Package ‘tergm’

June 12, 2019

Version 3.6.1

Date 2019-06-12

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

Depends ergm (>= 3.10.1), network (>= 1.15), networkDynamic (>= 0.10.0)

Imports robustbase (>= 0.93.5), coda (>= 0.19.2), nlme (>= 3.1.139), MASS (>= 7.3.51.4), statnet.common (>= 4.2.0)

LinkingTo ergm

Suggests lattice (>= 0.20.38), parallel, rmarkdown (>= 1.12), knitr (>= 1.22)

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.

License GPL-3 + file LICENSE

URL http://www.statnet.org

VignetteBuilder rmarkdown, knitr

RoxygenNote 6.1.1

Encoding UTF-8

Collate 'InitConstraint.R' 'InitErgmTerm.duration.R' 'InitMHP.DynMLE.R' 'InitMHP.DynMLE.blockdiag.R' 'InitMHP.DynMoME.R' 'coef.stergm.R' 'combine.networks.R' 'control.logLik.stergm.R' 'control.simulate.stergm.R' 'control.stergm.R' 'ergm.godfather.R' 'gof.stergm.R' 'impute.network.list.R' 'is.lasttoggle.R' 'logLik.stergm.R' 'mcmc.diagnostics.stergm.R' 'print.stergm.R' 'simulate.stergm.R' 'stergm.CMLE.R' 'stergm.EGMME.GD.R' 'stergm.EGMME.R' 'stergm.EGMME.SA.R' 'stergm.EGMME.initialfit.R' 'stergm.R' 'tergm-deprecated.R' 'stergm.getMCMCsample.R' 'stergm.utils.R'

'summary.statistics.networkDynamic.R' 'summary.stergm.R'
'zzz.R'

NeedsCompilation yes

Author Pavel N. Krivitsky [aut, cre] (<https://orcid.org/0000-0002-9101-3362>),
Mark S. Handcock [aut, ths],
David R. Hunter [ctb],
Steven M. Goodreau [ctb, ths],
Martina Morris [ctb, ths],
Nicole Bohme Carnegie [ctb],
Carter T. Butts [ctb],
Ayn Leslie-Cook [ctb],
Skye Bender-deMoll [ctb],
Li Wang [ctb],
Kirk Li [ctb]

Maintainer Pavel N. Krivitsky <pavel@uow.edu.au>

Repository CRAN

Date/Publication 2019-06-12 10:10:18 UTC

R topics documented:

`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')`

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.

Details

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.

More recently, there has been a number of extensions of ERGMs to model evolution of networks, including the temporal ERGM (TERGM) of Hanneke et al. (2010) and the separable termporal ERGM (STERGM) of Krivitsky and Handcock (2013). The latter model allows familiar ERGM terms and statistics to be reused in a dynamic context, interpreted in terms of formation and dissolution of ties. Krivitsky (2012) suggested a method for fitting dyanmic 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 evoluation. At this time, it implements, via the stergm function, the STERGMs, both a conditional MLE (CMLE) 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.

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

Author(s)

Pavel N. Krivitsky <krivitsky@stat.psu.edu> and
Mark S. Handcock <handcock@stat.ucla.edu>,
with contributions from
David R. Hunter <dhunter@stat.psu.edu>,
Steven M. Goodreau <goodreau@u.washington.edu>,
Martina Morris <morrism@u.washington.edu>,
Nicole Bohme Carnegie <nicole.carnegie@nyu.edu>, and
Ayn Leslie-Cook <aynlc3@uw.edu>

Maintainer: Pavel N. Krivitsky <krivitsky@stat.psu.edu>

References

- Hanneke S, Fu W, and Xing EP (2010). Discrete Temporal Models of Social Networks. *Electronic Journal of Statistics*, 2010, 4, 585-605. doi:10.1214/09-EJS548
- Krivitsky PN, Handcock MS (2013). A Separable Model for Dynamic Networks. *Journal of the Royal Statistical Society, Series B*, In Press. http://arxiv.org/abs/1011.1937

- 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
- Butts CT (2008). **network**: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). http://www.jstatsoft.org/v24/i02/.
- Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). A **statnet** Tutorial. *Journal of Statistical Software*, 24(8). http://www.jstatsoft.org/v24/i08/.
- Handcock MS, Hunter DR, Butts CT, Goodreau SM, Krivitsky P, and Morris M (2012). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. Statnet Project, Seattle, WA. Version 3, statnet.org.
- Handcock MS, Hunter DR, Butts CT, Goodreau SM, Krivitsky P, Morris M (2012). **statnet**: Software Tools for the Statistical Modeling of Network Data. Statnet Project, Seattle, WA. Version 3, statnet.org.
- 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). http://www.jstatsoft.org/v24/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). http://www.jstatsoft.org/v24/i04/.

`coef.stergm` *Separable Temporal Exponential Family Random Graph Models*

Description

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

Usage

```
## S3 method for class 'stergm'
coef(object, ...)

## S3 method for class 'stergm'
coefficients(object, ...)

## S3 method for class 'stergm'
print(x, digits = max(3, getOption("digits") - 3), ...)

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, ...)

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

Arguments

`object` A `stergm` fit.

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

`x` A `stergm` object.

`digits` Significant digits for coefficients

`nw` A `network` object (for EGMME); or `networkDynamic` object, a `network.list` object, or a `list` containing networks (for CMLE and CMPLE).

`stergm` understands the `lasttoggle` "API".

`formation, dissolution`
One-sided `ergm`-style formulas for the formation and dissolution models, respectively.

`constraints` A one-sided formula specifying one or more constraints on the support of the distribution of the networks being modeled, using syntax similar to the `formula` argument. Multiple constraints may be given, separated by "+" operators. 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 by passing a string with the function's name. In that case, arguments to the proposal should be specified through the `prop.args` argument to `control.ergm`.

The default is `~.`, for an unconstrained model.

See the ERGM constraints documentation for the constraints implemented in the **ergm** package. Other packages may add their own constraints.

For STERGMs in particular, the constraints apply to the post-formation and the post-dissolution network, rather than the final network. This means, for example, that if the degree of all vertices is constrained to be less than or equal to three, and a vertex begins a time step with three edges, then, even if one of its edges is dissolved during its time step, it won't be able to form another edge until the next time step. This behavior may change in the future.

Note that not all possible combinations of constraints are supported.

`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 `c(0,1)` if `nw` is a `networkDynamic` and to `1:length(nw)` (all transitions) if `nw` 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. Irregluarly spaced time series are not supported at this time.

offset.coef.form
: Numeric vector to specify offset formation parameters.

offset.coef.diss
: 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. If specified as "formation" or "dissolution", it copies the formula from the respective model. Any offset terms are removed automatically.

target.stats
: A vector specifying the values of the `targets` statistics that EGMME will try to match. Defaults to the statistics of `nw`. Unused for CMLE and CMPLE.

eval.loglik
: Whether or not to calculate the log-likelihood of a CMLE STERGM fit. See `ergm` for details. Can be set globally via `option(tergm.eval.loglik=...)`, falling back to `getOption("ergm.eval.loglik")` if not set.

control
: A list of control parameters for algorithm tuning. Constructed using `control.stergm`.

verbose
: logical or integer; if TRUE or positive, the program will print out progress information. Higher values result in more output.

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 avalible on the statnet wiki: `http://statnet.csde.washington.edu/workshops/SUNBELT/current/tergm/tergm_tutorial.pdf`
- For more usage examples, see the wiki page at `https://statnet.csde.washington.edu/trac/wiki/tergmUsage`

Value

`coef` and `coefficients` methods return parameter estimates extracted from `object` in the form of a list with two elements: `formation`, a vector of formation coefficients and `dissolution`, a vector of dissolution coefficients.

`stergm` returns an object of class `stergm` that is a list consisting of the following elements:

formation, dissolution
: Formation and dissolution formulas, respectively.

targets
: The targets formula.

target.stats
: The target statistics.

`estimate` The type of estimate.

`opt.history` A matrix containing the full trace of the EGMME optimization process: coefficients tried and target statistics simulated.

`sample` An `mcmc` object containing target statistics sampled at the estimate.

`covar` The full estimated variance-covariance matrix of the parameter estimates for EGMME. (Note that although the CMLE formation parameter estimates are independent of the dissolution parameter estimates due to the separability assumption, this is not necessarily the case for EGMME.)

`formation.fit, dissolution.fit`
For CMLE and CMPLE, `ergm` objects from fitting formation and dissolution, respectively. For EGMME, stripped down `ergm`-like lists.

`network` For `estimate=="EGMME"`, the original network; for `estimate=="CMLE"` or `estimate=="CMPLE"`, a `network.list` (a discrete series of networks) to which the model was fit.

`control` The control parameters used to fit the model.

See the method `print.stergm` for details on how an `stergm` object is printed. Note that the method `summary.stergm` returns a summary of the relevant parts of the `stergm` object in concise summary format.

Methods (by generic)

- `coef`: Extract parameter estimates.
- `coefficients`: An *alias* for the `coef` method.
- `print`: Print the parameter estimates.
- `summary`: Print the summary of the formation and the dissolution model fits.

References

- Krivitsky PN, Handcock MS (2010). A Separable Model for Dynamic Networks. http://arxiv.org/abs/1011.1937
- 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

```
# EGMME 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<-stergm(g0,formation = ~edges+degree(1), dissolution = ~edges,
               targets = ~edges+degree(1)+mean.age,
               target.stats=target.stats, estimate="EGMME",
               control=control.stergm(SA.plot.progress=TRUE))

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

# CMLE Example
data(samplk)

# Fit a transition from Time 1 to Time 2
samplk12 <- stergm(list(samplk1, samplk2),
                   formation=~edges+mutual+transitiveties+cyclicalties,
                   dissolution=~edges+mutual+transitiveties+cyclicalties,
                   estimate="CMLE")

mcmc.diagnostics(samplk12)
summary(samplk12)

# Fit a transition from Time 1 to Time 2 and from Time 2 to Time 3 jointly
samplk123 <- stergm(list(samplk1, samplk2, samplk3),
                    formation=~edges+mutual+transitiveties+cyclicalties,
                    dissolution=~edges+mutual+transitiveties+cyclicalties,
                    estimate="CMLE")

mcmc.diagnostics(samplk123)
summary(samplk123)
```

`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.burnin = NULL, MCMC.burnin.mul = NULL,
  MCMC.prop.weights.form = "default", MCMC.prop.args.form = NULL,
  MCMC.prop.weights.diss = "default", MCMC.prop.args.diss = NULL,
```

```
    MCMC.init.maxedges = 20000, MCMC.packagenames = c(),
    term.options = NULL, MCMC.init.maxchanges = 1e+06)

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

Arguments

`MCMC.burnin.min, MCMC.burnin.max, MCMC.burnin.pval, MCMC.burnin.add`
: Number of Metropolis-Hastings steps per phase (formation and dissolution) 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 (formation network or dissolution network). 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 had been taken. (Stopping immediately would bias the sampling.)

  To use a fixed number of steps, set both `MCMC.burnin.min` and `MCMC.burnin.max` to the desired number of steps.

`MCMC.burnin, MCMC.burnin.mul`
: No longer used. See `MCMC.burnin.min`, `MCMC.burnin.max`, `MCMC.burnin.pval`, `MCMC.burnin.pval`, and `MCMC.burnin.add`.

`MCMC.prop.weights.form, MCMC.prop.weights.diss`
: Specifies the proposal distribution used in the MCMC Metropolis-Hastings algorithm for formation and dissolution, respectively. Possible choices are `"TNT"` or `"random"`; the `"default"`. The `TNT` (tie / no tie) option puts roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling, whereas the `random` option puts equal weight on all possible dyads, though the interpretation of `random` may change according to the constraints in place. When no constraints are in place, the default is TNT, which appears to improve Markov chain mixing particularly for networks with a low edge density, as is typical of many realistic social networks.

`MCMC.prop.args.form, MCMC.prop.args.diss`
: An alternative, direct way of specifying additional arguments to proposals.

`MCMC.init.maxedges`
: Maximum number of edges expected in network.

`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.

`term.options` A list of additional arguments to be passed to term initializers. It can also be set globally via `option(ergm.term=list(...))`.

`MCMC.init.maxchanges`
Maximum number of toggles changes for which to allocate space.

Details

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

Value

A list with arguments as components.

See Also

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

`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.weights.form = "default", MCMC.prop.args.form = NULL,
  MCMC.prop.weights.diss = "default", MCMC.prop.args.diss = NULL,
  MCMC.init.maxedges = 20000, MCMC.init.maxchanges = 20000,
  MCMC.packagenames = c(), CMLE.MCMC.burnin = 1024 * 16,
  CMLE.MCMC.interval = 1024, CMLE.control = NULL,
  CMLE.control.form = control.ergm(init = init.form, MCMC.burnin =
  CMLE.MCMC.burnin, MCMC.interval = CMLE.MCMC.interval, MCMC.prop.weights =
  MCMC.prop.weights.form, MCMC.prop.args = MCMC.prop.args.form,
  MCMC.init.maxedges = MCMC.init.maxedges, MCMC.packagenames =
  MCMC.packagenames, parallel = parallel, parallel.type = parallel.type,
  parallel.version.check = parallel.version.check, force.main =
  force.main), CMLE.control.diss = control.ergm(init = init.diss,
  MCMC.burnin = CMLE.MCMC.burnin, MCMC.interval = CMLE.MCMC.interval,
  MCMC.prop.weights = MCMC.prop.weights.diss, MCMC.prop.args =
  MCMC.prop.args.diss, MCMC.init.maxedges = MCMC.init.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.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 = control.san(term.options =
term.options, SAN.maxit = SAN.maxit, SAN.prop.weights =
MCMC.prop.weights.form, SAN.prop.args = MCMC.prop.args.form,
SAN.init.maxedges = MCMC.init.maxedges, SAN.max.maxedges = Inf,
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), 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)
```

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`.
- 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 `control.ergm(init=coef(prev.fit))` can be used to "resume" an uncoverged `stergm` run, but see `enformulate.curved`.

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. If 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.weights.form, MCMC.prop.weights.diss
: Specifies the method to allocate probabilities of being proposed to dyads in the formation/dissolution phase. Defaults to "default", which picks a reasonable default for the specified constraint. Possible values include "TNT", "random", though not all values may be used with all possible constraints.

MCMC.prop.args.form, MCMC.prop.args.diss
: An alternative, direct way of specifying additional arguments to the proposal in the formation/dissolution phase.

MCMC.init.maxedges
: Maximum number of edges for which to allocate space.

MCMC.init.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
: Maximum number of Metropolis-Hastings steps per phase (formation and dissolution) per time step used in CMLE fitting.

CMLE.MCMC.interval
: Number of Metropolis-Hastings steps between successive draws when running MCMC MLE.

CMLE.control
: A convenience argument for specifying both CMLE.control.form and CMLE.control.diss at once. See control.ergm.

CMLE.control.form, CMLE.control.diss
: Control parameters used to fit the CMLE for the formation/dissolution ERGM. See control.ergm.

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 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 stergm{stergm} 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.MCMC.burnin.min, EGMME.MCMC.burnin.max,
: Number of Metropolis-Hastings steps per phase (formation and dissolution) 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 (formation network or dissolution network). 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 both EGMME.MCMC.burnin.min and EGMME.MCMC.burnin.max to the desired number of steps.

EGMME.MCMC.burnin.pval, EGMME.MCMC.burnin.add
: Number of Metropolis-Hastings steps per phase (formation and dissolution) 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 (formation network or dissolution network). 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 both EGMME.MCMC.burnin.min and EGMME.MCMC.burnin.max to the desired number of steps.

MCMC.burnin, MCMC.burnin.mul
: No longer used. See EGMME.MCMC.burnin.min, EGMME.MCMC.burnin.max, EGMME.MCMC.burnin.pval, 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.control defaults.

SAN.control
: 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 reprsenting 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) conductiong 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) conductiong 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.

`SA.phase2.jitter.mul`
: Jitter standard deviation of each parameter is this value times its standard deviation without jitter.

`SA.phase2.maxreljump`
: 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.

`SA.guard.mul`
: The multiplier for the range of parameter and statistics values to compute the guard width.

`SA.par.eff.pow`
: 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.

`SA.robust` Whether to use robust linear regression (for gradients) and covariance estimation.

`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 `mcmc.diagnostics.stergm` 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. It can also be set globally via `option(ergm.term=list(...))`.

`seed` Seed value (integer) for the random number generator. See `set.seed`

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

`parallel.type` API to use for parallel processing. Supported values are `"MPI"` and `"PSOCK"`. Defaults to using the `parallel` package default.

`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.

Details

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

`stergm`. The `control.simulate.stergm` function performs a similar function for `simulate.stergm`.

`control.tergm.godfather` *Control parameters for* `tergm.godfather()`.

Description

Returns a list of its arguments.

Usage

```
control.tergm.godfather(GF.init.maxedges.mul = 5)
```

Arguments

`GF.init.maxedges.mul`
: How much space is allocated for the edgelist of the final network. It is used adaptively, so should not be greater than `10`.

`ergm-constraints` *Formation and Dissolution Constraints for Exponential Family Random Graph Models*

Description

This page describes the network sample space constraints that are included with the `tergm` package. For more information, and instructions for using constraints, see `ergm-constraints` and `ergm`.

Constraints implemented in the `tergm` package

`atleast(nw)` *The Formation Constraint:* Preserve all ties in network `nw`. Only dyads that are not ties in `nw` may be changed.

`atmost(nw)` *The Dissolution Constraint:* Prevent all nonties in network `nw`. Only dyads that have ties in `nw` may be changed.

References

Krivitsky PN, Handcock MS (2010). A Separable Model for Dynamic Networks. http://arxiv.org/abs/1011.1937

Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). A **statnet** Tutorial. *Journal of Statistical Software*, 24(8). http://www.jstatsoft.org/v24/i08/.

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

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). http://www.jstatsoft.org/v24/i03/.

Krivitsky PN (2012). Exponential-Family Random Graph Models for Valued Networks. *Electronic Journal of Statistics*, 2012, 6, 1100-1128. doi:10.1214/12-EJS696

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). http://www.jstatsoft.org/v24/i04/.

`ergm-terms` *Temporally-Sensitive Terms used in Exponential Family Random Graph Models*

Description

Unlike ordinary `ergm-terms`, which take only a single network as an argument, the terms documented here also take into account the "ages" of extant ties in the network: the time elapsed since their formation.

As implemented, many of these terms cannot be used to "drive" the process of network evolution, but they can be used as target statistics to infer the terms that do. More concretely, they may appear in `targets=` or `monitor=` formulas of `stergm`, `simulate.stergm`, or `summary.formula` (with an ERGM formula), but they may not appear in their `formation=` and `dissolution=` formulas. These terms are marked with "(target-only)".

All terms listed here are binary.

Terms to represent network statistics included in the `tergm` package

`degrange.mean.age(from, to=+Inf, byarg=NULL, emptyval=0)` **(target-only)** *Average age of ties incident on nodes having degree in a given range:* The `from` and `to` arguments are 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. This term adds one network statistic to the model for each element of `from` (or `to`); the ith 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 by is a character string giving the name of an attribute in the network's vertex attribute list. If specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.

Because this average is undefined for a network that does not have any actors with degree in the specified range, the argument `emptyval` can be used to specify the value returned if this is the case. 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.

`degree.mean.age(d, by=NULL, emptyval=0)` **(target-only)** *Average age of ties incident on nodes having a given degree:* The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the ith 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 `by` is a character string giving the name of an attribute in the network's vertex attribute list. If specified, then separate degree statistics are calculated for nodes having each separate value of the attribute.

Because this average is undefined for a network that does not have any actors with degree `d[i]`, the argument `emptyval` can be used to specify the value returned if this is the case. 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.

`edges.ageinterval(from, to=+Inf)` **(dissolution- and target-only)** *Number of edges with age falling into a specified range:* 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)`. `from` and `to` may be scalars, vectors of the same length, or one of them must have length one, in which case it is recycled.

When used in the dissolution formula of a STERGM, it can be used to model a non-Markovian dissolution process, controlling the hazard function in the interval directly.

`edge.ages` **(target-only)** *Sum of ages of extant ties:* 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.

`edgecov.ages(x, attrname=NULL)` **(target-only)** *Weighted sum of ages of extant ties:* 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. See the help for the `edgecov` term for details for specifying the 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.

`edgecov.mean.age(x, attrname=NULL, emptyval=0)` **(target-only)** *Weighted average age of an extant tie:* 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. See the help for the `edgecov` term for details for specifying the covariate.

The behavior when there are negative weights is undefined.

Because this average is undefined for an empty network (or a network all of whose extant edges have been weighted 0), the argument `emptyval` can be used to specify the value returned if this is the case. 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.

`mean.age(emptyval=0)` **(target-only)** *Average age of an extant tie:* This term adds one statistic equaling the average, over all ties present in the network, of the amount of time elapsed since formation.

Because this average is undefined for an empty network, the argument `emptyval` can be used to specify the value returned if it is. 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.

References

- Handcock M. S., Hunter D. R., Butts C. T., Goodreau S. G., Krivitsky P. N. and Morris M. (2012). _Fit, Simulate and Diagnose Exponential-Family Models for Networks_. Version 3.1. Project home page at <URL: http://www.statnet.org>, <URL: CRAN.R-project.org/package=ergm>.
- 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`
- Krivitsky, P.N. (2012). Modeling Tie Duration in ERGM-Based Dynamic Network Models. *Pennsylvania State University Department of Statistics Technical Report*, 2012(2012-02). `http://stat.psu.edu/research/technical-report-files/2012-technical-reports/TR1201A.pdf`

See Also

`ergm-terms` (from the `ergm` package), `ergm`, `network`, `%v%`, `%n%`

`gof.stergm` *Goodness-of-fit methods for STERGM CMLE and CMPLE fits*

Description

For now, these are simple wrappers around `gof.ergm`, `print.gof`, `summary.gof`, and `plot.gof`, respectively, to run goodness-of-fit for formation and dissolution models separately. This may change in the future.

Usage

```
## S3 method for class 'stergm'
gof(object, ...)

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

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

## S3 method for class 'gof.stergm'
plot(x, ..., main = "Goodness-of-fit diagnostics")
```

Arguments

- `object` For `gof.stergm`, `stergm` conditional MLE (CMLE) or conditional MPLE (CMPLE) fit. For the others, a `gof.stergm` object returned by `gof.stergm`.
- `...` Additional arguments passed through to the respective functions in the `ergm` package.
- `x` A `gof.stergm` object returned by `gof.stergm`.
- `main` Gives the title of the goodness-of-fit plots, which will have "Formation:" and "Dissolution:" prepended to it.

Value

For `gof.stergm`, an object of class `gof.stergm`, which is simply a list with two named elements: `formation` and `dissolution`, each of them a `gof` returned by `gof.ergm`.

For the others, nothing.

See Also

`stergm()`, `ergm()`, `simulate.stergm()`, `ergm::print.gof()`, `ergm::plot.gof()`, summary.gof, mcmc.diagnostics.ergm

Examples

```
data(samplk)

# Fit a transition from Time 1 to Time 2
samplk12 <- stergm(list(samplk1, samplk2),
                   formation=~edges+mutual+transitiveties+cyclicalties,
                   dissolution=~edges+mutual+transitiveties+cyclicalties,
                   estimate="CMLE")

samplk12.gof <- gof(samplk12)

samplk12.gof

summary(samplk12.gof)

plot(samplk12.gof)

plot(samplk12.gof, plotlogodds=TRUE)
```

impute.network.list *Impute missing dyads in a series of networks*

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

nwl	A list of network objects or a network.list object.
imputers	A character vector giving one or more methods to impute missing dyads. Currenly 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 formation and dissolution 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.
nwl.prepend	An optional list of networks to treat as preceding those in `nwl`. They will not be imputed or returned, but they can be useful for imputing dyads in the first network in `nwl`, when using `"previous"` imputer.
nwl.append	An optional list of networks to treat as following those in `nwl`. They will not be imputed or returned, but they can be useful for imputing dyads in the last network in `nwl`, when using `"next"` imputer.

Value

A list of networks with missing dyads imputed.

See Also

network, is.na

---

logLik.stergm *A logLik method for stergm.*

Description

Functions to return the log-likelihood associated with a stergm CMLE fit, evaluating it if necessary. See logLik.ergm documentation for details and caveats.

logLikNull method computes the null model likelihood. See ergm::logLikNull().

Usage

```
## S3 method for class 'stergm'
logLik(object, add = FALSE, force.reeval = FALSE,
  eval.loglik = add || force.reeval, control = control.logLik.stergm(),
  ...)

## S3 method for class 'stergm'
logLikNull(object, control = control.logLik.stergm(),
  ...)
```

Arguments

object	A stergm fit, returned by stergm, for estimate="CMLE".
add	Logical: If TRUE, instead of returning the log-likelihood, return object with log-likelihood value set.
force.reeval	Logical: If TRUE, reestimate the log-likelihood even if object already has an estiamte.
eval.loglik	Logical: If TRUE, evaluate the log-likelihood if not set on object.
control	A list of control parameters for algorithm tuning. Constructed using control.logLik.ergm.
...	Other arguments to the likelihood functions.

Details

If the log-likelihood was not computed for object, produces an error unless eval.loglik=TRUE

Value

For logLik.stergm, add=FALSE (the default), a logLik object. If add=TRUE (the default), an ergm object or a stergm object with the log-likelihood set. For logLikNull.stergm, a logLik object.

References

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

See Also

`logLik`, `ergm.bridge.llr`, `ergm.bridge.dindstart.llk`

`mcmc.diagnostics.stergm` *Conduct MCMC diagnostics on an ergm or stergm fit*

Description

This function prints diagnistic information and creates simple diagnostic plots for the MCMC sampled statistics produced from a `stergm` fit.

Usage

```
## S3 method for class 'stergm'
mcmc.diagnostics(object, center = TRUE, esteq = TRUE,
  vars.per.page = 3, ...)
```

Arguments

`object`	A stergm object. See documentation for `stergm`.
`center`	Logical: If TRUE, ; center the samples on the observed statistics.
`esteq`	Logical: If TRUE, summarize the estimating equation values (evaluated at the MLE of any non-linear parameters), rather than their canonical components.
`vars.per.page`	Number of rows (one variable per row) per plotting page. Ignored if `latticeExtra` package is not installed.
`...`	Additional arguments, to be passed to plotting functions.

Details

The plots produced are a trace of the sampled output and a density estimate for each variable in the chain. The diagnostics printed include correlations and convergence diagnostics.

In fact, an `object` contains the matrix of statistics from the MCMC run as component `$sample`. This matrix is actually an object of class `mcmc` and can be used directly in the `coda` package to assess MCMC convergence. *Hence all MCMC diagnostic methods available in* `coda` *are available directly.* See the examples and `http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/coda-readme/`.

More information can be found by looking at the documentation of `stergm`.

Value

mcmc.diagnostics.ergm returns some degeneracy information, if it is included in the original object. The function is mainly used for its side effect, which is to produce plots and summary output based on those plots.

References

Raftery, A.E. and Lewis, S.M. (1995). The number of iterations, convergence diagnostics and generic Metropolis algorithms. In Practical Markov Chain Monte Carlo (W.R. Gilks, D.J. Spiegelhalter and S. Richardson, eds.). London, U.K.: Chapman and Hall.

This function is based on the coda package It is based on the the R function raftery.diag in coda. raftery.diag, in turn, is based on the FORTRAN program gibbsit written by Steven Lewis which is available from the Statlib archive.

See Also

ergm, stergm,network package, coda package, summary.ergm

---

simulate.stergm *Draw from the distribution of an Separable Temporal Exponential Family Random Graph Model*

Description

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

Usage

```
## S3 method for class 'stergm'
simulate(object, nsim = 1, seed = NULL,
  coef.form = object$formation.fit$coef,
  coef.diss = object$dissolution.fit$coef,
  constraints = object$constraints, monitor = object$targets,
  time.slices = 1, time.start = NULL, time.burnin = 0,
  time.interval = 1, control = control.simulate.stergm(),
  statsonly = NULL, output = c("networkDynamic", "stats", "changes",
  "final"), nw.start = NULL, stats.form = FALSE, stats.diss = FALSE,
  duration.dependent = NULL, verbose = FALSE, ...)

## 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(), statsonly = NULL,
```

```
  output = c("networkDynamic", "stats", "changes", "final"),
  stats.form = FALSE, stats.diss = FALSE, duration.dependent = NULL,
  verbose = FALSE, ...)

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

Arguments

`object` an object of type `stergm` giving a model fit or of type `network` giving the initial network.

`simulate.network` understands the `lasttoggle` "API".

`nsim` Number of replications (separate chains of networks) of the process to run and return. The `networkDynamic` method only supports `nsim=1`.

`seed` Random number integer seed. See `set.seed`.

`coef.form` Parameters for the model from which the post-formation network is drawn.

`coef.diss` As `coef.form`, but for the post-dissolution network.

`constraints` A one-sided formula specifying one or more constraints on the support of the distribution of the networks being modeled, using syntax similar to the `formula` argument. Multiple constraints may be given, separated by "+" operators. 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 by passing a string with the function's name. In that case, arguments to the proposal should be specified through the `prop.args` argument to `control.ergm`.

The default is `~.`, for an unconstrained model.

See the ERGM constraints documentation for the constraints implemented in the **ergm** package. Other packages may add their own constraints.

For STERGMs in particular, the constraints apply to the post-formation and the post-dissolution network, rather than the final network. This means, for example, that if the degree of all vertices is constrained to be less than or equal to three, and a vertex begins a time step with three edges, then, even if one of its edges is dissolved during its time step, it won't be able to form another edge until the next time step. This behavior may change in the future.

Note that not all possible combinations of constraints are supported.

`monitor` Either a one-sided formula specifying one or more terms whose value is to be monitored, or a string containing `"formation"` or `"dissolution"`, to monitor their respective terms, or `"all"` to monitor distinct terms from both.

`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 `time.burnin==0` and `time.interval==1` (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. Actual network will only be returned if `time.burnin==0`.

`time.interval` Number of time steps between successive recordings of network statistics. Actual network will only be returned if `time.interval==1`.

`control` A list of control parameters for algorithm tuning. Constructed using `control.simulate.stergm` or `control.simulate.network`.

`statsonly` Deprecated in favor of `output`.

`output` A character vector specifying output type: one of "networkDynamic" (the default), "stats", and "changes", with partial matching allowed. See Value section for details.

`nw.start` A specification for the starting network to be used by `simulate.stergm`, optional for EGMME fits, but required for CMLE and CMPLE fits:

- **a numeric index** use `i`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;
- `i` use `i`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;
- `"first"` **or** `"last"` the first or last time point used in fitting the model; or
- `network` specify the network directly.

`networkDynamic`s cannot be used as starting networks for `simulate.stergm` at this time. (They can be used as starting networks for `simulate.networkDynamic`, of course.)

`stats.form, stats.diss`
Logical: Whether to return formation/dissolution model statistics. This is not the recommended method: use `monitor` argument instead.

`duration.dependent`
Logical: Whether the model terms in formula or model are duration dependent. E.g., if a duration-dependent term is used in estimation/simulation model, the probability of forming or dissolving a tie may dependent on the age the dyad status.

`verbose` Logical: If TRUE, extra information is printed as the Markov chain progresses.

`...` Further arguments passed to or used by methods.

`formation, dissolution`
One-sided `ergm`-style formulas for the formation and dissolution models, respectively.

`time.offset` Argument specifying the offset between the point when the state of the network is sampled (`time.start`) and the the beginning of the spell that should be recorded for the newly simulated network state.

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 object, by default with the same model and parameters.

The starting network for the stergm object method (simulate.stergm) 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.)
- Othewise, the simulation starts at 0.

Value

Depends on the output argument:

"stats" If stats.form==FALSE and stats.diss==FALSE, an mcmc matrix with monitored statistics, and if either of them is TRUE, a list containing elements stats for statistics specified in the monitor argument, and stats.form and stats.diss for the respective formation and dissolution statistics. If stats.form==FALSE and stats.diss==FALSE and no monitored statistics are specified, an empty list is returned, with a warning. When nsim>1, an mcmc.list (or list of them) of the statistics is returned instead.

"networkDynamic"
A networkDynamic object representing the simulated process, with ties present in the initial network having onset -Inf and ties present at the end of the simulation having terminus +Inf. The method for networkDynamic returns the initial networkDynamic with simulated changes applied to it. The net.obs.period network attribute is updated (or added if not existing) to reflect the time period that was simulated. If the network does not have any persistent.ids defined for vertices, a vertex.pid will be attached in a vertex attribute named 'tergm_pid' to facilitate 'bookkeeping' between the networkDynamic argument and the simulated network time step. Additionally, attributes (attr, not network attributes) are attached as follows:

formation, dissolution, monitor: Formation, dissolution, and monitoring formulas used in the simulation, respectively.

stats, stats.form, stats.diss: Network statistics as above.

coef.form, coef.diss: Coefficients used in the simulation.

changes: A four-column matrix summarizing the changes in the "changes" output. (This may be removed in the future.)

When nsim>1, a network.list of these networkDynamics is returned.

"changes" An integer matrix with four columns (time, tail, head, and to), giving the time-stamped changes relative to the current network. to is 1 if a tie was formed and 0 if a tie was dissolved. The convention for time is that it gives the time point during which the change is effective. For example, a row c(5,2,3,1) 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 c(5,2,3,0) indicates that in that time, that tie was dissolved, so that it is was present at time point 4 and absent at time point 5. Additionally, same attributes (attr, not network attributes) as with output=="networkDynamic" are attached. When nsim>1, a list of these change matrices is returned.

"final" A network object representing the last network in the series generated. This is not implemented in the method for networkDynamic. lasttoggle attributes are also included. Additionally, attributes (attr, not network attributes) are attached as follows:

formation, dissolution, monitor: Formation, dissolution, and monitoring formulas used in the simulation, respectively.

stats, stats.form, stats.diss: Network statistics as above.

coef.form, coef.diss: Coefficients used in the simulation.

changes A four-column matrix summarizing the changes in the "changes" output. (This may be removed in the future.)

When nsim>1, a network.list of these networks is returned.

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,time.slices=S,verbose=TRUE)
```

---

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)
```

tergm.godfather *A function to apply a given series of changes to a network.*

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 `summary.formula`-style formula, with either a `network` or a `networkDynamic` as the LHS and statistics to be computed on the RHS. If LHS is a `networkDynamic`, it will be used to derive the changes to the network whose statistics are wanted. Otherwise, either `changes` or `toggles` must be specified, and the LHS `network` 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 `formula` is not a `networkDynamic`.
toggles	A matrix with three columns: time, tail, and head, giving the dyads which had changed. Can only be used if LHS of `formula` is not a `networkDynamic`.
start	Time from which to start applying changes. Note that the first set of changes will take effect at `start+1`. Defaults to the time point 1 before the earliest change passed.
end	Time from which to finish applying changes. Defaults to the last time point at which a change occurs.
end.network	Whether to return a network that results. Defaults to `FALSE`.
stats.start	Whether to return the network statistics at `start` (before any changes are applied) as the first row of the statistics matrix. Defaults to `FALSE`, to produce output similar to that of `simulate` for STERGMs when `output="stats"`, where initial network's statistics are not returned.
verbose	Whether to print progress messages.
control	A control list generated by `control.tergm.godfather`.

Value

If `end.network==FALSE` (the default), an `mcmc` object with the requested network statistics associed 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==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.stergm, simulate.network, simulate.networkDynamic

Examples

```
g1 <- network.initialize(10, dir=FALSE)
g1[1,2] <- 1
g1[3,4] <- 1
g1 %n% "time" <- 0
g1 %n% "lasttoggle" <- -1-rgeom(network.dyadcount(g1),1/4)

dc <- matrix(rnorm(100),10,10); dc <- dc+t(dc)

# Simulate a network, tracking its statistics.
simnet <- simulate(g1, formation=~edges, dissolution=~edges, coef.form=-1, coef.diss=1,
                   time.slices=50, monitor=~degree(1)+mean.age+degree.mean.age(1)+
                                         mean.age(log=TRUE)+degree.mean.age(1,log=TRUE)+
                                          degrange(1,3)+mean.age+degrange.mean.age(1,3)+
                                     mean.age(log=TRUE)+degrange.mean.age(1,3,log=TRUE)+
                                            edge.ages+edgecov(dc)+edgecov.ages(dc),
                   output="networkDynamic")

sim.stats <- attr(simnet, "stats")

print(head(sim.stats))
sim.stats <- as.matrix(sim.stats)

# Replay the simulation using a networkDynamic, monitoring a potentially different set of
# statistics (but same in this case).
gf1.stats <- tergm.godfather(simnet~degree(1)+mean.age+degree.mean.age(1)+
                                    mean.age(log=TRUE)+degree.mean.age(1,log=TRUE)+
                                    degrange(1,3)+mean.age+degrange.mean.age(1,3)+
                                    mean.age(log=TRUE)+degrange.mean.age(1,3,log=TRUE)+
                                    edge.ages+edgecov(dc)+edgecov.ages(dc),
                             start=0, end=50)

print(head(gf1.stats))
gf1.stats <- as.matrix(gf1.stats)

# Replay the simulation using the initial network + list of changes.

gf2.stats <- tergm.godfather(g1~degree(1)+mean.age+degree.mean.age(1)+
                                mean.age(log=TRUE)+degree.mean.age(1,log=TRUE)+
                                degrange(1,3)+mean.age+degrange.mean.age(1,3)+
                                mean.age(log=TRUE)+degrange.mean.age(1,3,log=TRUE)+
```

```
                              edge.ages+edgecov(dc)+edgecov.ages(dc),
                           start=0, end=50, changes=attr(simnet,"changes"))

print(head(gf2.stats))
gf2.stats <- as.matrix(gf2.stats)

# We can also compare them to the network statistics summarized.
summ.stats <- summary(simnet~degree(1)+mean.age+degree.mean.age(1)+
                           mean.age(log=TRUE)+degree.mean.age(1,log=TRUE)+
                           degrange(1,3)+mean.age+degrange.mean.age(1,3)+
                           mean.age(log=TRUE)+degrange.mean.age(1,3,log=TRUE)+
                           edge.ages+edgecov(dc)+edgecov.ages(dc), at=1:50)

print(head(summ.stats))

tol <- sqrt(.Machine$double.eps)
# If they aren't all identical, we are in trouble.
stopifnot(all.equal(sim.stats,gf1.stats),
          all.equal(sim.stats,gf2.stats),
          all.equal(sim.stats,summ.stats))
```

---

tergm_proposals *Metropolis-Hastings Proposal Methods for TERGM MCMC*

Description

tergm uses a Metropolis-Hastings (MH) algorithm provided by ergm to control the behavior of the Markov Chain Monte Carlo (MCMC) for sampling networks. The MCMC chain is intended to step around the sample space of possible networks, selecting a network at regular intervals to evaluate the statistics in the model. For each MCMC step, n ($n = 1$ in the simple case) toggles are proposed to change the dyad(s) to the opposite value. The probability of accepting the proposed change is determined by the MH acceptance ratio. The role of the different MH methods implemented in tergm is to vary how the sets of dyads are selected for toggle proposals. This is used in some cases to improve the performance (speed and mixing) of the algorithm, and in other cases to constrain the sample space.

Details

MH proposal methods implemented in the tergm package:

TODO: EXPLAIN TERGM PROPOSAL TYPES HERE

- InitErgmProposal.dissolution
- InitErgmProposal.dissolutionTNT
- InitErgmProposal.dissolutionMLE
- InitErgmProposal.dissolutionNonObservedMLE
- InitErgmProposal.formation
- InitErgmProposal.formationMLE

- InitErgmProposal.formationMLETNT
- InitErgmProposal.formationNonObservedMLE
- InitErgmProposal.formationTNT
- InitErgmProposal.dissolutionMLEblockdiag
- InitErgmProposal.dissolutionNonObservedMLEblockdiag
- InitErgmProposal.formationMLEblockdiag
- InitErgmProposal.formationMLEblockdiagTNT
- InitErgmProposal.formationNonObservedMLEblockdiag
- InitErgmProposal.dissolutionMLETNT
- InitErgmProposal.dissolutionMLEblockdiagTNT
- InitErgmProposal.dissolutionNonObservedMLETNT
- InitErgmProposal.dissolutionNonObservedMLEblockdiagTNT
- InitErgmProposal.formationNonObservedMLETNT
- InitErgmProposal.formationNonObservedMLEblockdiagTNT

References

Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). A **statnet** Tutorial. *Journal of Statistical Software*, 24(8). http://www.jstatsoft.org/v24/i08/.

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

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). http://www.jstatsoft.org/v24/i03/.

Krivitsky PN (2012). Exponential-Family Random Graph Models for Valued Networks. *Electronic Journal of Statistics*, 2012, 6, 1100-1128. doi:10.1214/12-EJS696

Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). http://www.jstatsoft.org/v24/i04/.

See Also

`tergm` package, `ergm`, `ergm-constraints`, and ergm's `ergm_proposal` help page

Index