$ iff $i \leftrightarrow k \leftrightarrow j$.

• Outgoing Shared Partner ("OSP"): vertex $k$ is an OSP shared partner of ordered pair $(i, j)$ iff $i \rightarrow k, j \rightarrow k$.

• Incoming Shared Partner ("ISP"): vertex $k$ is an ISP shared partner of ordered pair $(i, j)$ iff $k \rightarrow i, k \rightarrow j$. By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

**Note**

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

**See Also**

ergmTerm for index of model terms currently visible to the package.

**Keywords:** directed, layer-aware, undirected, binary

---

**dgwnspL-ergmTerm**

*Geometrically weighted non-edgewise shared partner distribution on layers*

**Description**

This term is just like gwespL and gwdspL except it adds a statistic equal to the geometrically weighted nonedgewise (that is, over dyads that do not have an edge) shared partner distribution with decay parameter. For a directed network, multiple shared partner definitions are possible. gdwspL and dgwnspL are aliases for consistency with ergm.

**Usage**

```r
# binary: dgwnspL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
# L.path=NULL, L.in_order=FALSE)

# binary: gwespL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
# L.path=NULL, L.in_order=FALSE)
```

**Arguments**

- **decay** nonnegative decay parameter for the shared partner or selected directed analogue count; required if fixed=TRUE and ignored with a warning otherwise.
fixed

This optional argument indicates whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is FALSE, which means the scale parameter is not fixed and thus the model is a curved exponential family.

cutoff

This optional argument sets the number of underlying NSP terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)

type

A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.

L.base

the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specification for the base

Ls.path, L.in_order

a vector of one or two formulas Ls.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order==TRUE, the first tie of the two-path must be the first element of Ls.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the type argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the relevant package):

- Outgoing Two-path ("OTP"): vertex k is an OTP shared partner of ordered pair (i, j) iff i → k → j. Also known as "transitive shared partner".
- Incoming Two-path ("ITP"): vertex k is an ITP shared partner of ordered pair (i, j) iff j → k → i. Also known as "cyclical shared partner"
- Reciprocated Two-path ("RTP"): vertex k is an RTP shared partner of ordered pair (i, j) iff i ↔ k ↔ j.
- Outgoing Shared Partner ("OSP"): vertex k is an OSP shared partner of ordered pair (i, j) iff i → k, j → k.
- Incoming Shared Partner ("ISP"): vertex k is an ISP shared partner of ordered pair (i, j) iff k → i, k → j. By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is
usually enabled by default.

See Also

\texttt{ergmTerm} for index of model terms currently visible to the package.

\textbf{Keywords:} directed, layer-aware, undirected, binary

---

\texttt{direct.network} \hspace{1cm} \textit{Returns a directed version of an undirected binary network}

\textbf{Description}

Returns a directed version of an undirected binary network

\textbf{Usage}

\begin{verbatim}
direct.network(x, rule = c("both", "upper", "lower"))
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
\item \texttt{x} a \texttt{network} object.
\item \texttt{rule} a string specifying how the network is to be constructed.
\end{itemize}

---

\texttt{dnspL-ergmTerm} \hspace{1cm} \textit{Non-edgewise shared partners and paths on layers}

\textbf{Description}

This term adds one network statistic to the model for each element in \texttt{d} where the \texttt{ith} such statistic equals the number of non-edges in the network with exactly \texttt{d[i]} shared partners. For a directed network, multiple shared partner definitions are possible. \texttt{nspL} and \texttt{dnspL} are aliases for consistency with \texttt{ergm}.

\textbf{Usage}

\begin{verbatim}
# binary: dnspL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)
# binary: nspL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)
\end{verbatim}
Arguments

da vector of distinct integers

type A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.

L.base the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specification for the base

L.s.path, L.in_order a vector of one or two formulas L.s.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order==TRUE, the first tie of the two-path must be the first element of L.s.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP", the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the type argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the relevent package):

- Outgoing Two-path ("OTP"): vertex $k$ is an OTP shared partner of ordered pair $(i, j)$ iff $i \rightarrow k \rightarrow j$. Also known as "transitive shared partner".
- Incoming Two-path ("ITP"): vertex $k$ is an ITP shared partner of ordered pair $(i, j)$ iff $j \rightarrow k \rightarrow i$. Also known as "cyclical shared partner"
- Reciprocated Two-path ("RTP"): vertex $k$ is an RTP shared partner of ordered pair $(i, j)$ iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner ("OSP"): vertex $k$ is an OSP shared partner of ordered pair $(i, j)$ iff $i \rightarrow k, j \rightarrow k$.
- Incoming Shared Partner ("ISP"): vertex $k$ is an ISP shared partner of ordered pair $(i, j)$ iff $k \rightarrow i, k \rightarrow j$. By default, outgoing two-paths ("OTP") are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

ergmTerm for index of model terms currently visible to the package.
Keywords: directed, layer-aware, undirected, binary

**Description**

This is a list of 318 network objects derived from contact diary data collected by Goeyvaerts et al. (2018). The study recruited households in Flanders and Brussels-Capital region with at least one child 12 or under. The networks are symmetrized.

**Usage**

data(Goeyvaerts)

**Format**

An object of class list of length 318.

**Nonstandard Network Attributes**

- included (logical) whether the network was included in Goeyvaerts’s analysis. (Two were excluded.)
- weekday (logical) whether the contact diary on which the network is based was collected on a weekday, as opposed to weekend.

**Nonstandard Vertex Attributes**

- age (numeric) the household member’s age.
- gender (character) the household member’s gender ("F"/"M").
- role (character) the household member’s inferred role ("Father"/"Mother"/"Child"/"Grandmother").

**Licenses and Citation**

When publishing results obtained using this data set, the original authors (Goeyvaerts et al. 2018) should be cited, along with this R package.

**Source**

The data were collected and by Goeyvaerts et al. (2018) and curated by Pietro Coletti.

**References**

gofN

See Also
vignette("Goeyvaerts_reproduction") for a vignette reproducing the Goeyvaerts analysis and performing diagnostics

gofN
Linear model diagnostics for multinetwork linear models

Description
gofN() performs a simulation to obtain Pearson residuals for the multivariate linear model for ERGM parameters, which can then be used for a variety of diagnostics and diagnostic plots developed by Krivitsky et al. (2022).

Usage
gofN(
object,
GOF = NULL,
subset = TRUE,
control = control.gofN.ergm(),
save_stats = FALSE,
...
)

## S3 method for class 'gofN'
x[i, j, ..., drop = FALSE]

## S3 method for class 'gofN'
augment(x, ...)

## S3 method for class 'gofN'
summary(object, by = NULL, ...)

Arguments

object an `ergm` object.
GOF a one-sided `ergm` formula specifying network statistics whose goodness of fit to test, or `NULL`; if `NULL`, uses the original model.
subset argument for the `N` term.
control See `control.gofN.ergm()`.
save_stats If `TRUE`, save the simulated network statistics; defaults to `FALSE` to save memory and disk space.
... additional arguments to functions (`simulate.ergm()` and `summary.ergm_model()`) for the constructor.
x a `gofN` object.
for the indexing operator, index of statistics to be kept in the subset.

for the indexing operator, index of networks to be kept in the subset.

whether the indexing operator should drop attributes and return simply a list.

a numeric or character vector, or a formula whose RHS gives an expression in terms of network attributes, used as a grouping variable for summarizing the values.

Value

An object of class `gofN`: a named list containing a list for every statistic in the specified GOF formula with the following elements vectors of length equal to the number of subnetworks:

- **observed**: For completely observed networks, their value of the statistic. For partially observed networks, the expected value of their imputations under the model.
- **fitted**: Expected value of the statistic under the model.
- **var**: Variance of the statistic under the model.
- **var.obs**: Conditional variance under imputation statistic.
- **pearson**: The Pearson residual computed from the above.
- **stats,stats.obs**: If `save_stats` control parameter is `TRUE`, the simulated statistics.

In addition, the following `attr`-style attributes are included:

- **nw**: The observed multinetwork object.
- **subset**: A logical vector giving the subset of networks that were used.
- **control**: Control parameters passed.

Methods (by generic)

- `[`: Extract a subset of statistics for which goodness-of-fit had been computed.
- `augment(gofN)`: a method for constructing a `tibble` of network attributes augmented with goodness of fit information. Columns include:

  - **network attributes**: the attributes of each of the networks
    - `.stat_name`: name of the simulated statistic
    - `.stat_id`: index of the simulated statistic in the `gofN` object
    - `.network_id`: index of the network in the networks for which `gofN` was run (excluding those not in the subset)
    - `.fitted`: predicted value for the statistic
    - `.observed`: either the observed (for completely observed networks) or the predicted conditional on observed (for partially observed networks) value of the statistic
    - `.pearson`: the standardised Pearson residual
    - `.var,.var.obs`: estimated unconditional and average conditional variance of the statistic
    - `.weight`: inverse of the variance of the residual
- `summary(gofN)`: A simple summary function.
gofN

References

See Also
plot.gofN() and autoplot.gofN() for plotting gofN objects to make residual plots; ergm::gof() for single-network goodness-of-fit simulations in ergm

Examples
data(samplk)
monks <- Networks(samplk1, samplk2, samplk3, samplk1, samplk2, samplk3, samplk1, samplk2, samplk3)
fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model
summary(fit.gof)
plot(fit.gof)
fit.gof <- gofN(fit, GOF=~triangles)
summary(fit.gof)
plot(fit.gof)

samplk1[,1]<-NA
samplk2[,2]<-NA
monks <- Networks(samplk1, samplk2, samplk3, samplk1, samplk2, samplk3, samplk1, samplk2, samplk3)
fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model
summary(fit.gof)
plot(fit.gof)
fit.gof <- gofN(fit, GOF=~triangles)
summary(fit.gof)
plot(fit.gof)
plot(fit.gof, against=~log(.fitted)) # Plot against transformed fitted values.

### If 'ggplot2' and 'ggrepel' are installed, illustrate the autoplot() method.
if(require("ggplot2") & & requireNamespace("ggrepel")){
  autoplot(fit.gof)
}

# Default is good enough in this case, but sometimes, we might want to set it higher. E.g.,
## Not run:
fit.gof <- gofN(fit, GOF=~edges, control=control.gofN.ergm(nsim=400))

## End(Not run)

### If 'generics' is installed, illustrate the augment() method.
if(require("generics")){
augment(fit.gof)
}
**gwb1degreeL-ergmTerm**  
*Geometrically weighted degree distribution for the first mode in a bi-partite (aka two-mode) network*

### Description

This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the decay parameter, which should be non-negative, for nodes in the first mode of a bipartite network. The first mode of a bipartite network object is sometimes known as the "actor" mode.

This term can only be used with undirected bipartite networks.

### Usage

```r
# binary: gwb1degreeL(decay, fixed=FALSE, cutoff=30, levels=NULL, Ls=NULL)
```

### Arguments

- **decay**: non-negative model parameter that is the same as theta_s in equation (14) in Hunter (2007).
- **fixed**: specify if the value supplied for decay may be fixed (if fixed=TRUE), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default).
- **attrname**: if specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
- **cutoff**: only relevant if fixed=FALSE. In that case it only uses this number of terms in computing the statistics to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See control.ergm.)
- **levels**: a list of layer specifications. If given, degree of a node i is considered to be the number of edges in all layers, combined.
- **Ls**: a list (constructed by list() or c() of one or more Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation).

### See Also

- **ergmTerm** for index of model terms currently visible to the package.

### Keywords:

- bipartite, curved, undirected, binary
**gwb2degreeL-ergmTerm**  
*Geometrically weighted degree distribution for the second mode in a bipartite (aka two-mode) network*

---

**Description**

This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the which should be non-negative, for nodes in the second mode of a bipartite network. The second mode of a bipartite network object is sometimes known as the "event" mode.

This term can only be used with undirected bipartite networks.

**Usage**

```r
# binary: gwb2degreeL(decay, fixed=FALSE, attrname=NULL, cutoff=30, levels=NULL, Ls=NULL)
```

**Arguments**

- **decay**: non-negative model parameter that is the same as theta_s in equation (14) in Hunter (2007).
- **fixed**: specify if the value supplied for decay may be fixed (if fixed=TRUE ), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default).
- **attrname**: if specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
- **cutoff**: only relevant if fixed=FALSE . In that case it only uses this number of terms in computing the statistics to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See control.ergm.)
- **levels**: a list of layer specifications. If given, degree of a node i is considered to be the number of edges in all layers, combined.
- **Ls**: a list (constructed by list() or c() of one or more Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation).

**See Also**

- `ergmTerm` for index of model terms currently visible to the package.

**Keywords:** bipartite, curved, undirected, binary
**gwdegreeL-ergmTerm**  
*Geometrically weighted degree distribution*

**Description**

This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the decay parameter.

This term can only be used with undirected networks.

**Usage**

```r
# binary: gwdegreeL(decay, fixed=FALSE, attrname=NULL, cutoff=30, levels=NULL)
```

**Arguments**

- **decay**
  - non-negative model parameter that is the same as theta_s in equation (14) in Hunter (2007).

- **fixed**
  - specify if the value supplied for decay may be fixed (if fixed=TRUE ), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default).

- **attrname**
  - if specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.

- **cutoff**
  - only relevant if fixed=FALSE . In that case it only uses this number of terms in computing the statistics to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See control.ergm.)

- **levels**
  - a list of layer specifications. If given, degree of a node i is considered to be the number of edges in all layers, combined.

**See Also**

ergmTerm for index of model terms currently visible to the package.

**Keywords:** curved, frequently-used, undirected, binary

---

**gwidegreeL-ergmTerm**  
*Geometrically weighted in-degree distribution*

**Description**

This term adds one network statistic to the model equal to the weighted in-degree distribution with decay parameter. This term can only be used with directed networks.
**Usage**

```r
# binary: gwodegreeL(decay, fixed=FALSE, attrname=NULL, cutoff=30, levels=NULL, Ls=NULL)
```

**Arguments**

- `decay`: non-negative model parameter that is the same as theta_s in equation (14) in Hunter (2007).
- `fixed`: specify if the value supplied for `decay` may be fixed (if `fixed=TRUE`), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default).
- `attrname`: if specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
- `cutoff`: only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See `control.ergm`.)
- `levels`: a list of layer specifications. If given, degree of a node i is considered to be the number of edges in all layers, combined.

**See Also**

- `ergmTerm` for index of model terms currently visible to the package.

**Keywords:** curved, directed, binary

---

**gwodegreeL-ergmTerm**  
*Geometrically weighted out-degree distribution*

This term adds one network statistic to the model equal to the weighted out-degree distribution with decay parameter. This term can only be used with directed networks.

**Usage**

```r
# binary: gwodegreeL(decay, fixed=FALSE, attrname=NULL, cutoff=30, levels=NULL, Ls=NULL)
```

**Arguments**

- `decay`: non-negative model parameter that is the same as theta_s in equation (14) in Hunter (2007).
- `fixed`: specify if the value supplied for `decay` may be fixed (if `fixed=TRUE`), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default).
- `attrname`: if specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
This term adds one network statistic to the model for each element in \( d \); the \( i \) th such statistic equals the number of nodes in the network of in-degree \( d[i] \), i.e. the number of nodes with exactly \( d[i] \) in-edges. This term can only be used with directed networks; for undirected networks see `degree`.

**Arguments**

- `d` a vector of distinct integers.
- `by` an optional character string giving the name of an attribute in the network's vertex attribute list.
- `homophily` only applied if by is specified. If set (homophile == `TRUE`), then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. Otherwise (the default), then separate degree statistics are calculated for nodes having each separate value of the attribute.
- `levels` if by is specified, which levels to consider.
- `Ls` either a Layer Logic specification formula (c.f. Layer Logic section in the `Layer()` documentation) or a list thereof (constructed by `list()` or `c()`). If given, degree of a node in each layer is considered.

**See Also**

- `ergmTerm` for index of model terms currently visible to the package.

**Keywords:** curved, directed, binary
## L-ergmTerm

### Evaluation on layers

**Description**

Evaluates the terms in `formula` on an observed or logical layers specified in `formula Ls` and sums the results elementwise.

**Usage**

```
# binary: L(formula, Ls=~.)
```

**Arguments**

- `formula`: a one-sided `ergm()`-style formula with the terms to be evaluated
- `Ls`: either a Layer Logic specification formula (c.f. Layer Logic section in the `Layer()` documentation) or a list thereof (constructed by `list()` or `c()`), on which to evaluate `formula`

**See Also**

- `ergmTerm` for index of model terms currently visible to the package.

**Keywords**: layer-aware, operator, binary

## Layer

### A multilayer network representation.

**Description**

A function for specifying the LHS of a multilayer (a.k.a. multiplex, a.k.a. multirelational, a.k.a. multivariate) ERGM in the framework of Krivitsky et al. (2020).

**Usage**

```
Layer(..., .symmetric = NULL, .bipartite = NULL, .active = NULL)
```

**Arguments**

- `...`: layer specification, in one of three formats:
  1. An (optionally named) list of identically-dimensioned networks.
  2. Several networks as (optionally named) arguments.
3. A single network, a character vector, and several optional arguments. Then, the layers are values of the named edge attributes. If the vector has named elements (e.g., c(a="advice", c="collaboration")), the layers will be renamed accordingly. The optional arguments .symmetric and .bipartite are then interpreted as described below.

- **.symmetric**: If the layer specification is via a single network with edge attributes and the network is directed, an optional logical vector to specify which of the layers should be treated as undirected.

- **.bipartite**: If the layer specification is via a single network with edge attributes and the network is unipartite, an optional integer vector to specify which of the layers should be treated as bipartite and how many b1 vertices there are.

- **.active**: A nodal attribute specification (? nodal_attributes) specifying which nodes on each network may have ties, or a list with an element for each network. The list will be recycled up to the number of layers.

**Value**

A network object with layer metadata.

**Specifying models for multilayer network**

In order to fit a model for multilayer networks, first use Layer construct an LHS network that ergm() will understand as multilayered.

Used in the formula directly, most, but not all, ergm terms will sum their statistics over the observed layers.

Some terms are layer-aware, however. By convention, layer-aware terms have capital L appended to them. For example, mutualL is a layer-aware generalization of mutual. These terms have one or more explicit (usually optional) layer specification arguments. By convention, an argument that requires one layer specification is named L= and one that requires a list of specifications (constructed by list() or c()) is named Ls=; and a specification of the form ~. is a placeholder for all observed layers.

Operator L(formula, Ls=...) can be used to evaluate arbitrary terms in the formula on specified layers.

Layer specification documentation follows.

**Layer Logic:**

Each formula’s right-hand side describes an observed layer or some "logical" layer, whose ties are a function of corresponding ties in observed layers. (Krivitsky et al. 2020)

The observed layers can be referenced either by name or by number (i.e., order in which they were passed to Layer). When referencing by number, enclose the number in quotation marks (e.g., "1") or backticks (e.g., "1").

Arithmetical, relational, and logical operators can be used to combine them. All listed operators are implemented, as well as functions abs, round, and sign. Standard operator precedence applies, so use of parentheses is recommended to ensure the logical expression is what it looks like.

**Important:** For performance reasons, ergm.multi’s Layer Logic implementation uses integer arithmetic. This means, in particular, that / will round down instead of returning a fraction (as
%% does in R, and round() function without a second argument (which can be negative to round to the nearest 10, 100, etc.) is not meaningful and will be ignored.
For example, if LHS is Layer(A=nwA, B=nwB), both ~`2` and ~B refer to nwB, while A&B refers to a “logical” layer that has ties that are in nwA but not in nwB.
Transpose function t applied to a directed layer will reverse the direction of all relations (transposing the sociomatrix). Unlike the others, it can only be used on an observed layer directly. For example, ~t(\`1`)&t(\`2`) is valid but ~t(\`1`&\`2`) is not.
At this time, logical expressions that produce complete graphs from empty graph inputs (e.g., A==B or !A) are not supported.

Summing layers:
Some of the terms that call for a list of layers (i.e., have Ls= arguments) will sum the statistic over the layers. For example, Layer(nw1,nw2)\sim\text{L}(\sim\text{edges}, c(\sim`1`,`\sim`2`\&!`1`)) produces the number of edges in layer 1 plus the number of edges in layer 2 but not in layer 1.
For these formulas, one can specify the layer’s weight on its left-handside. For example, Layer(nw1,nw2)\sim\text{L}(\sim\text{edges}, c(3\sim`1`,-1\sim`2`\&!`1`)) will produce three times the number of edges in layer 1, minus the number of edges in layer 2 but not in layer 1.

Note
The resulting network will be the "least common denominator" network: if not all layers have the same bipartedness, all layers will appear as unipartite to the statistics, and if any are directed, all will be. However, certain operator terms, particularly Symmetrize() and S(), can be used to construct a bipartite subgraph of a unipartite graph or change directedness.

References

See Also
Help on model specification for specific terms.

Examples
data(florentine)

# Method 1: list of networks
flo <- Layer(list(m = flomarriage, b = flobusiness))
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

# Method 2: networks as arguments
flo <- Layer(m = flomarriage, b = flobusiness)
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

# Method 3: edge attributes (also illustrating renaming):
flo <- flomarriage | flobusiness
flo[, names.eval="marriage"] <- as.matrix(flomarriage)
flo[, names.eval="business"] <- as.matrix(flobusiness)
Layer

flo # edge attributes
flo <- Layer(flo, c(m="marriage", b="business"))
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

### Specifying modes and mixed bipartitedness

# Suppose we have a two-mode network with 5 nodes on Mode 1 and 15
# on Mode 2, and suppose that we observe two layers, one only among
# actors of Mode 1 and the other bipartite between Modes 1 and 2.

# Construct the two layers' networks:
nw1 <- network.initialize(20, dir=FALSE)
nw12 <- network.initialize(20, dir=FALSE, bipartite=5)
nw1 %v% "mode" <- rep(1:2,c(5,15))

# For testing: the maximal set of edges for each type of network:
  nw1[1:5,1:5] <- 1
  nw12[1:5,6:20] <- 1

# The .active argument specifies the following:
# * nw1's vertices are only active if their mode=1 (i.e., 1-2, 2-1,
# and 2-2 can't have edges).
# * nw12's vertices are all active, but the network is bipartite,
# so constraints will be adjusted automatically.
lnw <- Layer(nw1, nw12, .active=list(~mode==1, ~TRUE))

summary(lnw~edges+ # 5*4/2+5*15 = 10+75 = 85
L(~edges, ~(1)+ # 5*4/2 = 10
L(~edges, ~(2)+ # 5*15 = 75
L(~edges, ~(1|2))+ # This logical layer has contents of both, so also 85.
L(~edges, ~(1&2)) # There is no overlap between the two layers, so 0.
)

# Layer-aware terms can be used:
nw1[,] <-0
nw1[1:2:3] <- 1
nw1[2:3] <- 1
nw12[,] <- 0
nw12[1,6:7] <- 1
nw12[2,6:7] <- 1

lnw <- Layer(nw1, nw12, .active=list(~mode==1, ~TRUE))

summary(lnw~L(~triangles, ~(1)+ # 1-2-3 triangle.
  L(~triangles, ~(1|2))+ # 1-2-3, 1-2-6, 1-2-7 triangles
dgwespl(L.base=~1, Ls.path=list(~2, ~2)) # 1-2-6 and 1-2-7 only
)

# Because the layers are represented as a block-diagonal matrix,
# this will only count triangles entirely contained within a single
# layer, i.e., 1-2-3:
summary(lnw-triangles)

# If you need to evaluate bipartite-only statistics on the second
# layer, you need to use the S() operator to select the bipartite
# view:
summary(lnw-L(-S(~b1degree(1:3)+b2degree(1:3),1:5~6:20), ~
`2`))

---

**Description**


**Usage**

data(Lazega)

**Format**

An object of class network of length 5.

**Details**

All relations are directed.

**Nonstandard Vertex Attributes**

- **age** (numeric) the lawyer’s age.
- **gender** (character) the lawyer’s gender (“man”/“woman”).
- **office** (character) in which of the firm’s three offices the lawyer is based (“Boston”/“Hartford”/“Providence”).
- **practice** (character) which area of law the lawyer practices (“corporate”/“litigation”).
- **school** (character) from which law school the lawyer graduated (“Harvard/Yale”/“UConnecticut”/“other”).
- **seniority** (numeric) the lawyer’s seniority rank in the firm (1 = high).
- **status** (character) the lawyer’s status in the firm (“associate”/“partner”).
- **yrs_frm** (numeric) the number of years the lawyer has been with the firm.

**Nonstandard Edge Attributes**

Each directed edge $i \rightarrow j$ has the following attributes:

- **advice** (logical) whether $i$ has reported receiving advice from $j$. (Note that as defined, advice flows from head of the directed edge to the tail.)
- **coworker** (logical) whether $i$ has reported receiving $j$’s assistance in preparing documents. (Note that as defined, assistance flows from head of the directed edge to the tail.)
- **friendship** (logical) whether $i$ considers $j$ a friend outside of work.
**Licenses and Citation**

When publishing results obtained using this data set, the original author (Lazega 2001) should be cited, along with this R package.

**Source**

This version of the dataset was retrieved from the RSiena web site and was compiled by Christopher Steven Marcum and Pavel N. Krivitsky for Krivitsky et al. (2020).

**References**


**Examples**

```r
data(Lazega)  # Construct a multilayer network for ergm(). (See `?Layer` for syntax.)
LLazega <- Layer(Lazega, c("advice", "coworker", "friendship"))
# Specify a layer logic model.
efit <- ergm(LLazega ~ L(~edges + mutual, ~advice) +
              L(~edges + mutual, ~coworker) +
              L(~edges + mutual, ~friendship) +
              L(~edges + mutual, ~advice&coworker) +
              L(~edges + mutual, ~advice&friendship) +
              L(~edges + mutual, ~coworker&friendship))
summary(efit)
```

---

**lm.gofN**

*Fit a linear model to the residuals in a gofN object.*

**Description**

This non-method runs a properly weighted linear model on the raw residuals of a gofN simulation for a multi-network ERGM fit.

**Usage**

```r
lm.gofN(formula, data, ...)  
```

**Arguments**

- `formula` an lm-style formula. See Details for interpretation.
- `data` a gofN object.
- `...` additional arguments to lm(), excluding weights.
Details

The formula’s RHS is evaluated in an environment comprising the network statistics used in the `gofN()` call (which refer to the raw residuals for the corresponding statistic) and the network attributes.

The LHS is handled in a nonstandard manner, designed to make it easier to reference the usually lengthy network statistics: first, it is evaluated in the formula’s environment. If the evaluation is successful and the result is numeric, these numbers are used as indices of the statistics in the `gofN` object to use on the RHS. If it is a character vector, it is treated as names of these statistics.

Value

A list of `lm` objects, one for each element of the vector on the LHS.

See Also

gofN() and related methods.

Examples

data(samplk)
# Add time indices:
samplk1 %n% "t" <- 1
samplk2 %n% "t" <- 2
samplk3 %n% "t" <- 3

monks <- Networks(samplk1, samplk2, samplk3)

fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model

# Is there a time effect we should incorporate?
fit.gof.lm <- lm.gofN((1:2)~t, data=fit.gof)

lapply(fit.gof.lm, summary)

Description

This function is to be considered experimental. Do NOT rely on it. It may, eventually, be moved to `ergm`, perhaps integrated into the `simulate` methods.
Usage

```r
marg_cond_sim(
    object,
    nsim = 1,
    obs.twostage = nsim/2,
    GOF = NULL,
    control = control.gofN.ergm(),
    save_stats = FALSE,
    ...
)
```

Arguments

- **object**: an `ergm` object.
- **nsim**: number of realizations.
- **obs.twostage, GOF, save_stats**: see `gofN()`.
- **control**: a control list returned by `control.gofN.ergm()`: note that `nsim` and `obs.twostage` parameters in the control list are ignored in favor of those passed to the function directly.
- **...**: additional arguments to `ergm_model()`, `simulate.ergm()`, and `summary.ergm_model()`.

Value

An object of similar structure as that returned by `gofN()`.

---

**mutualL-ergmTerm**  
**Mutuality**

Description

In binary ERGMs, equal to the number of pairs of actors $i$ and $j$ for which $(i \rightarrow j)$ and $(j \rightarrow i)$ both exist.

Usage

```r
# binary: mutualL(same=NULL, diff=FALSE, by=NULL, keep=NULL, Ls=NULL)
```

Arguments

- **same**: optional argument. If passed the name of a vertex attribute, only mutual pairs that match on the attribute are counted. Only one of `same` or `by` may be used. If both parameters are passed, `same` takes precedence. This parameter is affected by `diff`.
- **diff**: separate counts for each unique matching value can be obtained by using `diff=TRUE` with `same`. 
by each node is counted separately for each mutual pair in which it occurs and the counts are tabulated by unique values of the attribute if passed the name of a vertex attribute. This means that the sum of the mutual statistics when by is used will equal twice the standard mutual statistic. Only one of same or by may be used. If both parameters are passed, same takes precedence. This parameter is not affected by diff.

keep a numerical vector to specify which statistics should be kept whenever the mutual term would ordinarily result in multiple statistics.

Ls a list (constructed by list() or c() of one or two Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation). If given, the statistic will count the number of dyads where a tie in $Ls[[1]]$ reciprocates a tie in $Ls[[2]]$ and vice versa. (Note that dyad that has mutual ties in $Ls[[1]]$ and in $Ls[[2]]$ will add 2 to this statistic.) If a formula is given, it is replicated.

Details

This term can only be used with directed networks.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: directed, frequently-used, layer-aware, binary

Description

Evaluates the terms in formula on each of the networks joined using Networks function, and returns either a weighted sum or an lm-style linear model for the ERGM coefficients (Krivitsky et al. 2022). Its syntax follows that of lm closely, with sensible defaults.

The default formula (~1) sums the specified network statistics. If lm refers to any network attributes for which some networks have missing values, the term will stop with an error. This can be avoided by pre-filtering with subset, which controls which networks are affected by the term.

Usage

# binary: N(formula, lm=~1, subset=TRUE, weights=1, contrasts=NULL, offset=0, label=NULL, 
# .NetworkID=".NetworkID", .NetworkName=".NetworkName")

# valued: N(formula, lm=~1, subset=TRUE, weights=1, contrasts=NULL, offset=0, label=NULL, 
# .NetworkID=".NetworkID", .NetworkName=".NetworkName")
Arguments

.NetworkID, .NetworkName
Optional strings indicating the vertex attributes used to distinguish and name the networks; intended to be used by term developers.

formula
a one-sided \texttt{ergm}()-style formula with the terms to be evaluated

lm
a one-sided \texttt{lm}()-style formula whose RHS specifies the network-level predictors for the terms in the \texttt{ergm}() formula formula.

subset, contrasts
see \texttt{lm}().

offset
A constant, a vector of length equal to the number of networks, or a matrix whose number of rows is the number of networks and whose number of columns is the number of free parameters of the ERGM. It can be specified in \texttt{lm} as well.

weights
reserved for future use; attempting to change it will cause an error: at this time, there is no way to assign sampling weights to networks.

label
An optional parameter which will add a label to model parameters to help identify the term (which may have similar predictors but, say, a different network subset) in the output or a function that wraps the names.

Offsets and fixing parameters

By default, an \texttt{N(formula, lm)} term will add $p \times q$ free parameters, where $p$ is the number of free parameters (possibly curved) of the ERGM specified by \texttt{formula}, and $q$ is the number of parameters specified by the \texttt{lm} formula. That is, there would be one parameter for each combination of an ERGM parameter and a linear model parameter, in an ERGM-major order (i.e., for each ERGM parameter, the linear model parameters will be enumerated). For example, the term \texttt{gwesp()} has two free parameters: its coefficient and its decay rate. We can specify a model in which they depend on \(\log(n)\) as \texttt{N(\log(n))~gwesp}, resulting in the following 4 parameters, with the intercept for the linear model being implicit:

\[
\begin{align*}
\texttt{#> [1] "N(1)~gwesp"} & \quad \texttt{"N(log(n))~gwesp"} & \quad \texttt{"N(1)~gwesp.decay"} \\
\texttt{#> [4] "N(log(n))~gwesp.decay"}
\end{align*}
\]

If a different linear model is desired for different ERGM terms (e.g., some are to be affected by network size while others are not), multiple \texttt{N()} terms can be specified. This covers most such cases, but not all. For example, suppose that for the above model, we wish for its coefficient to depend on \(\log(n)\) but for the decay parameter not to. In this case, one can use the \texttt{offset()} decorator with partial offsetting. Then, specifying \texttt{offset(N(\log(n))~gwesp(), ~log(n)), 4)}, we get:

\[
\begin{align*}
\texttt{#> [1] "N(1)~gwesp"} & \quad \texttt{"N(log(n))~gwesp"} \\
\texttt{#> [3] "N(1)~gwesp.decay"} & \quad \texttt{"offset(N(log(n))~gwesp.decay)"}
\end{align*}
\]

Then, setting the corresponding \texttt{offset.coef = 0} will fix the coefficient of \(\log(n)\) for the decay parameter at 0, while allowing a constant decay parameter to be estimated.
Networks

Note

Care should be taken to avoid multicollinearity when using this operator. As with the `lm()` function, `lm` formulas have an implicit intercept, which can be suppressed by specifying `~ 0 + ...` or `~ -1 + ...` on the formula. When `lm` is given a model with intercept and a categorical predictor (including a logical one), it will use the first level (or FALSE) as the baseline, but if the model is without intercept, it will use all levels of the first categorical predictor. This is typically what is wanted in a linear regression, but for the `N` operator, this can be problematic if the ”intercept” effect is added by a different term. A workaround is to convert the categorical predictor to dummy variables before putting it into the `lm` formula.

References


See Also

`ergmTerm` for index of model terms currently visible to the package.

**Keywords:** directed, operator, undirected, binary, valued

`vignette("Goeyvaerts_reproduction")` for a demonstration.

---

Networks

*A multinetowrk network representation.*

Description

A function for specifying the LHS of a multi-network (a.k.a. multilevel) ERGM. Typically used in conjunction with the `N()` term operator.

Usage

```r
Networks(...)  
```

Arguments

... 

network specification, in one of two formats:

1. An (optionally named) list of networks with same directedness and bipartedness (but possibly different sizes).
2. Several networks as (optionally named) arguments.

Value

A network object with multinetowrk metadata.
network_view

Construct a "view" of a network.

Description

Returns a network with edges optionally filtered according to a specified criterion and with edge attributes optionally computed from other edge attributes.

Usage

network_view(x, ..., .clear = FALSE, .sep = ".")

Arguments

x

a network object.

... a list of attribute or filtering specifications. See Details.

.clear whether the edge attributes not set by this call should be deleted.

.sep when specifying via a character vector, use this as the separator for concatenating edge values.

Details

Attribute specification arguments have the form <newattrname> = <expr>, where <newattrname> specifies the name of the new edge attribute (or attribute to be overwritten) and <expr> can be one of the following:

- a function The function will be passed two arguments, the edgelist tibble and the network, and must return a vector of edge attribute values to be set on the edges in the order specified.

Examples

data(samplk)

# Method 1: list of networks
monks <- Networks(list(samplk1, samplk2))
ergm(monks ~ N(~edges))

# Method 2: networks as arguments
monks <- Networks(samplk1, samplk2)
ergm(monks ~ N(~edges))
a formula The expression on the RHS of the formula will be evaluated with names in it referencing
the edge attributes. The input network may be referenced as .nw. The expression’s result is
expected to be a vector of edge attribute values to be set on the edges in the order specified.

a character vector If of length one, the edge attribute with that name will simply be copied; if
greater than one, the attribute values will be concatenated with the .sep argument as the
separator.

an object enclosed in I() The object will be used directly to set the edge attribute.

Filtering arguments are specified the same way as attribute arguments, but they must be unnamed
arguments (i.e., must be passed without the =) and must return a logical or numeric vector suitable
for indexing the edge list. Multiple filtering arguments may be specified, and the edge will be kept
if it satisfies all. If the conjunction of the edge’s original states and the filtering results is ambiguous
(i.e., NA), it will be set as missing.

Value
A network object with modified edges and edge attributes.

Examples

data(florentine)
flo <- flomarriage
flo[,,add.edges=TRUE] <- as.matrix(flomarriage) | as.matrix(flobusiness)
flo[,, names.eval="m"] <- as.matrix(flomarriage)==1
flobusiness[3,5] <- NA
flobusiness[3,5] <- as.matrix(flobusiness)==1
flo
(flob <- network_view(flo, "b"))
(flobusiness) # for comparison

(flob <- network_view(flo, ~b&m))
(flobusiness & flomarriage) # for comparison

as.matrix(flob <- network_view(flo, bm=~b+m), attrname="bm")
(as.matrix(flobusiness) + as.matrix(flomarriage)) # for comparison

as.matrix(flob <- network_view(flo, ~b, bm=~b+m), attrname="bm")
as.matrix(flobusiness)*(1+as.matrix(flomarriage)) # for comparison

------

out-degree Out-degree
Description

This term adds one network statistic to the model for each element in \( d \); the \( i \)th such statistic equals the number of nodes in the network of out-degree \( d[i] \), i.e. the number of nodes with exactly \( d[i] \) out-edges.

Usage

# binary: odegreeL(d, by=NULL, homophily=FALSE, levels=NULL)

Arguments

d a vector of distinct integers
by a character string giving the name of an attribute in the network's vertex attribute list.
homophily If this is specified and homophily is TRUE, then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree statistics are calculated for nodes having each separate value of the attribute.

Ls either a Layer Logic specification formula (c.f. Layer Logic section in the Layer() documentation) or a list thereof (constructed by list() or c()). If given, degree of a node
levels list of layer specifications

Details

This term can only be used with directed networks; for undirected networks see degree.

If a list of layer specifications is given, degree of a node \( i \) is considered to be the number of edges in all layers, combined.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: categorical nodal attribute, directed, frequently-used, binary

plot.gofN  Plotting methods for gofN, making residual and scale-location plots.

Description

The plot() method uses R graphics.
The ggplot2::autoplot() method uses ggplot and ggrepel.
Usage

```r
## S3 method for class 'gofN'
plot(
  x,
  against = NULL,
  which = 1:2,
  col = 1,
  pch = 1,
  cex = 1,
  bg = 0,
  ...
)

## S3 method for class 'gofN'
autoplot(
  x,
  against = .fitted,
  which = 1:2,
  mappings = list(),
  geom_args = list(),
  id.n = 3,
  id.label = NULL
)
```

Arguments

- `x` a `gofN` object.
- `against` what the residuals should be plotted against. Note that different methods use different formats: see Details. Categorical (factor and ordered) values are visualised using boxplots, with ordered values also adding a smoothing line like the quantitative. Defaults to the fitted values.
- `which` which to plot (1 for residuals plot, 2 for $\sqrt{R_i}$ scale plot, and 3 for normal quantile-quantile plot).
- `col, pch, cex, bg` vector of values (wrapped in `I()`), network attribute, or a formula whose RHS gives an expression in terms of network attributes to plot against.
- `...` additional arguments to `plot()`, `qqnorm()`, and `qqline()`, and others.
- `ask` whether the user should be prompted between the plots.
- `id.n` maximum number of extreme points to label explicitly.
id.label: specification for how extreme points are to be labeled, defaulting to network’s index in the combined network.

main: a template for the plots’ titles; these use glue()’s templating, with {type} replaced with the type of plot and {name} replaced with the statistic.

xlab: horizontal axis label; defaults to a character representation of against.

ylim: vertical range for the plots, interpreted as in graphics::plot(); can be specified as a list with 3 elements, giving the range for the corresponding plot according to the plot numbers for the which= argument, and can be used to ensure that, e.g., diagnostic plots for different models are on the same scale.

cex.id: scaling factor for characters used to label extreme points; see plot.lm().

mappings: a named list of lists of mappings constructed by ggplot2::aes() overriding the defaults. See Details below.

geom_args: a named list of lists of arguments overriding the defaults for the individual geoms. See Details below.

Details

For the plot() method, against and id.label can be vectors of values (enclosed in I() to be used as is), a character string identifying a network attribute, or a formula whose RHS gives an expression in terms of network attributes to plot against. The against formula may also contain a .fitted variable which will be substituted with the fitted values.

For autoplot.gofN(), against and id.label are interpreted as expressions in terms of network attributes and values generated by augment.gofN(), included .fitted for the fitted values.

Value

autoplot.gofN() returns a list of ggplot objects that if printed render to diagnostic plots. If there is only one, the object itself is returned.

Customising autoplot.gofN()

autoplot.gofN() constructs the plots out of ggplot2::ggplot(), ggplot2::geom_point() (for numeric against), ggplot2::geom_boxplot() (for categorical or ordinal against), and ggplot2::geom_smooth() (for numeric or ordinal against), and ggrepel::geom_text_repel(). Mappings and arguments passed through mappings and geom_args override the respective defaults. They may have elements default (for ggplot()), point (for geom_point() and geom_boxplot()), smooth (for geom_smooth), and text (for geom_text_repel()).

See Also

gofN() for examples, plot.lm(), graphics::plot() for regression diagnostic plots and their parameters.
Description

A utility to facilitate argument completion of control lists, reexported from statnet.common.

Currently recognised control parameters

This list is updated as packages are loaded and unloaded.

Package ergm:

control.ergm drop, init, init.method, main.method, force.main, main.hessian, checkpoint, resume, MLE.samplesize, init.MLE.samplesize, MLE.type, MLE.maxit, MLE.nonvar, MLE.nonident, MLE.nonident.tol, MLE.covariance.samplesize, MLE.covariance.method, MLE.covariance.sim.burnin, MLE.covariance.sim.interval, MLE.constraints.ignore, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.interval, MCMC.burnin, MCMC.samplesize, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.return.stats, MCMC.runtime.traceplot, MCMC.maxedges, MCMC.addto.se, MCMC.packagenames, SAN.maxit, SAN.nsteps.times, SAN, MCMLE.termination, MCMLE.maxit, MCMLE.conv.min.pval, MCMLE.confidence, MCMLE.confidence.boost, MCMLE.confidence.boo, MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize, obs.MCMC.samplesize, obs.MCMC.samplesize, obs.MCMC.interval.mul, obs.MCMC.interval, obs.MCMC.burnin.mul, obs.MCMC.burnin, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density, obs.MCMC.min.depfac, MCMC.sampsize.boost.pow, MCMC.MCMC_metric, MCMC.MCMC.method, MCMC.MCMC.dampening, MCMC.MCMC.dampening.min.ess, MCMC.MCMC.dampening.level, MCMC.steplength.margin, MCMC.steplength, MCMC.steplength.parallel, MCMC.steplength.sequential, MCMC.density.guard.min, MCMC.density.guard, MCMC.effectiveSize, MCMC.interval, MCMC.burnin, MCMC.samplesize.per_theta, MCMC.samplesize.min, MCMC.samplesize, obs.MCMC.samplesize.per_theta, obs.MCMC.samplesize, obs.MCMC.samplesize, obs.MCMC.interval, obs.MCMC.burnin, MCMC.steplength.solver, MCMC.last.boost, MCMC.steplength.esteq, MCMC.steplength.miss.sample, MCMC.steplength.min, MCMC.effectiveSize.interval_drop, MCMC.save_intermediates, MCMC.nonvar, MCMC.nonident, MCMC.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.min_iterations, SA.max_iterations, SA.phase3_n, SA.interval, SA.burnin, SA.samplesize, CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.nsteps, CD.multiplicity, CD.nsteps.obs, CD.multiplicity.obs, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.reltol, CD.metric, CD.method, CD.dampening, CD.dampening.min.ess, CD.dampening.level, CD.steplength.margin, CD.steplength, CD.adaptive.epsilon, CD.steplength.esteq, CD.steplength.miss.sample, CD.steplength.min, CD.steplength.parallel, CD.steplength.solver, loglik, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.bridge bridge.nsteps, bridge.target.se, bridge.bidirectional, MCMC.burnin, MCMC.burnin.between, MCMC.interval, MCMC.samplesize, obs.MCMC.burnin, obs.MCMC.burnin.between, obs.MCMC.interval, obs.MCMC.samplesize, MCMC.prop, MCMC.prop.weights, MCMC.prop.args,
obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.godfather term.options

control.gof.ergm nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gof.formula nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.logLik.ergm bridge.nsteps, bridge.target.se, bridge.bidirectional, MCMC.burnin, MCMC.interval, MCMC.samplesize, obs.MCMC.samplesize, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT...

control.san SAN.maxit, SAN.tau, SAN.invrcov, SAN.invrcov.diag, SAN.nsteps.alloc, SAN.nsteps, SAN.samplesize, SAN.prop, SAN.prop.weights, SAN.prop.args, SAN.packagenames, SAN.ignore.finite.offsets, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.simulate MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.scl, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT...

control.simulate.ergm MCMC.burnin, MCMC.interval, MCMC.scale, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.scl, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT...

control.simulate.formula MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.scl, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT...

control.simulate.formula.ergm MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.scl, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT...
Package ergm.multi:

control.gofN nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gofN.ergm nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

See Also

statnet.common::snctrl()

split.network A split() method for network::network objects.

Description

Split a network into subnetworks on a factor.

Usage

## S3 method for class 'network'
split(x, f, drop = FALSE, sep = ".", lex.order = FALSE, ...)

Arguments

x a network::network object.

f, drop, sep, lex.order

see split(): note that f must have length equal to network.size(x).

... additional arguments, currently unused.

Value

A network.list containing the networks. These networks will inherit all vertex and edge attributes, as well as relevant network attributes.

See Also

network::get.inducedSubgraph()
Description

This term adds one statistic to the model, equal to the number of cross-layer two-stars or two-paths in the network.

Usage

```
# binary: twostarL(Ls, type, distinct=TRUE)
```

Arguments

- **Ls**: a list (constructed by `list()` or `c()`) of two Layer Logic specifications (c.f. Layer Logic section in the `Layer()` documentation) specifying the layers of interest.
- **type**: determines what is counted:
  1. "any" Number of configurations \((i - j), (i - k)\), where \((i - j)\) is in logical layer \(L_s[[1]]\) and \((i - k)\) is in logical layer \(L_s[[2]]\).
  2. "out" Number of configurations \((i \rightarrow j), (i \rightarrow k)\), where \((i \rightarrow j)\) is in logical layer \(L_s[[1]]\) and \((i \rightarrow k)\) is in logical layer \(L_s[[2]]\).
  3. "in" Number of configurations \((j \rightarrow i), (k \rightarrow i)\), where \((j \rightarrow i)\) is in logical layer \(L_s[[1]]\) and \((k \rightarrow i)\) is in logical layer \(L_s[[2]]\).
  4. "path" Number of configurations \((j \rightarrow i), (i \rightarrow k)\), where \((j \rightarrow i)\) is in logical layer \(L_s[[1]]\) and \((i \rightarrow k)\) is in logical layer \(L_s[[2]]\).

At this time, "any" is only supported for undirected networks, and if the network is undirected, type is ignored and "any" is assumed.

- **distinct**: if TRUE, \(j\) and \(k\) above are required to be distinct. That is, the constituent edges may not be coincident or reciprocal.

See Also

`ergmTerm` for index of model terms currently visible to the package.

**Keywords**: directed, layer-aware, undirected, binary
uncombine_network

Split up a network into a list of subgraphs

Description

Given a network created by `combine_networks()`, `uncombine_network()` returns a list of networks, preserving attributes that can be preserved.

Usage

```r
uncombine_network(
  nw,
  split.vattr = nw %n% ".blockID.vattr",
  names.vattr = nw %n% ".blockName.vattr",
  use.subnet.cache = FALSE
)
```

Arguments

- **nw**: a `network::network` created by `combine_networks()`.
- **split.vattr**: name of the vertex attribute on which to split, defaulting to the value of the ".blockID.vattr" network attribute.
- **names.vattr**: optional name of the vertex attribute to use as network names in the output list, defaulting to the value of the ".blockName.vattr" network attribute.
- **use.subnet.cache**: whether to use subnet cache if available; this is only safe to do if the network is not used for its edges but only for its vertex and network attributes.

Value

- a list of `network::networks` containing subgraphs on `split.vattr`. In particular,
  - their basic properties (directedness and bipartednes) are the same as those of the input network;
  - vertex attributes are split;
  - edge attributes are assigned to their respective edges in the returned networks.

If `split.vattr` is a vector, only the first element is used and it's "popped".

See Also

- `split.network()`
**Examples**

data(samplk)

ol <- combine_networks(list(samplk1, samplk2, samplk3))
image(as.matrix(ol))

ol <- uncombine_network(ol)

---

**upper_tri-ergmConstraint**

*Only dyads in the upper-triangle of the sociomatrix may be toggled*

---

**Description**

For a directed network, only dyads \((i, j)\) for which \(i < j\) may be toggled. Optional argument `attr` controls which subgraphs are thus restricted.

**Usage**

```
# upper_tri(attr = NULL)
```

**Arguments**

- `attr` a vertex attribute specification (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

**See Also**

- `ergmConstraint` for index of constraints and hints currently visible to the package.

**Keywords:** directed, dyad-independent
Index

* bipartite
  b1degreeL-ergmTerm, 5
  b2degree-ergmTerm, 6
  gwb1degreeL-ergmTerm, 26
  gwb2degreeL-ergmTerm, 27

* categorical nodal attribute
  b1degreeL-ergmTerm, 5
  b2degree-ergmTerm, 6
  degreeL-ergmTerm, 12
  idegreeL-ergmTerm, 30
  odegreeL-ergmTerm, 43

* curved
  gwb1degreeL-ergmTerm, 26
  gwb2degreeL-ergmTerm, 27
  gwdegreeL-ergmTerm, 28
  gwidegreeL-ergmTerm, 28
  gwodegreeL-ergmTerm, 29

* datasets
  Goeyvaerts, 22
  Lazega, 35

* directed
  CMBL-ergmTerm, 6
  ddspL-ergmTerm, 11
  degreeL-ergmTerm, 12
  despL-ergmTerm, 13
  dgwdspL-ergmTerm, 15
  dgwespL-ergmTerm, 16
  dgwnspL-ergmTerm, 18
  dnspL-ergmTerm, 20
  L-ergmTerm, 31
  mutualL-ergmTerm, 38
  twostarL-ergmTerm, 50

* frequently-used
  b1degreeL-ergmTerm, 5
  b2degree-ergmTerm, 6
  degreeL-ergmTerm, 12
  gwb1degreeL-ergmTerm, 26
  gwb2degreeL-ergmTerm, 27
  idegreeL-ergmTerm, 28
  odegreeL-ergmTerm, 30
  mutualL-ergmTerm, 38
  twostarL-ergmTerm, 50

* layer-aware
  CMBL-ergmTerm, 6
  ddspL-ergmTerm, 11
  despL-ergmTerm, 13
  dgwdspL-ergmTerm, 15
  dgwespL-ergmTerm, 16
  dgwnspL-ergmTerm, 18
  dnspL-ergmTerm, 20
  L-ergmTerm, 31
  mutualL-ergmTerm, 38
  twostarL-ergmTerm, 50

* models
  ergm.multi-package, 2

* operator
  L-ergmTerm, 31
  N-ergmTerm, 39

* package
  ergm.multi-package, 2

* undirected
  b1degreeL-ergmTerm, 5
  b2degree-ergmTerm, 6
  CMBL-ergmTerm, 6
  ddspL-ergmTerm, 11
  degreeL-ergmTerm, 12
  despL-ergmTerm, 13
  dgwdspL-ergmTerm, 15
  dgwespL-ergmTerm, 16
  dgwnspL-ergmTerm, 18
  dnspL-ergmTerm, 20
  gwb1degreeL-ergmTerm, 26
  gwb2degreeL-ergmTerm, 27

* dyad-independent
  upper_tri-ergmConstraint, 52

53
gwdegreeL-ermgTerm, 28
N-ermgTerm, 39
twostarL-ermgTerm, 50
[gofN (gofN), 23
.abs, 32
Arithmetical, 32
as_tibble.combined_networks, 4
.attr, 24
augment.gofN (gofN), 23
augment.gofN(), 46
autoplot.gofN (plot.gofN), 44
autoplot.gofN(), 25
b1degreeL-ermgTerm, 5
b2degree-ermgTerm, 6
c(), 5–7, 13, 26, 27, 30–32, 39, 44, 50
certain operator terms, 33
CML-ermgTerm, 6
combine_networks, 7
combine_networks(), 7, 51
combined_networks, 4, 5
combined_networks (combine_networks), 7
control.ermg, 26–30, 47
control.ermg.bridge, 47
control.ermg.godfather, 48
control.gof.ermg, 48
control.gof.formula, 48
control.gofN, 49
control.gofN (control.gofN.ermg), 9
control.gofN.ermg, 9, 49
control.gofN.ermg(), 10, 23, 38
control.logLik.ermg, 48
control.san, 48
control.simulate, 48
control.simulate.ermg, 48
control.simulate.ermg (), 11
control.simulate.formula, 48
control.simulate.formula.ermg, 48
ddspL-ermgTerm (ddspL-ermgTerm), 11
dergm, 9, 11, 23, 38
dergm (), 3, 31, 32, 40
dergm.multi-package, 2
dergm::edgecov, 8
dergm::gof(), 25
dergm_model(), 38
dergmConstraint, 52
ermgTerm, 5–7, 12–14, 16, 18, 20, 21, 26–31, 39, 41, 44, 50
ermgTerm?L, 3
ermgTerm?N, 3
esplL-ermgTerm (desplL-ermgTerm), 13
factor, 45
ggplot2::aes(), 46
ggplot2::autoplot(), 44
ggplot2::geom_boxplot(), 46
ggplot2::geom_point(), 46
ggplot2::geom_smooth(), 46
ggplot2::ggplot(), 46
ggrepel::geom_text_repel(), 46
glue(), 46
Goeyvaerts, 22
gofN, 23, 36, 37, 44, 45
gofN (), 3, 9, 23, 37, 38, 46
graphics::plot(), 46
gwb1degreeL-ermgTerm, 26
gwb2degreeL-ermgTerm, 27
gwdegreeL-ermgTerm, 28
gwdspl, 18
gwdsplL-ermgTerm (dgwdsplL-ermgTerm), 15
gwespl, 18
gwesplL-ermgTerm (dgwesplL-ermgTerm), 16
gwdegreeL-ermgTerm, 28
gwesplL-ermgTerm (dgwesplL-ermgTerm), 18
gwdegreeL-ermgTerm, 29
Help on model specification, 33, 42
I(), 43, 45, 46
idegreeL-ermgTerm, 30
InitErgmConstraint.upper_tri
(upper_tri-ergmConstraint), 52
InitErgmTerm.b1degreeL
(b1degreeL-ermgTerm), 5
InitErgmTerm.b2degreeL
(b2degree-ermgTerm), 6
set.MT_terms(), 11
sign, 32
simulate.ergm(), 23, 38
snctrl, 47
split(), 49
split.network, 49
split.network(), 8, 51
statnet.common::snctrl(), 49
summary.combined_networks
  (combine_networks), 7
summary.ergm_model(), 23, 38
summary.gofN(gofN), 23

t, 33
tibble, 24, 42
twoStarL-ergmTerm, 50
uncombine_network, 51
uncombine_network(), 8, 51
upper_tri-ergmConstraint, 52