

Package ‘tergm’

November 8, 2022

Version 4.1.1

Date 2022-11-07

Title Fit, Simulate and Diagnose Models for Network Evolution Based on Exponential-Family Random Graph Models

Depends ergm ($\geq 4.3.1$),
network ($\geq 1.18.0$),
networkDynamic ($\geq 0.11.2$)

Imports methods,
utils,
robustbase ($\geq 0.93.5$),
coda ($\geq 0.19.2$),
nlme ($\geq 3.1.139$),
MASS ($\geq 7.3.51.4$),
statnet.common ($\geq 4.4.0$),
purrr

LinkingTo ergm

Suggests lattice ($\geq 0.20.38$),
parallel,
rmarkdown (≥ 1.12),
knitr (≥ 1.22),
tibble,
testthat,
covr

BugReports <https://github.com/statnet/tergm/issues>

Description An integrated set of extensions to the 'ergm' package to analyze and simulate network evolution based on exponential-family random graph models (ERGM). 'tergm' is a part of the 'statnet' suite of packages for network analysis. See Krivitsky and Handcock (2014) <[doi:10.1111/rssb.12014](https://doi.org/10.1111/rssb.12014)> and Carnegie, Krivitsky, Hunter, and Goodreau (2015) <[doi:10.1080/10618600.2014.903087](https://doi.org/10.1080/10618600.2014.903087)>.

License GPL-3 + file LICENSE

URL <https://statnet.org>

VignetteBuilder rmarkdown, knitr

RoxygenNote 7.2.1

Roxygen list(markdown = TRUE)

Config/testthat/parallel true

Config/testthat/edition 3

Encoding UTF-8

R topics documented:

tergm-package	3
.extract.fd.formulae	4
Change-ergmTerm	5
combine_networks	6
control.simulate.network	8
control.simulate.tergm	10
control.stergm	12
control.tergm	20
control.tergm.godfather	27
Cross-ergmTerm	28
degrange.mean.age-ergmTerm	28
degree.mean.age-ergmTerm	29
discord-ergmHint	30
Diss-ergmTerm	30
edge.ages-ergmTerm	31
EdgeAges-ergmTerm	32
edgecov.ages-ergmTerm	32
edgecov.mean.age-ergmTerm	33
edges.ageinterval-ergmTerm	34
Form-ergmTerm	35
impute.network.list	35
is.durational	37
lasttoggle	38
mean.age-ergmTerm	38
NetSeries	39
nodefactor.mean.age-ergmTerm	40
nodemix.mean.age-ergmTerm	41
Persist-ergmTerm	42
simulate.network	42
simulate.tergm	46
snctrl	51
split.network	53
stergm	54
summary_formula.networkDynamic	57
tergm	58
tergm.godfather	61
uncombine_network	63

Index

65

tergm-package

Fit, Simulate and Diagnose Dynamic Network Models derived from Exponential-Family Random Graph Models

Description

`tergm` is a collection of extensions to the `ergm` package to fit, diagnose, and simulate models for dynamic networks — networks that evolve over time — based on exponential-family random graph models (ERGMs). For a list of functions type `help(package='tergm')`

Details

When publishing results obtained using this package, please cite the original authors as described in `citation(package="tergm")`.

All programs derived from this package must cite it.

An exponential-family random graph model (ERGM) postulates an exponential family over the sample space of networks of interest, and `ergm` package implements a suite of tools for modeling single networks using ERGMs.

There have been a number of extensions of ERGMs for modeling the evolution of networks, including the temporal ERGM (TERGM) of Hanneke et al. (2010) and the separable temporal ERGM (STERGM) of Krivitsky and Handcock (2014). The latter model allows familiar ERGM terms and statistics to be reused in a dynamic context, interpreted in terms of formation and dissolution (persistence) of ties. Krivitsky (2012) suggested a method for fitting dynamic models when only a cross-sectional network is available, provided some temporal information for it is available as well.

This package aims to implement these and other ERGM-based models for network evolution. At this time, it implements, via the `tergm` function, a general framework for modeling tie dynamics in temporal networks with flexible model specification (including (S)TERGMs). Estimation options include a conditional MLE (CMLE) approach for fitting to a series of networks and an Equilibrium Generalized Method of Moments Estimation (EGMME) for fitting to a single network with temporal information. For further development, see the referenced papers.

Temporal model specification in `tergm`

The operator terms implemented by `tergm` are `Form()`, `Persist()`, `Diss()`, `Cross()`, and `Change()`. These are used to specify how the `ergm` terms (`ergmTerm`) in a formula are evaluated across a network time-series. Note, you cannot use one of these operators within another temporal, so `Cross(~Form(~edges))` is not a valid specification. (Generally, nesting these operators within other operators will often not work; nesting other operators within them will almost always work, however.)

The durational terms are distinguished either by their name, `mean.age`, or their name extensions: `<name>.ages`, `<name>.mean.age`, and `<name>.age.interval`. In contrast to their eponymous terms in `ergm`, these durational terms take into account the elapsed time since each (term-relevant) dyad in the network was last toggled.

As currently implemented, the package does not support use of many durational terms during estimation, though it may work with some. But durational terms may be used as targets, monitors, or summary statistics. The ability to use these terms in the estimation of models is under development.

Compatibility with previous versions

If you previously used the `stergm()` function in this package, please note that `stergm()` has been superseded by the new `tergm` function, and has been deprecated. The dissolution formula in `stergm()` maps to the new `Persist()` operator in the `tergm()` function, **not** the `Diss()` operator.

For detailed information on how to download and install the software, go to the Statnet project website: <https://statnet.org>. A tutorial, support newsgroup, references and links to further resources are provided there.

References

Hanneke S, Fu W and Xing EP (2010). Discrete Temporal Models of Social Networks. *Electronic Journal of Statistics*, 2010, 4, 585-605. doi:10.1214/09EJS548

Krackhardt, D and Handcock, MS (2006) Heider vs Simmel: Emergent features in dynamic structures. ICML Workshop on Statistical Network Analysis. Springer, Berlin, Heidelberg, 2006.

Krivitsky PN & Handcock MS (2014) A Separable Model for Dynamic Networks. *Journal of the Royal Statistical Society, Series B*, 76(1): 29-46. doi:10.1111/rssb.12014

Krivitsky, PN (2012). Modeling of Dynamic Networks based on Egocentric Data with Durational Information. *Pennsylvania State University Department of Statistics Technical Report*, 2012(2012-01). <https://web.archive.org/web/20170830053722/https://stat.psu.edu/research/technical-report-files/2012-technical-reports/TR1201A.pdf>

Butts CT (2008). **network**: A Package for Managing Relational Data in . *Journal of Statistical Software*, 24(2). doi:10.18637/jss.v024.i02

Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). A **statnet** Tutorial. *Journal of Statistical Software*, 24(8). doi:10.18637/jss.v024.i08

Hunter, D. R. and Handcock, M. S. (2006) Inference in curved exponential family models for networks, *Journal of Computational and Graphical Statistics*, 15: 565-583

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). doi:10.18637/jss.v024.i03

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). doi:10.18637/jss.v024.i04

.extract.fd.formulae *An Internal Function for Extracting (Some) Formation and Dissolution Formulas from a Combined Formula*

Description

This function is used in `tergm.EGMME.initialfit` and also when targets or monitoring formulas are specified by characters. It makes a basic attempt to identify the formation and dissolution formulas within a larger combined formula (which may also include non-separable terms). Instances of `Form` at the top level (which may occur inside `offset`) contribute to the formation formula; instances of `Persist` and `Diss` at the top level (which may also occur inside `offset`) contribute to

the dissolution formula. All other terms are regarded as non-separable; this includes instances of Form, Persist, and Diss that occur inside other operator terms, including inside Offset, and also includes all interactions at the top level (for which the top level term is effectively the interaction operator * or :), whether or not they include Form, Persist, and/or Diss. The formation and dissolution formulas are obtained by adding the contributing terms, replacing Form and Persist with trivial operators that protect the environments of their formula arguments but have no effect on statistics or coefficient names (meaning the formulas effectively become cross-sectional), and replacing Diss by a similar operator that negates statistics. These are included in the return value as the form and pers elements of the list (the "dissolution" formula really being the persistence formula), which also includes the formula of non-separable terms as nonsep, and the formula of all terms after replacing Form, Persist, and Diss as described above as all.

If usage proves problematic, one may specify the monitoring and/or targets formulas explicitly (rather than by characters), and one may pass initial coefficient values for the EGMME to avoid running `tergm.EGMME.initialfit`.

Usage

```
.extract.fd.formulae(formula)
```

Arguments

formula a formula.

Value

A list containing form, pers, nonsep, and all formulas as described above.

Change-ergmTerm *The Change Operator Term*

Description

The Change Operator Term

Usage

```
# binary: Change(formula)
```

Arguments

formula a one-sided `ergm()`-style formula with the terms to be evaluated

Details

This term accepts a model formula and produces the corresponding model for a network constructed by taking the dyads that have changed between time steps.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

combine_networks	<i>A single block-diagonal network created by combining multiple networks</i>
------------------	---

Description

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

Usage

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

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

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

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

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.

<code>blockName.vattr</code>	if not NULL, the name of the vertex attribute into which to store the name of the network to which that vertex originally belonged.
<code>detect.edgecov</code>	if TRUE, combine network attributes that look like dyadic covariate (<code>ergm::edgecov</code>) matrices into a block-diagonal matrix.
<code>keep.unshared.attr</code>	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 <code>detect.edgecov==TRUE</code> .
<code>subnet.cache</code>	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 <code>uncombine_network()</code>), an expensive call to <code>split.network()</code> can be avoided, at the cost of storage.
<code>x, object</code>	a combined network.
<code>...</code>	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 `nwl`. In particular,

- the returned network's size is the sum of the input networks';
- its basic properties (directedness and bipartednes) 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, a two new vertex attributes, specified by `blockID.vattr` and (optionally) `blockName.vattr` contain, respectively, the index in `nwl` of the network from which that vertex came and its name, determined as follows:

1. If `nwl` 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.

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

```
data(samplk)

o1 <- combine_networks(list(samplk1, samplk2, samplk3))
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.simulate.network

Auxiliary for Controlling Separable Temporal ERGM Simulation

Description

Auxiliary function as user interface for fine-tuning STERGM simulation.

Usage

```
control.simulate.network(
  MCMC.burnin.min = 1000,
  MCMC.burnin.max = 1e+05,
  MCMC.burnin.pval = 0.5,
  MCMC.burnin.add = 1,
  MCMC.prop.form = ~discord + sparse,
  MCMC.prop.diss = ~discord + sparse,
  MCMC.prop.weights.form = "default",
  MCMC.prop.weights.diss = "default",
  MCMC.prop.args.form = NULL,
  MCMC.prop.args.diss = NULL,
  MCMC.maxedges = Inf,
  MCMC.maxchanges = 1e+06,
  term.options = NULL,
  MCMC.packagenames = c()
```



```

)

control.simulate.stergm(
  MCMC.burnin.min = NULL,
  MCMC.burnin.max = NULL,
  MCMC.burnin.pval = NULL,
  MCMC.burnin.add = NULL,
  MCMC.prop.form = NULL,
  MCMC.prop.diss = NULL,
  MCMC.prop.weights.form = NULL,
  MCMC.prop.weights.diss = NULL,
  MCMC.prop.args.form = NULL,
  MCMC.prop.args.diss = NULL,
  MCMC.maxedges = NULL,
  MCMC.maxchanges = NULL,
  term.options = NULL,
  MCMC.packagenames = NULL
)

```

Arguments

- `MCMC.burnin.min`, `MCMC.burnin.max`, `MCMC.burnin.pval`, `MCMC.burnin.add`
 Number of Metropolis-Hastings steps per time step used in simulation. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once `MCMC.burnin.min` steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than `MCMC.burnin.pval`), or `MCMC.burnin.max` steps are proposed, whichever comes first, the simulation is stopped after an additional `MCMC.burnin.add` times the number of elapsed steps have been taken. (Stopping immediately would bias the sampling.)
 To use a fixed number of steps, set `MCMC.burnin.min` and `MCMC.burnin.max` to the same value.
- `MCMC.prop.form` Hints and/or constraints for selecting and initializing the proposal.
- `MCMC.prop.weights.form`
 Specifies the proposal weighting scheme to be used in the MCMC Metropolis-Hastings algorithm. Possible choices may be determined by calling [`ergm_proposal_table`](#).
- `MCMC.prop.weights.diss`, `MCMC.prop.args.diss`, `MCMC.prop.diss`
 Ignored. These are included for backwards compatibility of calls to `control` functions only; they have no effect on `simulate` behavior.
- `MCMC.prop.args.form`
 An alternative, direct way of specifying additional arguments to proposals.
- `MCMC.maxedges` The maximum number of edges that may occur during the MCMC sampling. If this number is exceeded at any time, sampling is stopped immediately.
- `MCMC.maxchanges`
 Maximum number of changes for which to allocate space.
- `term.options` A list of additional arguments to be passed to term initializers. See [? `term.options`](#).

MCMC.packagenames

Names of packages in which to look for change statistic functions in addition to those autodetected. This argument should not be needed outside of very strange setups.

Details

This function is only used within a call to the `simulate` function. See the usage section in `simulate.stergm` for details.

These functions are included for backwards compatibility, and users are encouraged to use `control.simulate.tergm` or `control.simulate.formula.tergm` with the `simulate.tergm` family of functions instead. When a `control.simulate.stergm` or `control.simulate.network` object is passed to one of the `simulate.stergm` functions, the corresponding `simulate.tergm` function is invoked, and uses the formation proposal control arguments, ignoring the dissolution proposal control arguments.

Note: The old dissolution formula in `stergm` represents tie persistence. As a result it maps to the new `Persist()` operator in `tergm`, NOT the `Diss()` operator

Value

A list with arguments as components.

See Also

`simulate.stergm`, `simulate.formula.control.stergm` performs a similar function for `stergm`.

control.simulate.tergm

Auxiliary for Controlling Temporal ERGM Simulation

Description

Auxiliary function as user interface for fine-tuning TERGM simulation.

Usage

```
control.simulate.tergm(
  MCMC.burnin.min = NULL,
  MCMC.burnin.max = NULL,
  MCMC.burnin.pval = NULL,
  MCMC.burnin.add = NULL,
  MCMC.prop = NULL,
  MCMC.prop.weights = NULL,
  MCMC.prop.args = NULL,
  MCMC.maxedges = NULL,
  MCMC.maxchanges = NULL,
  term.options = NULL,
  MCMC.packagenames = NULL
```

```

)

control.simulate.formula.tergm(
  MCMC.burnin.min = 1000,
  MCMC.burnin.max = 1e+05,
  MCMC.burnin.pval = 0.5,
  MCMC.burnin.add = 1,
  MCMC.prop = ~discord + sparse,
  MCMC.prop.weights = "default",
  MCMC.prop.args = NULL,
  MCMC.maxedges = Inf,
  MCMC.maxchanges = 1e+06,
  term.options = NULL,
  MCMC.packagenames = c()
)

```

Arguments

- `MCMC.burnin.min`, `MCMC.burnin.max`, `MCMC.burnin.pval`, `MCMC.burnin.add`
 Number of Metropolis-Hastings steps per time step used in simulation. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once `MCMC.burnin.min` steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than `MCMC.burnin.pval`), or `MCMC.burnin.max` steps are proposed, whichever comes first, the simulation is stopped after an additional `MCMC.burnin.add` times the number of elapsed steps have been taken. (Stopping immediately would bias the sampling.)
 To use a fixed number of steps, set `MCMC.burnin.min` and `MCMC.burnin.max` to the same value.
- `MCMC.prop` Hints and/or constraints for selecting and initializing the proposal.
- `MCMC.prop.weights`
 Specifies the proposal weighting scheme to be used in the MCMC Metropolis-Hastings algorithm. Possible choices may be determined by calling [ergm_proposal_table](#).
- `MCMC.prop.args` An alternative, direct way of specifying additional arguments to the proposal.
- `MCMC.maxedges` The maximum number of edges that may occur during the MCMC sampling. If this number is exceeded at any time, sampling is stopped immediately.
- `MCMC.maxchanges`
 Maximum number of changes for which to allocate space.
- `term.options` A list of additional arguments to be passed to term initializers. See [? term.options](#).
- `MCMC.packagenames`
 Names of packages in which to look for change statistic functions in addition to those autodetected. This argument should not be needed outside of very strange setups.

Details

This function is only used within a call to the [simulate](#) function. See the usage section in [simulate.tergm](#) for details.

Value

A list with arguments as components.

See Also

[simulate.tergm](#), [simulate.formula.control.tergm](#) performs a similar function for [tergm](#).

 control.stergm

Auxiliary for Controlling Separable Temporal ERGM Fitting

Description

Auxiliary function as user interface for fine-tuning 'stergm' fitting.

Usage

```
control.stergm(
  init.form = NULL,
  init.diss = NULL,
  init.method = NULL,
  force.main = FALSE,
  MCMC.prop.form = ~discord + sparse,
  MCMC.prop.diss = ~discord + sparse,
  MCMC.prop.weights.form = "default",
  MCMC.prop.args.form = NULL,
  MCMC.prop.weights.diss = "default",
  MCMC.prop.args.diss = NULL,
  MCMC.maxedges = Inf,
  MCMC.maxchanges = 1e+06,
  MCMC.packagenames = c(),
  CMLE.MCMC.burnin = 1024 * 16,
  CMLE.MCMC.interval = 1024,
  CMLE.ergm = NULL,
  CMLE.form.ergm = control.ergm(init = init.form, MCMC.burnin = CMLE.MCMC.burnin,
    MCMC.interval = CMLE.MCMC.interval, MCMC.prop = MCMC.prop.form, MCMC.prop.weights =
    MCMC.prop.weights.form, MCMC.prop.args = MCMC.prop.args.form, MCMC.maxedges =
    MCMC.maxedges, MCMC.packagenames = MCMC.packagenames, parallel = parallel,
    parallel.type = parallel.type, parallel.version.check = parallel.version.check,
    parallel.inherit.MT = parallel.inherit.MT, force.main = force.main),
  CMLE.diss.ergm = control.ergm(init = init.diss, MCMC.burnin = CMLE.MCMC.burnin,
    MCMC.interval = CMLE.MCMC.interval, MCMC.prop = MCMC.prop.diss, MCMC.prop.weights =
    MCMC.prop.weights.diss, MCMC.prop.args = MCMC.prop.args.diss, MCMC.maxedges =
```

```
MCMC.maxedges, MCMC.packagenames = MCMC.packagenames, parallel = parallel,
parallel.type = parallel.type, parallel.version.check = parallel.version.check,
  parallel.inherit.MT = parallel.inherit.MT, force.main = force.main),
CMLE.NA.impute = c(),
CMLE.term.check.override = FALSE,
EGMME.main.method = c("Gradient-Descent"),
EGMME.initialfit.control = control.ergm(),
EGMME.MCMC.burnin.min = 1000,
EGMME.MCMC.burnin.max = 1e+05,
EGMME.MCMC.burnin.pval = 0.5,
EGMME.MCMC.burnin.add = 1,
MCMC.burnin = NULL,
MCMC.burnin.mul = NULL,
SAN.maxit = 4,
SAN.nsteps.times = 8,
SAN = control.san(term.options = term.options, SAN.maxit = SAN.maxit, SAN.prop =
  MCMC.prop.form, SAN.prop.weights = MCMC.prop.weights.form, SAN.prop.args =
  MCMC.prop.args.form, SAN.nsteps = round(sqrt(EGMME.MCMC.burnin.min *
  EGMME.MCMC.burnin.max)) * SAN.nsteps.times, SAN.packagenames = MCMC.packagenames,
  parallel = parallel, parallel.type = parallel.type, parallel.version.check =
  parallel.version.check, parallel.inherit.MT = FALSE),
SA.restarts = 10,
SA.burnin = 1000,
SA.plot.progress = FALSE,
SA.max.plot.points = 400,
SA.plot.stats = FALSE,
SA.init.gain = 0.1,
SA.gain.decay = 0.5,
SA.runlength = 25,
SA.interval.mul = 2,
SA.init.interval = 500,
SA.min.interval = 20,
SA.max.interval = 500,
SA.phase1.minruns = 4,
SA.phase1.tries = 20,
SA.phase1.jitter = 0.1,
SA.phase1.max.q = 0.1,
SA.phase1.backoff.rat = 1.05,
SA.phase2.levels.max = 40,
SA.phase2.levels.min = 4,
SA.phase2.max.mc.se = 0.001,
SA.phase2.repeats = 400,
SA.stepdown.maxn = 200,
SA.stepdown.p = 0.05,
SA.stop.p = 0.1,
SA.stepdown.ct = 5,
SA.phase2.backoff.rat = 1.1,
SA.keep.oh = 0.5,
```

```

SA.keep.min.runs = 8,
SA.keep.min = 0,
SA.phase2.jitter.mul = 0.2,
SA.phase2.maxreljump = 4,
SA.guard.mul = 4,
SA.par.eff.pow = 1,
SA.robust = FALSE,
SA.oh.memory = 1e+05,
SA.refine = c("mean", "linear", "none"),
SA.se = TRUE,
SA.phase3.samplesize.runs = 10,
SA.restart.on.err = TRUE,
term.options = NULL,
seed = NULL,
parallel = 0,
parallel.type = NULL,
parallel.version.check = TRUE,
parallel.inherit.MT = FALSE,
...
)

```

Arguments

`init.form`, `init.diss`

numeric or NA vector equal in length to the number of parameters in the formation/dissolution model or NULL (the default); the initial values for the estimation and coefficient offset terms. If NULL is passed, all of the initial values are computed using the method specified by `control$init.method`. If a numeric vector is given, the elements of the vector are interpreted as follows:

- Elements corresponding to terms enclosed in `offset()` are used as the fixed offset coefficients. These should match the offset values given in `offset.coef.form` and `offset.coef.diss`.
- Elements that do not correspond to offset terms and are not NA are used as starting values in the estimation.
- Initial values for the elements that are NA are fit using the method specified by `control$init.method`.

Passing coefficients from a previous run can be used to "resume" an uncovered `stergm` run.

`init.method` Estimation method used to acquire initial values for estimation. If NULL (the default), the initial values are computed using the edges dissolution approximation (Carnegie et al.) when appropriate; note that this relies on `.extract.fd.formulae` to identify the formation and dissolution parts of the formula; the user should be aware of its behavior and limitations. If `init.method` is set to "zeros", the initial values are set to zeros.

`force.main` Logical: If TRUE, then force MCMC-based estimation method, even if the exact MLE can be computed via maximum pseudolikelihood estimation.

`MCMC.prop.form` Hints and/or constraints for selecting and initializing the proposal.

MCMC.prop.weights.form	Specifies the proposal weighting to use.
MCMC.prop.args.form	A direct way of specifying arguments to the proposal.
MCMC.prop.weights.diss, MCMC.prop.args.diss, MCMC.prop.diss	Ignored.
MCMC.maxedges	The maximum number of edges that may occur during the MCMC sampling. If this number is exceeded at any time, sampling is stopped immediately.
MCMC.maxchanges	Maximum number of changes in dynamic network simulation for which to allocate space.
MCMC.packagenames	Names of packages in which to look for change statistic functions in addition to those autodetected. This argument should not be needed outside of very strange setups.
CMLE.MCMC.burnin	Burnin used in CMLE fitting.
CMLE.MCMC.interval	Number of Metropolis-Hastings steps between successive draws when running MCMC MLE.
CMLE.ergm	A convenience argument for specifying both CMLE.form.ergm and CMLE.diss.ergm at once. See control.ergm .
CMLE.form.ergm	Control parameters used to fit the CMLE. See control.ergm .
CMLE.diss.ergm	Ignored, with the exception of initial parameter values.
CMLE.NA.impute	In STERGM CMLE, missing dyads in transitioned-to networks are accommodated using methods of Handcock and Gile (2009), but a similar approach to transitioned-from networks requires much more complex methods that are not, currently, implemented. CMLE.NA.impute controls how missing dyads in transitioned-from networks are be imputed. See argument <code>imputers</code> of impute.network.list for details. By default, no imputation is performed, and the fitting stops with an error if any transitioned-from networks have missing dyads.
CMLE.term.check.override	The method <code>stergm{stergm}</code> uses at this time to fit a series of more than two networks requires certain assumptions to be made about the ERGM terms being used, which are tested before a fit is attempted. This test sometimes fails despite the model being amenable to fitting, so setting this option to TRUE overrides the tests.
EGMME.main.method	Estimation method used to find the Equilibrium Generalized Method of Moments estimator. Currently only "Gradient-Descent" is implemented.
EGMME.initialfit.control	Control object for the ergm fit in <code>tergm.EGMME.initialfit</code>

EGMME.MCMC.burnin.min, EGMME.MCMC.burnin.max,	Number of Metropolis-Hastings steps per time step used in EGMME fitting. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once EGMME.MCMC.burnin.min steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than EGMME.MCMC.burnin.pval), or EGMME.MCMC.burnin.max steps are proposed, whichever comes first, the simulation is stopped after an additional EGMME.MCMC.burnin.add times the number of elapsed steps had been taken. (Stopping immediately would bias the sampling.) To use a fixed number of steps, set EGMME.MCMC.burnin.min and EGMME.MCMC.burnin.max to the same value.
EGMME.MCMC.burnin.pval, EGMME.MCMC.burnin.add	Number of Metropolis-Hastings steps per time step used in EGMME fitting. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once EGMME.MCMC.burnin.min steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than EGMME.MCMC.burnin.pval), or EGMME.MCMC.burnin.max steps are proposed, whichever comes first, the simulation is stopped after an additional EGMME.MCMC.burnin.add times the number of elapsed steps had been taken. (Stopping immediately would bias the sampling.) To use a fixed number of steps, set EGMME.MCMC.burnin.min and EGMME.MCMC.burnin.max to the same value.
MCMC.burnin, MCMC.burnin.mul	No longer used. See EGMME.MCMC.burnin.min, EGMME.MCMC.burnin.max, EGMME.MCMC.burnin.pval, EGMME.MCMC.burnin.add and CMLE.MCMC.burnin and CMLE.MCMC.interval.
SAN.maxit	When target.stats argument is passed to ergm() , the maximum number of attempts to use san to obtain a network with statistics close to those specified.
SAN.nsteps.times	Multiplier for SAN.nsteps relative to MCMC.burnin. This lets one control the amount of SAN burn-in (arguably, the most important of SAN parameters) without overriding the other SAN defaults.
SAN	SAN control parameters. See control.san
SA.restarts	Maximum number of times to restart a failed optimization process.
SA.burnin	Number of time steps to advance the starting network before beginning the optimization.
SA.plot.progress, SA.plot.stats	Logical: Plot information about the fit as it proceeds. If SA.plot.progress==TRUE, plot the trajectories of the parameters and target statistics as the optimization progresses. If SA.plot.stats==TRUE, plot a heatmap representing correlations of target statistics and a heatmap representing the estimated gradient. Do NOT use these with non-interactive plotting devices like pdf . (In fact, it will refuse to do that with a warning.)

- `SA.max.plot.points`
If `SA.plot.progress==TRUE`, the maximum number of time points to be plotted. Defaults to 400. If more iterations elapse, they will be thinned to at most 400 before plotting.
- `SA.init.gain` Initial gain, the multiplier for the parameter update size. If the process initially goes crazy beyond recovery, lower this value.
- `SA.gain.decay` Gain decay factor.
- `SA.runlength` Number of parameter trials and updates per C run.
- `SA.interval.mul`
The number of time steps between updates of the parameters is set to be this times the mean duration of extant ties.
- `SA.init.interval`
Initial number of time steps between updates of the parameters.
- `SA.min.interval`, `SA.max.interval`
Upper and lower bounds on the number of time steps between updates of the parameters.
- `SA.phase1.minruns`
Number of runs during Phase 1 for estimating the gradient, before every gradient update.
- `SA.phase1.tries`
Number of runs trying to find a reasonable parameter and network configuration.
- `SA.phase1.jitter`
Initial jitter standard deviation of each parameter.
- `SA.phase1.max.q`
Q-value (false discovery rate) that a gradient estimate must obtain before it is accepted (since sign is what is important).
- `SA.phase1.backoff.rat`, `SA.phase2.backoff.rat`
If the run produces this relative increase in the approximate objective function, it will be backed off.
- `SA.phase2.levels.min`, `SA.phase2.levels.max`
Range of gain levels (subphases) to go through.
- `SA.phase2.max.mc.se`
Approximate precision of the estimates that must be attained before stopping.
- `SA.phase2.repeats`, `SA.stepdown.maxn`,
A gain level may be repeated multiple times (up to `SA.phase2.repeats`) if the optimizer detects that the objective function is improving or the estimating equations are not centered around 0, so slowing down the parameters at that point is counterproductive. To detect this it looks at the the window controlled by `SA.keep.oh`, thinning objective function values to get `SA.stepdown.maxn`, and 1) fitting a GLS model for a linear trend, with AR(2) autocorrelation and 2) conducting an approximate Hotelling's T^2 test for equality of estimating equation values to 0. If there is no significance for either at `SA.stepdown.p` `SA.stepdown.ct` runs in a row, the gain level (subphase) is allowed to end. Otherwise, the process continues at the same gain level.

SA.stepdown.p, SA.stepdown.ct	<p>A gain level may be repeated multiple times (up to SA.phase2.repeats) if the optimizer detects that the objective function is improving or the estimating equations are not centered around 0, so slowing down the parameters at that point is counterproductive. To detect this it looks at the the window controlled by SA.keep.oh, thinning objective function values to get SA.stepdown.maxn, and 1) fitting a GLS model for a linear trend, with AR(2) autocorrelation and 2) conducting an approximate Hotelling's T² test for equality of estimating equation values to 0. If there is no significance for either at SA.stepdown.p SA.stepdown.ct runs in a row, the gain level (subphase) is allowed to end. Otherwise, the process continues at the same gain level.</p>
SA.stop.p	<p>At the end of each gain level after the minimum, if the precision is sufficiently high, the relationship between the parameters and the targets is tested for evidence of local nonlinearity. This is the p-value used.</p> <p>If that test fails to reject, a Phase 3 run is made with the new parameter values, and the estimating equations are tested for difference from 0. If this test fails to reject, the optimization is finished.</p> <p>If either of these tests rejects, at SA.stop.p, optimization is continued for another gain level.</p>
SA.keep.oh, SA.keep.min, SA.keep.min.runs	<p>Parameters controlling how much of optimization history to keep for gradient and covariance estimation.</p> <p>A history record will be kept if it's at least one of the following:</p> <ul style="list-style-type: none"> • Among the last SA.keep.oh (a fraction) of all runs. • Among the last SA.keep.min (a count) records. • From the last SA.keep.min.runs (a count) optimization runs.
SA.phase2.jitter.mul	<p>Jitter standard deviation of each parameter is this value times its standard deviation without jitter.</p>
SA.phase2.maxreljump	<p>To keep the optimization from "running away" due to, say, a poor gradient estimate building on itself, if a magnitude of change (Mahalanobis distance) in parameters over the course of a run divided by average magnitude of change for recent runs exceeds this, the change is truncated to this amount times the average for recent runs.</p>
SA.guard.mul	<p>The multiplier for the range of parameter and statistics values to compute the guard width.</p>
SA.par.eff.pow	<p>Because some parameters have much, much greater effects than others, it improves numerical conditioning and makes estimation more stable to rescale the kth estimating function by $s_k = (\sum_{i=1}^q G_{i,k}^2 / V_{i,i})^{-p/2}$, where $G_{i,k}$ is the estimated gradient of the ith target statistics with respect to kth parameter. This parameter sets the value of p: 0 for no rescaling, 1 (default) for scaling by root-mean-square normalized gradient, and greater values for greater penalty.</p>
SA.robust	<p>Whether to use robust linear regression (for gradients) and covariance estimation.</p>

SA.oh.memory	Absolute maximum number of data points per thread to store in the full optimization history.
SA.refine	Method, if any, used to refine the point estimate at the end: "linear" for linear interpolation, "mean" for average, and "none" to use the last value.
SA.se	Logical: If TRUE (the default), get an MCMC sample of statistics at the final estimate and compute the covariance matrix (and hence standard errors) of the parameters. This sample is stored and can also be used by <code>mcmc.diagnostics()</code> to assess convergence.
SA.phase3.samplesize.runs	This many optimization runs will be used to determine whether the optimization has converged and to estimate the standard errors.
SA.restart.on.err	Logical: if TRUE (the default) an error somewhere in the optimization process will cause it to restart with a smaller gain value. Otherwise, the process will stop. This is mainly used for debugging
term.options	A list of additional arguments to be passed to term initializers. See <code>?term.options</code> .
seed	Seed value (integer) for the random number generator. See <code>set.seed</code> .
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 <code>parallel</code> package with PSOCK clusters. See ergm-parallel
parallel.version.check	Logical: If TRUE, check that the version of <code>ergm</code> 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 <code>set.MT_terms()</code> setting.
...	Additional arguments, passed to other functions This argument is helpful because it collects any control parameters that have been deprecated; a warning message is printed in case of deprecated arguments.

Details

This function is only used within a call to the `stergm` function. See the usage section in `stergm` for details. Generally speaking, `control.stergm` is remapped to `control.tergm`, with dissolution controls ignored and formation controls used as controls for the overall `tergm` process. An exception to this rule is the initial parameter values specified via `init.form`, `init.diss`, `CMLE.form.ergm$init`, and `CMLE.diss.ergm$init`, which will be remapped jointly with the `stergm()` arguments `offset.coef.form` and `offset.coef.diss` to determine the initial parameter values passed to `tergm`.

It is recommended that new code make use of `tergm` and `control.tergm` directly; `stergm` wrappers are included only for backwards compatibility.

Value

A list with arguments as components.

References

- Boer, P., Huisman, M., Snijders, T.A.B., and Zeggelink, E.P.H. (2003), StOCNET User's Manual. Version 1.4.
- Firth (1993), Bias Reduction in Maximum Likelihood Estimates. *Biometrika*, 80: 27-38.
- Hunter, D. R. and M. S. Handcock (2006), Inference in curved exponential family models for networks. *Journal of Computational and Graphical Statistics*, 15: 565-583.
- Hummel, R. M., Hunter, D. R., and Handcock, M. S. (2010), A Steplength Algorithm for Fitting ERGMs, Penn State Department of Statistics Technical Report.

See Also

[stergm](#), [tergm](#), [control.tergm](#). The [control.simulate.stergm](#) function performs a similar function for [simulate.tergm](#).

control.tergm

Auxiliary for Controlling Temporal ERGM Fitting

Description

Auxiliary function as user interface for fine-tuning 'tergm' fitting.

Usage

```
control.tergm(
  init = NULL,
  init.method = NULL,
  force.main = FALSE,
  MCMC.prop = ~discord + sparse,
  MCMC.prop.weights = "default",
  MCMC.prop.args = NULL,
  MCMC.maxedges = Inf,
  MCMC.maxchanges = 1e+06,
  MCMC.packagenames = c(),
  CMLE.MCMC.burnin = 1024 * 16,
  CMLE.MCMC.interval = 1024,
  CMLE.ergm = control.ergm(init = init, MCMC.burnin = CMLE.MCMC.burnin, MCMC.interval =
    CMLE.MCMC.interval, MCMC.prop = MCMC.prop, MCMC.prop.weights = MCMC.prop.weights,
    MCMC.prop.args = MCMC.prop.args, MCMC.maxedges = MCMC.maxedges, MCMC.packagenames =
    MCMC.packagenames, parallel = parallel, parallel.type = parallel.type,
    parallel.version.check = parallel.version.check, force.main = force.main),
  CMLE.NA.impute = c(),
  CMLE.term.check.override = FALSE,
  EGMME.main.method = c("Gradient-Descent"),
  EGMME.initialfit.control = control.ergm(),
  EGMME.MCMC.burnin.min = 1000,
  EGMME.MCMC.burnin.max = 1e+05,
```

```
EGMME.MCMC.burnin.pval = 0.5,
EGMME.MCMC.burnin.add = 1,
MCMC.burnin = NULL,
MCMC.burnin.mul = NULL,
SAN.maxit = 4,
SAN.nsteps.times = 8,
SAN = control.san(term.options = term.options, SAN.maxit = SAN.maxit, SAN.prop =
  MCMC.prop, SAN.prop.weights = MCMC.prop.weights, SAN.prop.args = MCMC.prop.args,
  SAN.nsteps = round(sqrt(EGMME.MCMC.burnin.min * EGMME.MCMC.burnin.max)) *
  SAN.nsteps.times, SAN.packagenames = MCMC.packagenames, parallel = parallel,
  parallel.type = parallel.type, parallel.version.check = parallel.version.check,
  parallel.inherit.MT = parallel.inherit.MT),
SA.restarts = 10,
SA.burnin = 1000,
SA.plot.progress = FALSE,
SA.max.plot.points = 400,
SA.plot.stats = FALSE,
SA.init.gain = 0.1,
SA.gain.decay = 0.5,
SA.runlength = 25,
SA.interval.mul = 2,
SA.init.interval = 500,
SA.min.interval = 20,
SA.max.interval = 500,
SA.phase1.minruns = 4,
SA.phase1.tries = 20,
SA.phase1.jitter = 0.1,
SA.phase1.max.q = 0.1,
SA.phase1.backoff.rat = 1.05,
SA.phase2.levels.max = 40,
SA.phase2.levels.min = 4,
SA.phase2.max.mc.se = 0.001,
SA.phase2.repeats = 400,
SA.stepdown.maxn = 200,
SA.stepdown.p = 0.05,
SA.stop.p = 0.1,
SA.stepdown.ct = 5,
SA.phase2.backoff.rat = 1.1,
SA.keep.oh = 0.5,
SA.keep.min.runs = 8,
SA.keep.min = 0,
SA.phase2.jitter.mul = 0.2,
SA.phase2.maxreljump = 4,
SA.guard.mul = 4,
SA.par.eff.pow = 1,
SA.robust = FALSE,
SA.oh.memory = 1e+05,
SA.refine = c("mean", "linear", "none"),
```

```

SA.se = TRUE,
SA.phase3.samplesize.runs = 10,
SA.restart.on.err = TRUE,
term.options = NULL,
seed = NULL,
parallel = 0,
parallel.type = NULL,
parallel.version.check = TRUE,
parallel.inherit.MT = FALSE
)

```

Arguments

<code>init</code>	<p>numeric or NA vector equal in length to the number of parameters in the model or NULL (the default); the initial values for the estimation and coefficient offset terms. If NULL is passed, all of the initial values are computed using the method specified by <code>control\$init.method</code>. If a numeric vector is given, the elements of the vector are interpreted as follows:</p> <ul style="list-style-type: none"> • Elements corresponding to terms enclosed in <code>offset()</code> are used as the fixed offset coefficients. These should match the offset values given in <code>offset.coef</code>. • Elements that do not correspond to offset terms and are not NA are used as starting values in the estimation. • Initial values for the elements that are NA are fit using the method specified by <code>control\$init.method</code>. <p>Passing coefficients from a previous run can be used to "resume" an uncovered <code>tergm</code> run.</p>
<code>init.method</code>	<p>Estimation method used to acquire initial values for estimation. If NULL (the default), the initial values are computed using the edges dissolution approximation (Carnegie et al.) when appropriate; note that this relies on <code>.extract.fd.formulae</code> to identify the formation and dissolution parts of the formula; the user should be aware of its behavior and limitations. If <code>init.method</code> is set to "zeros", the initial values are set to zeros.</p>
<code>force.main</code>	<p>Logical: If TRUE, then force MCMC-based estimation method, even if the exact MLE can be computed via maximum pseudolikelihood estimation.</p>
<code>MCMC.prop</code>	<p>Hints and/or constraints for selecting and initializing the proposal.</p>
<code>MCMC.prop.weights</code>	<p>Specifies the proposal weighting to use.</p>
<code>MCMC.prop.args</code>	<p>A direct way of specifying arguments to the proposal.</p>
<code>MCMC.maxedges</code>	<p>The maximum number of edges that may occur during the MCMC sampling. If this number is exceeded at any time, sampling is stopped immediately.</p>
<code>MCMC.maxchanges</code>	<p>Maximum number of changes permitted to occur during the simulation.</p>
<code>MCMC.packagenames</code>	<p>Names of packages in which to look for change statistic functions in addition to those autodetected. This argument should not be needed outside of very strange setups.</p>

- `CMLE.MCMC.burnin`
Burnin used in CMLE fitting.
- `CMLE.MCMC.interval`
Number of Metropolis-Hastings steps between successive draws when running MCMC MLE.
- `CMLE.ergm`
Control parameters used to fit the CMLE. See [control.ergm](#).
- `CMLE.NA.impute`
In TERGM CMLE, missing dyads in transitioned-to networks are accommodated using methods of Handcock and Gile (2009), but a similar approach to transitioned-from networks requires much more complex methods that are not, currently, implemented. `CMLE.NA.impute` controls how missing dyads in transitioned-from networks are be imputed. See argument `imputers` of [impute.network.list](#) for details.
By default, no imputation is performed, and the fitting stops with an error if any transitioned-from networks have missing dyads.
- `CMLE.term.check.override`
The method `tergm{tergm}` uses at this time to fit a series of more than two networks requires certain assumptions to be made about the ERGM terms being used, which are tested before a fit is attempted. This test sometimes fails despite the model being amenable to fitting, so setting this option to TRUE overrides the tests.
- `EGMME.main.method`
Estimation method used to find the Equilibrium Generalized Method of Moments estimator. Currently only "Gradient-Descent" is implemented.
- `EGMME.initialfit.control`
Control object for the `ergm` fit in `tergm.EGMME.initialfit`
- `EGMME.MCMC.burnin.min`, `EGMME.MCMC.burnin.max`,
Number of Metropolis-Hastings steps per time step used in EGMME fitting. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once `EGMME.MCMC.burnin.min` steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than `EGMME.MCMC.burnin.pval`), or `EGMME.MCMC.burnin.max` steps are proposed, whichever comes first, the simulation is stopped after an additional `EGMME.MCMC.burnin.add` times the number of elapsed steps had been taken. (Stopping immediately would bias the sampling.)
To use a fixed number of steps, set `EGMME.MCMC.burnin.min` and `EGMME.MCMC.burnin.max` to the same value.
- `EGMME.MCMC.burnin.pval`, `EGMME.MCMC.burnin.add`
Number of Metropolis-Hastings steps per time step used in EGMME fitting. By default, this is determined adaptively by keeping track of increments in the Hamming distance between the transitioned-from network and the network being sampled. Once `EGMME.MCMC.burnin.min` steps have elapsed, the increments are tested against 0, and when their average number becomes statistically indistinguishable from 0 (with the p-value being greater than `EGMME.MCMC.burnin.pval`), or `EGMME.MCMC.burnin.max` steps are proposed, whichever comes first, the simulation is stopped after an additional `EGMME.MCMC.burnin.add` times the num-

	ber of elapsed steps had been taken. (Stopping immediately would bias the sampling.)
	To use a fixed number of steps, set <code>EGMME.MCMC.burnin.min</code> and <code>EGMME.MCMC.burnin.max</code> to the same value.
<code>MCMC.burnin</code> , <code>MCMC.burnin.mul</code>	No longer used. See <code>EGMME.MCMC.burnin.min</code> , <code>EGMME.MCMC.burnin.max</code> , <code>EGMME.MCMC.burnin.pval</code> , <code>EGMME.MCMC.burnin.pval</code> , <code>EGMME.MCMC.burnin.add</code> and <code>CMLE.MCMC.burnin</code> and <code>CMLE.MCMC.interval</code> .
<code>SAN.maxit</code>	When <code>target.stats</code> argument is passed to <code>ergm()</code> , the maximum number of attempts to use <code>san</code> to obtain a network with statistics close to those specified.
<code>SAN.nsteps.times</code>	Multiplier for <code>SAN.nsteps</code> relative to <code>MCMC.burnin</code> . This lets one control the amount of SAN burn-in (arguably, the most important of SAN parameters) without overriding the other SAN defaults.
<code>SAN</code>	SAN control parameters. See <code>control.san</code>
<code>SA.restarts</code>	Maximum number of times to restart a failed optimization process.
<code>SA.burnin</code>	Number of time steps to advance the starting network before beginning the optimization.
<code>SA.plot.progress</code> , <code>SA.plot.stats</code>	Logical: Plot information about the fit as it proceeds. If <code>SA.plot.progress==TRUE</code> , plot the trajectories of the parameters and target statistics as the optimization progresses. If <code>SA.plot.stats==TRUE</code> , plot a heatmap representing correlations of target statistics and a heatmap representing the estimated gradient. Do NOT use these with non-interactive plotting devices like <code>pdf</code> . (In fact, it will refuse to do that with a warning.)
<code>SA.max.plot.points</code>	If <code>SA.plot.progress==TRUE</code> , the maximum number of time points to be plotted. Defaults to 400. If more iterations elapse, they will be thinned to at most 400 before plotting.
<code>SA.init.gain</code>	Initial gain, the multiplier for the parameter update size. If the process initially goes crazy beyond recovery, lower this value.
<code>SA.gain.decay</code>	Gain decay factor.
<code>SA.runlength</code>	Number of parameter trials and updates per C run.
<code>SA.interval.mul</code>	The number of time steps between updates of the parameters is set to be this times the mean duration of extant ties.
<code>SA.init.interval</code>	Initial number of time steps between updates of the parameters.
<code>SA.min.interval</code> , <code>SA.max.interval</code>	Upper and lower bounds on the number of time steps between updates of the parameters.
<code>SA.phase1.minruns</code>	Number of runs during Phase 1 for estimating the gradient, before every gradient update.

- SA.phase1.tries
Number of runs trying to find a reasonable parameter and network configuration.
- SA.phase1.jitter
Initial jitter standard deviation of each parameter.
- SA.phase1.max.q
Q-value (false discovery rate) that a gradient estimate must obtain before it is accepted (since sign is what is important).
- SA.phase1.backoff.rat, SA.phase2.backoff.rat
If the run produces this relative increase in the approximate objective function, it will be backed off.
- SA.phase2.levels.min, SA.phase2.levels.max
Range of gain levels (subphases) to go through.
- SA.phase2.max.mc.se
Approximate precision of the estimates that must be attained before stopping.
- SA.phase2.repeats, SA.stepdown.maxn,
A gain level may be repeated multiple times (up to SA.phase2.repeats) if the optimizer detects that the objective function is improving or the estimating equations are not centered around 0, so slowing down the parameters at that point is counterproductive. To detect this it looks at the the window controlled by SA.keep.oh, thinning objective function values to get SA.stepdown.maxn, and 1) fitting a GLS model for a linear trend, with AR(2) autocorrelation and 2) conducting an approximate Hotelling's T^2 test for equality of estimating equation values to 0. If there is no significance for either at SA.stepdown.p SA.stepdown.ct runs in a row, the gain level (subphase) is allowed to end. Otherwise, the process continues at the same gain level.
- SA.stepdown.p, SA.stepdown.ct
A gain level may be repeated multiple times (up to SA.phase2.repeats) if the optimizer detects that the objective function is improving or the estimating equations are not centered around 0, so slowing down the parameters at that point is counterproductive. To detect this it looks at the the window controlled by SA.keep.oh, thinning objective function values to get SA.stepdown.maxn, and 1) fitting a GLS model for a linear trend, with AR(2) autocorrelation and 2) conducting an approximate Hotelling's T^2 test for equality of estimating equation values to 0. If there is no significance for either at SA.stepdown.p SA.stepdown.ct runs in a row, the gain level (subphase) is allowed to end. Otherwise, the process continues at the same gain level.
- SA.stop.p
At the end of each gain level after the minimum, if the precision is sufficiently high, the relationship between the parameters and the targets is tested for evidence of local nonlinearity. This is the p-value used.
If that test fails to reject, a Phase 3 run is made with the new parameter values, and the estimating equations are tested for difference from 0. If this test fails to reject, the optimization is finished.
If either of these tests rejects, at SA.stop.p, optimization is continued for another gain level.
- SA.keep.oh, SA.keep.min, SA.keep.min.runs
Parameters controlling how much of optimization history to keep for gradient and covariance estimation.

A history record will be kept if it's at least one of the following:

- Among the last `SA.keep.oh` (a fraction) of all runs.
- Among the last `SA.keep.min` (a count) records.
- From the last `SA.keep.min.runs` (a count) optimization runs.

<code>SA.phase2.jitter.mul</code>	Jitter standard deviation of each parameter is this value times its standard deviation without jitter.
<code>SA.phase2.maxreljump</code>	To keep the optimization from "running away" due to, say, a poor gradient estimate building on itself, if a magnitude of change (Mahalanobis distance) in parameters over the course of a run divided by average magnitude of change for recent runs exceeds this, the change is truncated to this amount times the average for recent runs.
<code>SA.guard.mul</code>	The multiplier for the range of parameter and statistics values to compute the guard width.
<code>SA.par.eff.pow</code>	Because some parameters have much, much greater effects than others, it improves numerical conditioning and makes estimation more stable to rescale the k th estimating function by $s_k = (\sum_{i=1}^q G_{i,k}^2 / V_{i,i})^{-p/2}$, where $G_{i,k}$ is the estimated gradient of the i th target statistics with respect to k th parameter. This parameter sets the value of p : 0 for no rescaling, 1 (default) for scaling by root-mean-square normalized gradient, and greater values for greater penalty.
<code>SA.robust</code>	Whether to use robust linear regression (for gradients) and covariance estimation.
<code>SA.oh.memory</code>	Absolute maximum number of data points per thread to store in the full optimization history.
<code>SA.refine</code>	Method, if any, used to refine the point estimate at the end: "linear" for linear interpolation, "mean" for average, and "none" to use the last value.
<code>SA.se</code>	Logical: If TRUE (the default), get an MCMC sample of statistics at the final estimate and compute the covariance matrix (and hence standard errors) of the parameters. This sample is stored and can also be used by <code>mcmc.diagnostics()</code> to assess convergence.
<code>SA.phase3.samplesize.runs</code>	This many optimization runs will be used to determine whether the optimization has converged and to estimate the standard errors.
<code>SA.restart.on.err</code>	Logical: if TRUE (the default) an error somewhere in the optimization process will cause it to restart with a smaller gain value. Otherwise, the process will stop. This is mainly used for debugging
<code>term.options</code>	A list of additional arguments to be passed to term initializers. See ? term.options .
<code>seed</code>	Seed value (integer) for the random number generator. See set.seed .
<code>parallel</code>	Number of threads in which to run the sampling. Defaults to 0 (no parallelism). See the entry on parallel processing for details and troubleshooting.
<code>parallel.type</code>	API to use for parallel processing. Supported values are "MPI" and "PSOCK". Defaults to using the <code>parallel</code> 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

This function is only used within a call to the [tergm](#) function. See the usage section in [tergm](#) for details.

Value

A list with arguments as components.

References

Boer, P., Huisman, M., Snijders, T.A.B., and Zeggelink, E.P.H. (2003), StOCNET User's Manual. Version 1.4.

Firth (1993), Bias Reduction in Maximum Likelihood Estimates. *Biometrika*, 80: 27-38.

Hunter, D. R. and M. S. Handcock (2006), Inference in curved exponential family models for networks. *Journal of Computational and Graphical Statistics*, 15: 565-583.

Hummel, R. M., Hunter, D. R., and Handcock, M. S. (2010), A Steplength Algorithm for Fitting ERGMs, Penn State Department of Statistics Technical Report.

See Also

[tergm](#). The [control.simulate.tergm](#) function performs a similar function for [simulate.tergm](#).

control.tergm.godfather

Control parameters for [tergm.godfather\(\)](#).

Description

Returns a list of its arguments.

Usage

```
control.tergm.godfather(term.options = NULL)
```

Arguments

`term.options` A list of additional arguments to be passed to term initializers. See [? term.options](#).

 Cross-ergmTerm

The Crossection Operator Term

Description

The Crossection Operator Term

Usage

```
# binary: Cross(formula)
```

Arguments

formula a one-sided [ergm\(\)](#)-style formula with the terms to be evaluated

Details

This term accepts a model formula and produces the corresponding model for the cross-sectional network. It is mainly useful for CMLE estimation, and has no effect (i.e., `Cross(~TERM) == ~TERM`) for EGMME and dynamic simulation.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

 degrange.mean.age-ergmTerm

Average age of ties incident on nodes having degree in a given range

Description

Average age of ties incident on nodes having degree in a given range

Usage

```
# binary: degrange.mean.age(from, to=+Inf, byarg=NULL, emptyval=0)
```

Arguments

from, to	vectors of distinct integers or +Inf , for to . If one of the vectors has length 1, it is recycled to the length of the other. Otherwise, they must have the same length.
byarg	specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.). If specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
emptyval	can be used to specify the value returned if the network does not have any actors with degree in the specified range. This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest has no actors with specified degree.

Details

This term adds one network statistic to the model for each element of from (or to); the i th such statistic equals the average, among all ties incident on nodes with degree greater than or equal to from[i] but strictly less than to[i] , of the amount of time elapsed since the tie's formation. The optional argument

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

degree.mean.age-ergmTerm

Average age of ties incident on nodes having a given degree

Description

Average age of ties incident on nodes having a given degree

Usage

```
# binary: degree.mean.age(d, byarg=NULL, emptyval=0)
```

Arguments

d	a vector of distinct integers
byarg	specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.). If specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.
emptyval	can be used to specify the value returned if the network does not have any actors with degree in the specified range. This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest has no actors with specified degree.

Details

This term adds one network statistic to the model for each element in `d`; the i th such statistic equals the average, among all ties incident on nodes with degree exactly `d[i]`, of the amount of time elapsed since the tie's formation. The optional argument `byarg` specifies a vertex attribute (see [Specifying Vertex Attributes and Levels](#) for details). If specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

`discord-ergmHint` *Discordant dyads*

Description

Propose toggling discordant dyads with greater frequency (typically about 50 percent). May be used in dynamic fitting and simulation.

Usage

```
# discord
```

See Also

[ergmHint](#) for index of constraints and hints currently visible to the package.

Keywords: None

`Diss-ergmTerm` *The Dissolution Operator Term*

Description

The Dissolution Operator Term

Usage

```
# binary: Diss(formula)
```

Arguments

`formula` a one-sided [ergm\(\)](#)-style formula with the terms to be evaluated

Details

This term accepts a model formula and produces the corresponding model for the post-dissolution network (same as `Persist()`), but with all statistics negated.

Note: This is not the equivalent of the old style dissolution model, because the signs of the coefficients are reversed. So a larger positive coefficient for `Diss()` operator means more dissolution.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

edge.ages-ergmTerm	<i>Sum of ages of extant ties</i>
--------------------	-----------------------------------

Description

Sum of ages of extant ties

Usage

```
# binary: edge.ages
```

Details

This term adds one statistic equaling sum, over all ties present in the network, of the amount of time elapsed since formation.

Unlike [mean.age](#), this statistic is well-defined on an empty network. However, if used as a target, it appears to produce highly biased dissolution parameter estimates if the goal is to get an intended average duration.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

EdgeAges-ergmTerm *The EdgeAges Operator Term*

Description

The EdgeAges Operator Term

Usage

```
# binary: EdgeAges(formula)
```

Arguments

formula cross-sectional, dyad-independent model formula

Details

This term accepts a cross-sectional, dyad-independent model formula. The statistics of the EdgeAges term are equal to the sum over all extant ties of the tie age times the on-toggle change statistics for the tie under the given model formula.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

edgecov.ages-ergmTerm *Weighted sum of ages of extant ties*

Description

Weighted sum of ages of extant ties

Usage

```
# binary: edgecov.ages(x, attrname=NULL)
```

Arguments

x, attrname either a square matrix of covariates, one for each possible edge in the network, the name of a network attribute of covariates, or a network; if the latter, or if the network attribute named by x is itself a network, optional argument attrname provides the name of the quantitative edge attribute to use for covariate values (in this case, missing edges in x are assigned a covariate value of zero).

Details

This term adds one statistic equaling sum, over all ties present in the network, of the amount of time elapsed since formation, multiplied by a dyadic covariate.

"Weights" can be negative.

Unlike [edgecov.mean.age](#), this statistic is well-defined on an empty network. However, if used as a target, it appears to produce highly biased dissolution parameter estimates if the goal is to get an intended average duration.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

edgecov.mean.age-ergmTerm

Weighted average age of an extant tie

Description

Weighted average age of an extant tie

Usage

```
# binary: edgecov.mean.age(x, attrname=NULL, emptyval=0)
```

Arguments

<code>x, attrname</code>	either a square matrix of covariates, one for each possible edge in the network, the name of a network attribute of covariates, or a network; if the latter, or if the network attribute named by <code>x</code> is itself a network, optional argument <code>attrname</code> provides the name of the quantitative edge attribute to use for covariate values (in this case, missing edges in <code>x</code> are assigned a covariate value of zero).
<code>emptyval</code>	can be used to specify the value returned if the network is empty (or all extant edges have been weighted 0). This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest is empty and/or if only a few dyads have nonzero weights.

Details

This term adds one statistic equaling the average, over all ties present in the network, of the amount of time elapsed since formation, weighted by a (nonnegative) dyadic covariate.

The behavior when there are negative weights is undefined.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

edges.ageinterval-ergmTerm

Number of edges with age falling into a specified range

Description

Number of edges with age falling into a specified range

Usage

```
# binary: edges.ageinterval(from, to=+Inf)
```

Arguments

from, to parameters to specify the lower bound and strict upper bounds. Can be scalars, vectors of the same length, or one of them must have length one, in which case it is recycled.

Details

This term counts the number of edges in the network for which the time elapsed since formation is greater than or equal to from but strictly less than to . In other words, it is in the semiopen interval [from, to) .

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

Form-ergmTerm	<i>The Formation Operator Term</i>
---------------	------------------------------------

Description

The Formation Operator Term

Usage

```
# binary: Form(formula)
```

Arguments

formula a one-sided `ergm()`-style formula with the terms to be evaluated

Details

This term accepts a model formula and produces the corresponding model for the post-formation network: effectively a network containing both previous time step's ties and ties just formed, the union of the previous and current network. This is the equivalent of the old-style `formation` model.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

<code>impute.network.list</code>	<i>Impute missing dyads in a series of networks</i>
----------------------------------	---

Description

This function takes a list of networks with missing dyads and returns a list of networks with missing dyads imputed according to a list of imputation directives.

Usage

```
impute.network.list(  
  nwl,  
  imputers = c(),  
  nwl.prepend = list(),  
  nwl.append = list()  
)
```

Arguments

- `nw1` A list of `network` objects or a `network.list` object.
- `imputers` A character vector giving one or more methods to impute missing dyads. Currently implemented methods are as follows:
- `next` Impute the state of the same dyad in the next network in the list (or later, if that one is also missing). This imputation method is likely to lead to an underestimation of the tie-change rates. The last network in the list cannot be imputed this way.
 - `previous` Impute the state of the same dyad in the previous network in the list (or earlier, if that one is also missing). The first network in the list cannot be imputed this way.
 - `majority` Impute the missing dyad with the value of the majority among the non-missing dyads in that time step's network. A network that has exactly the same number of ties as non-missing non-ties cannot be imputed this way.
 - `0` Assume missing dyads are all non-ties.
 - `1` Assume missing dyads are all ties.
- If `length(imputers)>1` the specified imputation methods will be applied in succession. For example, `imputers=c("next", "previous", "majority", "0")` would first try to impute a missing dyad with the next time step's value. If it, and all of the later values for that dyad are missing, it will try to impute it with the previous time step's value. If it, and all of the earlier values for that dyad are missing as well, it will try to impute it with the value of the majority of non-missing dyads for that time step. If there is an exact tie, it will impute 0.
- `nw1.prepend` An optional list of networks to treat as preceding those in `nw1`. They will not be imputed or returned, but they can be useful for imputing dyads in the first network in `nw1`, when using "previous" imputer.
- `nw1.append` An optional list of networks to treat as following those in `nw1`. They will not be imputed or returned, but they can be useful for imputing dyads in the last network in `nw1`, when using "next" imputer.

Value

A list of networks with missing dyads imputed.

See Also

[network](#), [is.na](#)

is.durational *Testing for duration dependent models*

Description

These functions test whether an ERGM is duration dependent or not.

The method for NULL always returns FALSE by convention.

Usage

```
is.durational(object, ...)  
  
## S3 method for class ``NULL``  
is.durational(object, ...)  
  
## S3 method for class 'ergm_model'  
is.durational(object, ...)  
  
## S3 method for class 'ergm_state'  
is.durational(object, ...)  
  
## S3 method for class 'formula'  
is.durational(object, response = NULL, basis = ergm.getnetwork(object), ...)
```

Arguments

object	An ERGM formula, ergm_model object, or ergm_state object.
...	Unused at this time.
response, basis	See ergm() .

Value

TRUE if the ERGM terms in the model are duration dependent; FALSE otherwise.

Methods (by class)

- `is.durational(ergm_model)`: Test if the [ergm_model](#) has duration-dependent terms, which call for [lasttoggle](#) data structures.
- `is.durational(ergm_state)`: Test if the [ergm_state](#) has duration-dependent terms, which call for [lasttoggle](#) data structures.

lasttoggle	<i>Lasttoggle</i>
------------	-------------------

Description

A data structure used by `tergm` for tracking of limited information about dyad edge histories.

Details

The `tergm` package handles durational information attached to `network` objects by way of the `time` and `lasttoggle` network attributes. The `lasttoggle` data structure is a 3-column matrix; the first two columns are tails and heads (respectively) of dyads, and the third column is the last time at which the dyad was toggled. The default last toggle time is `-INT_MAX/2`. Last toggle times for non-edges are periodically cleared in the C code. The `time` network attribute is simply an integer, and together with the `lasttoggle` data it determines the age of an extant tie as `time + 1` minus the last toggle time for that dyad. The default value for `time` is 0.

mean.age-ergmTerm	<i>Average age of an extant tie</i>
-------------------	-------------------------------------

Description

Average age of an extant tie

Usage

```
# binary: mean.age(emptyval=0, log=FALSE)
```

Arguments

<code>emptyval</code>	can be used to specify the value returned if the network is empty. This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest is empty.
<code>log</code>	logical specifying if mean log age should be returned instead of mean age

Details

This term adds one statistic equaling the average, over all ties present in the network, of the amount of time elapsed since formation.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

NetSeries *A network series specification for conditional modeling.*

Description

A function for specifying the LHS of a temporal network series ERGM.

Usage

```
NetSeries(..., order = 1, NA.impute = NULL)
```

Arguments

...	series specification, in one of three formats: <ol style="list-style-type: none"> 1. A list of identically- dimensioned and directed networks. 2. Several networks as arguments. 3. A <code>networkDynamic</code> object and a numeric vector of time indices.
order	how many previous networks to store as an accessible covariate of the model.
NA.impute	How missing dyads in transitioned-from networks are be imputed when using conditional estimation. See argument <code>imputers</code> of <code>impute.network.list</code> for details.

Value

A network object with temporal metadata.

Note

It is not recommended to modify the network returned by `NetSeries` except by adding and removing edges, and even that must be done with some care, to avoid putting it into an inconsistent state.

It is almost always better to modify the original networks and regenerate the series.

See Also

[Help on model specification](#) for specific terms.

Examples

```
data(samp1k)

# Method 1: list of networks
monks <- NetSeries(list(samp1k1, samp1k2, samp1k3))
ergm(monks ~ Form(~edges)+Diss(~edges))
ergm(monks ~ Form(~edges)+Persist(~edges))

# Method 2: networks as arguments
```

```

monks <- NetSeries(samplk1,samplk2,samplk3)
ergm(monks ~ Form(~edges)+Diss(~edges))
ergm(monks ~ Form(~edges)+Persist(~edges))

# Method 3: networkDynamic and time points:
## TODO

```

```
nodefactor.mean.age-ergmTerm
```

Average ages of extant half-ties incident on nodes of specified attribute levels

Description

Average ages of extant half-ties incident on nodes of specified attribute levels

Usage

```
# binary: nodefactor.mean.age(attr, levels=NULL, emptyval=0, log=FALSE)
```

Arguments

<code>attr</code>	a vertex attribute specification (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
<code>levels</code>	controls what levels are included. Note that the default <code>levels</code> value for <code>nodefactor.mean.age</code> retains all levels, unlike the default for <code>nodefactor</code> , which omits the first level.
<code>emptyval</code>	can be used to specify the value returned if the network is empty. A different value may be specified for each level of <code>attr</code> . The length of <code>emptyval</code> should either be 1 (in which case that value is used for every level of <code>attr</code>) or should be equal to the number of retained levels of <code>attr</code> , in which case the i th value in <code>emptyval</code> is used for the i th retained level of <code>attr</code> . This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest is empty.
<code>log</code>	logical specifying if mean log age should be returned instead of mean age

Details

This term adds one statistic for each level of `attr`, equaling the average, over all half-ties incident on nodes of that level, of the amount of time elapsed since formation.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

 nodemix.mean.age-ergmTerm

Average ages of extant ties of specified mixing types

Description

Average ages of extant ties of specified mixing types

Usage

```
# binary: nodemix.mean.age(attr, b1levels=NULL, b2levels=NULL, levels=NULL,
#                               levels2=NULL, emptyval=0, log=FALSE)
```

Arguments

attr	a vertex attribute specification (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
b1levels, b2levels, levels, level2	control what statistics are included in the model and the order in which they appear. levels2 apply to all networks; levels applies to unipartite networks; b1levels and b2levels apply to bipartite networks (see Specifying Vertex attributes and Levels (?nodal_attributes) for details)
emptyval	can be used to specify the value returned if the network is empty. A different value may be specified for each mixing type of attr. The length of emptyval should either be 1 (in which case that value is used for every mixing type of attr) or should be equal to the number of retained mixing types of attr, in which case the i th value in emptyval is used for the i th retained mixing type of attr. This is, technically, an arbitrary value, but it should not have a substantial effect unless a non-negligible fraction of networks at the parameter configuration of interest is empty.
log	logical specifying if mean log age should be returned instead of mean age

Details

This term adds one statistic for each mixing type of attr, equaling the average, over all ties of that mixing type, of the amount of time elapsed since formation.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

Persist-ergmTerm	<i>The Persistence Operator Term</i>
------------------	--------------------------------------

Description

The Persistence Operator Term

Usage

```
# binary: Persist(formula)
```

Arguments

formula a one-sided [ergm\(\)](#)-style formula with the terms to be evaluated

Details

This term accepts a model formula and produces the corresponding model for the post-dissolution/persistence network: effectively the network containing ties that persisted since the last time step.

This is the equivalent of the old-style dissolution model. So a larger positive coefficient for `Persist()` operator means less dissolution. It produces the same results as the new `Diss()` operator, except the signs of the coefficients are negated.

See Also

[ergmTerm](#) for index of model terms currently visible to the package.

Keywords: None

simulate.network	<i>STERGM wrappers for TERGM simulation</i>
------------------	---

Description

The `simulate.network` and `simulate.networkDynamic` wrappers are provided for backwards compatibility. It is recommended that new code make use of the `simulate_formula.network` and `simulate_formula.networkDynamic` functions instead. See [simulate.tergm](#) for details on these new functions.

Usage

```
## S3 method for class 'network'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  formation,
  dissolution,
  coef.form,
  coef.diss,
  constraints = ~.,
  monitor = NULL,
  time.slices = 1,
  time.start = NULL,
  time.burnin = 0,
  time.interval = 1,
  time.offset = 1,
  control = control.simulate.network(),
  output = c("networkDynamic", "stats", "changes", "final", "ergm_state"),
  stats.form = FALSE,
  stats.diss = FALSE,
  verbose = FALSE,
  ...
)

## S3 method for class 'networkDynamic'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  formation,
  dissolution,
  coef.form = attr(object, "coef.form"),
  coef.diss = attr(object, "coef.diss"),
  constraints = ~.,
  monitor = NULL,
  time.slices = 1,
  time.start = NULL,
  time.burnin = 0,
  time.interval = 1,
  time.offset = 1,
  control = control.simulate.network(),
  output = c("networkDynamic", "stats", "changes", "final", "ergm_state"),
  stats.form = FALSE,
  stats.diss = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

object	an object of type <code>network</code> or <code>networkDynamic</code>
nsim	Number of replications (separate chains of networks) of the process to run and return. The <code>networkDynamic</code> method only supports <code>nsim=1</code> .
seed	Seed value (integer) for the random number generator. See <code>set.seed</code> .
formation, dissolution	One-sided <code>ergm</code> -style formulas for the formation and dissolution models, respectively. The dissolution model is parameterized in terms of tie persistence.
coef.form	Parameters for the formation model.
coef.diss	Parameters for the dissolution (persistence) model.
constraints	<p>A formula specifying one or more constraints on the support of the distribution of the networks being modeled. Multiple constraints may be given, separated by “+” and “-” operators. See <code>ergmConstraint</code> for the detailed explanation of their semantics and also for an indexed list of the constraints visible to the <code>ergm</code> package.</p> <p>The default is to have no constraints except those provided through the <code>ergmlhs</code> API.</p> <p>Together with the model terms in the formula and the reference measure, the constraints define the distribution of networks being modeled.</p> <p>It is also possible to specify a proposal function directly either by passing a string with the function’s name (in which case, arguments to the proposal should be specified through the <code>MCMC.prop.args</code> argument to the relevant control function, or by giving it on the LHS of the hints formula to <code>MCMC.prop</code> argument to the control function. This will override the one chosen automatically.</p> <p>Note that not all possible combinations of constraints and reference measures are supported. However, for relatively simple constraints (i.e., those that simply permit or forbid specific dyads or sets of dyads from changing), arbitrary combinations should be possible.</p>
monitor	A one-sided formula specifying one or more terms whose value is to be monitored. If <code>monitor</code> is specified as a character (one of “formation”, “dissolution”, and “all”) then the function <code>.extract.fd.formulae</code> is used to determine the corresponding formula; the user should be aware of its behavior and limitations.
time.slices	Number of time slices (or statistics) to return from each replication of the dynamic process. See below for return types. Defaults to 1, which, if <code>time.burnin==0</code> and <code>time.interval==1</code> (the defaults), advances the process one time step.
time.start	An optional argument specifying the time point at which the simulation is to start. See Details for further information.
time.burnin	Number of time steps to discard before starting to collect network statistics.
time.interval	Number of time steps between successive recordings of network statistics.
time.offset	Argument specifying the offset between the point when the state of the network is sampled (<code>time.start</code>) and the the beginning of the spell that should be recorded for the newly simulated network state.
control	A list of control parameters for algorithm tuning, constructed using <code>control.simulate.network</code> . These are mapped to <code>control.simulate.formula.tergm</code> controls by assigning:

	<ul style="list-style-type: none"> • MCMC.prop.form to MCMC.prop, • MCMC.prop.args.form to MCMC.prop.args, and • MCMC.prop.weights.form to MCMC.prop.weights.
output	A character vector specifying output type: one of "networkDynamic" (the default), "stats", "changes", "final", and "ergm_state", with partial matching allowed.
stats.form, stats.diss	Logical: Whether to return formation/dissolution model statistics. This is not the recommended method: use the monitor argument instead. Note that if either stats.form or stats.diss is TRUE, all generative model statistics will be returned.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, with higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
...	Further arguments passed to or used by methods.

Details

Note that return values may be structured differently than in past versions.

Remember that in `stergm`, the dissolution formula is parameterized in terms of tie persistence: negative coefficients imply lower rates of persistence and positive coefficients imply higher rates. The dissolution effects are simply the negation of these coefficients.

Because the old dissolution formula in `stergm` represents tie persistence, it maps to the new `Persist()` operator in the `tergm` function, NOT the `Diss()` operator

Value

Depends on the output argument. See `simulate.tergm` for details. Note that some formation/dissolution separated information is also attached to the return value for calls made through `simulate.network` and `simulate.networkDynamic` in an attempt to increase backwards compatibility.

Examples

```
logit<-function(p)log(p/(1-p))
coef.form.f<-function(coef.diss,density) -log(((1+exp(coef.diss))/(density/(1-density))))-1)

# Construct a network with 20 nodes and 20 edges
n<-20
target.stats<-edges<-20
g0<-network.initialize(n,dir=TRUE)
g1<-san(g0~edges,target.stats=target.stats,verbose=TRUE)

S<-10

# To get an average duration of 10...
duration<-10
```

```

coef.diss<-logit(1-1/duration)

# To get an average of 20 edges...
dyads<-network.dyadcount(g1)
density<-edges/dyads
coef.form<-coef.form.f(coef.diss,density)

# ... coefficients.
print(coef.form)
print(coef.diss)

# Simulate a networkDynamic
dynsim<-simulate(g1,formation=~edges,dissolution=~edges,
                 coef.form=coef.form,coef.diss=coef.diss,
                 time.slices=S,verbose=TRUE)

# "Resume" the simulation.
dynsim2<-simulate(dynsim,formation=~edges,dissolution=~edges,time.slices=S,verbose=TRUE)

```

simulate.tergm	<i>Draw from the distribution of a Temporal Exponential Family Random Graph Model</i>
----------------	---

Description

`simulate` is used to draw from temporal exponential family random network models in their natural parameterizations. See `tergm` for more information on these models.

Usage

```

## S3 method for class 'tergm'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  coef = coefficients(object),
  constraints = object$constraints,
  monitor = object$targets,
  time.slices = 1,
  time.start = NULL,
  time.burnin = 0,
  time.interval = 1,
  control = control.simulate.tergm(),
  output = c("networkDynamic", "stats", "changes", "final", "ergm_state"),
  nw.start = NULL,
  stats = FALSE,
  verbose = FALSE,

```

```
    ...
  )

## S3 method for class 'network'
simulate_formula(
  object,
  nsim = 1,
  seed = NULL,
  coef = NULL,
  constraints = ~.,
  monitor = NULL,
  time.slices = 1,
  time.start = NULL,
  time.burnin = 0,
  time.interval = 1,
  time.offset = 1,
  control = control.simulate.formula.tergm(),
  output = c("networkDynamic", "stats", "changes", "final", "ergm_state"),
  stats = FALSE,
  verbose = FALSE,
  ...,
  basis = ergm.getnetwork(object),
  dynamic = FALSE
)

## S3 method for class 'networkDynamic'
simulate_formula(
  object,
  nsim = 1,
  seed = NULL,
  coef = attr(basis, "coef"),
  constraints = ~.,
  monitor = NULL,
  time.slices = 1,
  time.start = NULL,
  time.burnin = 0,
  time.interval = 1,
  time.offset = 1,
  control = control.simulate.formula.tergm(),
  output = c("networkDynamic", "stats", "changes", "final", "ergm_state"),
  stats = FALSE,
  verbose = FALSE,
  ...,
  basis = eval_lhs.formula(object),
  dynamic = FALSE
)
```

Arguments

object	for <code>simulate.tergm</code> , an object of type <code>tergm</code> giving a model fit; for <code>simulate_formula.network</code> and <code>simulate_formula.networkDynamic</code> , a formula specifying the model <code>simulate_formula.network</code> understands the <code>lasttoggle</code> "API".
nsim	Number of replications (separate chains of networks) of the process to run and return. The <code>networkDynamic</code> method only supports <code>nsim=1</code> .
seed	Seed value (integer) for the random number generator. See <code>set.seed</code> .
coef	Parameters for the model.
constraints	A formula specifying one or more constraints on the support of the distribution of the networks being modeled. Multiple constraints may be given, separated by "+" and "-" operators. See <code>ergmConstraint</code> for the detailed explanation of their semantics and also for an indexed list of the constraints visible to the <code>ergm</code> package. The default is to have no constraints except those provided through the <code>ergmlhs</code> API. Together with the model terms in the formula and the reference measure, the constraints define the distribution of networks being modeled. It is also possible to specify a proposal function directly either by passing a string with the function's name (in which case, arguments to the proposal should be specified through the <code>MCMC.prop.args</code> argument to the relevant control function, or by giving it on the LHS of the hints formula to <code>MCMC.prop</code> argument to the control function. This will override the one chosen automatically. Note that not all possible combinations of constraints and reference measures are supported. However, for relatively simple constraints (i.e., those that simply permit or forbid specific dyads or sets of dyads from changing), arbitrary combinations should be possible.
monitor	A one-sided formula specifying one or more terms whose value is to be monitored. If <code>monitor</code> is specified as a character (one of "formation", "dissolution", and "all") then the function <code>.extract.fd.formulae</code> is used to determine the corresponding formula; the user should be aware of its behavior and limitations.
time.slices	Number of time slices (or statistics) to return from each replication of the dynamic process. See below for return types. Defaults to 1, which, if <code>time.burnin==0</code> and <code>time.interval==1</code> (the defaults), advances the process one time step.
time.start	An optional argument specifying the time point at which the simulation is to start. See Details for further information.
time.burnin	Number of time steps to discard before starting to collect network statistics.
time.interval	Number of time steps between successive recordings of network statistics.
control	A list of control parameters for algorithm tuning. Constructed using <code>control.simulate.tergm</code> or <code>control.simulate_formula.tergm</code> . For backwards compatibility, control lists from <code>control.simulate.stergm</code> and <code>control.simulate.network</code> are allowed in calls to <code>simulate.tergm</code> ; they are mapped to <code>control.simulate.tergm</code> by assigning: <ul style="list-style-type: none"> • <code>MCMC.prop.form</code> to <code>MCMC.prop</code>, • <code>MCMC.prop.args.form</code> to <code>MCMC.prop.args</code>,

	<ul style="list-style-type: none"> • <code>MCMC.prop.weights.form</code> to <code>MCMC.prop.weights</code>.
<code>output</code>	A character vector specifying output type: one of "networkDynamic" (the default), "stats", "changes", "final", and "ergm_state", with partial matching allowed. See Value section for details.
<code>nw.start</code>	<p>A specification for the starting network to be used by <code>simulate.tergm</code>, optional for EGMME fits, but required for CMLE and CMPLLE fits:</p> <p>a numeric index <code>i</code> use <code>i</code>th time-point's network, where the first network in the series used to fit the model is defined to be at the first time point;</p> <p>"first" or "last" the first or last time point used in fitting the model; or</p> <p>network specify the network directly.</p> <p><code>networkDynamic</code>s cannot be used as starting networks for <code>simulate.tergm</code> at this time. (They can be used as starting networks for <code>simulate_formula.networkDynamic</code>, of course.)</p>
<code>stats</code>	Logical: Whether to return model statistics. This is not the recommended method: use <code>monitor</code> argument instead.
<code>verbose</code>	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, with higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
<code>...</code>	Further arguments passed to or used by methods.
<code>time.offset</code>	Argument specifying the offset between the point when the state of the network is sampled (<code>time.start</code>) and the the beginning of the spell that should be recorded for the newly simulated network state.
<code>basis</code>	For the network and networkDynamic methods, the network to start the simulation from. (If <code>basis</code> is missing, the default is the left hand side of the object argument.)
<code>dynamic</code>	Logical; if TRUE, dynamic simulation is performed in <code>tergm</code> ; if FALSE (the default), ordinary <code>ergm</code> simulation is performed instead. Note that when <code>dynamic=FALSE</code> , default argument values for <code>ergm</code> 's <code>simulate</code> methods are used.

Details

The dynamic process is run forward and the results are returned. For the method for `networkDynamic`, the simulation is resumed from the last generated time point of `basis` (or the left hand side of object if `basis` is missing), by default with the same model and parameters.

The starting network for the `tergm` object method (`simulate.tergm`) is determined by the `nw.start` argument.

- If `time.start` is specified, it is used as the initial time index of the simulation.
- If `time.start` is not specified (is NULL), then if the object carries a time stamp from which to start or resume the simulation, either in the form of a "time" network attribute (for the `network` method — see the `lasttoggle` "API") or in the form of an `net.obs.period` network attribute (for the `networkDynamic` method), this attribute will be used. (If specified, `time.start` will override it with a warning.)
- Otherwise, the simulation starts at 0.

Value

Depends on the output argument:

"stats"	If <code>stats == FALSE</code> , an <code>mcmc</code> matrix with monitored statistics, and if <code>stats == TRUE</code> , a list containing elements <code>stats</code> for statistics specified in the <code>monitor</code> argument, and <code>stats.gen</code> for the model statistics. If <code>stats == FALSE</code> and no monitored statistics are specified, an empty list is returned, with a warning. When <code>nsim > 1</code> , an <code>mcmc.list</code> (or list of them) of the statistics is returned instead.
"networkDynamic"	<p>A <code>networkDynamic</code> object representing the simulated process, with ties present in the initial network having onset <code>-Inf</code> and ties present at the end of the simulation having terminus <code>+Inf</code>. The method for <code>networkDynamic</code> returns the initial <code>networkDynamic</code> with simulated changes applied to it. The <code>net.obs.period</code> network attribute is updated (or added if not existing) to reflect the time period that was simulated. If the network does not have any <code>persistent.ids</code> defined for vertices, a <code>vertex.pid</code> will be attached in a vertex attribute named <code>'tergm_pid'</code> to facilitate 'bookkeeping' between the <code>networkDynamic</code> argument and the simulated network time step. Additionally, attributes (<code>attr</code>, not network attributes) are attached as follows:</p> <p><code>formula, monitor</code>: Model and monitoring formulas used in the simulation, respectively.</p> <p><code>stats, stats.gen</code>: Network statistics as above.</p> <p><code>coef</code>: Coefficients used in the simulation.</p> <p><code>changes</code>: A four-column matrix summarizing the changes in the "changes" output. (This may be removed in the future.)</p> <p>When <code>nsim > 1</code>, a <code>network.list</code> of these <code>networkDynamics</code> is returned.</p>
"changes"	An integer matrix with four columns (<code>time</code> , <code>tail</code> , <code>head</code> , and <code>to</code>), giving the time-stamped changes relative to the current network. <code>to</code> is 1 if a tie was formed and 0 if a tie was dissolved. The convention for <code>time</code> is that it gives the time point during which the change is effective. For example, a row <code>c(5, 2, 3, 1)</code> indicates that between time 4 and 5, a tie from node 2 to node 3 was formed, so that it was absent at time point 4 and present at time point 5; while a row <code>c(5, 2, 3, 0)</code> indicates that in that time, that tie was dissolved, so that it was present at time point 4 and absent at time point 5. Additionally, the same attributes (<code>attr</code> , not network attributes) as with <code>output == "networkDynamic"</code> are attached. When <code>nsim > 1</code> , a list of these change matrices is returned.
"final"	A <code>network</code> object representing the last network in the series generated. <code>lasttoggle</code> and time attributes are also included. Additionally, the same attributes (<code>attr</code> , not network attributes) as with <code>output == "networkDynamic"</code> are attached. When <code>nsim > 1</code> , a <code>network.list</code> of these <code>networks</code> is returned.
"ergm_state"	The <code>ergm_state</code> object resulting from the simulation. Attributes are attached as for other output types.

Note that when using `simulate_formula.networkDynamic` with either "final" or "ergm_state" for output, the nodes included in these objects are those produced by `network.collapse` at the start time.

Examples

```

data(samplk)

# Fit a transition from Time 1 to Time 2
samplk12 <- tergm(list(samplk1, samplk2)~
  Form(~edges+mutual+transitiveties+cyclicalities)+
  Diss(~edges+mutual+transitiveties+cyclicalities),
  estimate="CMLE")

# direct simulation from tergm object
sim1 <- simulate(samplk12, nw.start="last")

# equivalent simulation from formula with network LHS;
# must pass dynamic=TRUE for tergm simulation
sim2 <- simulate(samplk2 ~ Form(~edges+mutual+transitiveties+cyclicalities) +
  Diss(~edges+mutual+transitiveties+cyclicalities),
  coef = coef(samplk12),
  dynamic=TRUE)

# the default simulate output is a networkDynamic, and we can simulate
# with a networkDynamic LHS as well
sim3 <- simulate(sim2 ~ Form(~edges+mutual+transitiveties+cyclicalities) +
  Diss(~edges+mutual+transitiveties+cyclicalities),
  coef = coef(samplk12),
  dynamic=TRUE)

```

 snctrl

Statnet Control

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, MPLE.samplesize, init.MPLE.samplesize, MPLE.type, MPLE.maxit, MPLE.nonvar, MPLE.nonident, MPLE.nonident.tol, MPLE.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.pval, 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.bo
 MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize.m
 obs.MCMC.samplesize, obs.MCMC.effectiveSize, 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, MCMLE.min.depfac,
 MCMLE.sampszie.boost.pow, MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac, MCMLE.metric,
 MCMLE.method, MCMLE.dampening, MCMLE.dampening.min.ess, MCMLE.dampening.level,
 MCMLE.steplength.margin, MCMLE.steplength, MCMLE.steplength.parallel, MCMLE.sequential,
 MCMLE.density.guard.min, MCMLE.density.guard, MCMLE.effectiveSize, obs.MCMLE.effectiveSize,
 MCMLE.interval, MCMLE.burnin, MCMLE.samplesize.per_theta, MCMLE.samplesize.min,
 MCMLE.samplesize, obs.MCMLE.samplesize.per_theta, obs.MCMLE.samplesize.min,
 obs.MCMLE.samplesize, obs.MCMLE.interval, obs.MCMLE.burnin, MCMLE.steplength.solver,
 MCMLE.last.boost, MCMLE.steplength.esteq, MCMLE.steplength.miss.sample, MCMLE.steplength.min,
 MCMLE.effectiveSize.interval_drop, MCMLE.save_intermediates, MCMLE.nonvar, MCMLE.nonident,
 MCMLE.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.niterations,
 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, obs.MCMC.interval, obs.MCMC.burnin,
 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.san SAN.maxit, SAN.tau, SAN.invcov, SAN.invcov.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.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC,

MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges,
 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.maxr,
 MCMC.effectiveSize.burnin.pval, 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.maxedges,
 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.maxr,
 MCMC.effectiveSize.burnin.pval, 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.maxedges,
 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.maxr,
 MCMC.effectiveSize.burnin.pval, 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.maxedges,
 MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel,
 parallel.type, parallel.version.check, parallel.inherit.MT, ...

See Also

`statnet.common::snctrl()`

split.network	A <code>split()</code> method for <code>network::network</code> 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()`

stergm	<i>Separable Temporal Exponential Family Random Graph Models (Deprecated)</i>
--------	---

Description

`stergm` is used for finding Separable Temporal ERGMs' (STERGMs) Conditional MLE (CMLE) (Krivitsky and Handcock, 2014) and Equilibrium Generalized Method of Moments Estimator (EGMME) (Krivitsky, 2009). This function is deprecated in favor of `tergm()`, whose special case it is, and may be removed in a future version.

Usage

```
stergm(
  nw,
  formation,
  dissolution,
  constraints = ~.,
  estimate,
  times = NULL,
  offset.coef.form = NULL,
  offset.coef.diss = NULL,
  targets = NULL,
  target.stats = NULL,
  eval.loglik = NVL(getOption("tergm.eval.loglik"), getOption("ergm.eval.loglik")),
  control = control.stergm(),
  verbose = FALSE,
  ...,
  SAN.offsets = NULL
)
```

Arguments

`nw` A `network` object (for EGMME); or `networkDynamic` object, a `network.list` object, or a `list` containing networks (for CMLE and CMPL).
`stergm` understands the `lasttoggle` "API".

formation, dissolution	One-sided ergm -style formulas for the formation and dissolution models, respectively. In <code>stergm</code> , the dissolution formula is parameterized in terms of tie persistence: negative coefficients imply lower rates of persistence and positive coefficients imply higher rates. The dissolution effects are simply the negation of these coefficients.
constraints	<p>A formula specifying one or more constraints on the support of the distribution of the networks being modeled. Multiple constraints may be given, separated by “+” and “-” operators. See ergmConstraint for the detailed explanation of their semantics and also for an indexed list of the constraints visible to the ergm package.</p> <p>The default is to have no constraints except those provided through the ergmlhs API.</p> <p>Together with the model terms in the formula and the reference measure, the constraints define the distribution of networks being modeled.</p> <p>It is also possible to specify a proposal function directly either by passing a string with the function’s name (in which case, arguments to the proposal should be specified through the <code>MCMC.prop.args</code> argument to the relevant control function, or by giving it on the LHS of the hints formula to <code>MCMC.prop</code> argument to the control function. This will override the one chosen automatically.</p> <p>Note that not all possible combinations of constraints and reference measures are supported. However, for relatively simple constraints (i.e., those that simply permit or forbid specific dyads or sets of dyads from changing), arbitrary combinations should be possible.</p>
estimate	One of "EGMME" for Equilibrium Generalized Method of Moments Estimation, based on a single network with some temporal information and making an assumption that it is a product of a STERGM process running to its stationary (equilibrium) distribution; "CMLE" for Conditional Maximum Likelihood Estimation, modeling a transition between two networks, or "CMPLE" for Conditional Maximum PseudoLikelihood Estimation, using MPLE instead of MLE. CMPLE is extremely inaccurate at this time.
times	For CMLE and CMPLE estimation, times or indexes at which the networks whose transition is to be modeled are observed. Default to <code>c(0, 1)</code> if <code>nw</code> is a networkDynamic and to <code>1:length(nw)</code> (all transitions) if <code>nw</code> is a network.list or a list . Unused for EGMME. Note that at this time, the selected time points will be treated as temporally adjacent. Irregularly spaced time series are not supported at this time.
<code>offset.coef.form</code>	Numeric vector to specify offset formation parameters.
<code>offset.coef.diss</code>	Numeric vector to specify offset dissolution parameters.
targets	One-sided ergm -style formula specifying statistics whose moments are used for the EGMME. Unused for CMLE and CMPLE. Targets is required for EGMME estimation. It may contain any valid ergm terms. Any offset terms are used only during the preliminary SAN run; they are removed automatically for the EGMME proper. If <code>targets</code> is specified as a character (one of "formation"

and "dissolution") then the function `.extract.fd.formulae` is used to determine the corresponding formula; the user should be aware of its behavior and limitations.

<code>target.stats</code>	A vector specifying the values of the targets statistics that EGMME will try to match. Defaults to the statistics of <code>nw</code> . Unused for CMLE and CMPL.
<code>eval.loglik</code>	Whether or not to calculate the log-likelihood of a CMLE STERGM fit. See ergm for details. Can be set globally via <code>option(tergm.eval.loglik=...)</code> , falling back to <code>getOption("ergm.eval.loglik")</code> if not set.
<code>control</code>	A list of control parameters for algorithm tuning. Constructed using <code>control.stergm</code> . Remapped to <code>control.tergm</code> .
<code>verbose</code>	A logical or an integer to control the amount of progress and diagnostic information to be printed. <code>FALSE/0</code> produces minimal output, with higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
<code>...</code>	Additional arguments, to be passed to lower-level functions.
<code>SAN.offsets</code>	Offset coefficients (if any) to use during the SAN run.

Details

The `stergm` function uses a pair of formulas, formation and dissolution to model tie-dynamics. The dissolution formula, however, is parameterized in terms of tie persistence: negative coefficients imply lower rates of persistence and positive coefficients imply higher rates. The dissolution effects are simply the negation of these coefficients, but the discrepancy between the terminology and interpretation has always been unfortunate, and we have fixed this in the new `tergm` function.

If you are making the transition from old `stergm` to new `tergm`, note that the dissolution formula in `stergm` maps to the new `Persist()` operator in the `tergm` function, NOT the `Diss()` operator.

Model Terms See [ergm](#) and [ergm-terms](#) for details. At this time, only linear ERGM terms are allowed.

- For a brief demonstration, please see the `tergm` package vignette: `browseVignettes(package='tergm')`
- A more detailed tutorial is available on the statnet wiki: https://statnet.org/Workshops/tergm/tergm_tutorial.html

Value

`stergm` returns an object of class `tergm`; see `tergm()` for details and methods.

References

Krivitsky P.N. and Handcock M.S. (2014) A Separable Model for Dynamic Networks. *Journal of the Royal Statistical Society, Series B*, 76(1): 29-46. doi:10.1111/rssb.12014

Krivitsky, P.N. (2012). Modeling of Dynamic Networks based on Egocentric Data with Durational Information. *Pennsylvania State University Department of Statistics Technical Report*, 2012(2012-01). <https://web.archive.org/web/20170830053722/https://stat.psu.edu/research/technical-report-files/2012-technical-reports/TR1201A.pdf>

See Also

ergm, network, \

summary_formula.networkDynamic

Calculation of networkDynamic statistics.

Description

A method for [summary_formula](#) to calculate the specified statistics for an observed [networkDynamic](#) at the specified time point(s). See [ergm-terms](#) for more information on the statistics that may be specified.

Usage

```
## S3 method for class 'networkDynamic'
summary_formula(object, at, ..., basis = NULL)
```

Arguments

object	An formula object with a networkDynamic as its LHS. (See summary_formula for more details.)
at	A vector of time points at which to calculate the statistics.
...	Further arguments passed to or used by methods.
basis	An optional networkDynamic object relative to which the statistics should be calculated.

Value

A matrix with `length(at)` rows, one for each time point in `at`, and columns for each term of the formula, containing the corresponding statistics measured on the network.

See Also

[ergm\(\)](#), [networkDynamic](#), [ergm-terms](#), [summary.formula](#)

Examples

```
# create a toy dynamic network
my.nD <- network.initialize(100,directed=FALSE)
activate.vertices(my.nD, onset=0, terminus = 10)
add.edges.active(my.nD,tail=1:2,head=2:3,onset=5,terminus=8)

# use a summary formula to display number of isolates and edges
# at discrete time points
summary(my.nD~isolates+edges, at=1:10)
```

Description

`tergm` is used for finding Temporal ERGMs' (TERGMs) and Separable Temporal ERGMs' (STERGMs) Conditional MLE (CMLE) (Krivitsky and Handcock, 2010) and Equilibrium Generalized Method of Moments Estimator (EGMME) (Krivitsky, 2009).

Usage

```
tergm(
  formula,
  constraints = ~.,
  estimate,
  times = NULL,
  offset.coef = NULL,
  targets = NULL,
  target.stats = NULL,
  SAN.offsets = NULL,
  eval.loglik = NVL(getOption("tergm.eval.loglik"), getOption("ergm.eval.loglik")),
  control = control.tergm(),
  verbose = FALSE,
  ...
)
```

Arguments

<code>formula</code>	an ERGM formula.
<code>constraints</code>	<p>A formula specifying one or more constraints on the support of the distribution of the networks being modeled. Multiple constraints may be given, separated by “+” and “-” operators. See ergmConstraint for the detailed explanation of their semantics and also for an indexed list of the constraints visible to the ergm package.</p> <p>The default is to have no constraints except those provided through the ergmlhs API.</p> <p>Together with the model terms in the formula and the reference measure, the constraints define the distribution of networks being modeled.</p> <p>It is also possible to specify a proposal function directly either by passing a string with the function’s name (in which case, arguments to the proposal should be specified through the <code>MCMC.prop.args</code> argument to the relevant control function, or by giving it on the LHS of the hints formula to <code>MCMC.prop</code> argument to the control function. This will override the one chosen automatically.</p> <p>Note that not all possible combinations of constraints and reference measures are supported. However, for relatively simple constraints (i.e., those that simply permit or forbid specific dyads or sets of dyads from changing), arbitrary combinations should be possible.</p>

<code>estimate</code>	One of "EGMME" for Equilibrium Generalized Method of Moments Estimation, based on a single network with some temporal information and making an assumption that it is a product of a TERGM process running to its stationary (equilibrium) distribution; "CMLE" for Conditional Maximum Likelihood Estimation, modeling a transition between two networks, or "CMPLE" for Conditional Maximum PseudoLikelihood Estimation, using MPLE instead of MLE. CMPLE is extremely inaccurate at this time.
<code>times</code>	For CMLE and CMPLE estimation, times or indexes at which the networks whose transition is to be modeled are observed. Default to <code>c(0,1)</code> if <code>nw</code> is a <code>networkDynamic</code> and to <code>1:length(nw)</code> (all transitions) if <code>nw</code> is a <code>network.list</code> or a <code>list</code> . Unused for EGMME. Note that at this time, the selected time points will be treated as temporally adjacent. Irregularly spaced time series are not supported at this time.
<code>offset.coef</code>	Numeric vector to specify offset parameters.
<code>targets</code>	One-sided <code>ergm</code> -style formula specifying statistics whose moments are used for the EGMME. Unused for CMLE and CMPLE. Targets is required for EGMME estimation. It may contain any valid <code>ergm</code> terms. Any offset terms are used only during the preliminary SAN run; they are removed automatically for the EGMME proper. If <code>targets</code> is specified as a character (one of "formation" and "dissolution") then the function <code>.extract.fd.formulae</code> is used to determine the corresponding formula; the user should be aware of its behavior and limitations.
<code>target.stats</code>	A vector specifying the values of the <code>targets</code> statistics that EGMME will try to match. Defaults to the statistics of <code>nw</code> . Unused for CMLE and CMPLE.
<code>SAN.offsets</code>	Offset coefficients (if any) to use during the SAN run.
<code>eval.loglik</code>	Whether or not to calculate the log-likelihood of a CMLE TERGM fit. See <code>ergm</code> for details. Can be set globally via <code>option(tergm.eval.loglik=...)</code> , falling back to <code>getOption("ergm.eval.loglik")</code> if not set.
<code>control</code>	A list of control parameters for algorithm tuning. Constructed using <code>control.tergm</code> .
<code>verbose</code>	A logical or an integer to control the amount of progress and diagnostic information to be printed. <code>FALSE/0</code> produces minimal output, with higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
<code>...</code>	Additional arguments, to be passed to lower-level functions.

Details

Model Terms See `ergm` and `ergm-terms` for details. At this time, only linear ERGM terms are allowed.

- For a brief demonstration, please see the `tergm` package vignette: `browseVignettes(package='tergm')`
- A more detailed tutorial is available on the statnet wiki: https://statnet.org/Workshops/tergm/tergm_tutorial.html

Value

`tergm` returns an object of class `tergm` that inherits from `ergm` and has the usual methods (`coef.ergm()`, `summary.ergm()`, `mcmc.diagnostics()`, etc.) implemented for it. Note that `gof()` only works for the CMLE method.

References

- Krackhardt, D and Handcock, MS (2006) Heider vs Simmel: Emergent features in dynamic structures. ICML Workshop on Statistical Network Analysis. Springer, Berlin, Heidelberg, 2006.
- Hanneke S, Fu W, and Xing EP (2010). Discrete Temporal Models of Social Networks. *Electronic Journal of Statistics*, 2010, 4, 585-605. doi:10.1214/09EJS548
- Krivitsky P.N. and Handcock M.S. (2014) A Separable Model for Dynamic Networks. *Journal of the Royal Statistical Society, Series B*, 76(1): 29-46. doi:10.1111/rssb.12014
- Krivitsky, P.N. (2012). Modeling of Dynamic Networks based on Egocentric Data with Durational Information. *Pennsylvania State University Department of Statistics Technical Report*, 2012(2012-01). <http://stat.psu.edu/research/technical-report-files/2012-technical-reports/modeling-of-dynamic-networks-based-on-egocentric-data-with-durational-information>

See Also

`ergm()`, `network()`, `%v%`, `%n%`, `ergm-terms`

Examples

```
## Not run:
# EGME Example
par(ask=FALSE)
n<-30
g0<-network.initialize(n,dir=FALSE)

#           edges, degree(1), mean.age
target.stats<-c( n*1/2, n*0.6, 20)

dynfit<-tergm(g0 ~ Form(~edges + degree(1)) + Diss(~edges),
              targets = ~edges+degree(1)+mean.age,
              target.stats=target.stats, estimate="EGME",
              control=control.tergm(SA.plot.progress=TRUE))

par(ask=TRUE)
mcmc.diagnostics(dynfit)
summary(dynfit)

## End(Not run)

# CMLE Example
data(samplk)

# Fit a transition from Time 1 to Time 2
samplk12 <- tergm(list(samplk1, samplk2)~
                  Form(~edges+mutual+transitivities+cyclicalities)+
```

```

                                Diss(~edges+mutual+transitivities+cyclicalities),
                                estimate="CMLE")

mcmc.diagnostics(samplk12)
summary(samplk12)

samplk12.gof <- gof(samplk12)

samplk12.gof

plot(samplk12.gof)

plot(samplk12.gof, plotlogodds=TRUE)

# Fit a transition from Time 1 to Time 2 and from Time 2 to Time 3 jointly
samplk123 <- tergm(list(samplk1, samplk2, samplk3)~
                    Form(~edges+mutual+transitivities+cyclicalities)+
                    Diss(~edges+mutual+transitivities+cyclicalities),
                    estimate="CMLE")

mcmc.diagnostics(samplk123)
summary(samplk123)

```

tergm.godfather	<i>A function to apply a given series of changes to a network.</i>
-----------------	--

Description

Gives the network a series of timed proposals it can't refuse. Returns the statistics of the network, and, optionally, the final network.

Usage

```

tergm.godfather(
  formula,
  changes = NULL,
  toggles = changes[, -4, drop = FALSE],
  start = NULL,
  end = NULL,
  end.network = FALSE,
  stats.start = FALSE,
  verbose = FALSE,
  control = control.tergm.godfather()
)

```

Arguments

formula	An <code>summary.formula</code> -style formula, with either a <code>network</code> or a <code>networkDynamic</code> as the LHS and statistics to be computed on the RHS. If LHS is a <code>networkDynamic</code> , it will be used to derive the changes to the network whose statistics are wanted. Otherwise, either <code>changes</code> or <code>toggles</code> must be specified, and the LHS <code>network</code> will be used as the starting network.
changes	A matrix with four columns: time, tail, head, and new value, describing the changes to be made. Can only be used if LHS of <code>formula</code> is not a <code>networkDynamic</code> .
toggles	A matrix with three columns: time, tail, and head, giving the dyads which had changed. Can only be used if LHS of <code>formula</code> is not a <code>networkDynamic</code> .
start	Time from which to start applying changes. Note that the first set of changes will take effect at <code>start + 1</code> . Defaults to the time point 1 before the earliest change passed.
end	Time at which to finish applying changes. Defaults to the last time point at which a change occurs.
end.network	Whether to return the network that results. Defaults to FALSE.
stats.start	Whether to return the network statistics at <code>start</code> (before any changes are applied) as the first row of the statistics matrix. Defaults to FALSE, to produce output similar to that of <code>simulate</code> for TERGMs when <code>output="stats"</code> , where initial network's statistics are not returned.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, with higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
control	A control list generated by <code>control.tergm.godfather</code> .

Value

If `end.network` is FALSE (the default), an `mcmc` object with the requested network statistics associated with the network series produced by applying the specified changes. Its `mcmc` attributes encode the timing information: so `start(out)` gives the time point associated with the first row returned, and `end(out)` out the last. The "thinning interval" is always 1.

If `end.network` is TRUE, return a `network` object with `lasttoggle` "extension", representing the final network, with a matrix of statistics described in the previous paragraph attached to it as an `attr`-style attribute "stats".

See Also

`simulate.tergm()`, `simulate_formula.network()`, `simulate_formula.networkDynamic()`

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

```
uncombine_network(  
  nw,  
  ignore.nattr = c("bipartite", "directed", "hyper", "loops", "mnext", "multiple", "n",  
    ".subnetcache"),  
  ignore.vattr = c(),  
  ignore.eattr = c(),  
  split.vattr = ".NetworkID",  
  names.vattr = NULL  
)
```

Arguments

`nw` a `network::network` created by `combine_networks()`.
`ignore.nattr`, `ignore.vattr`, `ignore.eattr`
 network, vertex, and edge attributes not to be processed as described below.
`split.vattr` name of the vertex attribute on which to split.
`names.vattr` optional name of the vertex attribute to use as network names in the output list.

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(samp1k)
```

```
o1 <- combine_networks(list(samp1k1, samp1k2, samp1k3))  
image(as.matrix(o1))
```

```
o1 <- uncombine_network(o1)
```


Index

- * **durational**
 - Change-ergmTerm, 5
 - Cross-ergmTerm, 28
 - degrange.mean.age-ergmTerm, 28
 - degree.mean.age-ergmTerm, 29
 - Diss-ergmTerm, 30
 - edge.ages-ergmTerm, 31
 - EdgeAges-ergmTerm, 32
 - edgecov.ages-ergmTerm, 32
 - edgecov.mean.age-ergmTerm, 33
 - edges.ageinterval-ergmTerm, 34
 - Form-ergmTerm, 35
 - mean.age-ergmTerm, 38
 - nodefactor.mean.age-ergmTerm, 40
 - nodemix.mean.age-ergmTerm, 41
 - Persist-ergmTerm, 42
- * **dyad-independent**
 - discord-ergmHint, 30
- * **manip**
 - impute.network.list, 35
- * **models**
 - control.simulate.network, 8
 - control.simulate.tergm, 10
 - summary_formula.networkDynamic, 57
 - tergm-package, 3
- * **model**
 - is.durational, 37
- * **operator**
 - Change-ergmTerm, 5
 - Cross-ergmTerm, 28
 - Diss-ergmTerm, 30
 - EdgeAges-ergmTerm, 32
 - Form-ergmTerm, 35
 - Persist-ergmTerm, 42
- * **package**
 - tergm-package, 3
 - .extract.fd.formulae, 4, 14, 22, 44, 48, 56, 59
 - %n%, 60
- %v%, 60
- attr, 50
- Change-ergmTerm, 5
- coef.ergm(), 60
- combine_networks, 6
- combine_networks(), 6, 63
- combined_networks (combine_networks), 6
- control.ergm, 15, 23, 51
- control.ergm.bridge, 52
- control.ergm.godfather, 52
- control.gof.ergm, 52
- control.gof.formula, 52
- control.logLik.ergm, 52
- control.san, 16, 24, 52
- control.simulate, 52
- control.simulate.ergm, 53
- control.simulate.formula, 53
- control.simulate.formula.ergm, 53
- control.simulate.formula.tergm, 44, 48
- control.simulate.formula.tergm (control.simulate.tergm), 10
- control.simulate.network, 8, 44, 48
- control.simulate.stergm, 20, 48
- control.simulate.stergm (control.simulate.network), 8
- control.simulate.tergm, 10, 27, 48
- control.stergm, 10, 12, 56
- control.tergm, 12, 20, 20, 56, 59
- control.tergm.godfather, 27, 62
- control\$init.method, 14, 22
- Cross-ergmTerm, 28
- degrange.mean.age-ergmTerm, 28
- degree.mean.age-ergmTerm, 29
- discord-ergmConstraint (discord-ergmHint), 30
- discord-ergmHint, 30
- Diss-ergmTerm, 30

- edge.ages-ergmTerm, 31
- EdgeAges-ergmTerm, 32
- edgecov.ages-ergmTerm, 32
- edgecov.mean.age, 33
- edgecov.mean.age-ergmTerm, 33
- edges.ageinterval-ergmTerm, 34
- end, 62
- ergm, 3, 19, 27, 44, 55, 56, 59
- ergm(), 5, 16, 24, 28, 30, 35, 37, 42, 57, 60
- ergm::edgecov, 7
- ergm_model, 37
- ergm_proposal_table, 9, 11
- ergm_state, 37, 50
- ergmConstraint, 44, 48, 55, 58
- ergmHint, 30
- ergmlhs, 44, 48, 55, 58
- ergmTerm, 3, 6, 28–35, 38, 40–42

- Form-ergmTerm, 35
- formula, 57

- gof(), 60

- Help on model specification, 39

- impute.network.list, 15, 23, 35, 39
- InitErgmConstraint.discord
 (discord-ergmHint), 30
- InitErgmTerm.Cross (Cross-ergmTerm), 28
- InitErgmTerm.degrange.mean.age
 (degrange.mean.age-ergmTerm),
 28
- InitErgmTerm.degree.mean.age
 (degree.mean.age-ergmTerm), 29
- InitErgmTerm.edge.ages
 (edge.ages-ergmTerm), 31
- InitErgmTerm.EdgeAges
 (EdgeAges-ergmTerm), 32
- InitErgmTerm.edgecov.ages
 (edgecov.ages-ergmTerm), 32
- InitErgmTerm.edgecov.mean.age
 (edgecov.mean.age-ergmTerm), 33
- InitErgmTerm.edges.ageinterval
 (edges.ageinterval-ergmTerm),
 34
- InitErgmTerm.mean.age
 (mean.age-ergmTerm), 38
- InitErgmTerm.nodefactor.mean.age
 (nodefactor.mean.age-ergmTerm),
 40

- InitErgmTerm.nodemix.mean.age
 (nodemix.mean.age-ergmTerm), 41
- is.durational, 37
- is.na, 36

- lasttoggle, 37, 38, 48–50, 54, 62
- list, 54, 55, 59

- mcmc, 50, 62
- mcmc.diagnostics(), 19, 26, 60
- mcmc.list, 50
- mean.age, 31
- mean.age-ergmTerm, 38

- net.obs.period, 49, 50
- NetSeries, 39
- network, 36, 38, 44, 49, 50, 54, 62
- network(), 60
- network.list, 36, 50, 54, 55, 59
- network::get.inducedSubgraph(), 54
- network::network, 6, 7, 53, 63
- network::print.network(), 7
- network::print.summary.network(), 8
- network::summary.network(), 8
- networkDynamic, 39, 44, 48–50, 54, 55, 57,
 59, 62
- nodefactor.mean.age-ergmTerm, 40
- nodemix.mean.age-ergmTerm, 41

- parallel processing, 19, 26
- pdf, 16, 24
- Persist-ergmTerm, 42
- persistent.ids, 50
- print.combined_networks
 (combine_networks), 6
- print.summary.combined_networks
 (combine_networks), 6

- san, 16, 24
- set.MT_terms(), 19, 27
- set.seed, 19, 26, 44, 48
- simulate, 10, 12, 46, 62
- simulate.formula, 10, 12
- simulate.network, 42
- simulate.networkDynamic
 (simulate.network), 42
- simulate.stergm, 10
- simulate.stergm(simulate.network), 42
- simulate.tergm, 10, 12, 20, 27, 42, 45, 46

`simulate.tergm()`, 62
`simulate_formula.network`
 (`simulate.tergm`), 46
`simulate_formula.network()`, 62
`simulate_formula.networkDynamic`
 (`simulate.tergm`), 46
`simulate_formula.networkDynamic()`, 62
`snctrl`, 51
`split()`, 53
`split.network`, 53
`split.network()`, 7, 63
`start`, 62
`statnet.common::snctrl()`, 53
`stergm`, 10, 14, 15, 19, 20, 54, 54, 56
`stergm()`, 4
`summary.combined_networks`
 (`combine_networks`), 6
`summary.ergm()`, 60
`summary_formula`, 57, 62
`summary_formula`
 (`summary_formula.networkDynamic`),
 57
`summary_formula`, 57
`summary_formula.networkDynamic`, 57

`tergm`, 3, 4, 12, 20, 22, 23, 27, 46, 48, 49, 56,
 58, 58, 60
`tergm()`, 4, 54, 56
`tergm-package`, 3
`tergm.godfather`, 61
`tergm.godfather()`, 27

`uncombine_network`, 63
`uncombine_network()`, 7, 63