Package 'ergm'

May 10, 2022

Version 4.2.1

Date 2022-05-10

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

Depends R (>= 4.0),
network (>= 1.17.0)

Imports robustbase (>= 0.93.7),
coda (>= 0.19.4),
trust (>= 0.1.8),
Matrix (>= 1.3.2),
lpSolveAPI (>= 5.5.2.0.17.7),
MASS (>= 7.3.53.1),
statnet.common (>= 4.6.0),
rle (>= 0.9.2),
purrr (>= 0.3.4),
rlang (>= 0.4.10),
memoise (>= 2.0.0),
tibble (>= 3.1.0),
magrittr (>= 2.0.1),
parallel,
methods,
knitr,
digest,
stringr

Suggests lattice (>= 0.20.41),
latticeExtra (>= 0.6.29),
sna (>= 2.6),
snow (>= 0.4.3),
latentnet (>= 2.10.5),
rmarkdown,
Rmpi (>= 0.6.9.1),
testthat (>= 3.0.2),
tergm (>= 4.0.0),
ergm.count (>= 4.0),
ergm.userterms (>= 3.10.0),
networkDynamic (>= 0.10.1),

> withr,
> covr,
> Rglpk,
> slam

SystemRequirements OpenMPI

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

Description An integrated set of tools to analyze and simulate networks based on exponential-family random graph models (ERGMs). 'ergm' is a part of the Statnet suite of packages for network analysis. See Hunter, Handcock, Butts, Goodreau, and Morris (2008) <doi:10.18637/jss.v024.i03> and Krivitsky, Hunter, Morris, and Klumb (2021) <arXiv:2106.04997>.

License GPL-3 + file LICENSE

License_is_FOSS yes

License_restricts_use no

URL <https://statnet.org>

VignetteBuilder knitr

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE)

Config/testthat/parallel true

Config/testthat/edition 3

Encoding UTF-8

Collate 'InitErgmConstraint.R'
'InitErgmConstraint.blockdiag.R'
'InitErgmConstraint.hints.R'
'InitErgmProposal.R'
'InitErgmProposal.dyadnoise.R'
'InitErgmReference.R'
'ergm-deprecated.R'
'InitErgmTerm.R'
'InitErgmTerm.auxnet.R'
'InitErgmTerm.bipartite.R'
'InitErgmTerm.bipartite.degree.R'
'InitErgmTerm.blockop.R'
'InitErgmTerm.coincidence.R'
'InitErgmTerm.dgw_sp.R'
'InitErgmTerm.extra.R'
'InitErgmTerm.indices.R'
'InitErgmTerm.interaction.R'
'InitErgmTerm.operator.R'
'InitErgmTerm.spcache.R'
'InitErgmTerm.test.R'
'InitErgmTerm.transitiveties.R'
'InitWtErgmProposal.R'

'InitWtErgmTerm.R'
'InitWtErgmTerm.operator.R'
'InitWtErgmTerm.test.R'
'anova.ergm.R'
'anova.ergmlist.R'
'approx.hotelling.diff.test.R'
'as.network.numeric.R'
'build_term_index.R'
'check.ErgmTerm.R'
'control.ergm.R'
'control.ergm.bridge.R'
'control.gof.R'
'control.logLik.ergm.R'
'control.san.R'
'control.simulate.R'
'data.R'
'ergm-defunct.R'
'ergm-internal.R'
'ergm-options.R'
'ergm-package.R'
'ergm-terms-index.R'
'ergm.CD.fixed.R'
'ergm.Cprepare.R'
'ergm.MCMCse.R'
'ergm.MCMLE.R'
'ergm.R'
'ergm.allstats.R'
'ergm.auxstorage.R'
'ergm.bounddeg.R'
'ergm.bridge.R'
'ergm.design.R'
'ergm.errors.R'
'ergm.estimate.R'
'ergm.eta.R'
'ergm.etagrad.R'
'ergm.etagradmult.R'
'ergm.etamap.R'
'ergm.geodistn.R'
'ergm.getCDsample.R'
'ergm.getMCMCsample.R'
'ergm.getnetwork.R'
'ergm.initialfit.R'
'ergm.llik.R'
'ergm.llik.obs.R'
'ergm.logitreg.R'
'ergm.mple.R'
'ergm.pen.glm.R'
'ergm.phase12.R'

'ergm.pl.R'
'ergm.robmon.R'
'ergm.san.R'
'ergm.stepping.R'
'ergm.stocapprox.R'
'ergm.utility.R'
'ergmMPLE.R'
'ergm_estfun.R'
'ergm_keyword.R'
'ergm_model.R'
'ergm_model.utils.R'
'ergm_proposal.R'
'ergm_response.R'
'ergm_state.R'
'ergmlhs.R'
'formula.utils.R'
'get.node.attr.R'
'godfather.R'
'gof.ergm.R'
'is.curved.R'
'is.dyad.independent.R'
'is.inCH.R'
'is.valued.R'
'logLik.ergm.R'
'mcmc.diagnostics.ergm.R'
'network.list.R'
'network.update.R'
'nonidentifiability.R'
'nparam.R'
'obs.constraints.R'
'parallel.utils.R'
'param_names.R'
'predict.ergm.R'
'print.ergm.R'
'print.network.list.R'
'print.summary.ergm.R'
'rank_test.ergm.R'
'rlebdm.R'
'simulate.ergm.R'
'simulate.formula.R'
'summary.ergm.R'
'summary.ergm_model.R'
'summary.network.list.R'
'summary.statistics.network.R'
'to_ergm_Cdouble.R'
'vcov.ergm.R'
'wtd.median.R'
'zzz.R'

R **topics documented:**

---

ergm-package   *Fit, Simulate and Diagnose Exponential-Family Models for Networks*

Description

[ergm](#) is a collection of functions to plot, fit, diagnose, and simulate from exponential-family random graph models (ERGMs). For a list of functions type: help(package='ergm')

Details

For a complete list of the functions, use library(help="ergm") or read the rest of the manual. For a simple demonstration, use demo(packages="ergm").

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

All programs derived from this package must cite it. Please see the file LICENSE and http://statnet.org/attribution.

Recent advances in the statistical modeling of random networks have had an impact on the empirical study of social networks. Statistical exponential family models (Strauss and Ikeda 1990) are a generalization of the Markov random network models introduced by Frank and Strauss (1986), which in turn derived from developments in spatial statistics (Besag, 1974). These models recognize the complex dependencies within relational data structures. To date, the use of stochastic network models for networks has been limited by three interrelated factors: the complexity of realistic models, the lack of simulation tools for inference and validation, and a poor understanding of the inferential properties of nontrivial models.

This manual introduces software tools for the representation, visualization, and analysis of network data that address each of these previous shortcomings. The package relies on the [network](#) package which allows networks to be represented in . The [ergm](#) package implements maximum likelihood estimates of ERGMs to be calculated using Markov Chain Monte Carlo (via [ergm](#)). The package

also provides tools for simulating networks (via `simulate.ergm`) and assessing model goodness-of-fit (see `mcmc.diagnostics` and `gof.ergm`).

A number of Statnet Project packages extend and enhance `ergm`. These include `tergm` (Temporal ERGM), which provides extensions for modeling evolution of networks over time; `ergm.count`, which facilitates exponential family modeling for networks whose dyadic measurements are counts; and `ergm.userterms`, which allows users to implement their own ERGM terms.

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

Author(s)

Mark S. Handcock <handcock@stat.ucla.edu>,
David R. Hunter <dhunter@stat.psu.edu>,
Carter T. Butts <buttsc@uci.edu>,
Steven M. Goodreau <goodreau@u.washington.edu>,
Pavel N. Krivitsky <pavel@statnet.org>, and
Martina Morris <morrism@u.washington.edu>

Maintainer: Pavel N. Krivitsky <pavel@statnet.org>

References

Krivitsky P. N., Hunter D. R., Morris M., Klumb C. (2021). "ergm 4.0: New features and improvements." arXiv:2106.04997. `https://arxiv.org/abs/2106.04997`

Admiraal R, Handcock MS (2007). **networksis**: Simulate bipartite graphs with fixed marginals through sequential importance sampling. Statnet Project, Seattle, WA. Version 1, `https://statnet.org`.

Bender-deMoll S, Morris M, Moody J (2008). Prototype Packages for Managing and Animating Longitudinal Network Data: **dynamicnetwork** and **rSoNIA**. *Journal of Statistical Software*, 24(7). doi:10.18637/jss.v024.i07

Besag, J., 1974, Spatial interaction and the statistical analysis of lattice systems (with discussion), *Journal of the Royal Statistical Society, B*, 36, 192-236.

Boer P, Huisman M, Snijders T, Zeggelink E (2003). StOCNET: an open software system for the advanced statistical analysis of social networks. Groningen: ProGAMMA / ICS, version 1.4 edition.

Butts CT (2007). **sna**: Tools for Social Network Analysis. R package version 2.3-2. `https://cran.r-project.org/package=sna`

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

Butts C (2015). **network**: Classes for Relational Data. The Statnet Project (`https://statnet.org`). R package version 1.12.0, `https://cran.r-project.org/package=network`.

Frank, O., and Strauss, D.(1986). Markov graphs. *Journal of the American Statistical Association*, 81, 832-842.

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

Goodreau SM, Kitts J, Morris M (2008b). Birds of a Feather, or Friend of a Friend? Using Exponential Random Graph Models to Investigate Adolescent Social Networks. *Demography*, 45, in press.

Handcock, M. S. (2003) Assessing Degeneracy in Statistical Models of Social Networks, Working Paper \#39, Center for Statistics and the Social Sciences, University of Washington. `https://csss.uw.edu/research/working-papers/assessing-degeneracy-statistical-models-social-networks`

Handcock MS (2003b). **degreenet**: Models for Skewed Count Distributions Relevant to Networks. Statnet Project, Seattle, WA. Version 1.0, `https://statnet.org`.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003a). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. Statnet Project, Seattle, WA. Version 3, `https://statnet.org`.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003b). **statnet**: Software Tools for the Statistical Modeling of Network Data. Statnet Project, Seattle, WA. Version 3, `https://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). doi:10.18637/jss.v024.i03

Krivitsky PN, Handcock MS (2007). **latentnet**: Latent position and cluster models for statistical networks. Seattle, WA. Version 2, `https://statnet.org`.

Krivitsky PN (2012). Exponential-Family Random Graph Models for Valued Networks. *Electronic Journal of Statistics*, 2012, 6, 1100-1128. c("\Sexpr[results=rd,stage=build]tools:::Rd_expr_doi(\"#1\")", "10.1214/12-EJS696")\ifelse{text}{doi:10.1214/12-EJS696 <https://doi.org/10.1214/12-EJS696>}{\ifelse{latex}{\href{http EJS696}{doi:10.1214\out{\slash{}}12\out{\-}EJS696}}{\href{https://doi.org/10.1214/12-EJS696}{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). doi:10.18637/jss.v024.i04

Strauss, D., and Ikeda, M.(1990). Pseudolikelihood estimation for social networks. *Journal of the American Statistical Association*, 85, 204-212.

.dyads-ergmConstraint   *A meta-constraint indicating handling of arbitrary dyadic constraints*

Description

This is a flag in the proposal table indicating that the proposal can enforce arbitrary combinations of dyadic constraints. It cannot be invoked directly by the user.

See Also

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

---

absdiff-ergmTerm   *Absolute difference in nodal attribute*

Description

This term adds one network statistic to the model equaling the sum of abs(attr[i]-attr[j])^pow for all edges (i,j) in the network.

Usage

```
# binary: absdiff(attr,
#                 pow=1)

# valued: absdiff(attr,
#                 pow=1,
#                 form="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
pow	power to which to take the absolute difference
form	character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See [ergm-options](#) for how to invoke the old behaviour.

See Also

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

---

absdiffcat-ergmTerm   *Categorical absolute difference in nodal attribute*

Description

This term adds one statistic for every possible nonzero distinct value of abs(attr[i]-attr[j]) in the network. The value of each such statistic is the number of edges in the network with the corresponding absolute difference.

Usage

```
# binary: absdiffcat(attr,
#                base=NULL,
#                levels=NULL)

# valued: absdiffcat(attr,
#                base=NULL,
#                levels=NULL,
#                form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

base   deprecated

levels specifies which nonzero difference to include in or exclude from the model. (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See ergm-options for how to invoke the old behaviour.

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

See Also

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

---

altkstar-ergmTerm   *Alternating k-star*

Description

Add one network statistic to the model equal to a weighted alternating sequence of k-star statistics with weight parameter lambda.

Usage

```
# binary: altkstar(lambda,
#                fixed=FALSE)
```

Arguments

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

Details

This is the version given in Snijders et al. (2006). The gwdegree and altkstar produce mathematically equivalent models, as long as they are used together with the edges (or kstar(1)) term, yet the interpretation of the gwdegree parameters is slightly more straightforward than the interpretation of the altkstar parameters. For this reason, we recommend the use of the gwdegree instead of altkstar. See Section 3 and especially equation (13) of Hunter (2007) for details.

Note

This term can only be used with undirected networks.

See Also

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

---

anova.ergm   *ANOVA for ERGM Fits*

Description

Compute an analysis of variance table for one or more ERGM fits.

Usage

```
## S3 method for class 'ergm'
anova(object, ..., eval.loglik = FALSE)

## S3 method for class 'ergmlist'
anova(object, ..., eval.loglik = FALSE)
```

Arguments

object, ...	objects of class [ergm](#), usually, a result of a call to [ergm](#).
eval.loglik	a logical specifying whether the log-likelihood will be evaluated if missing.

Details

Specifying a single object gives a sequential analysis of variance table for that fit. That is, the reductions in the residual sum of squares as each term of the formula is added in turn are given in the rows of a table, plus the residual sum of squares.

The table will contain F statistics (and P values) comparing the mean square for the row to the residual mean square.

If more than one object is specified, the table has a row for the residual degrees of freedom and sum of squares for each model. For all but the first model, the change in degrees of freedom and sum of squares is also given. (This only make statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

Optionally the table can include test statistics. Normally the F statistic is most appropriate, which compares the mean square for a row to the residual sum of squares for the largest model considered. If scale is specified chi-squared tests can be used. Mallows' C_p statistic is the residual sum of squares plus twice the estimate of σ^2 times the residual degrees of freedom.

If any of the objects do not have estimated log-likelihoods, produces an error, unless eval.loglik=TRUE.

Value

An object of class "anova" inheriting from class "data.frame".

Warning

The comparison between two or more models will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and 's default of na.action = na.omit is used, and anova.ergmlist will detect this with an error.

See Also

The model fitting function ergm, anova, logLik.ergm for adding the log-likelihood to an existing ergm object.

Examples

```
data(molecule)
molecule %v% "atomic type" <- c(1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3)
fit0 <- ergm(molecule ~ edges)
anova(fit0)
fit1 <- ergm(molecule ~ edges + nodefactor("atomic type"))
anova(fit1)

fit2 <- ergm(molecule ~ edges + nodefactor("atomic type") +  gwesp(0.5,
  fixed=TRUE), eval.loglik=TRUE) # Note the eval.loglik argument.
anova(fit0, fit1)
anova(fit0, fit1, fit2)
```

```
approx.hotelling.diff.test
```
Approximate Hotelling T^2-Test for One or Two Population Means

---

Description

A multivariate hypothesis test for a single population mean or a difference between them. This version attempts to adjust for multivariate autocorrelation in the samples.

Usage

```
approx.hotelling.diff.test(
  x,
  y = NULL,
  mu0 = 0,
  assume.indep = FALSE,
  var.equal = FALSE,
  ...
)
```

Arguments

x	a numeric matrix of data values with cases in rows and variables in columns.
y	an optinal matrix of data values with cases in rows and variables in columns for a 2-sample test.
mu0	an optional numeric vector: for a 1-sample test, the poulation mean under the null hypothesis; and for a 2-sample test, the difference between population means under the null hypothesis; defaults to a vector of 0s.
assume.indep	if TRUE, performs an ordinary Hotelling's test without attempting to account for autocorrelation.
var.equal	for a 2-sample test, perform the pooled test: assume population variance-covariance matrices of the two variables are equal.
...	additional arguments, passed on to [spectrum0.mvar()](), etc.; in particular, order.max= can be used to limit the order of the AR model used to estimate the effective sample size.

Value

An object of class htest with the following information:

statistic	The T^2 statistic.
parameter	Degrees of freedom.
p.value	P-value.
method	Method specifics.
null.value	Null hypothesis mean or mean difference.

alternative	Always "two.sided".
estimate	Sample difference.
covariance	Estimated variance-covariance matrix of the estimate of the difference.
covariance.x	Estimated variance-covariance matrix of the estimate of the mean of x.
covariance.y	Estimated variance-covariance matrix of the estimate of the mean of y.

It has a print method print.htest().

Note

For mcmc.list input, the variance for this test is estimated with unpooled means. This is not strictly correct.

References

Hotelling, H. (1947). Multivariate Quality Control. In C. Eisenhart, M. W. Hastay, and W. A. Wallis, eds. Techniques of Statistical Analysis. New York: McGraw-Hill.

See Also

t.test()

---

as.network.numeric   *Create a Simple Random network of a Given Size*

Description

as.network.numeric creates a random Bernoulli network of the given size as an object of class network.

Usage

```
## S3 method for class 'numeric'
as.network(
  x,
  directed = TRUE,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE,
  ignore.eval = TRUE,
  names.eval = NULL,
  edge.check = FALSE,
  density = NULL,
  init = NULL,
  numedges = NULL,
  ...
)
```

Arguments

`x`	count; the number of nodes in the network
`directed`	logical; should edges be interpreted as directed?
`hyper`	logical; are hyperedges allowed? Currently ignored.
`loops`	logical; should loops be allowed? Currently ignored.
`multiple`	logical; are multiplex edges allowed? Currently ignored.
`bipartite`	count; should the network be interpreted as bipartite? If present (i.e., non-NULL) it is the count of the number of actors in the bipartite network. In this case, the number of nodes is equal to the number of actors plus the number of events (with all actors preceding all events). The edges are then interpreted as nondirected.
`ignore.eval`	logical; ignore edge values? Currently ignored.
`names.eval`	optionally, the name of the attribute in which edge values should be stored. Currently ignored.
`edge.check`	logical; perform consistency checks on new edges?
`density`	numeric; the probability of a tie for Bernoulli networks. If neither density nor `init` is given, it defaults to the number of nodes divided by the number of dyads (so the expected number of ties is the same as the number of nodes.)
`init`	numeric; the log-odds of a tie for Bernoulli networks. It is only used if density is not specified.
`numedges`	count; if present, sample the Bernoulli network conditional on this number of edges (rather than independently with the specified probability).
`...`	additional arguments

Details

The network will not have vertex, edge or network attributes. These can be added with operators such as %v%, %n%, %e%.

Value

An object of class `network`

References

Butts, C.T. 2002. "Memory Structures for Relational Data in R: Classes and Interfaces" Working Paper.

See Also

`network`

Examples

```
#Draw a random directed network with 25 nodes
g<-network(25)
#Draw a random undirected network with density 0.1
g<-network(25, directed=FALSE, density=0.1)
#Draw a random bipartite network with 4 actors and 6 events and density 0.1
g<-network(10, bipartite=4, density=0.1)
```

---

asymmetric-ergmTerm   *Asymmetric dyads*

Description

This term adds one network statistic to the model equal to the number of pairs of actors for which exactly one of $(i{\rightarrow}j)$ or $(j{\rightarrow}i)$ exists.

Usage

```
# binary: asymmetric(attr=NULL, diff=FALSE, keep=NULL, levels=NULL)
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) If specified, only symmetric pairs that match on the vertex attribute are counted.

diff   Used in the same way as for the nodematch term. (See nodematch (?nodematch) for details.)

keep   deprecated

level   Used in the same way as for the nodematch term. (See nodematch (?nodematch) for details.)

Note

This term can only be used with directed networks.

The argument keep is retained for backwards compatibility and may be removed in a future version. When both keep and levels are passed, levels overrides keep.

See Also

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

---

atleast-ergmTerm   *Number of dyads with values greater than or equal to a threshold*

Description

Adds the number of statistics equal to the length of `threshold` equaling to the number of dyads whose values equal or exceed the corresponding element of `threshold` .

Usage

```
# valued: atleast(threshold=0)
```

Arguments

threshold   vector of numerical values

See Also

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

---

atmost-ergmTerm   *Number of dyads with values less than or equal to a threshold*

Description

Adds the number of statistics equal to the length of `threshold` equaling to the number of dyads whose values equal or are exceeded by the corresponding element of `threshold` .

Usage

```
# valued: atmost(threshold=0)
```

Arguments

threshold   a vector of numerical values

See Also

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

---

attrcov-ergmTerm   *Edge covariate by attribute pairing*

Description

This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network, where the covariate value for a given edge is determined by its mixing type on `attr`. Undirected networks are regarded as having undirected mixing, and it is assumed that `mat` is symmetric in that case.

This term can be useful for simulating large networks with many mixing types, where `nodemix` would be slow due to the large number of statistics, and `edgecov` cannot be used because an adjacency matrix would be too big.

Usage

```
# binary: attrcov(attr, mat)
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?`nodal_attributes`) for details.)
mat	a matrix of covariates with the same dimensions as a mixing matrix for `attr`

See Also

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

B-ergmTerm   *Wrap binary terms for use in valued models*

Description

Wraps binary `ergm` terms for use in valued models, with `formula` specifying which terms are to be wrapped and `form` specifying how they are to be used and how the binary network they are evaluated on is to be constructed.

Usage

```
# valued: B(formula, form)
```

Arguments

formula   a one-sided [ergm()](#)-style formula whose RHS contains the binary ergm terms to be evaluated. Which terms may be used depends on the argument `form`

form   One of three values:

- `"sum"`: see section "Generalizations of binary terms" in [ergmTerm](#) help; all terms in `formula` must be dyad-independent.
- `"nonzero"`: section "Generalizations of binary terms" in [ergmTerm](#) help; any binary ergm terms may be used in `formula` .
- a one-sided formula value-dependent network. `form` must contain one "valued" ergm term, with the following properties:
  - dyadic independence;
  - dyadwise contribution of either 0 or 1; and
  - dyadwise contribution of 0 for a 0-valued dyad.

  Formally, this means that it is expressable as

$$g(y) = \sum_{i,j} f_{i,j}(y_{i,j}),$$

  where for all i, j, and y, $f_{i,j}(y_{i,j})$ is either 0 or 1 and, in particular, $f_{i,j}(0) = 0$.
  Examples of such terms include `nonzero` , `ininterval()` , `atleast()` , `atmost()` , `greaterthan()` , `lessthen()` , and `equalto()` .
  Then, the value of the statistic will be the value of the statistics in `formula` evaluated on a binary network that is defined to have an edge if and only if the corresponding dyad of the valued network adds 1 to the valued term in `form` .

Details

For example, `B(~nodecov("a"), form="sum")` is equivalent to `nodecov("a", form="sum")` and similarly with `form="nonzero"` .

When a valued implementation is available, it should be preferred, as it is likely to be faster.

See Also

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

---

b1concurrent-ergmTerm   *Concurrent node count for the first mode in a bipartite network*

Description

This term adds one network statistic to the model, equal to the number of nodes in the first mode of the network with degree 2 or higher. The first mode of a bipartite network object is sometimes known as the "actor" mode. This term can only be used with undirected bipartite networks.

Usage

```
# binary: b1concurrent(by=NULL, levels=NULL)
```

Arguments

by   optional argument specifying a vertex attribute (see Specifying Vertex attributes
            and Levels (?nodal_attributes) for details). It functions just like the by ar-
            gument of the b1degree term. Without the optional argument, this statistic is
            equivalent to b1mindegree(2) .

levels   TODO (See Specifying Vertex attributes and Levels ([?nodal_attributes](#)) for
            details.)

See Also

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

---

b1cov-ergmTerm   *Main effect of a covariate for the first mode in a bipartite network*

Description

This term adds a single network statistic for each quantitative attribute or matrix column to the
model equaling the total value of attr(i) for all edges /eqn(i,j) in the network. This term may only
be used with bipartite networks. For categorical attributes, see b1factor .

Usage

```
# binary: b1cov(attr)

# valued: b1cov(attr, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
            for details.)

form   character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See [ergm-options](#) for how
to invoke the old behaviour.

See Also

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

---

b1degrange-ergmTerm   *Degree range for the first mode in a bipartite network*

Description

This term adds one network statistic to the model for each element of from (or to); the ith such statistic equals the number of nodes of the first mode ("actors") in the network of degree greater than or equal to from[i] but strictly less than to[i] , i.e. with edge count in semiopen interval [from,to) .

This term can only be used with bipartite networks; for directed networks see idegrange and odegrange . For undirected networks, see degrange , and see b2degrange for degrees of the second mode ("events").

Usage

```
# binary: b1degrange(from, to=`+Inf`, by=NULL, homophily=FALSE, levels=NULL)
```

Arguments

from, to	vectors of distinct integers. If one of the vectors have length 1, it is recycled to the length of the other. Otherwise, it must have the same length.
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

b1degree-ergmTerm   *Degree for the first mode in a bipartite network*

Description

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

Usage

```
# binary: b1degree(d, by=NULL, levels=NULL)
```

Arguments

d	a vector of distinct integers.
by	a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified then each node's degree is tabulated only with other nodes having the same value of the by attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Note

This term can only be used with undirected bipartite networks.

See Also

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

---

b1degrees-ergmConstraint

Preserve the actor degree for bipartite networks

Description

For bipartite networks, preserve the degree for the first mode of each vertex of the given network, while allowing the degree for the second mode to vary.

Usage

```
# b1degrees
```

See Also

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

---

b1dsp-ergmTerm *Dyadwise shared partners for dyads in the first bipartition*

Description

This term adds one network statistic to the model for each element in d ; the ith such statistic equals the number of dyads in the first bipartition with exactly d[i] shared partners. (Those shared partners, of course, must be members of the second bipartition.) This term can only be used with bipartite networks.

Usage

```
# binary: b1dsp(d)
```

Arguments

d   a vector of distinct integers.

See Also

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

---

b1factor-ergmTerm *Factor attribute effect for the first mode in a bipartite network*

Description

This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attr attribute. Each of these statistics gives the number of times a node with that attribute in the first mode of the network appears in an edge. The first mode of a bipartite network object is sometimes known as the "actor" mode.

Usage

```
# binary: b1factor(attr, base=1, levels=-1)

# valued: b1factor(attr, base=1, levels=-1, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                for details.)
base   deprecated
levels   this optional argument controls which levels of the attribute attributes and Levels
                ([?nodal_attributes](#)) for details.)
form   character how to aggregate tie values in a valued ERGM

Note

To include all attribute values is usually not a good idea, because the sum of all such statistics equals the number of edges and hence a linear dependency would arise in any model also including edges. The default, levels=-1, is therefore to omit the first (in lexicographic order) attribute level. To include all levels, pass either levels=TRUE (i.e., keep all levels) or levels=NULL (i.e., do not filter levels).

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

This term can only be used with undirected bipartite networks.

See Also

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

---

b1mindegree-ergmTerm   *Minimum degree for the first mode in a bipartite network*

Description

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

Usage

```
# binary: b1mindegree(d)
```

Arguments

d   a vector of distinct integers.

Note

This term can only be used with undirected bipartite networks.

See Also

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

b1nodematch-ergmTerm	*Nodal attribute-based homophily effect for the first mode in a bipartite network*

Description

This term is introduced in Bomiriya et al (2014). With the default `alpha` and `beta` values, this term will simply be a homophily based two-star statistic. This term adds one statistic to the model unless `diff` is set to `TRUE` , in which case the term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attr` attribute.

Usage

```
# binary: b1nodematch(attr, diff=FALSE, keep=NULL, alpha=1, beta=1, byb2attr=NULL,
#                     levels=NULL)
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
diff	by default, one statistic will be added to the model. If `diff` is set to `TRUE`, one statistic will be added for each unique value of the `attr` attribute
keep	deprecated
alpha, beta	optional discount parameters both of which take values from [0, 1], only one should be set at one time
byb2attr	specifies a second mode categorical attribute. Setting this argument will separate the orginal statistics based on the values of the set second mode attribute— i.e. for example, if `diff` is `FALSE` , then the sum of all the statistics for each level of this second-mode attribute will be equal to the original b1nodematch statistic where byb2attr set to `NULL` .
levels	select a subset of `attr` values to include. (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Details

If an `alpha` discount parameter is used, each of these statistics gives the sum of the number of common second-mode nodes raised to the power `alpha` for each pair of first-mode nodes with that attribute. If a `beta` discount parameter is used, each of these statistics gives half the sum of the number of two-paths with two first-mode nodes with that attribute as the two ends of the two path raised to the power `beta` for each edge in the network.

Note

This term can only be used with undirected bipartite networks.

The argument keep is retained for backwards compatibility and may be removed in a future version. When both keep and levels are passed, levels overrides keep.

See Also

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

---

b1sociality-ergmTerm   *Degree*

Description

This term adds one network statistic for each node in the first bipartition, equal to the number of ties of that node. This term can only be used with bipartite networks. For directed networks, see sender and receiver. For unipartite networks, see sociality.

Usage

```
# binary: b1sociality(nodes=-1)

# valued: b1sociality(nodes=-1, form="sum")
```

Arguments

nodes   By default, nodes=-1 means that the statistic for the first node (in the second bipartition) will be omitted, but this argument may be changed to control which statistics are included. The nodes argument is interpreted using the new UI for level specification (see Specifying Vertex Attributes and Levels (?nodal_attributes) for details), where both the attribute and the sorted unique values are the vector of vertex indices (nb1 + 1):n , where nb1 is the size of the first bipartition and n is the total number of nodes in the network. Thus nodes=120 will include only the statistic for the 120th node in the second biparition, while nodes=I(120) will include only the statistic for the 120th node in the entire network.

form   character how to aggregate tie values in a valued ERGM

See Also

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

---

b1star-ergmTerm   *k-Stars for the first mode in a bipartite network*

Description

This term adds one network statistic to the model for each element in k . The i th such statistic
counts the number of distinct k[i] -stars whose center node is in the first mode of the network.
The first mode of a bipartite network object is sometimes known as the "actor" mode. A k -star is
defined to be a center node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $\{N, O_i\}$
exist for $i = 1, \ldots, k$. If args is specified then the count is over the number of k-stars (with center
node in the first mode) where all nodes have the same value of the attribute. This term can only be
used for undirected bipartite networks.

Usage

```
# binary: b1star(k, attr=NULL, levels=NULL)
```

Arguments

k	a vector of distinct integers
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Note

b1star(1) is equal to b2star(1) and to edges .

See Also

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

---

b1starmix-ergmTerm   *Mixing matrix for k-stars centered on the first mode of a bipartite net-*
work

Description

This term counts all k-stars in which the b2 nodes (called events in some contexts) are homophilous
in the sense that they all share the same value of attr . However, the b1 node (in some contexts, the
actor) at the center of the k-star does NOT have to have the same value as the b2 nodes; indeed, the
values taken by the b1 nodes may be completely distinct from those of the b2 nodes, which allows
for the use of this term in cases where there are two separate nodal attributes, one for the b1 nodes
and another for the b2 nodes (in this case, however, these two attributes should be combined to form
a single nodal attribute, attr). A different statistic is created for each value of attr seen in a b1
node, even if no k-stars are observed with this value.

Usage

```
# binary: b1starmix(k, attr, base=NULL, diff=TRUE)
```

Arguments

k   only a single value of k is allowed

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
             for details.)

base   deprecated

diff   whether a different statistic is created for each value seen in a b2 node. When
             diff=TRUE, the default, a different statistic is created for each value and thus the
             behavior of this term is reminiscent of the nodemix term, from which it takes its
             name; when diff=FALSE , all homophilous k-stars are counted together, though
             these k-stars are still categorized according to the value of the central b1 node.

Note

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and levels are passed, levels overrides base.

See Also

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

---

b1twostar-ergmTerm   *Two-star census for central nodes centered on the first mode of a bi-*
                              partite network

Description

This term takes two nodal attributes. Assuming that there are n_1 values of b1attr among the b1
nodes and n_2 values of b2attr among the b2 nodes, then the total number of distinct categories of
two stars according to these two attributes is $n_1(n_2)(n_2+1)/2$. By default, this model term creates
a distinct statistic counting each of these categories.

Usage

```
# binary: b1twostar(b1attr, b2attr, base=NULL, b1levels=NULL, b2levels=NULL, levels2=NULL)
```

Arguments

b1attr   b1 nodes (actors in some contexts) (see Specifying Vertex attributes and Levels
             (?nodal_attributes) for details)

b2attr   b2 nodes (events in some contexts). If b2attr is not passed, it is assumed to be
             the same as b1attr .

b1levels, b2levels, base, levels2
             used to leave some of the categories out (see Specifying Vertex attributes and
             Levels (?nodal_attributes) for details)

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels2 are passed, levels2 overrides base.

See Also

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

---

b2concurrent-ergmTerm   *Concurrent node count for the second mode in a bipartite network*

Description

This term adds one network statistic to the model, equal to the number of nodes in the second mode of the network with degree 2 or higher. The second mode of a bipartite network object is sometimes known as the "event" mode. Without the optional argument, this statistic is equivalent to b2mindegree(2).

Usage

```
# binary: b2concurrent(by=NULL)
```

Arguments

by   This optional argument specifie a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details); it functions just like the by argument of the b2degree term.

Note

This term can only be used with undirected bipartite networks.

See Also

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

---

b2cov-ergmTerm   *Main effect of a covariate for the second mode in a bipartite network*

Description

This term adds a single network statistic for each quantitative attribute or matrix column to the model equaling the total value of attr(j) for all edges (i, j) in the network. This term may only be used with bipartite networks. For categorical attributes, see b2factor.

Usage

```
# binary: b2cov(attr)

# valued: b2cov(attr, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See ergm-options for how to invoke the old behaviour.

See Also

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

---

b2degrange-ergmTerm   *Degree range for the second mode in a bipartite network*

Description

This term adds one network statistic to the model for each element of from (or to); the i th such statistic equals the number of nodes of the second mode ("events") in the network of degree greater than or equal to from[i] but strictly less than to[i] , i.e. with edge count in semiopen interval [from,to) .

This term can only be used with bipartite networks; for directed networks see idegrange and odegrange . For undirected networks, see degrange , and see b1degrange for degrees of the first mode ("actors").

Usage

```
# binary: b2degrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
```

Arguments

`from, to`	vectors of distinct integers. If one of the vectors have length 1, it is recycled to the length of the other. Otherwise, it must have the same length.
`by, homophily`	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and `homophily` is `TRUE` , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and `homophily` is `FALSE` (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
`levels`	TODO (See Specifying Vertex attributes and Levels ([?nodal_attributes](#)) for details.)

See Also

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

---

b2degree-ergmTerm	*Degree for the second mode in a bipartite network*

Description

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

Usage

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

Arguments

`d`	a vector of distinct integers
`by`	this optional term specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified then each node's degree is tabulated only with other nodes having the same value of the by attribute.

Note

This term can only be used with undirected bipartite networks.

See Also

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

---

b2degrees-ergmConstraint

Preserve the receiver degree for bipartite networks

Description

For bipartite networks, preserve the degree for the second mode of each vertex of the given network, while allowing the degree for the first mode to vary.

Usage

```
# b2degrees
```

See Also

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

---

b2dsp-ergmTerm   *Dyadwise shared partners for dyads in the second bipartition*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of dyads in the second bipartition with exactly d[i] shared partners. (Those shared partners, of course, must be members of the first bipartition.) This term can only be used with bipartite networks.

Usage

```
# binary: b2dsp(d)
```

Arguments

d   a vector of distinct integers

See Also

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

---

b2factor-ergmTerm   *Factor attribute effect for the second mode in a bipartite network*

Description

This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attr attribute. Each of these statistics gives the number of times a node with that attribute in the second mode of the network appears in an edge. The second mode of a bipartite network object is sometimes known as the "event" mode.

Usage

```
# binary: b2factor(attr, base=1, levels=-1)

# valued: b2factor(attr, base=1, levels=-1, form="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
base	deprecated
levels	this optional argument controls which levels of the attribute attributes and Levels ([?nodal_attributes](#)) for details.)
form	character how to aggregate tie values in a valued ERGM

Note

To include all attribute values is usually not a good idea, because the sum of all such statistics equals the number of edges and hence a linear dependency would arise in any model also including edges. The default, levels=-1, is therefore to omit the first (in lexicographic order) attribute level. To include all levels, pass either levels=TRUE (i.e., keep all levels) or levels=NULL (i.e., do not filter levels).

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

This term can only be used with undirected bipartite networks.

See Also

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

---

b2mindegree-ergmTerm   *Minimum degree for the second mode in a bipartite network*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of nodes in the second mode of a bipartite network with at least degree d[i] . The second mode of a bipartite network object is sometimes known as the "event" mode.

Usage

```
# binary: b2mindegree(d)
```

Arguments

d   a vector of distinct integers

Note

This term can only be used with undirected bipartite networks.

See Also

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

---

b2nodematch-ergmTerm   *Nodal attribute-based homophily effect for the second mode in a bipartite network*

Description

This term is introduced in Bomiriya et al (2014). With the default alpha and beta values, this term will simply be a homophily based two-star statistic. This term adds one statistic to the model unless diff is set to TRUE , in which case the term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attr attribute.

Usage

```
# binary: b2nodematch(attr, diff=FALSE, keep=NULL, alpha=1, beta=1, byb1attr=NULL,
#                      levels=NULL)
```

Arguments

diff   by default, one statistic will be added to the model. If `diff` is set to `TRUE`, one
                statistic will be added for each unique value of the `attr` attribute

keep   deprecated

alpha, beta   optional discount parameters both of which take values from [0, 1], only one
                should be set at one time

byb2attr   specifies a second mode categorical attribute. Setting this argument will separate
                the orginal statistics based on the values of the set second mode attribute— i.e.
                for example, if `diff` is `FALSE` , then the sum of all the statistics for each level
                of this second-mode attribute will be equal to the original b1nodematch statistic
                where `byb2attr` set to `NULL` .

levels   select a subset of `attr` values to include. (See Specifying Vertex attributes and
                Levels ([?nodal_attributes](#)) for details.)

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                for details.)

Details

If an `alpha` discount parameter is used, each of these statistics gives the sum of the number of
common first-mode nodes raised to the power `alpha` for each pair of second-mode nodes with that
attribute. If a `beta` discount parameter is used, each of these statistics gives half the sum of the
number of two-paths with two second-mode nodes with that attribute as the two ends of the two
path raised to the power `beta` for each edge in the network.

Note

This term can only be used with undirected bipartite networks.

The argument `keep` is retained for backwards compatibility and may be removed in a future version.
When both `keep` and `levels` are passed, `levels` overrides `keep`.

See Also

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

---

b2sociality-ergmTerm   *Degree*

Description

This term adds one network statistic for each node in the second bipartition, equal to the number of
ties of that node. For directed networks, see `sender` and `receiver` . For unipartite networks, see
`sociality` .

Usage

```
# binary: b2sociality(nodes=-1)

# valued: b2sociality(nodes=-1, form="sum")
```

Arguments

nodes   By default, nodes=-1 means that the statistic for the first node (in the sec-
         ond bipartition) will be omitted, but this argument may be changed to con-
         trol which statistics are included. The nodes argument is interpreted using
         the new UI for level specification (see Specifying Vertex Attributes and Lev-
         els (?nodal_attributes) for details), where both the attribute and the sorted
         unique values are the vector of vertex indices (nb1 + 1):n , where nb1 is the
         size of the first bipartition and n is the total number of nodes in the network.
         Thus nodes=120 will include only the statistic for the 120th node in the second
         biparition, while nodes=I(120) will include only the statistic for the 120th node
         in the entire network.

form   character how to aggregate tie values in a valued ERGM

Note

This term can only be used with undirected bipartite networks.

See Also

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

---

    b2star-ergmTerm   *k-Stars for the second mode in a bipartite network*

Description

This term adds one network statistic to the model for each element in k . The i th such statistic
counts the number of distinct k[i] -stars whose center node is in the second mode of the network.
The second mode of a bipartite network object is sometimes known as the "event" mode. A k -star
is defined to be a center node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties
$\{N, O_i\}$ exist for $i = 1, \ldots, k$. This term can only be used for undirected bipartite networks.

Usage

```
# binary: b2star(k, attr=NULL, levels=NULL)
```

Arguments

k	a vector of distinct integers
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) then the count is over the number of k-stars (with center node in the second mode) where all nodes have the same value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Note

b2star(1) is equal to b1star(1) and to edges.

See Also

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

---

b2starmix-ergmTerm	*Mixing matrix for k-stars centered on the second mode of a bipartite network*

Description

This term is exactly the same as b1starmix except that the roles of b1 and b2 are reversed.

Usage

```
# binary: b2starmix(k, attr, base=NULL, diff=TRUE)
```

Arguments

k	only a single value of k is allowed
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
base	deprecated
diff	whether a different statistic is created for each value seen in a b1 node. When diff=TRUE, the default, a different statistic is created for each value and thus the behavior of this term is reminiscent of the nodemix term, from which it takes its name; when diff=FALSE, all homophilous k-stars are counted together, though these k-stars are still categorized according to the value of the central b1 node.

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

See Also

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

---

b2twostar-ergmTerm   *Two-star census for central nodes centered on the second mode of a*
                            bipartite network

Description

This term is exactly the same as b1twostar except that the roles of b1 and b2 are reversed.

Usage

```
# binary: b2twostar(b1attr, b2attr, base=NULL, b1levels=NULL, b2levels=NULL, levels2=NULL)
```

Arguments

b1attr   b1 nodes (actors in some contexts) (see Specifying Vertex attributes and Levels
                (?nodal_attributes) for details)

b2attr   b2 nodes (events in some contexts). If b1attr is not passed, it is assumed to be
                the same as b2attr .

b1levels, b2levels, base, levels2
                used to leave some of the categories out (see Specifying Vertex attributes and
                Levels (?nodal_attributes) for details)

Note

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and levels are passed, levels overrides base.

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and levels2 are passed, levels2 overrides base.

See Also

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

---

balance-ergmTerm   *Balanced triads*

Description

This term adds one network statistic to the model equal to the number of triads in the network that
are balanced. The balanced triads are those of type 102 or 300 in the categorization of Davis and
Leinhardt (1972). For details on the 16 possible triad types, see ?triad.classify in the {sna}
package. For an undirected network, the balanced triads are those with an odd number of ties (i.e.,
1 and 3).

Usage

```
# binary: balance
```

See Also

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

---

bd-ergmConstraint   *Constrain maximum and minimum vertex degree*

Description

Condition on the number of inedge or outedges posessed by a node. See Placing Bounds on Degrees section for more information. ([?ergmConstraint](#))

Usage

```
# bd(attribs, maxout, maxin, minout, minin)
```

Arguments

attribs   a matrix of logicals with dimension (n_nodes, attrcount) for the attributes on
                  which we are conditioning, where attrcount is the number of distinct attributes
                  values to condition on.

maxout, maxin, minout, minin
                  matrices of alter attributes with the same dimension as attribs when used in
                  conjunction with attribs. Otherwise, vectors of integers specifying the rele-
                  vant limits. If the vector is of length 1, the limit is applied to all nodes. If an
                  individual entry is NA, then there is no restriction of that kind is applied.

See Also

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

BDStratTNT-ergmProposal

                          TNT proposal with degree bounds

Description

Implements a TNT proposal with any subset of the following features:

1. upper on degree, specified via the [bd](#) constraint's maxout, maxin, and attribs arguments

2. stratification of proposals according to mixing type on a vertex attribute, specified via the [strat](#) hint;

3. fixation of specified mixing types on a(nother) vertex attribute, specified via the [blocks](#) con-
   straint.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	sparse	bdmax blocks strat	-3	BDStratTNT	c
Bernoulli	bdmax sparse	blocks strat	0	BDStratTNT	c
Bernoulli	blocks sparse	bdmax strat	0	BDStratTNT	c
Bernoulli	strat sparse	bdmax blocks	0	BDStratTNT	c

`Bernoulli-ergmReference`

Bernoulli reference

Description

Specifies each dyad's baseline distribution to be Bernoulli with probability of the tie being 0.5 . This is the only reference measure used in binary mode.

Usage

```
# Bernoulli
```

See Also

ergmReference for index of reference distributions currently visible to the package.

`blockdiag-ergmConstraint`

Force a block-diagonal structure on the network

Description

Only dyads (i, j) for which `attr(i)==attr(j)` can have edges.

Note that the current implementation requires that blocks be contiguous for unipartite graphs, and for bipartite graphs, they must be contiguous within a partition and must have the same ordering in both partitions. (They do not, however, require that all blocks be represented in both partitions, but those that overlap must have the same order.)

If multiple block-diagonal constraints are given, or if `attr` is a vector with multiple attribute names, blocks will be constructed on all attributes matching.

Usage

```
# blockdiag(attr)
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                 for details.)

See Also

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

---

blocks-ergmConstraint   *Constrain "blocks" of dyads*

Description

Any dyad whose toggle would produce a nonzero change statistic for a nodemix term with the same
arguments will be fixed. Note that the levels2 argument has a different default value for blocks
than it does for nodemix .

Usage

```
# blocks(attr=NULL, levels=NULL, levels2=FALSE, b1levels=NULL, b2levels=NULL)
```

Arguments

attr   quantitative attribute (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 at-
                 tributes and Levels (?nodal_attributes) for details)

See Also

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

---

check.ErgmTerm   *Ensures an Ergm Term and its Arguments Meet Appropriate Conditions*

Description

Helper functions for implementing [ergm()](#) terms, to check whether the term can be used with the specified network. For information on ergm terms, see [ergmTerm](#). ergm.checkargs, ergm.checkbipartite, and ergm.checkderected are helper functions for an old API and are deprecated. Use check.ErgmTerm.

Usage

```
check.ErgmTerm(
  nw,
  arglist,
  directed = NULL,
  bipartite = NULL,
  nonnegative = FALSE,
  varnames = NULL,
  vartypes = NULL,
  defaultvalues = list(),
  required = NULL,
  dep.inform = rep(FALSE, length(required)),
  dep.warn = rep(FALSE, length(required)),
  argexpr = NULL
)
```

Arguments

nw	the network that term X is being checked against
arglist	the list of arguments for term X
directed	logical, whether term X requires a directed network; default=NULL
bipartite	whether term X requires a bipartite network (T or F); default=NULL
nonnegative	whether term X requires a network with only nonnegative weights; default=FALSE
varnames	the vector of names of the possible arguments for term X; default=NULL
vartypes	the vector of types of the possible arguments for term X, separated by commas; an empty string ("") or NA disables the check for that argument, and also see Details; default=NULL
defaultvalues	the list of default values for the possible arguments of term X; default=list()
required	the logical vector of whether each possible argument is required; default=NULL
dep.inform, dep.warn	
	a list of length equal to the number of arguments the term can take; if the corresponding element of the list is not FALSE, a [message()](#) or a [warning()](#) respectively will be issued if the user tries to pass it; if the element is a character string, it will be used as a suggestion for replacement.
argexpr	optional call typically obtained by calling substitute(arglist).

Details

The check.ErgmTerm function ensures for the [InitErgmTerm](#).X function that the term X:

- is applicable given the 'directed' and 'bipartite' attributes of the given network
- is not applied to a directed bipartite network
- has an appropiate number of arguments
- has correct argument types if arguments where provided
- has default values assigned if defaults are available

by halting execution if any of the first 3 criteria are not met.

As a convenience, if an argument is optional *and* its default is NULL, then NULL is assumed to be an acceptable argument type as well.

Value

A list of the values for each possible argument of term X; user provided values are used when given, default values otherwise. The list also has an attr(,"missing") attribute containing a named logical vector indicating whether a particular argument had been set to its default. If argexpr= argument is provided, attr(,"exprs") attribute is also returned, containing expressions.

---

cohab *Target statistics and model fit to a hypothetical 50,000-node network population with 50,000 nodes based on egocent*

Description

This dataset consists of three objects, each based on data from King County, Washington, USA (where Seattle is located) derived from the National Survey of Family Growth (NSFG) (https://www.cdc.gov/nchs/nsfg/index.htm). The full dataset cannot be released publicly, so some aspects of these objects are simulated based on the real data. These objects may be used to illustrate that network modeling may be performed using data that are collected on egos only, i.e., without directly observing information about alters in a network except for information reported from egos. The hypothetical population reepresented by this dataset consists of only a subset of individuals, as categorized by their age, race / ethnicity / immigration status, and gender and sexual identity.

Usage

    data(cohab)

Details

The three objects are

cohab_MixMat Mixing matrix on 'race'. Based on ego reports of the race / ethnicity / immigration status of their cohabiting partners, this matrix gives counts of ego-alter ties by the race of each individual for a hypothetical population. These counts are based on the NSFG mixing matrix. Only five categories of the 'race' variable are included here: Black, Black immigrant, Hispanic, Hispanic immigrant, and White.

cohab_PopWts Data frame of demographic characteristics together with relative counts (weights) in a hypothetical population. Individuals are classified according to five variables: age in years, race (same five categories of race / ethnicity / immigration status as above), sex (Male or Female), sexual identity (Female, Male who has sex with Females, or Male who has sex with Males or Females), and number of model-predicted persistent partnerships with non-cohabiting partners (0 or 1, where 1 means any nonzero value; the number is capped at 3), and number of partners (0 or 1).

cohab_TargetStats Vector of target (expected) statistics for a 15-term ERGM applied to a network of 50,000 nodes in which a tie represents a cohabitation relationship between two nodes. It is assumed for the purposes of these statistics that only male-female cohabitation relationships are allowed and that no individual may have such a relationship with more than one person. That is, each node must have degree zero or one. The ergm formula is: `~ edges + nodefactor("sex.ident", levels = 3) + nodecov("age") + nodecov("agesq") + nodefactor("race", levels = -5) + nodefactor("othr.net.deg", levels = -1) + nodematch("race", diff = TRUE) + absdiff("sqrt.age.adj")`

References

Krivitsky, P.N., Hunter, D.R., Morris, M., and Klumb, C. (2021). *ergm 4.0: New Features and Improvements*. arXiv

National Center for Health Statistics (NCHS). (2020). 2006-2015 National Survey of Family Growth Public-Use Data and Documentation. Hyattsville, MD: CDC National Center for Health Statistics. Retrieved from https://www.cdc.gov/nchs/nsfg/index.htm

See Also

ergm

---

coincidence-ergmTerm   *Coincident node count for the second mode in a bipartite (aka two-mode) network*

Description

By default this term adds one network statistic to the model for each pair of nodes of mode two. It is equal to the number of (first mode) mutual partners of that pair. The first mode of a bipartite network object is sometimes known as the "actor" mode and the seconds as the "event" mode. So this is the number of actors going to both events in the pair. This term can only be used with undirected bipartite networks.

Usage

```
# binary: coincidence(levels=NULL,active=0)
```

Arguments

levels   specifies which pairs of nodes in mode two to include. (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

active   selects pairs for which the observed count is at least active . Ignored if levels is specified. (Thus, indices passed as levels should correspond to indices when levels = NULL and active = 0.)

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See ergm-options for how to invoke the old behaviour.

See Also

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

---

concurrent-ergmTerm   *Concurrent node count*

Description

This term adds one network statistic to the model, equal to the number of nodes in the network with degree 2 or higher. This term can only be used with undirected networks.

Usage

```
# binary: concurrent(by=NULL, levels=NULL)
```

Arguments

by   this optional argument specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) It functions just like the by argument of the degree term.

See Also

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

---

concurrentties-ergmTerm

Concurrent tie count

Description

This term adds one network statistic to the model, equal to the number of ties incident on each actor beyond the first. This term can only be used with undirected networks.

Usage

```
# binary: concurrentties(by=NULL, levels=NULL)
```

Arguments

by	a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.); it functions just like the by argument of the degree term
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

CondB1Degree-ergmProposal

MHp for b1degree constraints

Description

MHp for $constraints = b1degrees$. For bipartite networks, randomly select an edge B1i, B2j and an empty dyad with the same node B1i, B1i, B2k, and propose to toggle both B1i, B2j and B1i, B2k. This ensures that the degrees of individual nodes in mode 1 are preserved.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	b1degrees		0	random	c

CondB2Degree-ergmProposal

MHp for b2degree constraints

Description

MHp for $constraints = b2degrees$. For bipartite networks, randomly select an edge B1j, B2i and an empty dyad with the same node B2i, B1k, B2i, and propose to toggle both B1j, B2i and B1k, B2i. This ensures that the degrees of individual nodes in mode 2 are preserved.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	b2degrees		0	random	c

CondDegree-ergmProposal

MHp for degree constraints

Description

MHp for $constraints = degree$. Propose either 4 toggles (MH_CondDegreeTetrad) or 6 toggles (MH_CondDegreeHexad) at once. For undirected networks, propose 4 toggles (MH_CondDegreeTetrad). MH_CondDegreeTetrad selects two edges with no nodes in common, A1-A2 and B1-B2, s.t. A1-B2 and B1-A2 are not edges, and propose to replace the former two by the latter two. MH_CondDegreeHexad selects three edges A1->A2, B1->B2, C1->C2 at random and rotate them to A1->B2, B1->C2, and C1->A2.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	degrees		0	random	c
Bernoulli	idegrees odegrees		0	random	c
Bernoulli	b1degrees b2degrees		0	random	c

CondDegreeDist-ergmProposal
                              MHp for degreedist constraints

Description

MHp for $constraints = degreedist$. Randomly select a node (T) and its edge (E). If the head node of the edge (H) has 1 degree more than another randomly select node (A), and A is disconnected to both T and H, then propose to toggle E and the dyad between T and A.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	degreedist		0	random	c

CondDegreeMix-ergmProposal
                              MHp for degree mix constraints

Description

MHp for $constraints = degreesmix$. Similar to InitErgmProposal.CondDegree, except that the toggle is proposed only if the mixing matrix of degrees is preserved before and after the toggle.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	degreesmix		0	random	c

```
CondInDegree-ergmProposal
```
MHp for idegree constraints

Description

MHp for $constraints = idegrees$. For directed networks, randomly select two dyads with a common head node, one having an edge one not, and propose to swap the tie from one tail to the other.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	idegrees		0	random	c

```
CondInDegreeDist-ergmProposal
```
MHp for idegreedist constraints

Description

MHp for $constraints = idegreedist$. For directed networks, similar to `InitErgmProposal.CondDegreeDist`, except for indegree case

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	idegreedist		0	random	c

`CondOutDegree-ergmProposal`

MHp for odegree constraints

Description

MHp for $constraints = odegrees$. For directed networks, randomly select two dyads with a common tail node, one having an edge and one not, and propose to swap the tie from one head to the other.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	odegrees		0	random	c

`CondOutDegreeDist-ergmProposal`

MHp for odegreedist constraints

Description

MHp for $constraints = odegreedist$. For directed networks, similar to `InitErgmProposal.CondDegreeDist`, except for outdegree case

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	odegreedist		0	random	c

ConstantEdges-ergmProposal
MHp for edges constraints

Description

MHp for $constraints = edges$. Propose pairs of toggles that keep number of edges the same. This is done by: a. choosing an existing edge at random; b. repeatedly choosing dyads at random until one is found that does not have an edge; and c. proposing toggling both these dyads. Note that step b. will be very inefficient if the network is nearly complete, so this proposal is NOT recommended for such networks. However, most network datasets are sparse, so this is not likely to be an issue.

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	edges	bd	0	random	c

---

control.ergm
Auxiliary for Controlling ERGM Fitting

Description

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

Usage

```
control.ergm(
  drop = TRUE,
  init = NULL,
  init.method = NULL,
 main.method = c("MCMLE", "Robbins-Monro", "Stochastic-Approximation", "Stepping"),
  force.main = FALSE,
  main.hessian = TRUE,
  checkpoint = NULL,
  resume = NULL,
  MPLE.samplesize = .Machine$integer.max,
  init.MPLE.samplesize = function(d, e) max(sqrt(d), e, 40) * 8,
  MPLE.type = c("glm", "penalized", "logitreg"),
  MPLE.maxit = 10000,
  MPLE.nonvar = c("warning", "message", "error"),
  MPLE.nonident = c("warning", "message", "error"),
  MPLE.nonident.tol = 1e-10,
```

```
       MPLE.constraints.ignore = FALSE,
       MCMC.prop = trim_env(~sparse),
       MCMC.prop.weights = "default",
       MCMC.prop.args = list(),
       MCMC.interval = NULL,
       MCMC.burnin = EVL(MCMC.interval * 16),
       MCMC.samplesize = NULL,
       MCMC.effectiveSize = NULL,
       MCMC.effectiveSize.damp = 10,
       MCMC.effectiveSize.maxruns = 16,
       MCMC.effectiveSize.burnin.pval = 0.2,
       MCMC.effectiveSize.burnin.min = 0.05,
       MCMC.effectiveSize.burnin.max = 0.5,
       MCMC.effectiveSize.burnin.nmin = 16,
       MCMC.effectiveSize.burnin.nmax = 128,
       MCMC.effectiveSize.burnin.PC = FALSE,
       MCMC.effectiveSize.burnin.scl = 32,
       MCMC.effectiveSize.order.max = NULL,
       MCMC.return.stats = TRUE,
       MCMC.runtime.traceplot = FALSE,
       MCMC.maxedges = Inf,
       MCMC.addto.se = TRUE,
       MCMC.packagenames = c(),
       SAN.maxit = 4,
       SAN.nsteps.times = 8,
       SAN = control.san(term.options = term.options, SAN.maxit = SAN.maxit,
        SAN.prop.weights = MCMC.prop.weights, SAN.prop.args = MCMC.prop.args, SAN.nsteps =
        EVL(MCMC.burnin, 16384) * SAN.nsteps.times, SAN.samplesize = EVL(MCMC.samplesize,
        1024), SAN.packagenames = MCMC.packagenames, parallel = parallel, parallel.type =
         parallel.type, parallel.version.check = parallel.version.check),
      MCMLE.termination = c("confidence", "Hummel", "Hotelling", "precision", "none"),
      MCMLE.maxit = 60,
      MCMLE.conv.min.pval = 0.5,
      MCMLE.confidence = 0.99,
      MCMLE.confidence.boost = 2,
      MCMLE.confidence.boost.threshold = 1,
      MCMLE.confidence.boost.lag = 4,
      MCMLE.NR.maxit = 100,
      MCMLE.NR.reltol = sqrt(.Machine$double.eps),
      obs.MCMC.mul = 1/4,
      obs.MCMC.samplesize.mul = sqrt(obs.MCMC.mul),
      obs.MCMC.samplesize = EVL(round(MCMC.samplesize * obs.MCMC.samplesize.mul)),
      obs.MCMC.effectiveSize = NVL3(MCMC.effectiveSize, . * obs.MCMC.mul),
      obs.MCMC.interval.mul = sqrt(obs.MCMC.mul),
      obs.MCMC.interval = EVL(round(MCMC.interval * obs.MCMC.interval.mul)),
      obs.MCMC.burnin.mul = sqrt(obs.MCMC.mul),
      obs.MCMC.burnin = EVL(round(MCMC.burnin * obs.MCMC.burnin.mul)),
      obs.MCMC.prop = MCMC.prop,
```

```
  obs.MCMC.prop.weights = MCMC.prop.weights,
  obs.MCMC.prop.args = MCMC.prop.args,
  obs.MCMC.impute.min_informative = function(nw) network.size(nw)/4,
  obs.MCMC.impute.default_density = function(nw) 2/network.size(nw),
 MCMLE.min.depfac = 2,
 MCMLE.sampsize.boost.pow = 0.5,
 MCMLE.MCMC.precision = if (startsWith("confidence", MCMLE.termination[1])) 0.1 else
    0.005,
 MCMLE.MCMC.max.ESS.frac = 0.1,
 MCMLE.metric = c("lognormal", "logtaylor", "Median.Likelihood", "EF.Likelihood",
    "naive"),
 MCMLE.method = c("BFGS", "Nelder-Mead"),
 MCMLE.dampening = FALSE,
 MCMLE.dampening.min.ess = 20,
 MCMLE.dampening.level = 0.1,
 MCMLE.steplength.margin = 0.05,
 MCMLE.steplength = NVL2(MCMLE.steplength.margin, 1, 0.5),
 MCMLE.steplength.parallel = c("observational", "never"),
 MCMLE.sequential = TRUE,
 MCMLE.density.guard.min = 10000,
 MCMLE.density.guard = exp(3),
 MCMLE.effectiveSize = 64,
 obs.MCMLE.effectiveSize = NVL3(MCMLE.effectiveSize, . * obs.MCMC.mul),
 MCMLE.interval = 1024,
 MCMLE.burnin = MCMLE.interval * 16,
 MCMLE.samplesize.per_theta = 32,
 MCMLE.samplesize.min = 256,
 MCMLE.samplesize = NULL,
 obs.MCMLE.samplesize.per_theta = round(MCMLE.samplesize.per_theta *
    obs.MCMC.samplesize.mul),
 obs.MCMLE.samplesize.min = 256,
 obs.MCMLE.samplesize = NULL,
 obs.MCMLE.interval = round(MCMLE.interval * obs.MCMC.interval.mul),
 obs.MCMLE.burnin = round(MCMLE.burnin * obs.MCMC.burnin.mul),
 MCMLE.steplength.solver = c("glpk", "lpsolve"),
 MCMLE.last.boost = 4,
 MCMLE.steplength.esteq = TRUE,
 MCMLE.steplength.miss.sample = function(x1) ceiling(sqrt(ncol(rbind(x1)))),
 MCMLE.steplength.min = 1e-04,
 MCMLE.effectiveSize.interval_drop = 2,
 MCMLE.save_intermediates = NULL,
 MCMLE.nonvar = c("message", "warning", "error"),
 MCMLE.nonident = c("warning", "message", "error"),
 MCMLE.nonident.tol = 1e-10,
 SA.phase1_n = NULL,
 SA.initial_gain = NULL,
 SA.nsubphases = 4,
 SA.niterations = NULL,
```

```
SA.phase3_n = NULL,
SA.interval = 1024,
SA.burnin = SA.interval * 16,
SA.samplesize = 1024,
RM.phase1n_base = 7,
RM.phase2n_base = 100,
RM.phase2sub = 7,
RM.init_gain = 0.5,
RM.phase3n = 500,
RM.interval = 1024,
RM.burnin = RM.interval * 16,
RM.samplesize = 1024,
Step.maxit = 50,
Step.gridsize = 100,
Step.interval = 1024,
Step.burnin = Step.interval * 16,
Step.samplesize = 1024,
CD.samplesize.per_theta = 128,
obs.CD.samplesize.per_theta = 128,
CD.nsteps = 8,
CD.multiplicity = 1,
CD.nsteps.obs = 128,
CD.multiplicity.obs = 1,
CD.maxit = 60,
CD.conv.min.pval = 0.5,
CD.NR.maxit = 100,
CD.NR.reltol = sqrt(.Machine$double.eps),
CD.metric = c("naive", "lognormal", "logtaylor", "Median.Likelihood",
  "EF.Likelihood"),
CD.method = c("BFGS", "Nelder-Mead"),
CD.dampening = FALSE,
CD.dampening.min.ess = 20,
CD.dampening.level = 0.1,
CD.steplength.margin = 0.5,
CD.steplength = 1,
CD.adaptive.epsilon = 0.01,
CD.steplength.esteq = TRUE,
CD.steplength.miss.sample = function(x1) ceiling(sqrt(ncol(rbind(x1)))),
CD.steplength.min = 1e-04,
CD.steplength.parallel = c("observational", "always", "never"),
CD.steplength.solver = c("glpk", "lpsolve"),
loglik = control.logLik.ergm(),
term.options = NULL,
seed = NULL,
parallel = 0,
parallel.type = NULL,
parallel.version.check = TRUE,
parallel.inherit.MT = FALSE,
```

```
    ...
)
```

Arguments

drop   Logical: If TRUE, terms whose observed statistic values are at the extremes of
                their possible ranges are dropped from the fit and their corresponding parameter
                estimates are set to plus or minus infinity, as appropriate. This is done because
                maximum likelihood estimates cannot exist when the vector of observed statistic
                lies on the boundary of the convex hull of possible statistic values.

init   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 [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. Note that offset coefficients alone can be more conve-
                  niently specified using [ergm()]() argument offset.coef. If both offset.coef
                  and init arguments are given, values in offset.coef will take precedence.
                • 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 [ergm()]() run, though checkpoint and 'resume' would be better un-
                der most circumstances.

init.method   A chatacter vector or NULL. The default method depends on the reference mea-
                sure used. For the binary ("Bernoulli") ERGMs, with dyad-independent con-
                straints, it's maximum pseudo-likelihood estimation (MPLE). Other valid values
                include "zeros" for a 0 vector of appropriate length and "CD" for contrastive di-
                vergence. If passed explicitly, this setting overrides the reference's limitations.

                Valid initial methods for a given reference are set by the InitErgmReference.*
                function.

main.method   One of "MCMLE" (default),"Robbins-Monro", "Stochastic-Approximation", or
                "Stepping". Chooses the estimation method used to find the MLE. MCMLE at-
                tempts to maximize an approximation to the log-likelihood function. Robbins-Monro
                and Stochastic-Approximation are both stochastic approximation algorithms
                that try to solve the method of moments equation that yields the MLE in the case
                of an exponential family model. Another alternative is a partial stepping algo-
                rithm (Stepping) as in Hummel et al. (2012). The direct use of the likelihood
                function has many theoretical advantages over stochastic approximation, but the
                choice will depend on the model and data being fit. See Handcock (2000) and
                Hunter and Handcock (2006) for details.

                Note that in recent versions of ERGM, the enhancements of Stepping have been
                folded into the default MCMLE, which is able to handle more modeling scenarios;
                and Stepping may be removed in a future version.

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

main.hessian   Logical: If TRUE, then an approximate Hessian matrix is used in the MCMC-
                  based estimation method.

checkpoint   At the start of every iteration, save the state of the optimizer in a way that will
                  allow it to be resumed. The name is passed through sprintf() with iteration
                  number as the second argument. (For example, checkpoint="step_%03d.RData"
                  will save to step_001.RData, step_002.RData, etc.)

resume   If given a file name of an RData file produced by checkpoint, the optimizer will
                  attempt to resume after restoring the state. Control parameters from the saved
                  state will be reused, except for those whose value passed via control.ergm()
                  had change from the saved run. Note that if the network, the model, or some
                  critical settings differ between runs, the results may be undefined.

MPLE.samplesize, init.MPLE.samplesize
                  These parameters control the maximum number of dyads (potential ties) that
                  will be used by the MPLE to construct the predictor matrix for its logistic re-
                  gression. In general, the algorithm visits dyads in a systematic sample that, if
                  it does not hit one of these limits, will visit every informative dyad. If a limit
                  is exceeded, case-control approximation to the likelihood, comprising all edges
                  and those non-edges that have been visited by the algorithm before the limit was
                  exceeded will be used.

                  MPLE.samplesize limits the number of dyads visited, unless the MPLE is being
                  computed for the purpose of being the initial value for MCMC-based estimation,
                  in which case init.MPLE.samplesize is used instead, All of these can be spec-
                  ified either as numbers or as function(d,e) taking the number of informative
                  dyads and informative edges. Specifying or returning a larger number than the
                  number of informative dyads is safe.

MPLE.type   One of "glm", "penalized", or "logitreg". Chooses method of calculating
                  MPLE. "glm" is the usual formal logistic regression called via [glm](glm), whereas
                  "penalized" uses the bias-reduced method of Firth (1993) as originally imple-
                  mented by Meinhard Ploner, Daniela Dunkler, Harry Southworth, and Georg
                  Heinze in the "logistf" package. "logitreg" is an "in-house" implementation
                  that is slower and probably less stable but supports nonlinear logistic regression.
                  It is invoked automatically when the model has curved terms.

MPLE.maxit   Maximum number of iterations for "logitreg" implementation of MPLE.

MPLE.nonident, MPLE.nonident.tol, MPLE.nonvar, MCMLE.nonident, MCMLE.nonident.tol, MCMLE.nonvar
                  A rudimentary nonidentifiability/multicollinearity diagnostic. If MPLE.nonident.tol
                  > 0, test the MPLE covariate matrix or the CD statistics matrix has linearly de-
                  pendent columns via QR decomposition with tolerance MPLE.nonident.tol.
                  This is often (not always) indicative of a non-identifiable (multicollinear) model.
                  If nonidentifiable, depending on MPLE.nonident issue a warning, an error, or a
                  message specifying the potentially redundant statistics. Before the diagnostic
                  is performed, covariates that do not vary (i.e., all-zero columns) are dropped,
                  with their handling controlled by MPLE.nonvar. The corresponding MCMLE.*
                  arguments provide a similar diagnostic for the unconstrained MCMC sample's
                  estimating functions.

MPLE.constraints.ignore

If TRUE, MPLE will ignore all dyad-independent constraints except for those due to attributes missingness. This can be used to avert evaluating and storing the [rlebdm](#)s for very large networks except where absolutely necessary. Note that this can be very dangerous unless you know what you are doing.

MCMC.prop   Specifies the proposal (directly) and/or a series of "hints" about the structure of the model being sampled. The specification is in the form of a one-sided formula with hints separated by + operations. If the LHS exists and is a string, the proposal to be used is selected directly.

A common and default "hint" is ~sparse, indicating that the network is sparse and that the sample should put roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling.

MCMC.prop.weights

Specifies the proposal distribution used in the MCMC Metropolis-Hastings algorithm. Possible choices depending on selected reference and constraints arguments of the [ergm()](#) function, but often include "TNT" and "random", and the "default" is to use the one with the highest priority available.

MCMC.prop.args   An alternative, direct way of specifying additional arguments to proposal.

MCMC.interval   Number of proposals between sampled statistics. Increasing interval will reduces the autocorrelation in the sample, and may increase the precision in estimates by reducing MCMC error, at the expense of time. Set the interval higher for larger networks.

MCMC.burnin   Number of proposals before any MCMC sampling is done. It typically is set to a fairly large number.

MCMC.samplesize

Number of network statistics, randomly drawn from a given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm. Increasing sample size may increase the precision in the estimates by reducing MCMC error, at the expense of time. Set it higher for larger networks, or when using parallel functionality.

MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.p

Set MCMC.effectiveSize to a non-NULL value to adaptively determine the burn-in and the MCMC length needed to get the specified effective size; 50 is a reasonable value. In the adaptive MCMC mode, MCMC is run forward repeatedly (MCMC.samplesize*MCMC.interval steps, up to MCMC.effectiveSize.maxruns times) until the target effective sample size is reached or exceeded.

After each run, the returned statistics are mapped to the estimating function scale, then an exponential decay model is fit to the scaled statistics to find that burn-in which would reduce the difference between the initial values of statistics and their equilibrium values by a factor of MCMC.effectiveSize.burnin.scl of what it initially was, bounded by MCMC.effectiveSize.min and MCMC.effectiveSize.max as proportions of sample size. If the best-fitting decay exceeds MCMC.effectiveSize.max, the exponential model is considered to be unsuitable and MCMC.effectiveSize.min is used.

A Geweke diagnostic is then run, after thinning the sample to MCMC.effectiveSize.burnin.nmax. If this Geweke diagnostic produces a p-value higher than MCMC.effectiveSize.burnin.pval, it is accepted.

If `MCMC.effectiveSize.burnin.PC>0`, instead of using the full sample for burn-in estimation, at most this many principal components are used instead.

The effective size of the post-burn-in sample is computed via Vats, Flegal, and Jones (2015), and compared to the target effective size. If it is not matched, the MCMC run is resumed, with the additional draws needed linearly extrapolated but weighted in favor of the baseline `MCMC.samplesize` by the weighting factor `MCMC.effectiveSize.damp` (higher = less damping). Lastly, if after an MCMC run, the number of samples equals or exceeds `2*MCMC.samplesize`, the chain will be thinned by 2 until it falls below that, while doubling `MCMC.interval`. `MCMC.effectiveSize.order.max` can be used to set the order of the AR model used to estimate the effective sample size and the variance for the Geweke diagnostic.

MCMC.return.stats

Logical: If TRUE, return the matrix of MCMC-sampled network statistics. This matrix should have `MCMC.samplesize` rows. This matrix can be used directly by the coda package to assess MCMC convergence.

MCMC.runtime.traceplot

Logical: If TRUE, plot traceplots of the MCMC sample after every MCMC MLE iteration.

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.addto.se   Whether to add the standard errors induced by the MCMC algorithm to the estimates' standard errors.

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.

SAN.maxit   When `target.stats` argument is passed to ergm(), the maximum number of attempts to use [san](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   Control arguments to [san](san). See control.san for details.

MCMLE.termination

The criterion used for terminating MCMLE estimation:

- `"Hummel"` Terminate when the Hummel step length is 1 for two consecutive iterations. For the last iteration, the sample size is boosted by a factor of `MCMLE.last.boost`. See Hummel et. al. (2012).

Note that this criterion is incompatible with `MCMLE.steplength` $\neq 1$ or `MCMLE.steplength.margin` $=$ NULL.

- `"Hotelling"` After every MCMC sample, an autocorrelation-adjusted Hotelling's T^2 test for equality of MCMC-simulated network statistics to observed is conducted, and if its P-value exceeds `MCMLE.conv.min.pval`, the estimation is considered to have converged and finishes. This was the default option in ergm version 3.1.

- "precision" Terminate when the estimated loss in estimating precision due to using MCMC standard errors is below the precision bound specified by `MCMLE.MCMC.precision`, and the Hummel step length is 1 for two consecutive iterations. See `MCMLE.MCMC.precision` for details. This feature is in experimental status until we verify the coverage of the standard errors.

Note that this criterion is incompatible with `MCMLE.steplength` $\neq 1$ or `MCMLE.steplength.margin` $=$ NULL.

- "confidence": Performs an equivalence test to prove with level of confidence `MCMLE.confidence` that the true value of the deviation of the simulated mean value parameter from the observed is within an ellipsoid defined by the inverse-variance-covariance of the sufficient statistics multiplied by a scaling factor `control$MCMLE.MCMC.precision` (which has a different default).
- "none" Stop after `MCMLE.maxit` iterations.

MCMLE.maxit   Maximum number of times the parameter for the MCMC should be updated by maximizing the MCMC likelihood. At each step the parameter is changed to the values that maximizes the MCMC likelihood based on the current sample.

MCMLE.conv.min.pval

     The P-value used in the Hotelling test for early termination.

MCMLE.confidence

     The confidence level for declaring convergence for "confidence" methods.

MCMLE.confidence.boost

     The maximum increase factor in sample size (or target effective size, if enabled) when the "confidence" termination criterion is either not approaching the tolerance region or is unable to prove convergence.

MCMLE.confidence.boost.threshold, MCMLE.confidence.boost.lag

     Sample size or target effective size will be increaed if the distance from the tolerance region fails to decrease more than MCMLE.confidence.boost.threshold in this many successive iterations.

MCMLE.NR.maxit, MCMLE.NR.reltol

     The method, maximum number of iterations and relative tolerance to use within the `optim` routine in the MLE optimization. Note that by default, ergm uses `trust`, and falls back to `optim` only when `trust` fails.

obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMLE.effectiveSize, obs.MCMC.samplesiz

     Corresponding MCMC parameters and settings used for the constrained sample when unobserved data are present in the estimation routine. By default, they are controlled by the `*.mul` parameters, as fractions of the corresponding settings for the unconstrained (standard) MCMC.

     These can, in turn, be controlled by obs.MCMC.mul, which can be used to set the overal multiplier for the number of MCMC steps in the constrained sample; one half of its effect applies to the burn-in and interval and the other half to the total sample size. For example, for obs.MCMC.mul=1/4 (the default), obs.MCMC.samplesize is set to $\sqrt{1/4} = 1/2$ that of obs.MCMC.samplesize, and obs.MCMC.burnin and obs.MCMC.interval are set to $\sqrt{1/4} = 1/2$ of their respective unconstrained sampling parameters. When MCMC.effectiveSize or MCMLE.effectiveSize are given, their corresponding obs parameters are set to them multiplied by obs.MCMC.mul.

obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density

> Controls for imputation of missing dyads for initializing MCMC sampling. If
> numeric, obs.MCMC.impute.min_informative specifies the minimum number
> dyads that need to be non-missing before sample network density is used as the
> imputation density. It can also be specified as a function that returns this value.
> obs.MCMC.impute.default_density similarly controls the imputation density
> when number of non-missing dyads is too low.

MCMLE.min.depfac, MCMLE.sampsize.boost.pow

> When using adaptive MCMC effective size, and methods that increase the MCMC
> sample size, use MCMLE.sampsize.boost.pow as the power of the boost amount
> (relative to the boost of the target effective size), but ensure that sample size is
> no less than MCMLE.min.depfac times the target effective size.

MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac

> MCMLE.MCMC.precision is a vector of upper bounds on the standard errors in-
> duced by the MCMC algorithm, expressed as a percentage of the total standard
> error. The MCMLE algorithm will terminate when the MCMC standard errors
> are below the precision bound, and the Hummel step length is 1 for two consec-
> utive iterations. This is an experimental feature.
>
> If effective sample size is used (see MCMC.effectiveSize), then ergm may in-
> crease the target ESS to reduce the MCMC standard error.

MCMLE.metric   Method to calculate the loglikelihood approximation. See Hummel et al (2010)
                    for an explanation of "lognormal" and "naive".

MCMLE.method   Deprecated. By default, ergm uses trust, and falls back to optim with Nelder-
                    Mead method when trust fails.

MCMLE.dampening

> (logical) Should likelihood dampening be used?

MCMLE.dampening.min.ess

> The effective sample size below which dampening is used.

MCMLE.dampening.level

> The proportional distance from boundary of the convex hull move.

MCMLE.steplength.margin

> The extra margin required for a Hummel step to count as being inside the convex
> hull of the sample. Set this to 0 if the step length gets stuck at the same value
> over several iteraions. Set it to NULL to use fixed step length. Note that this
> parameter is required to be non-NULL for MCMLE termination using Hummel
> or precision criteria.

MCMLE.steplength

> Multiplier for step length (on the mean-value parameter scale), which may (for
> values less than one) make fitting more stable at the cost of computational effi-
> ciency.
>
> If MCMLE.steplength.margin is not NULL, the step length will be set using the
> algorithm of Hummel et al. (2010). In that case, it will serve as the maximum
> step length considered. However, setting it to anything other than 1 will preclude
> using Hummel or precision as termination criteria.

MCMLE.steplength.parallel

> Whether parallel multisection search (as opposed to a bisection search) for the
> Hummel step length should be used if running in multiple threads. Possible

values (partially matched) are "never", and (default) "observational" (i.e., when missing data MLE is used).

MCMLE.sequential

Logical: If TRUE, the next iteration of the fit uses the last network sampled as the starting network. If FALSE, always use the initially passed network. The results should be similar (stochastically), but the TRUE option may help if the target.stats in the [ergm()](#) function are far from the initial network.

MCMLE.density.guard.min, MCMLE.density.guard

A simple heuristic to stop optimization if it finds itself in an overly dense region, which usually indicates ERGM degeneracy: if the sampler encounters a network configuration that has more than MCMLE.density.guard.min edges and whose number of edges is exceeds the observed network by more than MCMLE.density.guard, the optimization process will be stopped with an error.

MCMLE.effectiveSize, MCMLE.effectiveSize.interval_drop, MCMLE.burnin, MCMLE.interval, MCMLE.samplesiz

Sets the corresponding MCMC.* parameters when main.method="MCMLE" (the default). Used because defaults may be different for different methods. MCMLE.samplesize.per_theta controls the MCMC sample size (not target effective size) as a function of the number of (curved) parameters in the model, and MCMLE.samplesize.min sets the minimum sample size regardless of their number.

MCMLE.steplength.solver

The linear program solver to use for MCMLE step length calculation. Can be either "glpk" to use **Rglpk** or "lpsolve" to use **lpSolveAPI**. **Rglpk** can be orders of magnitude faster, particularly for models with many parameters and with large sample sizes, so it is used where available; but it requires an external library to install under some operating systems, so fallback to **lpSolveAPI** provided.

MCMLE.last.boost

For the Hummel termination criterion, increase the MCMC sample size of the last iteration by this factor.

MCMLE.steplength.esteq

For curved ERGMs, should the estimating function values be used to compute the Hummel step length? This allows the Hummel stepping algorithm converge when some sufficient statistics are at 0.

MCMLE.steplength.miss.sample

In fitting the missing data MLE, the rules for step length become more complicated. In short, it is necessary for *all* points in the constrained sample to be in the convex hull of the unconstrained (though they may be on the border); and it is necessary for their centroid to be in its interior. This requires checking a large number of points against whether they are in the convex hull, so to speed up the procedure, a sample is taken of the points most likely to be outside it. This parameter specifies the sample size or a function of the unconstrained sample matrix to determine the sample size.

MCMLE.steplength.min

Stops MCMLE estimation when the step length gets stuck below this minimum value.

MCMLE.save_intermediates

Every iteration, after MCMC sampling, save the MCMC sample and some miscellaneous information to a file with this name. This is mainly useful for diag-

nostics and debugging. The name is passed through [sprintf()](#) with iteration number as the second argument. (For example, MCMLE.save_intermediates="step_%03d.RData" will save to step_001.RData, step_002.RData, etc.)

SA.phase1_n   Number of MCMC samples to draw in Phase 1 of the stochastic approximation algorithm. Defaults to 7 plus 3 times the number of terms in the model. See Snijders (2002) for details.

SA.initial_gain

Initial gain to Phase 2 of the stochastic approximation algorithm. See Snijders (2002) for details.

SA.nsubphases   Number of sub-phases in Phase 2 of the stochastic approximation algorithm. Defaults to MCMLE.maxit. See Snijders (2002) for details.

SA.niterations   Number of MCMC samples to draw in Phase 2 of the stochastic approximation algorithm. Defaults to 7 plus the number of terms in the model. See Snijders (2002) for details.

SA.phase3_n   Sample size for the MCMC sample in Phase 3 of the stochastic approximation algorithm. See Snijders (2002) for details.

SA.burnin, SA.interval, SA.samplesize

Sets the corresponding MCMC.* parameters when main.method="Stochastic-Approximation".

RM.phase1n_base, RM.phase2n_base, RM.phase2sub, RM.init_gain, RM.phase3n

The Robbins-Monro control parameters are not yet documented.

RM.burnin, RM.interval, RM.samplesize

Sets the corresponding MCMC.* parameters when main.method="Robbins-Monro".

Step.maxit   Maximum number of iterations (steps) allowed by the "Stepping" method.

Step.gridsize   Integer N such that the "Stepping" style of optimization chooses a step length equal to the largest possible multiple of $1/N$. See Hummel et al. (2012) for details.

Step.burnin, Step.interval, Step.samplesize

Sets the corresponding MCMC.* parameters when main.method="Stepping".

CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.r

Miscellaneous tuning parameters of the CD sampler and optimizer. These have the same meaning as their MCMLE.* and MCMC.* counterparts.

Note that only the Hotelling's stopping criterion is implemented for CD.

CD.nsteps, CD.multiplicity

Main settings for contrastive divergence to obtain initial values for the estimation: respectively, the number of Metropolis–Hastings steps to take before reverting to the starting value and the number of tentative proposals per step. Computational experiments indicate that increasing CD.multiplicity improves the estimate faster than increasing CD.nsteps — up to a point — but it also samples from the wrong distribution, in the sense that while as CD.nsteps$\to \infty$, the CD estimate approaches the MLE, this is not the case for CD.multiplicity.

In practice, MPLE, when available, usually outperforms CD for even a very high CD.nsteps (which is, in turn, not very stable), so CD is useful primarily when MPLE is not available. This feature is to be considered experimental and in flux.

The default values have been set experimentally, providing a reasonably stable, if not great, starting values.

CD.nsteps.obs, CD.multiplicity.obs

When there are missing dyads, `CD.nsteps` and `CD.multiplicity` must be set to a relatively high value, as the network passed is not necessarily a good start for CD. Therefore, these settings are in effect if there are missing dyads in the observed network, using a higher default number of steps.

loglik   See `control.ergm.bridge`

term.options   A list of additional arguments to be passed to term initializers. See `? term.options`.

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 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.

...   A dummy argument to catch deprecated or mistyped control parameters.

Details

This function is only used within a call to the `ergm()` function. See the usage section in `ergm()` for details.

Value

A list with arguments as components.

References

- Snijders, T.A.B. (2002), Markov Chain Monte Carlo Estimation of Exponential Random Graph Models. Journal of Social Structure. Available from `https://www.cmu.edu/joss/content/articles/volume3/Snijders.pdf`.
- 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. (2012), Improving Simulation-Based Algorithms for Fitting ERGMs, Journal of Computational and Graphical Statistics, 21: 920-939.
- Kristoffer Sahlin. Estimating convergence of Markov chain Monte Carlo simulations. Master's Thesis. Stockholm University, 2011. `https://www2.math.su.se/matstat/reports/master/2011/rep2/report.pdf`

See Also

`ergm()`. The `control.simulate` function performs a similar function for `simulate.ergm`; `control.gof` performs a similar function for `gof`.

---

control.ergm.bridge   *Auxiliaries for Controlling* ergm.bridge.llr() *and* logLik.ergm()

Description

Auxiliary functions as user interfaces for fine-tuning the ergm.bridge.llr() algorithm, which approximates log likelihood ratios using bridge sampling.

By default, the bridge sampler inherits its control parameters from the ergm() fit; control.logLik.ergm() allows the user to selectively override them.

Usage

```
control.ergm.bridge(
  bridge.nsteps = 16,
  bridge.target.se = NULL,
  bridge.bidirectional = TRUE,
  MCMC.burnin = MCMC.interval * 128,
 MCMC.burnin.between = max(ceiling(MCMC.burnin/sqrt(bridge.nsteps)), MCMC.interval *
    16),
  MCMC.interval = 128,
  MCMC.samplesize = 16384,
  obs.MCMC.burnin = obs.MCMC.interval * 128,
  obs.MCMC.burnin.between = max(ceiling(obs.MCMC.burnin/sqrt(bridge.nsteps)),
    obs.MCMC.interval * 16),
  obs.MCMC.interval = MCMC.interval,
  obs.MCMC.samplesize = MCMC.samplesize,
  MCMC.prop = trim_env(~sparse),
  MCMC.prop.weights = "default",
  MCMC.prop.args = list(),
  obs.MCMC.prop = MCMC.prop,
  obs.MCMC.prop.weights = MCMC.prop.weights,
  obs.MCMC.prop.args = MCMC.prop.args,
  MCMC.maxedges = Inf,
  MCMC.packagenames = c(),
  term.options = list(),
  seed = NULL,
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE,
  ...
)

control.logLik.ergm(
  bridge.nsteps = 16,
  bridge.target.se = NULL,
```

```
      bridge.bidirectional = TRUE,
      MCMC.burnin = NULL,
      MCMC.interval = NULL,
      MCMC.samplesize = NULL,
      obs.MCMC.samplesize = MCMC.samplesize,
      obs.MCMC.interval = MCMC.interval,
      obs.MCMC.burnin = MCMC.burnin,
      MCMC.prop = NULL,
      MCMC.prop.weights = NULL,
      MCMC.prop.args = NULL,
      obs.MCMC.prop = MCMC.prop,
      obs.MCMC.prop.weights = MCMC.prop.weights,
      obs.MCMC.prop.args = MCMC.prop.args,
      MCMC.maxedges = NULL,
      MCMC.packagenames = NULL,
      term.options = NULL,
      seed = NULL,
      parallel = NULL,
      parallel.type = NULL,
      parallel.version.check = TRUE,
      parallel.inherit.MT = FALSE,
      ...
  )
```

Arguments

bridge.nsteps   Number of geometric bridges to use.

bridge.target.se

If not NULL, if the estimated MCMC standard error of the likelihood estimate exceeds this, repeat the bridge sampling, accumulating samples.

bridge.bidirectional

Whether the bridge sampler first bridges from from to to, then from to to from (skipping the first burn-in), etc. if multiple attempts are required.

MCMC.burnin   Number of proposals before any MCMC sampling is done. It typically is set to a fairly large number.

MCMC.burnin.between

Number of proposals between the bridges; typically, less and less is needed as the number of steps decreases.

MCMC.interval   Number of proposals between sampled statistics.

MCMC.samplesize

Number of network statistics, randomly drawn from a given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm.

obs.MCMC.burnin, obs.MCMC.burnin.between, obs.MCMC.interval, obs.MCMC.samplesize

The obs versions of these arguments are for the unobserved data simulation algorithm.

MCMC.prop   Specifies the proposal (directly) and/or a series of "hints" about the structure of the model being sampled. The specification is in the form of a one-sided

formula with hints separated by + operations. If the LHS exists and is a string, the proposal to be used is selected directly.

A common and default "hint" is ~sparse, indicating that the network is sparse and that the sample should put roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling.

MCMC.prop.weights

Specifies the proposal distribution used in the MCMC Metropolis-Hastings algorithm. Possible choices depending on selected reference and constraints arguments of the [ergm()](#) function, but often include "TNT" and "random", and the "default" is to use the one with the highest priority available.

MCMC.prop.args   An alternative, direct way of specifying additional arguments to proposal.

obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args

The obs versions of these arguments are for the unobserved data simulation algorithm.

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.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. See [? term.options](#).

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 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.

...   A dummy argument to catch deprecated or mistyped control parameters.

Details

control.ergm.bridge() is only used within a call to the [ergm.bridge.llr()](#), [ergm.bridge.dindstart.llk()](#), or [ergm.bridge.0.llk()](#) functions.

control.logLik.ergm() is only used within a call to the [logLik.ergm()](#).

Value

A list with arguments as components.

See Also

[ergm.bridge.llr()](#)

[logLik.ergm](#)

---

control.ergm.godfather

Control parameters for [ergm.godfather()](#).

Description

Returns a list of its arguments.

Usage

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

Arguments

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

---

control.gof   *Auxiliary for Controlling ERGM Goodness-of-Fit Evaluation*

Description

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

The control.gof.ergm version is intended to be used with [gof.ergm()](#) specifically and will "in-herit" as many control parameters from [ergm](#) fit as possible().

Usage

```
control.gof.formula(
  nsim = 100,
  MCMC.burnin = 10000,
  MCMC.interval = 1000,
  MCMC.batch = 0,
  MCMC.prop = trim_env(~sparse),
  MCMC.prop.weights = "default",
  MCMC.prop.args = list(),
  MCMC.maxedges = Inf,
  MCMC.packagenames = c(),
  MCMC.runtime.traceplot = FALSE,
  network.output = "network",
  seed = NULL,
```

```
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)

control.gof.ergm(
  nsim = 100,
  MCMC.burnin = NULL,
  MCMC.interval = NULL,
  MCMC.batch = NULL,
  MCMC.prop = NULL,
  MCMC.prop.weights = NULL,
  MCMC.prop.args = NULL,
  MCMC.maxedges = NULL,
  MCMC.packagenames = NULL,
  MCMC.runtime.traceplot = FALSE,
  network.output = "network",
  seed = NULL,
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)
```

Arguments

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

MCMC.burnin   Number of proposals before any MCMC sampling is done. It typically is set to
                        a fairly large number.

MCMC.interval   Number of proposals between sampled statistics.

MCMC.batch   if not 0 or NULL, sample about this many networks per call to the lower-level
                        code; this can be useful if output= is a function, where it can be used to limit
                        the number of networks held in memory at any given time.

MCMC.prop   Specifies the proposal (directly) and/or a series of "hints" about the structure
                        of the model being sampled. The specification is in the form of a one-sided
                        formula with hints separated by + operations. If the LHS exists and is a string,
                        the proposal to be used is selected directly.

                        A common and default "hint" is ~sparse, indicating that the network is sparse
                        and that the sample should put roughly equal weight on selecting a dyad with or
                        without a tie as a candidate for toggling.

MCMC.prop.weights

                        Specifies the proposal distribution used in the MCMC Metropolis-Hastings al-
                        gorithm. Possible choices depending on selected reference and constraints
                        arguments of the ergm() function, but often include "TNT" and "random", and
                        the "default" is to use the one with the highest priority available.
```

| | |
|---|---|
| MCMC.prop.args | An alternative, direct way of specifying additional arguments to 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.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. |
| MCMC.runtime.traceplot | |
| | Logical: If TRUE, plot traceplots of the MCMC sample. |
| network.output | R class with which to output networks. The options are "network" (default) and "edgelist.compressed" (which saves space but only supports networks without vertex attributes) |
| 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 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 gof function. See the usage section in gof for details.

## Value

A list with arguments as components.

## See Also

gof. The control.simulate function performs a similar function for simulate.ergm; control.ergm performs a similar function for ergm.

---

| control.san | *Auxiliary for Controlling SAN* |
|---|---|

---

## Description

Auxiliary function as user interface for fine-tuning simulated annealing algorithm.

**Usage**

```
control.san(
  SAN.maxit = 4,
  SAN.tau = 1,
  SAN.invcov = NULL,
  SAN.invcov.diag = FALSE,
  SAN.nsteps.alloc = function(nsim) 2^seq_len(nsim),
  SAN.nsteps = 2^19,
  SAN.samplesize = 2^12,
  SAN.prop = trim_env(~sparse),
  SAN.prop.weights = "default",
  SAN.prop.args = list(),
  SAN.packagenames = c(),
  SAN.ignore.finite.offsets = TRUE,
  term.options = list(),
  seed = NULL,
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)
```

Arguments

SAN.maxit   Number of temperature levels to use.

SAN.tau   Tuning parameter, specifying the temperature of the process during the *penultimate* iteration. (During the last iteration, the temperature is set to 0, resulting in a greedy search, and during the previous iterations, the temperature is set to SAN.tau*(iterations left after this one).

SAN.invcov   Initial inverse covariance matrix used to calculate Mahalanobis distance in determining how far a proposed MCMC move is from the target.stats vector. If NULL, initially set to the identity matrix. In either case, during subsequent runs, it is estimated empirically.

SAN.invcov.diag
                   Whether to only use the diagonal of the covariance matrix. It seems to work better in practice.

SAN.nsteps.alloc
                   Either a numeric vector or a function of the number of runs giving a sequence of relative lengths of simulated annealing runs.

SAN.nsteps   Number of MCMC proposals for all the annealing runs combined.

SAN.samplesize   Number of realisations' statistics to obtain for tuning purposes.

SAN.prop   Specifies the proposal (directly) and/or a series of "hints" about the structure of the model being sampled. The specification is in the form of a one-sided formula with hints separated by + operations. If the LHS exists and is a string, the proposal to be used is selected directly.

A common and default "hint" is ~sparse, indicating that the network is sparse and that the sample should put roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling.

SAN.prop.weights
Specifies the proposal distribution used in the SAN Metropolis-Hastings algorithm. Possible choices depending on selected reference and constraints arguments of the [ergm()](#) function, but often include "TNT" and "random", and the "default" is to use the one with the highest priority available.

SAN.prop.args   An alternative, direct way of specifying additional arguments to proposal.

SAN.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.

SAN.ignore.finite.offsets
Whether SAN should ignore (treat as 0) finite offsets.

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

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 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 [san](#) function. See the usage section in [san](#) for details.

Value

A list with arguments as components.

See Also

[san](#)

control.simulate.ergm *Auxiliary for Controlling ERGM Simulation*

Description

Auxiliary function as user interface for fine-tuning ERGM simulation. `control.simulate`, `control.simulate.formula`, and `control.simulate.formula.ergm` are all aliases for the same function.

While the others supply a full set of simulation settings, `control.simulate.ergm` when passed as a control parameter to `simulate.ergm()` allows some settings to be inherited from the ERGM stimation while overriding others.

Usage

```
control.simulate.formula.ergm(
  MCMC.burnin = MCMC.interval * 16,
  MCMC.interval = 1024,
  MCMC.prop = trim_env(~sparse),
  MCMC.prop.weights = "default",
  MCMC.prop.args = list(),
  MCMC.batch = NULL,
  MCMC.effectiveSize = NULL,
  MCMC.effectiveSize.damp = 10,
  MCMC.effectiveSize.maxruns = 1000,
  MCMC.effectiveSize.burnin.pval = 0.2,
  MCMC.effectiveSize.burnin.min = 0.05,
  MCMC.effectiveSize.burnin.max = 0.5,
  MCMC.effectiveSize.burnin.nmin = 16,
  MCMC.effectiveSize.burnin.nmax = 128,
  MCMC.effectiveSize.burnin.PC = FALSE,
  MCMC.effectiveSize.burnin.scl = 1024,
  MCMC.effectiveSize.order.max = NULL,
  MCMC.maxedges = Inf,
  MCMC.packagenames = c(),
  MCMC.runtime.traceplot = FALSE,
  network.output = "network",
  term.options = NULL,
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE,
  ...
)

control.simulate(
  MCMC.burnin = MCMC.interval * 16,
  MCMC.interval = 1024,
```

```
      MCMC.prop = trim_env(~sparse),
      MCMC.prop.weights = "default",
      MCMC.prop.args = list(),
      MCMC.batch = NULL,
      MCMC.effectiveSize = NULL,
      MCMC.effectiveSize.damp = 10,
      MCMC.effectiveSize.maxruns = 1000,
      MCMC.effectiveSize.burnin.pval = 0.2,
      MCMC.effectiveSize.burnin.min = 0.05,
      MCMC.effectiveSize.burnin.max = 0.5,
      MCMC.effectiveSize.burnin.nmin = 16,
      MCMC.effectiveSize.burnin.nmax = 128,
      MCMC.effectiveSize.burnin.PC = FALSE,
      MCMC.effectiveSize.burnin.scl = 1024,
      MCMC.effectiveSize.order.max = NULL,
      MCMC.maxedges = Inf,
      MCMC.packagenames = c(),
      MCMC.runtime.traceplot = FALSE,
      network.output = "network",
      term.options = NULL,
      parallel = 0,
      parallel.type = NULL,
      parallel.version.check = TRUE,
      parallel.inherit.MT = FALSE,
      ...
    )

    control.simulate.formula(
      MCMC.burnin = MCMC.interval * 16,
      MCMC.interval = 1024,
      MCMC.prop = trim_env(~sparse),
      MCMC.prop.weights = "default",
      MCMC.prop.args = list(),
      MCMC.batch = NULL,
      MCMC.effectiveSize = NULL,
      MCMC.effectiveSize.damp = 10,
      MCMC.effectiveSize.maxruns = 1000,
      MCMC.effectiveSize.burnin.pval = 0.2,
      MCMC.effectiveSize.burnin.min = 0.05,
      MCMC.effectiveSize.burnin.max = 0.5,
      MCMC.effectiveSize.burnin.nmin = 16,
      MCMC.effectiveSize.burnin.nmax = 128,
      MCMC.effectiveSize.burnin.PC = FALSE,
      MCMC.effectiveSize.burnin.scl = 1024,
      MCMC.effectiveSize.order.max = NULL,
      MCMC.maxedges = Inf,
      MCMC.packagenames = c(),
      MCMC.runtime.traceplot = FALSE,
```

```
    network.output = "network",
    term.options = NULL,
    parallel = 0,
    parallel.type = NULL,
    parallel.version.check = TRUE,
    parallel.inherit.MT = FALSE,
    ...
  )

  control.simulate.ergm(
    MCMC.burnin = NULL,
    MCMC.interval = NULL,
    MCMC.scale = 1,
    MCMC.prop = NULL,
    MCMC.prop.weights = NULL,
    MCMC.prop.args = NULL,
    MCMC.batch = NULL,
    MCMC.effectiveSize = NULL,
    MCMC.effectiveSize.damp = 10,
    MCMC.effectiveSize.maxruns = 1000,
    MCMC.effectiveSize.burnin.pval = 0.2,
    MCMC.effectiveSize.burnin.min = 0.05,
    MCMC.effectiveSize.burnin.max = 0.5,
    MCMC.effectiveSize.burnin.nmin = 16,
    MCMC.effectiveSize.burnin.nmax = 128,
    MCMC.effectiveSize.burnin.PC = FALSE,
    MCMC.effectiveSize.burnin.scl = 1024,
    MCMC.effectiveSize.order.max = NULL,
    MCMC.maxedges = Inf,
    MCMC.packagenames = NULL,
    MCMC.runtime.traceplot = FALSE,
    network.output = "network",
    term.options = NULL,
    parallel = 0,
    parallel.type = NULL,
    parallel.version.check = TRUE,
    parallel.inherit.MT = FALSE,
    ...
  )
```

Arguments

`MCMC.burnin`	Number of proposals before any MCMC sampling is done. It typically is set to a fairly large number.
`MCMC.interval`	Number of proposals between sampled statistics.
`MCMC.prop`	Specifies the proposal (directly) and/or a series of "hints" about the structure of the model being sampled. The specification is in the form of a one-sided formula with hints separated by + operations. If the LHS exists and is a string,

the proposal to be used is selected directly.

A common and default "hint" is ~sparse, indicating that the network is sparse and that the sample should put roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling.

MCMC.prop.weights

Specifies the proposal distribution used in the MCMC Metropolis-Hastings algorithm. Possible choices depending on selected reference and constraints arguments of the [ergm()](#) function, but often include "TNT" and "random", and the "default" is to use the one with the highest priority available.

MCMC.prop.args An alternative, direct way of specifying additional arguments to proposal.

MCMC.batch if not 0 or NULL, sample about this many networks per call to the lower-level code; this can be useful if output= is a function, where it can be used to limit the number of networks held in memory at any given time.

MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.pv

Set MCMC.effectiveSize to a non-NULL value to adaptively determine the burn-in and the MCMC length needed to get the specified effective size; 50 is a reasonable value. In the adaptive MCMC mode, MCMC is run forward repeatedly (MCMC.samplesize*MCMC.interval steps, up to MCMC.effectiveSize.maxruns times) until the target effective sample size is reached or exceeded.

After each run, the returned statistics are mapped to the estimating function scale, then an exponential decay model is fit to the scaled statistics to find that burn-in which would reduce the difference between the initial values of statistics and their equilibrium values by a factor of MCMC.effectiveSize.burnin.scl of what it initially was, bounded by MCMC.effectiveSize.min and MCMC.effectiveSize.max as proportions of sample size. If the best-fitting decay exceeds MCMC.effectiveSize.max, the exponential model is considered to be unsuitable and MCMC.effectiveSize.min is used.

A Geweke diagnostic is then run, after thinning the sample to MCMC.effectiveSize.burnin.nmax. If this Geweke diagnostic produces a p-value higher than MCMC.effectiveSize.burnin.pval, it is accepted.

If MCMC.effectiveSize.burnin.PC>0, instead of using the full sample for burn-in estimation, at most this many principal components are used instead.

The effective size of the post-burn-in sample is computed via Vats, Flegal, and Jones (2015), and compared to the target effective size. If it is not matched, the MCMC run is resumed, with the additional draws needed linearly extrapolated but weighted in favor of the baseline MCMC.samplesize by the weighting factor MCMC.effectiveSize.damp (higher = less damping). Lastly, if after an MCMC run, the number of samples equals or exceeds 2*MCMC.samplesize, the chain will be thinned by 2 until it falls below that, while doubling MCMC.interval. MCMC.effectiveSize.order.max can be used to set the order of the AR model used to estimate the effective sample size and the variance for the Geweke diagnostic.

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.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.

MCMC.runtime.traceplot

Logical: If TRUE, plot traceplots of the MCMC sample.

network.output   R class with which to output networks. The options are "network" (default) and "edgelist.compressed" (which saves space but only supports networks without vertex attributes)

term.options   A list of additional arguments to be passed to term initializers. See `? term.options`.

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

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

parallel.version.check

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

parallel.inherit.MT

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

...   A dummy argument to catch deprecated or mistyped control parameters.

MCMC.scale   For `control.simulate.ergm()` inheriting `MCMC.burnin` and `MCMC.interval` from the `ergm` fit, the multiplier for the inherited values. This can be useful because MCMC parameters used in the fit are tuned to generate a specific effective sample size for the sufficient statistic in a large MCMC sample, so the inherited values might not generate independent realisations.

Details

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

Value

A list with arguments as components.

See Also

simulate.ergm, simulate.formula. control.ergm performs a similar function for ergm; control.gof performs a similar function for gof.

---

ctriple-ergmTerm   *Cyclic triples*

Description

By default, this term adds one statistic to the model, equal to the number of cyclic triples in the network, defined as a set of edges of the form $\{(i{\rightarrow}j),(j{\rightarrow}k),(k{\rightarrow}i)\}$.

Usage

```
# binary: ctriple(attr=NULL, diff=FALSE, levels=NULL)

# binary: ctriad
```

Arguments

attr, diff   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                 for details.) If attr is specified and diff is FALSE , then the statistic is the num-
                 ber of cyclic triples where all three nodes have the same value of the attribute. If
                 attr is specified and diff is TRUE , then one statistic is added to the model for
                 each value of attr, equal to the number of cyclic triples where all three nodes
                 have that value of the attribute.

levels   specifies the value of attr to consider if attr is passed and diff=TRUE. (See
                 Specifying Vertex attributes and Levels ([?nodal_attributes](#)) for details.)

Note

This term can only be used with directed networks.

for all directed networks, triangle is equal to ttriple+ctriple , so at most two of these three terms can be in a model.

See Also

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

Curve-ergmTerm   *Impose a curved structure on term parameters*

Description

Arguments may have the same forms as in the API, but for convenience, alternative forms are accepted.

If the model in formula is curved, then the outputs of this operator term's map argument will be used as inputs to the curved terms of the formula model.

Curve is an obsolete alias and may be deprecated and removed in a future release.

Usage

```
# binary: Curve(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf, cov=NULL)

# binary: Parametrise(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf,
#            cov=NULL)

# binary: Parametrize(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf,
#            cov=NULL)

# valued: Curve(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf, cov=NULL)

# valued: Parametrise(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf,
#            cov=NULL)

# valued: Parametrize(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf,
#            cov=NULL)
```

Arguments

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

params   a named list whos names are the curved parameter names, may also be a char-
                    acter vector with names.

map   the mapping from curved to canonical. May have the following forms:

- a function(x, n, ...) treated as in the API: called with x set to the
  curved parameter vector, n to the length of output expected, and cov , if
  present, passed in ... . The function must return a numeric vector of length
  n .
- a numeric vector to fix the output coefficients, like in an offset.
- a character string to select (partially-matched) one of predefined forms.
  Currently, the defined forms include:
    - "rep" recycle the input vector to the length of the output vector as a
      rep function would.

gradient   its gradient function. It is optional if map is constant or one of the predefined
                    forms; otherwise it must have one of the following forms:

- a function(x, n, ...) treated as in the API: called with x set to the
  curved parameter vector, n to the length of output expected, and cov , if
  present, passed in ... . The function must return a numeric matrix with
  length(params) rows and n columns.
- a numeric matrix to fix the gradient; this is useful when map is linear.
- a character string to select (partially-matched) one of predefined forms.
  Currently, the defined forms include:
    - "linear" calculate the (constant) gradient matrix using finite differ-
      ences. Note that this will be done only once at the initialization stage,
      so use only if you are certain map is, in fact, linear.

minpar, maxpar   the minimum and maximum allowed curved parameter values. The parameters
                    will be recycled to the appropriate length.

cov optional

See Also

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

---

cycle-ergmTerm	*k-Cycle Census*

Description

This term adds one network statistic to the model for each value of k , corresponding to the number of k -cycles (or, alternately, semicycles) in the graph.

This term can be used with either directed or undirected networks.

Usage

```
# binary: cycle(k, semi=FALSE)
```

Arguments

k	a vector of integers giving the cycle lengths to count. Directed cycle lengths may range from 2 to N (the network size); undirected cycle lengths and semicycle lengths may range from 3 to N ; length 2 semicycles are not currently supported.
semi	an optional logical indicating whether semicycles (rather than directed cycles) should be counted; this is ignored in the undirected case.
directed	2-cycles are equivalent to mutual dyads.

See Also

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

---

cyclicalties-ergmTerm	*Cyclical ties*

Description

This term adds one statistic, equal to the number of ties $i \rightarrow j$ such that there exists a two-path from j to i . (Related to the ttriple term.)

Usage

```
# binary: cyclicalties(attr=NULL, levels=NULL)
```

```
# valued: cyclicalties(threshold=0)
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) If set, all three nodes involved (i , j , and the node on the two-path) must match on this attribute in order for $i \rightarrow j$ to be counted.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

cyclicalweights-ergmTerm

Cyclical weights

Description

This statistic implements the cyclical weights statistic, like that defined by Krivitsky (2012), Equation 13, but with the focus dyad being $y_{j,i}$ rather than $y_{i,j}$. For each option, the first (and the default) is more stable but also more conservative, while the second is more sensitive but more likely to induce a multimodal distribution of networks.

Usage

```
# valued: cyclicalweights(twopath="min", combine="max", affect="min")
```

Arguments

twopath	the minimum of the constituent dyads ("min") or their geometric mean ("geomean")
combine	the maximum of the 2-path strengths ("max") or their sum ("sum")
affected	the minimum of the focus dyad and the combined strength of the two paths ("min") or their geometric mean ("geomean")

See Also

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

---

ddsp-ergmTerm   *Directed dyadwise shared partners*

Description

This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of dyads in the network with exactly d[i] shared partners.

Usage

```
# binary: ddsp(d, type="OTP")
```

Arguments

d   a vector of distinct integers

type   one of c("OTP", "ITP", "RTP", "OSP", "ISP")

Note

This term can only be used with directed networks.

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

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

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

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

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

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

See Also

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

---

degcor-ergmTerm   *Degree Correlation*

Description

This term adds one network statistic equal to the correlation of the degrees of all pairs of nodes in the network which are tied. Only coded for undirected networks.

Usage

```
# binary: degcor
```

See Also

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

---

degcrossprod-ergmTerm *Degree Cross-Product*

Description

This term adds one network statistic equal to the mean of the cross-products of the degrees of all pairs of nodes in the network which are tied. Only coded for undirected networks.

Usage

```
# binary: degcrossprod
```

See Also

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

---

degrange-ergmTerm   *Degree range*

Description

This term adds one network statistic to the model for each element of from (or to); the i th such statistic equals the number of nodes in the network of degree greater than or equal to from[i] but strictly less than to[i] , i.e. with edges in semiopen interval [from,to) .

Usage

```
# binary: degrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
```

Arguments

from, to	vectors of distinct integers. If one of the vectors have length 1, it is recycled to the length of the other. Otherwise, it must have the same length.
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Details

This term can only be used with undirected networks; for directed networks see idegrange and odegrange . This term can be used with bipartite networks, and will count nodes of both first and second mode in the specified degree range. To count only nodes of the first mode ("actors"), use b1degrange and to count only those fo the second mode ("events"), use b2degrange .

See Also

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

---

degree-ergmTerm *Degree*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of nodes in the network of degree d[i] , i.e. with exactly d[i] edges. This term can only be used with undirected networks; for directed networks see idegree and odegree .

Usage

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

Arguments

d	vector of distinct integers
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

degree1.5-ergmTerm *Degree to the 3/2 power*

Description

This term adds one network statistic to the model equaling the sum over the actors of each actor's degree taken to the 3/2 power (or, equivalently, multiplied by its square root). This term is an undirected analog to the terms of Snijders et al. (2010), equations (11) and (12). This term can only be used with undirected networks.

Usage

```
# binary: degree1.5
```

See Also

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

degreedist	*Computes and Returns the Degree Distribution Information for a Given Network*

Description

The degreedist generic computes and returns the degree distribution (number of vertices in the network with each degree value) for a given network. This help page documents the function. For help about the ERGM sample space constraint with that name, try help("degreedist-constraint").

Usage

```
degreedist(object, ...)

## S3 method for class 'network'
degreedist(object, print = TRUE, ...)
```

Arguments

object   a network object or some other object for which degree distribution is meaningful.

...   Additional arguments to functions.

print   logical, whether to print the degree distribution.

Value

If directed, a matrix of the distributions of in and out degrees; this is row bound and only contains degrees for which one of the in or out distributions has a positive count. If bipartite, a list containing the degree distributions of b1 and b2. Otherwise, a vector of the positive values in the degree distribution

Methods (by class)

- network: Method for network objects.

Examples

```
data(faux.mesa.high)
degreedist(faux.mesa.high)
```

`degreedist-ergmConstraint`

Preserve the degree distribution of the given network

Description

Only networks whose degree distributions are the same as those in the network passed in the model formula have non-zero probability.

Usage

```
# degreedist
```

See Also

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

`degrees-ergmConstraint`

Preserve the degree of each vertex of the given network

Description

Only networks whose vertex degrees are the same as those in the network passed in the model formula have non-zero probability. If the network is directed, both indegree and outdegree are preserved.

Usage

```
# degrees
```

```
# nodedegrees
```

See Also

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

---

density-ergmTerm   *Density*

Description

This term adds one network statistic equal to the density of the network. For undirected networks, density equals kstar(1) or edges divided by $n(n-1)/2$; for directed networks, density equals edges or istar(1) or ostar(1) divided by $n(n-1)$.

Usage

```
# binary: density
```

See Also

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

---

desp-ergmTerm   *Directed edgewise shared partners*

Description

This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of edges in the network with exactly d[i] shared partners.

Usage

```
# binary: desp(d, type="OTP")
```

Arguments

d	a vector of distinct integers
type	one of c("OTP", "ITP", "RTP", "OSP", "ISP").

Note

This term can only be used with directed networks.

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

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

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

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

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

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

See Also

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

---

dgwdsp-ergmTerm   *Geometrically weighted dyadwise shared partner distribution*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter decay parameter.

Usage

```
# binary: dgwdsp(decay, fixed=FALSE, cutoff=30, type="OTP")
```

Arguments

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

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

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

type   one of c("OTP", "ITP", "RTP", "OSP", "ISP")

Note

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

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

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

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

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

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

The GWDSP statistic is equal to the sum of GWNSP plus GWESP.

The decay parameter was called alpha prior to ergm 3.7.

See Also

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

---

dgwesp-ergmTerm   *Geometrically weighted edgewise shared partner distribution*

Description

This term adds a statistic equal to the geometrically weighted edgewise (not dyadwise) shared partner distribution with decay parameter decay parameter.

Usage

```
# binary: dgwesp(decay, fixed=FALSE, cutoff=30, type="OTP")
```

Arguments

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

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

cutoff	This optional argument sets the number of underlying ESP terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)
type	one of c("OTP", "ITP", "RTP", "OSP", "ISP").

Note

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

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

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

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

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

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

The decay parameter was called alpha prior to ergm 3.7.

See Also

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

---

dgwnsp-ergmTerm   *Geometrically weighted non-edgewise shared partner distribution*

Description

This term is just like gwesp and gwdsp except it adds a statistic equal to the geometrically weighted nonedgewise (that is, over dyads that do not have an edge) shared partner distribution with decay parameter decay parameter.

Usage

```
# binary: dgwnsp(decay, fixed=FALSE, cutoff=30, type="OTP")
```

Arguments

decay	nonnegative decay parameter for the shared partner or selected directed analogue count; required if `fixed=TRUE` and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying NSP terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See `?control.ergm`.)
type	one of `c("OTP", "ITP", "RTP", "OSP", "ISP")`

Note

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

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

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

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

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

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

The decay parameter was called `alpha` prior to `ergm` 3.7.

See Also

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

---

diff-ergmTerm   *Difference*

Description

For values of pow other than 0 , this term adds one network statistic to the model, equaling the sum, over directed edges (i, j) , of sign.action(attr[i]-attr[j])^pow if dir is "t-h" and of sign.action(attr[j]-attr[i])^pow if "h-t" . That is, the argument dir determines which vertex's attribute is subtracted from which, with tail being the origin of a directed edge and head being its destination, and bipartite networks' edges being treated as going from the first part (b1) to the second (b2).

If pow==0 , the exponentiation is replaced by the signum function: +1 if the difference is positive, 0 if there is no difference, and -1 if the difference is negative. Note that this function is applied after the sign.action . The comparison is exact, so when using calculated values of attr , ensure that values that you want to be considered equal are, in fact, equal.

Usage

```
# binary: diff(attr, pow=1, dir="t-h", sign.action="identity")

# valued: diff(attr, pow=1, dir="t-h", sign.action="identity", form ="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
pow	exponent for the node difference
dir	determines which vertix's attribute is subtracted from which. Accepts: "t-h" (the default), "tail-head" , "b1-b2", "h-t" , "head-tail" , and "b2-b1" .
sign.action	one of "identity", "abs", "posonly", "negonly". The following sign.actions are possible:

- "identity" (the default) no transformation of the difference regardless of sign
- "abs" absolute value of the difference: equivalent to the absdiff term
- "posonly" positive differences are kept, negative differences are replaced by 0
- "negonly" negative differences are kept, positive differences are replaced by 0

form	character how to aggregate tie values in a valued ERGM

Note

this term may not be meaningful for unipartite undirected networks unless sign.action=="abs" . When used on such a network, it behaves as if all edges were directed, going from the lower-indexed vertex to the higher-indexed vertex.

See Also

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

DiscUnif-ergmProposal *TODO*

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
DiscUnif			0	random	c

DiscUnif-ergmReference

Discrete Uniform reference

Description

Specifies each dyad's baseline distribution to be discrete uniform between a and b (both inclusive): $h(y) = 1$, with the support being a, a+1, ..., b-1, b.

Usage

```
# DiscUnif(a,b)
```

Arguments

a, b   minimum and maximum to the baseline discrete uniform distribution, both inclusive. Both values must be finite.

See Also

ergmReference for index of reference distributions currently visible to the package.

`DiscUnif2-ergmProposal`
                                          TODO

Description

   TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
DiscUnif			-1	random2	c

`DiscUnifNonObserved-ergmProposal`
                                          TODO

Description

   TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
DiscUnif	observed		0	random	c

DistRLE-ergmProposal   *TODO*

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
StdNormal		.dyads	0	random	c
Unif		.dyads	0	random	c
Unif		.dyads	-3	random	c
DiscUnif		.dyads	-3	random	c
StdNormal		.dyads	-3	random	c
Poisson		.dyads	-3	random	c
Binomial		.dyads	-3	random	c

dnsp-ergmTerm   *Directed non-edgewise shared partners*

Description

This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of non-edges in the network with exactly d[i] shared partners.

Usage

```
# binary: dnsp(d, type="OTP")
```

Arguments

d   a vector of distinct integers

type   one of c("OTP", "ITP", "RTP", "OSP", "ISP").

Note

This term can only be used with directed networks.

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

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

See Also

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

---

   dsp-ergmTerm   *Dyadwise shared partners*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of dyads in the network with exactly d[i] shared partners. This term can be used with directed and undirected networks.

Usage

```
# binary: dsp(d)
```

Arguments

d   a vector of distinct integers

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) dyad (i,j) in a directed graph, the number of shared partners counted by dsp is the number of nodes k that have edges i -> k -> j. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see ddsp.

See Also

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

`dyadcov-ergmTerm`   *Dyadic covariate*

Description

This term adds three statistics to the model, each equal to the sum of the covariate values for all dyads occupying one of the three possible non-empty dyad states (mutual, upper-triangular asymmetric, and lower-triangular asymmetric dyads, respectively), with the empty or null state serving as a reference category. If the network is undirected, x is either a matrix of edgewise covariates, or a network; if the latter, optional argument attrname provides the name of the edge attribute to use for edge values. This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The edgecov and dyadcov terms are equivalent for undirected networks.

Usage

```
# binary: dyadcov(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, 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).

See Also

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

`dyadnoise-ergmConstraint`

A soft constraint to adjust the sampled distribution for dyad-level noise with known perturbation probabilities

---

Description

It is assumed that the observed LHS network is a noisy observation of some unobserved true network, with p01 giving the dyadwise probability of erroneously observing a tie where the true network had a non-tie and p10 giving the dyadwise probability of erroneously observing a nontie where the true network had a tie.

Usage

```
# dyadnoise(p01, p10)
```

Arguments

p01, p10   can both be scalars or both be adjacency matrices of the same dimension as that
                of the LHS network giving these probabilities.

Note

See Karwa et al. (2016) for an application.

See Also

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

---

dyadnoise-ergmProposal
TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	dyadnoise		0	random	c

---

dyadnoiseTNT-ergmProposal
TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	dyadnoise sparse		1	TNT	c

Dyads-ergmConstraint   *Constrain fixed or varying dyad-independent terms*

Description

This is an "operator" constraint that takes one or two ergmTerm dyad-independent formulas. For the terms in the `vary=` formula, only those that change at least one of the terms will be allowed to vary, and all others will be fixed. If both formulas are given, the dyads that vary either for one or for the other will be allowed to vary. Note that a formula passed to `Dyads` without an argument name will default to `fix=`.

Usage

```
# Dyads(fix=NULL, vary=NULL)
```

Arguments

fix, vary   formula with only dyad-independent terms

See Also

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

---

ecoli   *Two versions of an E. Coli network dataset*

Description

This network data set comprises two versions of a biological network in which the nodes are operons in *Escherichia Coli* and a directed edge from one node to another indicates that the first encodes the transcription factor that regulates the second.

Usage

```
data(ecoli)
```

Details

The network object `ecoli1` is directed, with 423 nodes and 519 arcs. The object `ecoli2` is an undirected version of the same network, in which all arcs are treated as edges and the five isolated nodes (which exhibit only self-regulation in `ecoli1`) are removed, leaving 418 nodes.

Licenses and Citation

When publishing results obtained using this data set, the original authors (Salgado et al, 2001; Shen-Orr et al, 2002) should be cited, along with this R package.

Source

The data set is based on the RegulonDB network (Salgado et al, 2001) and was modified by Shen-Orr et al (2002).

References

Salgado et al (2001), Regulondb (version 3.2): Transcriptional Regulation and Operon Organization in Escherichia Coli K-12, *Nucleic Acids Research*, 29(1): 72-74.

Shen-Orr et al (2002), Network Motifs in the Transcriptional Regulation Network of Escerichia Coli, *Nature Genetics*, 31(1): 64-68.

%Saul and Filkov (2007)

%Hummel et al (2010)

---

edgecov-ergmTerm   *Edge covariate*

Description

This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The edgecov term applies to both directed and undirected networks. For undirected networks the covariates are also assumed to be undirected. The edgecov and dyadcov terms are equivalent for undirected networks.

Usage

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

# valued: edgecov(x, attrname=NULL, form="sum")
```

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, 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).

form   character how to aggregate tie values in a valued ERGM

See Also

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

---

edges-ergmConstraint *Preserve the edge count of the given network*

Description

Only networks having the same number of edges as the network passed in the model formula have non-zero probability.

Usage

```
# edges
```

See Also

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

---

edges-ergmTerm *Number of edges in the network*

Description

This term adds one network statistic equal to the number of edges (i.e. nonzero values) in the network. For undirected networks, edges is equal to kstar(1); for directed networks, edges is equal to both ostar(1) and istar(1).

Usage

```
# binary: edges

# valued: nonzero

# valued: edges
```

See Also

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

---

egocentric-ergmConstraint

Preserve values of dyads incident on vertices with given attribute

---

Description

Preserve values of dyads incident on vertices with attribute attr being TRUE or if attrname is NULL , the vertex attribute "na" being FALSE.

Usage

```
# egocentric(attr=NULL, direction="both")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
direction	one of "both", "out" and "in", only applies to directed networks. "out" only preserves the out-dyads of those actors and "in" preserves their in-dyads.

See Also

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

---

enformulate.curved-deprecated

Convert a curved ERGM into a form suitable as initial values for the same ergm. Deprecated in 4.0.0.

Description

The generic enformulate.curved converts an [ergm](#) object or formula of a model with curved terms to the variant in which the curved parameters embedded into the formula and are removed from the parameter vector. This is the form that used to be required by [ergm](#) calls.

Usage

```
enformulate.curved(object, ...)

## S3 method for class 'ergm'
enformulate.curved(object, ...)

## S3 method for class 'formula'
enformulate.curved(object, theta, ...)
```

Arguments

object	An [ergm](#) object or an ERGM formula. The curved terms of the given formula (or the formula used in the fit) must have all of their arguments passed by name.
...	Unused at this time.
theta	Curved model parameter configuration.

Details

Because of a current kludge in [ergm](#), output from one run cannot be directly passed as initial values (control.ergm(init=)) for the next run if any of the terms are curved. One workaround is to embed the curved parameters into the formula (while keeping fixed=FALSE) and remove them from control.ergm(init=).

This function automates this process for curved ERGM terms included with the [ergm](#) package. It does not work with curved terms not included in ergm.

Value

A list with the following components:

formula	The formula with curved parameter estimates incorporated.
theta	The coefficient vector with curved parameter estimates removed.

See Also

[ergm](#), [simulate.ergm](#)

---

equalto-ergmTerm	*Number of dyads with values equal to a specific value (within tolerance)*

Description

Adds one statistic equal to the number of dyads whose values are within tolerance of value , i.e., between value-tolerance and value+tolerance , inclusive.

Usage

```
# valued: equalto(value=0, tolerance=0)
```

Arguments

value	numerical threshold
tolerance	numerical threshold

See Also

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

---

ergm   *Exponential-Family Random Graph Models*

Description

[ergm](#) is used to fit exponential-family random graph models (ERGMs), in which the probability of a given network, y, on a set of nodes is $h(y)\exp\{\eta(\theta)\cdot g(y)\}/c(\theta)$, where $h(y)$ is the reference measure (usually $h(y) = 1$), $g(y)$ is a vector of network statistics for y, $\eta(\theta)$ is a natural parameter vector of the same length (with $\eta(\theta) = \theta$ for most terms), and $c(\theta)$ is the normalizing constant for the distribution. [ergm](#) can return a maximum pseudo-likelihood estimate, an approximate maximum likelihood estimate based on a Monte Carlo scheme, or an approximate contrastive divergence estimate based on a similar scheme. (For an overview of the package, see [ergm-package](#).)

Usage

```
ergm(
  formula,
  response = NULL,
  reference = ~Bernoulli,
  constraints = ~.,
  obs.constraints = ~. - observed,
  offset.coef = NULL,
  target.stats = NULL,
  eval.loglik = getOption("ergm.eval.loglik"),
  estimate = c("MLE", "MPLE", "CD"),
  control = control.ergm(),
  verbose = FALSE,
  ...,
  basis = ergm.getnetwork(formula)
)

is.ergm(object)

## S3 method for class 'ergm'
nobs(object, ...)

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

## S3 method for class 'ergm'
vcov(object, sources = c("all", "model", "estimation"), ...)
```

Arguments

formula   An R [formula](#) object, of the form y ~ <model terms>, where y is a [network](#) object or a matrix that can be coerced to a [network](#) object. For the details on the possible <model terms>, see [ergmTerm](#) and Morris, Handcock and Hunter

(2008) for binary ERGM terms and Krivitsky (2012) for valued ERGM terms (terms for weighted edges). To create a network object in R, use the network() function, then add nodal attributes to it using the %v% operator if necessary. Enclosing a model term in offset() fixes its value to one specified in offset.coef. (A second argument—a logical or numeric index vector—can be used to select *which* of the parameters within the term are offsets.)

response Either a character string, a formula, or NULL (the default), to specify the response attributes and whether the ERGM is binary or valued. Interpreted as follows:

NULL Model simple presence or absence, via a binary ERGM.

character string The name of the edge attribute whose value is to be modeled. Type of ERGM will be determined by whether the attribute is logical (TRUE/FALSE) for binary or numeric for valued.

a formula must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is evaluated in the formula's environment with the network's edge attributes accessible as variables. The optional NAME specifies the name of the edge attribute into which the results should be stored, with the default being a concise version of EXPR. Normally, the type of ERGM is determined by whether the result of evaluating EXPR is logical or numeric, but the optional TYPE can be used to override by specifying a scalar of the type involved (e.g., TRUE for binary and 1 for valued).

reference A one-sided formula specifying the reference measure ($h(y)$) to be used. See help for ERGM reference measures implemented in the **ergm** package.

constraints A formula specifying one or more constraints on the support of the distribution of the networks being modeled, using syntax similar to the formula argument, on the right-hand side. Multiple constraints may be given, separated by "+" and "-" operators. (See ERGM constraints for the explanation of their semantics.) 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 prop.args argument to control.ergm) or by giving it on the LHS of the constraints formula, in which case it will override the one chosen automatically.

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.

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.

obs.constraints

A one-sided formula specifying one or more constraints or other modification *in addition* to those specified by constraints that had affected the observation process for the network, using syntax similar to the formula argument. Multiple constraints may be given, separated by "+" operators.

This allows the domain of the integral in the numerator of the partially obseved network face-value likelihoods of Handcock and Gile (2010) and Karwa et al. (2017) to be specified explicitly.

The default is ~observed, to constrain the integral to only integrate over the missing dyads. (It is dropped automatically if the network is completely observed.)

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 obs.prop.args argument to control.ergm.

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

Note that not all possible combinations of constraints and reference measures are supported.

offset.coef	A vector of coefficients for the offset terms.
target.stats	vector of "observed network statistics," if these statistics are for some reason different than the actual statistics of the network on the left-hand side of formula. Equivalently, this vector is the mean-value parameter values for the model. If this is given, the algorithm finds the natural parameter values corresponding to these mean-value parameters. If NULL, the mean-value parameters used are the observed statistics of the network in the formula.
eval.loglik	Logical: For dyad-dependent models, if TRUE, use bridge sampling to evaluate the log-likelihoood associated with the fit. Has no effect for dyad-independent models. Since bridge sampling takes additional time, setting to FALSE may speed performance if likelihood values (and likelihood-based values like AIC and BIC) are not needed. Can be set globally via option(ergm.eval.loglik=...), which is set to TRUE when the package is loaded. (See options?ergm.)
estimate	If "MPLE," then the maximum pseudolikelihood estimator is returned. If "MLE" (the default), then an approximate maximum likelihood estimator is returned. For certain models, the MPLE and MLE are equivalent, in which case this argument is ignored. (To force MCMC-based approximate likelihood calculation even when the MLE and MPLE are the same, see the force.main argument of control.ergm. If "CD" (*EXPERIMENTAL*), the Monte-Carlo contrastive divergence estimate is returned.)
control	A list of control parameters for algorithm tuning, typically constructed with control.ergm(). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility snctrl() (StatNet ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
...	Additional arguments, to be passed to lower-level functions.
basis	a value (usually a network) to override the LHS of the formula.
object	an ergm object.

x, digits	See [print()]().
	Automatically called when an object of class [ergm]() is printed. Currently, summarizes the size of the MCMC sample, the θ vector governing the selection of the sample, and the Monte Carlo MLE. The optional digits argument specifies the significant digits for coefficients
sources	For the vcov method, specify whether to return the covariance matrix from the ERGM model, the estimation process, or both combined.

Value

[ergm]() returns an object of class [ergm]() that is a list consisting of the following elements:

coef	The Monte Carlo maximum likelihood estimate of θ, the vector of coefficients for the model parameters.
sample	The $n \times p$ matrix of network statistics, where n is the sample size and p is the number of network statistics specified in the model, generated by the last iteration of the MCMC-based likelihood maximization routine. These statistics are centered with respect to the observed statistics or target.stats, unless missing data MLE is used.
sample.obs	As sample, but for the constrained sample.
iterations	The number of Newton-Raphson iterations required before convergence.
MCMCtheta	The value of θ used to produce the Markov chain Monte Carlo sample. As long as the Markov chain mixes sufficiently well, sample is roughly a random sample from the distribution of network statistics specified by the model with the parameter equal to MCMCtheta. If estimate="MPLE" then MCMCtheta equals the MPLE.
loglikelihood	The approximate change in log-likelihood in the last iteration. The value is only approximate because it is estimated based on the MCMC random sample.
gradient	The value of the gradient vector of the approximated loglikelihood function, evaluated at the maximizer. This vector should be very close to zero.
covar	Approximate covariance matrix for the MLE, based on the inverse Hessian of the approximated loglikelihood evaluated at the maximizer.
failure	Logical: Did the MCMC estimation fail?
network	Network passed on the left-hand side of formula. If target.stats are passed, it is replaced by the network returned by [san()]().
newnetworks	A list of the final networks at the end of the MCMC simulation, one for each thread.
newnetwork	The first (possibly only) element of newnetworks.
coef.init	The initial value of θ.
est.cov	The covariance matrix of the model statistics in the final MCMC sample.
coef.hist, steplen.hist, stats.hist, stats.obs.hist	
	For the MCMLE method, the history of coefficients, Hummel step lengths, and average model statistics for each iteration..
control	The control list passed to the call.

etamap	The set of functions mapping the true parameter theta to the canonical parameter eta (irrelevant except in a curved exponential family model)
formula	The original formula entered into the ergm function.
target.stats	The target.stats used during estimation (passed through from the Arguments)
target.esteq	Used for curved models to preserve the target mean values of the curved terms. It is identical to target.stats for non-curved models.
constrained	The list of constraints implied by the constraints used by original ergm call
constraints	Constraints used during estimation (passed through from the Arguments)
reference	The reference measure used during estimation (passed through from the Arguments)
estimate	The estimation method used (passed through from the Arguments).
offset	vector of logical telling which model parameters are to be set at a fixed value (i.e., not estimated).
drop	If control$drop=TRUE, a numeric vector indicating which terms were dropped due to to extreme values of the corresponding statistics on the observed network, and how:

0 The term was not dropped.

-1 The term was at its minimum and the coefficient was fixed at -Inf.

+1 The term was at its maximum and the coefficient was fixed at +Inf.

estimable	A logical vector indicating which terms could not be estimated due to a constraints constraint fixing that term at a constant value.
null.lik	Log-likelihood of the null model. Valid only for unconstrained models.
mle.lik	The approximate log-likelihood for the MLE. The value is only approximate because it is estimated based on the MCMC random sample.

Methods (by generic)

- nobs: Return the number of informative dyads of a model fit.
- print:
- vcov: extracts the variance-covariance matrix of parameter estimates.

Notes on model specification

Although each of the statistics in a given model is a summary statistic for the entire network, it is rarely necessary to calculate statistics for an entire network in a proposed Metropolis-Hastings step. Thus, for example, if the triangle term is included in the model, a census of all triangles in the observed network is never taken; instead, only the change in the number of triangles is recorded for each edge toggle.

In the implementation of ergm, the model is initialized in R, then all the model information is passed to a C program that generates the sample of network statistics using MCMC. This sample is then returned to R, which implements a simple Newton-Raphson algorithm to approximate the MLE. An alternative style of maximum likelihood estimation is to use a stochastic approximation algorithm. This can be chosen with the control.ergm(style="Robbins-Monro") option.

The mechanism for proposing new networks for the MCMC sampling scheme, which is a Metropolis-Hastings algorithm, depends on two things: The `constraints`, which define the set of possible networks that could be proposed in a particular Markov chain step, and the weights placed on these possible steps by the proposal distribution. The former may be controlled using the `constraints` argument described above. The latter may be controlled using the `prop.weights` argument to the `control.ergm` function.

The package is designed so that the user could conceivably add additional proposal types.

References

Admiraal R, Handcock MS (2007). **networksis**: Simulate bipartite graphs with fixed marginals through sequential importance sampling. Statnet Project, Seattle, WA. Version 1. `https://statnet.org`.

Bender-deMoll S, Morris M, Moody J (2008). Prototype Packages for Managing and Animating Longitudinal Network Data: **dynamicnetwork** and **rSoNIA**. *Journal of Statistical Software*, 24(7). doi:10.18637/jss.v024.i07

Butts CT (2007). **sna**: Tools for Social Network Analysis. R package version 2.3-2. `https://cran.r-project.org/package=sna`.

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

Butts C (2015). **network**: The Statnet Project (https://statnet.org). R package version 1.12.0, `https://cran.r-project.org/package=network`.

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

Goodreau SM, Kitts J, Morris M (2008b). Birds of a Feather, or Friend of a Friend? Using Exponential Random Graph Models to Investigate Adolescent Social Networks. *Demography*, 45, in press.

Handcock, M. S. (2003) *Assessing Degeneracy in Statistical Models of Social Networks*, Working Paper \#39, Center for Statistics and the Social Sciences, University of Washington. `https://csss.uw.edu/research/working-papers/assessing-degeneracy-statistical-models-social-networks`

Handcock MS (2003b). **degreenet**: Models for Skewed Count Distributions Relevant to Networks. Statnet Project, Seattle, WA. Version 1.0, `https://statnet.org`.

Handcock MS and Gile KJ (2010). Modeling Social Networks from Sampled Data. *Annals of Applied Statistics*, 4(1), 5-25. doi:10.1214/08AOAS221

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003a). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. Statnet Project, Seattle, WA. Version 2, `https://statnet.org`.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003b). **statnet**: Software Tools for the Statistical Modeling of Network Data. Statnet Project, Seattle, WA. Version 2, `https://statnet.org`.

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). doi:10.18637/jss.v024.i03

Karwa V, Krivitsky PN, and Slavkovi\'c AB (2017). Sharing Social Network Data: Differentially Private Estimation of Exponential-Family Random Graph Models. *Journal of the Royal Statistical Society, Series C*, 66(3):481–500. doi:10.1111/rssc.12185

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

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

Snijders, T.A.B. (2002), Markov Chain Monte Carlo Estimation of Exponential Random Graph Models. Journal of Social Structure. Available from https://www.cmu.edu/joss/content/articles/volume3/Snijders.pdf.

See Also

network, %v%, %n%, ergmTerm, ergmMPLE, summary.ergm()

Examples

```
#
# load the Florentine marriage data matrix
#
data(flo)
#
# attach the sociomatrix for the Florentine marriage data
# This is not yet a network object.
#
flo
#
# Create a network object out of the adjacency matrix
#
flomarriage <- network(flo,directed=FALSE)
flomarriage
#
# print out the sociomatrix for the Florentine marriage data
#
flomarriage[,]
#
# create a vector indicating the wealth of each family (in thousands of lira)
# and add it as a covariate to the network object
#
flomarriage %v% "wealth" <- c(10,36,27,146,55,44,20,8,42,103,48,49,10,48,32,3)
flomarriage
#
# create a plot of the social network
#
plot(flomarriage)
#
# now make the vertex size proportional to their wealth
#
plot(flomarriage, vertex.cex=flomarriage %v% "wealth" / 20, main="Marriage Ties")
#
```

```
# Use 'data(package = "ergm")' to list the data sets in a
#
data(package="ergm")
#
# Load a network object of the Florentine data
#
data(florentine)
#
# Fit a model where the propensity to form ties between
# families depends on the absolute difference in wealth
#
gest <- ergm(flomarriage ~ edges + absdiff("wealth"))
summary(gest)
#
# add terms for the propensity to form 2-stars and triangles
# of families
#
gest <- ergm(flomarriage ~ kstar(1:2) + absdiff("wealth") + triangle)
summary(gest)

# import synthetic network that looks like a molecule
data(molecule)
# Add a attribute to it to mimic the atomic type
molecule %v% "atomic type" <- c(1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3)
#
# create a plot of the social network
# colored by atomic type
#
plot(molecule, vertex.col="atomic type",vertex.cex=3)

# measure tendency to match within each atomic type
gest <- ergm(molecule ~ edges + kstar(2) + triangle + nodematch("atomic type"))
summary(gest)

# compare it to differential homophily by atomic type
gest <- ergm(molecule ~ edges + kstar(2) + triangle
                        + nodematch("atomic type",diff=TRUE))
summary(gest)


# Extract parameter estimates as a numeric vector:
coef(gest)
# Sources of variation in parameter estimates:
vcov(gest, sources="model")
vcov(gest, sources="estimation")
vcov(gest, sources="all") # the default
```

---

ergm-options   *Global options and term options for the* ergm *package*

Description

Options set via the built-in options() functions that affect ergm estimation and options that control the behavior of some terms.

Global options and defaults

ergm.eval.loglik = TRUE Whether ergm() and similar functions will evaluate the likelihood of the fitted model. Can be overridden for a specific call by passing eval.loglik argument directly.

ergm.loglik.warn_dyads = TRUE Whether log-likelihood evaluation should issue a warning when the effective number of dyads that can vary in the sample space is poorly defined, such as if the degree sequence is constrained.

ergm.cluster.retries = 5 **ergm**'s parallel routines implement rudimentary fault-tolerance. This option controls the number of retries for a cluster call before giving up.

ergm.term = list() The default term options below.

Term options

Term options can be set in three places, in the order of precedence from high to low:

1. As a term argument (not always). For example, gw.cutoff below can be set in a gwesp term by gwesp(..., cutoff=X).

2. For functions such as summary that take ergm formulas but do not take a control list, the named arguments passed in as .... E.g, summary(nw~gwesp(.5,fix=TRUE), gw.cutoff=60) will evaluate the GWESP statistic with its cutoff set to 60.

3. As an element in a term.options= list passed via a control function such as control.ergm() or, for functions that do not, in a list with that argument name. E.g., summary(nw~gwesp(.5,fix=TRUE), term.options=list(gw.cutoff=60)) has the same effect.

4. As an element in a global option list ergm.term above.

The following options are in use by terms in the ergm package:

version A string that can be interpreted as an R package version. If set, the term will attempt to emulate its behavior as it was that version of ergm. Not all past version behaviors are available.

gw.cutoff In geometrically weighted terms (gwesp, gwdegree, etc.) the highest number of shared partners, degrees, etc. for which to compute the statistic. This usually defaults to 30.

cache.sp Whether the gwesp, dgwesp, and similar terms need should use a cache for the dyadwise number of shared partners. This usually improves performance significantly and therefore defaults to TRUE, but it can be disabled.

interact.dependent Whether to allow and how to handle the user attempting to interact dyad-dependent terms (e.g., absdiff("age"):triangles or absdiff("age")*triangles as opposed to absdiff("age"):nodefactor("sex")). Possible values are "error" (the default), "message", and "warning", for their respective actions, and "silent" for simply processing the term.

ergm-parallel *Parallel Processing in the* ergm *Package*

Description

Using clusters multiple CPUs or CPU cores to speed up ERGM estimation and simulation.

The ergm.getCluster function is usually called internally by the ergm process (in ergm_MCMC_sample) and will attempt to start the appropriate type of cluster indicated by the control.ergm settings. It will also check that the same version of ergm is installed on each node.

The ergm.stopCluster shuts down a cluster, but only if ergm.getCluster was responsible for starting it.

The ergm.restartCluster restarts and returns a cluster, but only if ergm.getCluster was responsible for starting it.

nthreads is a simple generic to obtain the number of parallel processes represented by its argument, keeping in mind that having no cluster (e.g., NULL) represents one thread.

Usage

```
ergm.getCluster(control = NULL, verbose = FALSE, stop_on_exit = parent.frame())

ergm.stopCluster(..., verbose = FALSE)

ergm.restartCluster(control = NULL, verbose = FALSE)

set.MT_terms(n)

get.MT_terms()

nthreads(clinfo = NULL, ...)

## S3 method for class 'cluster'
nthreads(clinfo = NULL, ...)

## S3 method for class '`NULL`'
nthreads(clinfo = NULL, ...)

## S3 method for class 'control.list'
nthreads(clinfo = NULL, ...)
```

Arguments

control   a control.ergm (or similar) list of parameter values from which the parallel
                 settings should be read; can also be NULL, in which case an existing cluster is
                 used if started, or no cluster otherwise.

verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
stop_on_exit	An environment or NULL. If an environment, defaulting to that of the calling function, the cluster will be stopped when the calling the frame in question exits.
...	not currently used
n	an integer specifying the number of threads to use; 0 (the starting value) disables multithreading, and -1 or NA sets it to the number of CPUs detected.
clinfo	a cluster or another object.

Details

For estimation that require MCMC, ergm can take advantage of multiple CPUs or CPU cores on the system on which it runs, as well as computing clusters through one of two mechanisms:

Running MCMC chains in parallel Packages parallel and snow are used to to facilitate this, all cluster types that they support are supported.

  The number of nodes used and the parallel API are controlled using the parallel and parallel.type arguments passed to the control functions, such as [control.ergm()](control.ergm).

  The [ergm.getCluster()](ergm.getCluster) function is usually called internally by the ergm process (in [ergm_MCMC_sample()](ergm_MCMC_sample)) and will attempt to start the appropriate type of cluster indicated by the [control.ergm()](control.ergm) settings. The [ergm.stopCluster()](ergm.stopCluster) is helpful if the user has directly created a cluster.

  Further details on the various cluster types are included below.

Multithreaded evaluation of model terms Rather than running multiple MCMC chains, it is possible to attempt to accelerate sampling by evaluating qualified terms' change statistics in multiple threads run in parallel. This is done using the OpenMP API.

  However, this introduces a nontrivial amont of computational overhead. See below for a list of the major factors affecting whether it is worthwhile.

Generally, the two approaches should not be used at the same time without caution. In particular, by default, cluster slave nodes will not "inherit" the multithreading setting; but parallel.inherit.MT= control parameter can override that. Their relative advantages and disadvantages are as follows:

- Multithreading terms cannot take advantage of clusters but only of CPUs and cores.

- Parallel MCMC chains produce several independent chains; multithreading still only produces one.

- Multithreading terms actually accellerates sampling, including the burn-in phase; parallel MCMC's multiple burn-in runs are effectively "wasted".

Value

[set.MT_terms()](set.MT_terms) returns the previous setting, invisibly.

[get.MT_terms()](get.MT_terms) returns the current setting.

Different types of clusters

PSOCK clusters The parallel package is used with PSOCK clusters by default, to utilize multiple cores on a system. The number of cores on a system can be determined with the detectCores() function.

This method works with the base installation of R on all platforms, and does not require additional software.

For more advanced applications, such as clusters that span multiple machines on a network, the clusters can be initialized manually, and passed into ergm() and others using the parallel control argument. See the second example below.

MPI clusters To use MPI to accelerate ERGM sampling, pass the control parameter parallel.type="MPI". ergm requires the snow and Rmpi packages to communicate with an MPI cluster.

Using MPI clusters requires the system to have an existing MPI installation. See the MPI documentation for your particular platform for instructions.

To use ergm() across multiple machines in a high performance computing environment, see the section "User initiated clusters" below.

User initiated clusters A cluster can be passed into ergm() with the parallel control parameter. ergm() will detect the number of nodes in the cluster, and use all of them for MCMC sampling. This method is flexible: it will accept any cluster type that is compatible with snow or parallel packages.

When is multithreading terms worthwhile?

- The more terms with statistics the model has, the more benefit from parallel execution.

- The more expensive the terms in the model are, the more benefit from parallel execution. For example, models with terms like gwdsp will generally get more benefit than models where all terms are dyad-independent.

- Sampling more dense networks will generally get more benefit than sparse networks. Network size has little, if any, effect.

- More CPUs/cores usually give greater speed-up, but only up to a point, because the amount of overhead grows with the number of threads; it is often better to "batch" the terms into a smaller number of threads than possible.

- Any other workload on the system will have a more severe effect on multithreaded execution. In particular, do not run more threads than CPUs/cores that you want to allocate to the tasks.

- Under Windows, even compiling with OpenMP appears to introduce unacceptable amounts of overhead, so it is disabled for Windows at compile time. To enable, *delete* src/Makevars.win and recompile from scratch.

Note

The this is a setting global to the ergm package and all of its C functions, including when called from other packages via the Linking-To mechanism.

Examples

```
# Uses 2 SOCK clusters for MCMLE estimation
```

```
data(faux.mesa.high)
nw <- faux.mesa.high
fauxmodel.01 <- ergm(nw ~ edges + isolates + gwesp(0.2, fixed=TRUE),
                     control=control.ergm(parallel=2, parallel.type="PSOCK"))
summary(fauxmodel.01)
```

---

ergm.allstats   *Calculate all possible vectors of statistics on a network for an ERGM*

Description

ergm.allstats produces a matrix of network statistics for an arbitrary statnet exponential-family
random graph model. One possible use for this function is to calculate the exact loglikelihood
function for a small network via the ergm.exact function.

Usage

```
ergm.allstats(
  formula,
  zeroobs = TRUE,
  force = FALSE,
  maxNumChangeStatVectors = 2^16,
  ...
)
```

Arguments

formula   an formula object of the form y ~ <model terms>, where y is a network object
                   or a matrix that can be coerced to a network object. For the details on the
                   possible <model terms>, see ergmTerm. To create a network object in , use
                   the network() function, then add nodal attributes to it using the %v% operator if
                   necessary.

zeroobs   Logical: Should the vectors be centered so that the network passed in the formula
                   has the zero vector as its statistics?

force   Logical: Should the algorithm be run even if it is determined that the problem
                   may be very large, thus bypassing the warning message that normally terminates
                   the function in such cases?

maxNumChangeStatVectors
                   Maximum possible number of distinct values of the vector of statistics. It's good
                   to use a power of 2 for this.

...   further arguments; not currently used.

Details

The mechanism for doing this is a recursive algorithm, where the number of levels of recursion is equal to the number of possible dyads that can be changed from 0 to 1 and back again. The algorithm starts with the network passed in formula, then recursively toggles each edge twice so that every possible network is visited.

ergm.allstats should only be used for small networks, since the number of possible networks grows extremely fast with the number of nodes. An error results if it is used on a directed network of more than 6 nodes or an undirected network of more than 8 nodes; use force=TRUE to override this error.

Value

Returns a list object with these two elements:

weights   integer of counts, one for each row of statmat telling how many networks share
            the corresponding vector of statistics.

statmat   matrix in which each row is a unique vector of statistics.

See Also

ergm.exact

Examples

```
# Count by brute force all the edge statistics possible for a 7-node
# undirected network
mynw <- network(matrix(0,7,7),dir=FALSE)
system.time(a <- ergm.allstats(mynw~edges))

# Summarize results
rbind(t(a$statmat),a$weights)

# Each value of a$weights is equal to 21-choose-k,
# where k is the corresponding statistic (and 21 is
# the number of dyads in an 7-node undirected network).
# Here's a check of that fact:
as.vector(a$weights - choose(21, t(a$statmat)))

# Simple ergm.exact outpuf for this network.
# We know that the loglikelihood for my empty 7-node network
# should simply be -21*log(1+exp(eta)), so we may check that
# the following two values agree:
-21*log(1+exp(.1234))
ergm.exact(.1234, mynw~edges, statmat=a$statmat, weights=a$weights)
```

---

ergm.bridge.llr   *Bridge sampling to evaluate ERGM log-likelihoods and log-likelihood*
                           ratios

Description

ergm.bridge.llr uses bridge sampling with geometric spacing to estimate the difference between
the log-likelihoods of two parameter vectors for an ERGM via repeated calls to simulate.formula.ergm.

ergm.bridge.0.llk is a convenience wrapper that returns the log-likelihood of configuration θ
relative to the reference measure. That is, the configuration with $\theta = 0$ is defined as having log-
likelihood of 0.

ergm.bridge.dindstart.llk is a wrapper that uses a dyad-independent ERGM as a starting point
for bridge sampling to estimate the log-likelihood for a given dyad-dependent model and parameter
configuration. Note that it only handles binary ERGMs (response=NULL) and with constraints
(constraints=) that that do not induce dyadic dependence.

Usage

```
ergm.bridge.llr(
  object,
  response = NULL,
  reference = ~Bernoulli,
  constraints = ~.,
  from,
  to,
  obs.constraints = ~. - observed,
  target.stats = NULL,
  basis = ergm.getnetwork(object),
  verbose = FALSE,
  ...,
  llronly = FALSE,
  control = control.ergm.bridge()
)

ergm.bridge.0.llk(
  object,
  response = NULL,
  reference = ~Bernoulli,
  coef,
  ...,
  llkonly = TRUE,
  control = control.ergm.bridge(),
  basis = ergm.getnetwork(object)
)

ergm.bridge.dindstart.llk(
```

```
    object,
    response = NULL,
    constraints = ~.,
    coef,
    obs.constraints = ~. - observed,
    target.stats = NULL,
    dind = NULL,
    coef.dind = NULL,
    basis = ergm.getnetwork(object),
    ...,
    llkonly = TRUE,
    control = control.ergm.bridge(),
    verbose = FALSE
)
```

Arguments

object   A model formula. See [ergm](#) for details.

response   Either a character string, a formula, or NULL (the default), to specify the response
                attributes and whether the ERGM is binary or valued. Interpreted as follows:

> NULL Model simple presence or absence, via a binary ERGM.

> **character string** The name of the edge attribute whose value is to be modeled.
> Type of ERGM will be determined by whether the attribute is [logical](#)
> (TRUE/FALSE) for binary or [numeric](#) for valued.

> **a formula** must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is
> evaluated in the formula's environment with the network's edge attributes
> accessible as variables. The optional NAME specifies the name of the edge
> attribute into which the results should be stored, with the default being a
> concise version of EXPR. Normally, the type of ERGM is determined by
> whether the result of evaluating EXPR is logical or numeric, but the optional
> TYPE can be used to override by specifying a scalar of the type involved
> (e.g., TRUE for binary and 1 for valued).

reference   A one-sided formula specifying the reference measure ($h(y)$) to be used. (De-
                faults to ~Bernoulli.)

constraints, obs.constraints

> One-sided formulas specifying one or more constraints on the support of the
> distribution of the networks being simulated and on the observation process re-
> spectively. See the documentation for a similar argument for [ergm](#) for more
> information.

from, to   The initial and final parameter vectors.

target.stats   A vector of sufficient statistics to be used in place of those of the network in the
                formula.

basis   An optional [network](#) object to start the Markov chain. If omitted, the default is
                the left-hand-side of the object.

verbose   A logical or an integer to control the amount of progress and diagnostic in-
                formation to be printed. FALSE/0 produces minimal output, wit higher values

producing more detail. Note that very high values (5+) may significantly slow down processing.

...	Further arguments to `ergm.bridge.llr` and [`simulate.formula.ergm`]().
llronly	Logical: If TRUE, only the estiamted log-ratio will be returned by `ergm.bridge.llr`.
control	A list of control parameters for algorithm tuning, typically constructed with [`control.ergm.bridge()`](). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility [`snctrl()`]() (Stat-Net ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
coef	A vector of coefficients for the configuration of interest.
llkonly	Whether only the estiamted log-likelihood should be returned by the `ergm.bridge.0.llk` and `ergm.bridge.dindstart.llk`. (Defaults to TRUE.)
dind	A one-sided formula with the dyad-independent model to use as a starting point. Defaults to the dyad-independent terms found in the formula `object` with an overal density term (`edges`) added if not redundant.
coef.dind	Parameter configuration for the dyad-independent starting point. Defaults to the MLE of `dind`.

Value

If `llronly=TRUE` or `llkonly=TRUE`, these functions return the scalar log-likelihood-ratio or the log-likelihood. Otherwise, they return a list with the following components:

llr	The estimated log-ratio.
llr.vcov	The estimated variance of the log-ratio due to MCMC approximation.
llrs	A list of lists (1 per attempt) of the estimated log-ratios for each of the `bridge.nsteps` bridges.
llrs.vcov	A list of lists (1 per attempt) of the estimated variances of the estimated log-ratios for each of the `bridge.nsteps` bridges.
paths	A list of lists (1 per attempt) with two elements: `theta`, a numeric matrix with `bridge.nsteps` rows, with each row being the respective bridge's parameter configuration; and `weight`, a vector of length `bridge.nsteps` containing its weight.
Dtheta.Du	The gradient vector of the parameter values with respect to position of the bridge.

`ergm.bridge.0.llk` result list also includes an `llk` element, with the log-likelihood itself (with the reference distribution assumed to have likelihood 0).

`ergm.bridge.dindstart.llk` result list also includes an `llk` element, with the log-likelihood itself and an `llk.dind` element, with the log-likelihood of the nearest dyad-independent model.

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

simulate.formula.ergm

---

ergm.design   *Obtain the set of informative dyads based on the network structure.*

Description

Note that this function is not recommended for general use, since it only supports only one way of specifying observational structure—through NA edges. It is likely to be deprecated in the future.

Usage

```
ergm.design(nw, ...)
```

Arguments

nw   a network object.

...   term options.

Value

ergm.design returns a rlebdm of informative (non-missing, non fixed) dyads.

---

ergm.exact   *Calculate the exact loglikelihood for an ERGM*

Description

ergm.exact calculates the exact loglikelihood, evaluated at eta, for the statnet exponential-family random graph model represented by formula.

Usage

```
ergm.exact(eta, formula, statmat = NULL, weights = NULL, ...)
```

Arguments

eta	vector of canonical parameter values at which the loglikelihood should be evaluated.
formula	an link{formula} object of the form y ~ <model terms>, where y is a network object or a matrix that can be coerced to a [network](#) object. For the details on the possible <model terms>, see [ergmTerm](#). To create a [network](#) object in , use the network() function, then add nodal attributes to it using the %v% operator if necessary.
statmat	if NULL, call [ergm.allstats](#) to generate all possible graph statistics for the networks in this model.
weights	In case statmat is not NULL, this should be the vector of counts corresponding to the rows of statmat. If statmat is NULL, this is generated by the call to [ergm.allstats](#).
...	further arguments; not currently used.

Details

ergm.exact should only be used for small networks, since the number of possible networks grows extremely fast with the number of nodes. An error results if it is used on a directed network of more than 6 nodes or an undirected network of more than 8 nodes; use force=TRUE to override this error.

In case this function is to be called repeatedly, for instance by an optimization routine, it is preferable to call [ergm.allstats](#) first, then pass statmat and weights explicitly to avoid repeatedly calculating these objects.

Value

Returns the value of the exact loglikelihood, evaluated at eta, for the statnet exponential-family random graph model represented by formula.

See Also

[ergm.allstats](#)

Examples

```
# Count by brute force all the edge statistics possible for a 7-node
# undirected network
mynw <- network(matrix(0,7,7),dir=FALSE)
system.time(a <- ergm.allstats(mynw~edges))

# Summarize results
rbind(t(a$statmat),a$weights)

# Each value of a$weights is equal to 21-choose-k,
# where k is the corresponding statistic (and 21 is
# the number of dyads in an 7-node undirected network).
# Here's a check of that fact:
as.vector(a$weights - choose(21, t(a$statmat)))
```

```
# Simple ergm.exact outpuf for this network.
# We know that the loglikelihood for my empty 7-node network
# should simply be -21*log(1+exp(eta)), so we may check that
# the following two values agree:
-21*log(1+exp(.1234))
ergm.exact(.1234, mynw~edges, statmat=a$statmat, weights=a$weights)
```

---

ergm.getnetwork   *Acquire and verify the network from the LHS of an* ergm *formula and verify that it is a valid network.*

Description

The function function ensures that the network in a given formula is valid; if so, the network is returned; if not, execution is halted with warnings.

Usage

```
ergm.getnetwork(formula, loopswarning = TRUE)
```

Arguments

formula   a two-sided formula whose LHS is a network, an object that can be coerced to a network, or an expression that evaluates to one.

loopswarning   whether warnings about loops should be printed (TRUE or FALSE); defaults to TRUE.

Value

A network object constructed by evaluating the LHS of the model formula in the formula's environment.

---

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

Description

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

Usage

```
ergm.godfather(
  formula,
  changes = NULL,
  response = NULL,
  end.network = FALSE,
  stats.start = FALSE,
  changes.only = FALSE,
  verbose = FALSE,
  control = control.ergm.godfather()
)
```

Arguments

formula   An ergm-style formula, with a network on its LHS.

changes   Either a matrix with three columns: tail, head, and new value, describing the changes to be made; or a list of such matrices to apply these changes in a sequence. For binary network models, the third column may be omitted. In that case, the changes are treated as toggles. Note that if a list is passed, it must either be all of changes or all of toggles.

response   Either a character string, a formula, or NULL (the default), to specify the response attributes and whether the ERGM is binary or valued. Interpreted as follows:

NULL Model simple presence or absence, via a binary ERGM.

character string The name of the edge attribute whose value is to be modeled. Type of ERGM will be determined by whether the attribute is logical (TRUE/FALSE) for binary or numeric for valued.

a formula must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is evaluated in the formula's environment with the network's edge attributes accessible as variables. The optional NAME specifies the name of the edge attribute into which the results should be stored, with the default being a concise version of EXPR. Normally, the type of ERGM is determined by whether the result of evaluating EXPR is logical or numeric, but the optional TYPE can be used to override by specifying a scalar of the type involved (e.g., TRUE for binary and 1 for valued).

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 ERGMs when output="stats", where initial network's statistics are not returned.

changes.only   Whether to return network statistics or only their changes relative to the initial network.

verbose   A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.

control   A list of control parameters for algorithm tuning, typically constructed with
               `control.ergm.godfather()`. Its documentation gives the the list of recog-
               nized control parameters and their meaning. The more generic utility `snctrl()`
               (StatNet ConTRoL) also provides argument completion for the available control
               functions and limited argument name checking.

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, 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

`tergm::tergm.godfather()`, `simulate.ergm()`, `simulate.formula()`

Examples

```
data(florentine)
ergm.godfather(flomarriage~edges+absdiff("wealth")+triangles,
               changes=list(cbind(1:2,2:3),
                            cbind(3,5),
                            cbind(3,5),
                            cbind(1:2,2:3)),
               stats.start=TRUE)
```

---

ergmConstraint   *Sample Space Constraints for Exponential-Family Random Graph*
                           Models

Description

This page describes how to specify the constraints on the network sample space (the set of possible
networks Y, the set of networks y for which $h(y) > 0$) and sometimes the baseline weights $h(y)$
to functions in the `ergm` package. It also provides an indexed list of the constraints visible to the
ergm's API.

Specifying constraints

In an exponential-family random graph model (ERGM), the probability or density of a given net-
work, $y \in Y$, on a set of nodes is

$$h(y) \exp \eta(\theta) \cdot g(y)/\kappa(\theta),$$

where $h(y)$ is the reference distribution (particularly for valued network models), $g(y)$ is a vector
of network statistics for y, $\eta(\theta)$ is a natural parameter vector of the same length (with $\eta(\theta) \equiv \theta$

for most terms), $\cdot$ is the dot product, and $\kappa(\theta)$ is the normalizing constant for the distribution. A complete ERGM specification requires a list of network statistics $g(y)$ and (if applicable) their $\eta(\theta)$ mappings provided by a formula of `ergmTerm`s; and, optionally, sample space $\mathcal{Y}$ and reference distribution $h(y)$ information provided by `ergmConstraint`s and, for valued ERGMs, by `ergmReference`s. Constraints typically affect Y, or, equivalently, set $h(y) = 0$ for some y, but some ("soft" constraints) set $h(y)$ to values other than 0 and 1.

A constraints formula is a one- or two-sided formula whose left-hand side is an optional direct selection of the `InitErgmProposal` function and whose right-hand side is a series of one or more terms separated by `"+"` and `"-"` operators, specifying the constraint.

The sample space (over and above the reference distribution) is determined by iterating over the constraints terms from left to right, each term updating it as follows:

- If the constraint introduces complex dependence structure (e.g., constrains degree or number of edges in the network), then this constraint always restricts the sample space. It may only have a `"+"` sign.

- If the constraint only restricts the set of dyads that may vary in the sample space (e.g., block-diagonal structure or fixing specific dyads at specific values) and has a `"+"` sign, the set of dyads that may vary is restricted to those that may vary according to this constraint *and* all the constraints to date.

- If the constraint only restricts the set of dyads that may vary in the sample space but has a `"-"` sign, the set of dyads that may vary is expanded to those that may vary according to this constraint *or* all the constraints up to date.

For example, a constraints formula ~a-b+c-d with all constraints dyadic will allow dyads permitted by either a or b but only if they are also permitted by c; as well as all dyads permitted by d. If A, B, C, and D were logical matrices, the matrix of variable dyads would be equal to ((A|B)&C)|D.

Terms with a positive sign can be viewed as "adding" a constraint while those with a negative sign can be viewed as "relaxing" a constraint.

The constraints formula can also contain a .. If present, `%ergmlhs%` attributes `constraints` and `constraints.obs` will be substituted in its place.

Constraints visible to the package

Term	Package	Description	Concepts
Dyads(fix=NULL, vary=NULL)	ergm	Constrain fixed or varying dyad-independent terms	directed dyad-independent operator undirected
b1degrees	ergm	Preserve the actor degree for bipartite networks	bipartite
b2degrees	ergm	Preserve the receiver degree for bipartite networks	bipartite
bd(attribs, maxout, maxin, minout, minin)	ergm	Constrain maximum and minimum vertex degree	directed undirected

blockdiag(attr)	ergm	Force a block-diagonal structure on the network	directed dyad-independent undi-rected
blocks(attr=NULL, levels=NULL, levels2=FALSE, b1levels=NULL, b2levels=NULL)	ergm	Constrain "blocks" of dyads	directed dyad-independent undi-rected
degreedist	ergm	Preserve the degree distribution of the given net-work	directed undi-rected
degrees nodedegrees	ergm	Preserve the degree of each vertex of the given net-work	directed undi-rected
dyadnoise(p01, p10)	ergm	A soft constraint to adjust the sampled distribution for dyad-level noise with known perturbation prob-abilities	directed dyad-independent soft undi-rected
edges	ergm	Preserve the edge count of the given network	
egocentric(attr=NULL, direction="both")	ergm	Preserve values of dyads incident on vertices with given attribute	directed dyad-independent undi-rected
fixallbut(free.dyads)	ergm	Preserve the dyad status in all but the given edges	directed dyad-independent undi-rected
fixedas(present, absent)	ergm	Preserve and preclude edges	directed dyad-independent undi-rected
hamming	ergm	Preserve the hamming distance to the given network (BROKEN: Do NOT Use)	directed undi-rected
idegreedist	ergm	Preserve the indegree distribution	directed
idegrees	ergm	Preserve indegree for directed networks	directed
observed	ergm	Preserve the observed dyads of the given network	directed dyad-independent undi-rected

odegreedist	ergm	Preserve the outdegree distribution	directed
odegrees	ergm	Preserve outdegree for directed networks	directed

All constraints:

Term	dir	dyad-indep	op	undir	bip	soft
Dyads	o	o	o	o		
b1degrees					o	
b2degrees					o	
bd	o			o		
blockdiag	o	o		o		
blocks	o	o		o		
degreedist	o			o		
degrees	o			o		
dyadnoise	o	o		o		o
edges						
egocentric	o	o		o		
fixallbut	o	o		o		
fixedas	o	o		o		
hamming	o			o		
idegreedist	o					
idegrees	o					
observed	o	o		o		
odegreedist	o					
odegrees	o					

Constraints by keywords:

directed Dyads, bd, blockdiag, blocks, degreedist, degrees, dyadnoise, egocentric, fixallbut, fixedas, hamming, idegreedist, idegrees, observed, odegreedist, odegrees

dyad-independent Dyads, blockdiag, blocks, dyadnoise, egocentric, fixallbut, fixedas, observed

operator Dyads

undirected Dyads, bd, blockdiag, blocks, degreedist, degrees, dyadnoise, egocentric, fixallbut, fixedas, hamming, observed

bipartite b1degrees, b2degrees

soft dyadnoise

References

- 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.

- 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

- Karwa V, Krivitsky PN, and Slavkovi\'c AB (2016). Sharing Social Network Data: Differentially Private Estimation of Exponential-Family Random Graph Models. *Journal of the Royal Statistical Society, Series C*, 66(3): 481-500. doi:10.1111/rssc.12185

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

- 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

---

ergmHint	*MCMC Hints for Exponential-Family Random Graph Models*

Description

This page describes how to provide to the **ergm**'s MCMC algorithms information about the sample space.

"Hints" for MCMC

In an exponential-family random graph model (ERGM), the probability or density of a given network, $y \in Y$, on a set of nodes is

$$h(y) \exp \eta(\theta) \cdot g(y)/\kappa(\theta),$$

where $h(y)$ is the reference distribution (particularly for valued network models), $g(y)$ is a vector of network statistics for y, $\eta(\theta)$ is a natural parameter vector of the same length (with $\eta(\theta) \equiv \theta$ for most terms), $\cdot$ is the dot product, and $\kappa(\theta)$ is the normalizing constant for the distribution. A complete ERGM specification requires a list of network statistics $g(y)$ and (if applicable) their $\eta(\theta)$ mappings provided by a formula of ergmTerms; and, optionally, sample space $\mathcal{Y}$ and reference distribution $h(y)$ information provided by ergmConstraints and, for valued ERGMs, by ergmReferences.

It is often the case that there is additional information available about the distribution of networks being modelled. For example, you may be aware that the network is sparse or that there are strata among the dyads. "Hints", typically passed on the right-hand side of MCMC.prop and obs.MCMC.prop arguments to control.ergm(), control.simulate.ergm(), and others, allow this information to be provided. By default, hint sparse is in effect.

Unlike constraints, model terms, and reference distributions, "hints" do not affect the specification of the model. That is, regardless of what "hints" may or may not be in effect, the sample space and the probabilities within it are the same. However, "hints" may affect the MCMC proposal distribution used by the samplers.

Note that not all proposals support all "hints": and if the most suitable proposal available cannot incorporate a particular "hint", a warning message will be printed.

"Hints" use the same underlying API as constraints, and, if present, %ergmlhs% attributes constraints and constraints.obs will be substituted in its place.

Hints available to the package

The following hints are known to **ergm** at this time:

Term	Package	Description	Concepts
sparse	ergm	Sparse network	dyad-independent
strat(attr=NULL, pmat=NULL, empirical=FALSE)	ergm	Stratified dyads	dyad-independent

References

- 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.

- 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

- Karwa V, Krivitsky PN, and Slavkovi\'c AB (2016). Sharing Social Network Data: Differentially Private Estimation of Exponential-Family Random Graph Models. *Journal of the Royal Statistical Society, Series C*, 66(3): 481-500. doi:10.1111/rssc.12185

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

- 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

---

ergmKeyword	*Keywords defined for Exponential-Family Random Graph Models*

Description

This collects all defined keywords defined for the ERGM and derived packages

Possible keywords defined by the ERGM and derived packages

name	short	description	popular	package
binary	bin	suitable for binary ERGMs	TRUE	ergm
bipartite	bip	suitable for bipartite networks	TRUE	ergm
categorical nodal attribute	cat nodal attr	involves a categorical nodal attribute	FALSE	ergm

continuous	cont	a continuous distribution for edge values	FALSE	ergm
curved	curved	is a curved term	FALSE	ergm
directed	dir	suitable for directed networks	TRUE	ergm
discrete	discrete	a discrete distribution for edge values	FALSE	ergm
dyad-independent	dyad-indep	does not induce dyadic dependence	TRUE	ergm
finite	fin	finite edge values only	FALSE	ergm
frequently-used	freq	is frequently used	FALSE	ergm
nonnegative	nneg	only meaningful for nonnegative edge values	FALSE	ergm
operator	op	a term operator	TRUE	ergm
positive	pos	only meaningful for positive edge values	FALSE	ergm
quantitative nodal attribute	quant nodal attr	involves a quantitative nodal attribute	FALSE	ergm
soft	soft	a constraint that does not necessarily forbid specific networks outright but reweights their probabilities	FALSE	ergm
triad-related	triad rel	involves triangles, two-paths, and other triadic structures	FALSE	ergm
valued	val	suitable for valued ERGMs	TRUE	ergm
undirected	undir	suitable for undirected networks	TRUE	ergm

ergmMPLE	*ERGM Predictors and response for logistic regression calculation of MPLE*

Description

Return the predictor matrix, response vector, and vector of weights that can be used to calculate the MPLE for an ERGM.

Usage

```
ergmMPLE(
  formula,
  constraints = ~.,
  obs.constraints = ~-observed,
  output = c("matrix", "array", "dyadlist", "fit"),
  expand.bipartite = FALSE,
  control = control.ergm(),
  verbose = FALSE,
  ...,
  basis = ergm.getnetwork(formula)
)
```

Arguments

formula, constraints, obs.constraints

> An ERGM formula and (optional) constraint specification formulas. See [ergm](#).

output   Character, partially matched. See Value.

expand.bipartite

> Logical. Specifies whether the output matrices (or array slices) representing dyads for bipartite networks are represented as rectangular matrices with first mode vertices in rows and second mode in columns, or as square matrices with dimension equalling the total number of vertices, containing with structural NAs or 0s within each mode.

control   A list of control parameters for algorithm tuning, typically constructed with [control.ergm()](#). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()](#) (StatNet Control) also provides argument completion for the available control functions and limited argument name checking.

verbose   A logical or an integer to control the amount of progress and diagnostic information to be printed. `FALSE`/`0` produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.

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

basis   a value (usually a [network](#)) to override the LHS of the formula.

Details

The MPLE for an ERGM is calculated by first finding the matrix of change statistics. Each row of this matrix is associated with a particular pair (ordered or unordered, depending on whether the network is directed or undirected) of nodes, and the row equals the change in the vector of network statistics (as defined in `formula`) when that pair is toggled from a 0 (no edge) to a 1 (edge), holding all the rest of the network fixed. The MPLE results if we perform a logistic regression in which the predictor matrix is the matrix of change statistics and the response vector is the observed network (i.e., each entry is either 0 or 1, depending on whether the corresponding edge exists or not).

Using `output="matrix"`, note that the result of the fit may be obtained from the [glm](#) function, as shown in the examples below.

Value

If `output=="matrix"` (the default), then only the response, predictor, and weights are returned; thus, the MPLE may be found by hand or the vector of change statistics may be used in some other way. To save space, the algorithm will automatically search for any duplicated rows in the predictor matrix (and corresponding response values). `ergmMPLE` function will return a list with three elements, `response`, `predictor`, and `weights`, respectively the response vector, the predictor matrix, and a vector of weights, which are really counts that tell how many times each corresponding response, predictor pair is repeated.

If `output=="dyadlist"`, as `"matrix"`, but rather than coalescing the duplicated rows, every relation in the network that is not fixed and is observed will have its own row in `predictor` and element

in `response` and `weights`, and `predictor` matrix will have two additional rows at the start, `tail` and `head`, indicating to which dyad the row and the corresponding elements pertain.

If `output=="array"`, a list with similarly named three elements is returned, but `response` is formatted into a sociomatrix; `predictor` is a 3-dimensional array of with cell `predictor[t,h,k]` containing the change score of term k for dyad (`t,h`); and `weights` is also formatted into a sociomatrix, with an element being 1 if it is to be added into the pseudolikelihood and 0 if it is not.

In particular, for a unipartite network, cells corresponding to self-loops, i.e., `predictor[i,i,k]` will be `NA` and `weights[i,i]` will be 0; and for a unipartite undirected network, lower triangle of each `predictor[,,k]` matrix will be set to `NA`, with the lower triangle of `weights` being set to 0.

To all of the above output types, `attr(., "etamap")` is attached containing the mapping and offset information.

If `output=="fit"`, then `ergmMPLE` simply calls the `ergm` function with the `estimate="MPLE"` option set, returning an object of class `ergm` that gives the fitted pseudolikelihood model.

See Also

`ergm`, `glm`

Examples

```
data(faux.mesa.high)
formula <- faux.mesa.high ~ edges + nodematch("Sex") + nodefactor("Grade")
mplesetup <- ergmMPLE(formula)

# Obtain MPLE coefficients "by hand":
coef(glm(mplesetup$response ~ . - 1, data = data.frame(mplesetup$predictor),
         weights = mplesetup$weights, family="binomial"))

# Check that the coefficients agree with the output of the ergm function:
coef(ergmMPLE(formula, output="fit"))

# We can also format the predictor matrix into an array:
mplearray <- ergmMPLE(formula, output="array")

# The resulting matrices are big, so only print the first 8 actors:
mplearray$response[1:8,1:8]
mplearray$predictor[1:8,1:8,]
mplearray$weights[1:8,1:8]

if(require(tergm)){
# Constraints are handled:
faux.mesa.high%v%"block" <- seq_len(network.size(faux.mesa.high)) %/% 4
mplearray <- ergmMPLE(faux.mesa.high~edges, constraints=~blockdiag("block"), output="array")
mplearray$response[1:8,1:8]
mplearray$predictor[1:8,1:8,]
mplearray$weights[1:8,1:8]

# Or, a dyad list:
faux.mesa.high%v%"block" <- seq_len(network.size(faux.mesa.high)) %/% 4
mplearray <- ergmMPLE(faux.mesa.high~edges, constraints=~blockdiag("block"), output="dyadlist")
```

```
mplearray$response[1:8]
mplearray$predictor[1:8,]
mplearray$weights[1:8]
}

# Curved terms produce predictors on the canonical scale:
formula2 <- faux.mesa.high ~ gwesp
mplearray <- ergmMPLE(formula2, output="array")
# The resulting matrices are big, so only print the first 5 actors:
mplearray$response[1:5,1:5]
mplearray$predictor[1:5,1:5,1:3]
mplearray$weights[1:5,1:5]
```

---

ergmProposal	*Metropolis-Hastings Proposal Methods for ERGM MCMC*

Description

[ergm](#) uses a Metropolis-Hastings (MH) algorithm 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 [ergm](#) 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.

Implemented proposals for ergm models

Proposal	Reference	Enforced	May Enforce	Priority	Weight	Class
BDStratTNT	Bernoulli	sparse	bdmax blocks strat	-3	BDStratTNT	c
BDStratTNT	Bernoulli	bdmax sparse	blocks strat	0	BDStratTNT	c
BDStratTNT	Bernoulli	blocks sparse	bdmax strat	0	BDStratTNT	c
BDStratTNT	Bernoulli	strat sparse	bdmax blocks	0	BDStratTNT	c
CondB1Degree	Bernoulli	b1degrees		0	random	c
CondB2Degree	Bernoulli	b2degrees		0	random	c
CondDegree	Bernoulli	degrees		0	random	c
CondDegree	Bernoulli	idegrees odegrees		0	random	c
CondDegree	Bernoulli	b1degrees b2degrees		0	random	c
CondDegreeDist	Bernoulli	degreedist		0	random	c
CondDegreeMix	Bernoulli	degreesmix		0	random	c
CondInDegree	Bernoulli	idegrees		0	random	c
CondInDegreeDist	Bernoulli	idegreedist		0	random	c
CondOutDegree	Bernoulli	odegrees		0	random	c
CondOutDegreeDist	Bernoulli	odegreedist		0	random	c
ConstantEdges	Bernoulli	edges	bd	0	random	c

DiscUnif	DiscUnif			0	random	c
DiscUnif2	DiscUnif			-1	random2	c
DiscUnifNonObserved	DiscUnif	observed		0	random	c
DistRLE	StdNormal		.dyads	0	random	c
DistRLE	Unif		.dyads	0	random	c
DistRLE	Unif		.dyads	-3	random	c
DistRLE	DiscUnif		.dyads	-3	random	c
DistRLE	StdNormal		.dyads	-3	random	c
DistRLE	Poisson		.dyads	-3	random	c
DistRLE	Binomial		.dyads	-3	random	c
HammingConstantEdges	Bernoulli	edges hamming		0	random	c
HammingTNT	Bernoulli	hamming sparse		0	random	c
StdNormal	StdNormal			0	random	c
TNT	Bernoulli	sparse	.dyads bd	-1	TNT	c
Unif	Unif			0	random	c
UnifNonObserved	Unif	observed		0	random	c
dyadnoise	Bernoulli	dyadnoise		0	random	c
dyadnoiseTNT	Bernoulli	dyadnoise sparse		1	TNT	c
randomtoggle	Bernoulli		.dyads bd	-2	random	c

References

- 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*.

- 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

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

- 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

See Also

ergm package, ergm, ergmConstraint, ergm_proposal

ergmReference | *Reference Measures for Exponential-Family Random Graph Models*

Description

This page describes how to specify the reference measures (baseline distributions) (the set of possible networks Y and the baseline weights $h(y)$ to functions in the ergm package. It also provides an indexed list of the references visible to the **ergm**'s API.

Specifying reference measures

In an exponential-family random graph model (ERGM), the probability or density of a given network, $y \in Y$, on a set of nodes is

$$h(y) \exp \eta(\theta) \cdot g(y)/\kappa(\theta),$$

where $h(y)$ is the reference distribution (particularly for valued network models), $g(y)$ is a vector of network statistics for y, $\eta(\theta)$ is a natural parameter vector of the same length (with $\eta(\theta) \equiv \theta$ for most terms), $\cdot$ is the dot product, and $\kappa(\theta)$ is the normalizing constant for the distribution. A complete ERGM specification requires a list of network statistics $g(y)$ and (if applicable) their $\eta(\theta)$ mappings provided by a formula of ergmTerms; and, optionally, sample space $\mathcal{Y}$ and reference distribution $h(y)$ information provided by ergmConstraints and, for valued ERGMs, by ergmReferences.

The reference measure $(Y, h(y))$ is specified on the right-hand side of a one-sided formula passed typically as the reference argument.

Reference measures visible to the package

Term	Package	Description	Concepts
Bernoulli	ergm	Bernoulli reference	discrete finite non-negative
DiscUnif(a,b)	ergm	Discrete Uniform reference	discrete finite
StdNormal	ergm	Standard Normal reference	continuous
Unif(a,b)	ergm	Continuous Uniform reference	continuous

All references:

Term	bin	discrete	fin	nneg	cont
Bernoulli	o	o	o	o	
DiscUnif		o	o		
StdNormal					o
Unif					o

References by keywords:

binary Bernoulli

discrete Bernoulli, DiscUnif

finite Bernoulli, DiscUnif

nonnegative Bernoulli

continuous StdNormal, Unif

References

- 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

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

See Also

ergm, network, sna, summary.ergm, print.ergm, \%v\%, \%n\%

---

ergmTerm   *Terms used in Exponential Family Random Graph Models*

Description

This page explains how to specify the network statistics $g(y)$ to functions in the ergm package and packages that extend it. It also provides an indexed list of the possible terms (and hence network statistics) visible to the **ergm** API. Terms can also be searched via search.ergmTerms.

Specifying models

In an exponential-family random graph model (ERGM), the probability or density of a given network, $y \in Y$, on a set of nodes is

$$h(y) \exp \eta(\theta) \cdot g(y) / \kappa(\theta),$$

where $h(y)$ is the reference distribution (particularly for valued network models), $g(y)$ is a vector of network statistics for y, $\eta(\theta)$ is a natural parameter vector of the same length (with $\eta(\theta) \equiv \theta$ for most terms), $\cdot$ is the dot product, and $\kappa(\theta)$ is the normalizing constant for the distribution. A complete ERGM specification requires a list of network statistics $g(y)$ and (if applicable) their $\eta(\theta)$ mappings provided by a formula of ergmTerms; and, optionally, sample space $\mathcal{Y}$ and reference distribution $h(y)$ information provided by ergmConstraints and, for valued ERGMs, by ergmReferences.

Network statistics $g(y)$ and mappings $\eta(\theta)$ are specified by a formula object, of the form y ~ <term 1> + <term 2> ..., where y is a network object or a matrix that can be coerced to a network object, and <term 1>, <term 2>, etc, are each terms chosen from the list given below. To create a network object in , use the network function, then add nodal attributes to it using the %v% operator if necessary.

Operator terms:

Operator terms like B and F take formulas with other ergm terms as their arguments and transform them by modifying their inputs (e.g., the network they evaluate) and/or their outputs.

By convention, their names are capitalized and CamelCased.

Interactions:

For binary ERGMs, interactions between ergm terms can be specified in a manner similar to lm and others, as using the : and * operators. However, they must be interpreted carefully, especially for dyad-dependent terms. (Interactions involving curved terms are not supported at this time.)

Generally, if term a has p_a statistics and b has p_b, a:b will add $p_a \times p_b$ statistics to the model, corresponding to each element of $g_a(y)$ interacted with each element of $g_b(y)$.

The interaction is defined as follows. Dyad-independent terms can be expressed in the general form $g(y; x) = \sum_{i,j} x_{i,j} y_{i,j}$ for some edge covariate matrix x,

$$g_{a:b}(y) = \sum_{i,j} x_{a,i,j} x_{b,i,j} y_{i,j}.$$

In other words, rather than being a product of their sufficient statistics $(g_a(y)g_b(y))$, it is a dyad-wise product of their dyad-level effects.

This means that an interaction between two dyad-independent terms can be interpreted the same way as it would be in the corresponding logistic regression for each potential edge. However, for undirected networks in particular, this may lead to somewhat counterintuitive results. For example, given two nodal covariates "a" and "b" (whose values for node i are denoted a_i and b_i, respectively), nodecov("a") adds one statistic of the form $\sum_{i,j}(a_i + a_j)y_{i,j}$ and analogously for nodecov("b"), so nodecov("a"):nodecov("b") produces

$$\sum_{i,j}(a_i + a_j)(b_i + b_j)y_{i,j}.$$

Binary and valued ERGM terms:

ergm functions such as ergm and simulate (for ERGMs) may operate in two modes: binary and weighted/valued, with the latter activated by passing a non-NULL value as the response argument, giving the edge attribute name to be modeled/simulated.

Generalizations of binary terms:

Binary ERGM statistics cannot be used directly in valued mode and vice versa. However, a substantial number of binary ERGM statistics — particularly the ones with dyadic independence — have simple generalizations to valued ERGMs, and have been adapted in ergm. They have the same form as their binary ERGM counterparts, with an additional argument: form, which, at this time, has two possible values: "sum" (the default) and "nonzero". The former creates a statistic of the form $\sum_{i,j} x_{i,j} y_{i,j}$, where $y_{i,j}$ is the value of dyad (i, j) and $x_{i,j}$ is the term's covariate associated with it. The latter computes the binary version, with the edge considered to be present if its value is not 0. Valued version of some binary ERGM terms have an argument threshold, which sets the value above which a dyad is conidered to have a tie. (Value less than or equal to threshold is considered a nontie.)

The B() operator term documented below can be used to pass other binary terms to valued models, and is more flexible, at the cost of being somewhat slower.

Nodal attribute levels and indices:

Terms taking a categorical nodal covariate also take the `levels` argument. (There are analogous `b1levels` and `b2levels` arguments for some terms that apply to bipartite networks, and the `levels2` argument for mixing terms.) The `levels` argument can be used to control the set and the ordering of attribute levels.

Terms that allow the selection of nodes do so with the `nodes` argument, which is interpreted in the same way as the `levels` argument, where the categories are the relevant nodal indices themselves.

Both `levels` and `nodes` use the new level selection UI. (See Specifying Vertex attributes and Levels (?

`nodal_attributes`) for details.)

Legacy arguments:
The legacy `base` and `keep` arguments are deprecated as of version 3.10, and replaced by the `levels` UI. The `levels` argument provides consistent and flexible mechanisms for specifying which attribute levels to exclude (previously handled by `base`) and include (previously handled by `keep`). If `levels` or `nodes` argument is given, then `base` and `keep` arguments are ignored. The legacy arguments will most likely be removed in a future version.

Note that this exact behavior is new in version 3.10, and it differs slightly from older versions: previously if both `levels` and `base`/`keep` were given, `levels` argument was applied first and then applied the `base`/`keep` argument. Since version 3.10, `base`/`keep` would be ignored, even if old term behavior is invoked (as described in the next section).

Term versioning:

When a term's behavior has changed from prior version, it is often possible to invoke the old behavior by setting and/or passing a `version` term option, giving the verison (constructed by `as.package_version`) desired.

Custom `ergm` terms:

Users and other packages may build custom terms, and package `ergm.userterms` provides tools for implementing them.

The current recommendation for any package implementing additional terms is to document the term with Roxygen comments and a name in the form `termName-ergmTerm`. This ensures that `help("ergmTerm")` will list ERGM terms available from all loaded packages.

Terms included in the `ergm` package

As noted above, a cross-referenced HTML version of the term documentation is also available via `vignette('ergm-term-crossRef')` and terms can also be searched via `search.ergmTerms`.

Term index (plain):

Term	Package	Description	Concepts
`absdiff(attr, pow) (bin)` `absdiff(attr, pow, form)` `(val)`	ergm	Absolute difference in nodal attribute	directed dyad-independent quantitative nodal attribute undirected

`absdiffcat(attr, base, levels)` (bin) `absdiffcat(attr, base, levels, form)` (val)	ergm	Categorical absolute difference in nodal attribute	categorical nodal attribute directed dyad-independent undirected
`altkstar(lambda, fixed)` (bin)	ergm	Alternating k-star	categorical nodal attribute curved undirected
`asymmetric(attr, diff, keep, levels)` (bin)	ergm	Asymmetric dyads	directed dyad-independent triad-related
`atleast(threshold)` (val)	ergm	Number of dyads with values greater than or equal to a threshold	directed dyad-independent undirected
`atmost(threshold)` (val)	ergm	Number of dyads with values less than or equal to a threshold	directed dyad-independent undirected
`attrcov(attr, mat)` (bin)	ergm	Edge covariate by attribute pairing	directed dyad-independent undirected
`b1concurrent(by, levels)` (bin)	ergm	Concurrent node count for the first mode in a bipartite network	bipartite categorical nodal attribute undirected

`b1cov(attr)` (bin) `b1cov(attr, form)` (val)	ergm	Main effect of a covariate for the first mode in a bipartite network	bipartite dyad-independent frequently-used quantitative nodal attribute undirected
`b1degrange(from, to, by, homophily, levels)` (bin)	ergm	Degree range for the first mode in a bipartite network	bipartite undirected
`b1degree(d, by, levels)` (bin)	ergm	Degree for the first mode in a bipartite network	bipartite categorical nodal attribute frequently-used undirected
`b1dsp(d)` (bin)	ergm	Dyadwise shared partners for dyads in the first bipartition	bipartite undirected
`b1factor(attr, base, levels)` (bin) `b1factor(attr, base, levels, form)` (val)	ergm	Factor attribute effect for the first mode in a bipartite network	bipartite categorical nodal attribute dyad-independent frequently-used undirected
`b1mindegree(d)` (bin)	ergm	Minimum degree for the first mode in a bipartite network	bipartite undirected
`b1nodematch(attr, diff, keep, alpha, beta, byb2attr, levels)` (bin)	ergm	Nodal attribute-based homophily effect for the first mode in a bipartite network	bipartite categorical nodal attribute dyad-independent frequently-used undirected

`b1sociality(nodes)` (bin) `b1sociality(nodes, form)` (val)	ergm	Degree	bipartite dyad-independent undirected
`b1star(k, attr, levels)` (bin)	ergm	k-Stars for the first mode in a bipartite network	bipartite categorical nodal attribute undirected
`b1starmix(k, attr, base, diff)` (bin)	ergm	Mixing matrix for k-stars centered on the first mode of a bipartite network	bipartite categorical nodal attribute undirected
`b1twostar(b1attr, b2attr, base, b1levels, b2levels, levels2)` (bin)	ergm	Two-star census for central nodes centered on the first mode of a bipartite network	bipartite categorical nodal attribute undirected
`b2concurrent(by)` (bin)	ergm	Concurrent node count for the second mode in a bipartite network	bipartite frequently-used undirected
`b2cov(attr)` (bin) `b2cov(attr, form)` (val)	ergm	Main effect of a covariate for the second mode in a bipartite network	bipartite dyad-independent frequently-used quantitative nodal attribute undirected
`b2degrange(from, to, by, homophily, levels)` (bin)	ergm	Degree range for the second mode in a bipartite network	bipartite undirected

`b2degree(d, by)` (bin)	ergm	Degree for the second mode in a bipartite network	bipartite categorical nodal attribute frequently-used undirected
`b2dsp(d)` (bin)	ergm	Dyadwise shared partners for dyads in the second bipartition	bipartite undirected
`b2factor(attr, base, levels)` (bin) `b2factor(attr, base, levels, form)` (val)	ergm	Factor attribute effect for the second mode in a bipartite network	bipartite categorical nodal attribute dyad-independent frequently-used undirected
`b2mindegree(d)` (bin)	ergm	Minimum degree for the second mode in a bipartite network	bipartite undirected
`b2nodematch(attr, diff, keep, alpha, beta, byb1attr, levels)` (bin)	ergm	Nodal attribute-based homophily effect for the second mode in a bipartite network	bipartite categorical nodal attribute dyad-independent frequently-used undirected
`b2sociality(nodes)` (bin) `b2sociality(nodes, form)` (val)	ergm	Degree	bipartite dyad-independent undirected
`b2star(k, attr, levels)` (bin)	ergm	k-Stars for the second mode in a bipartite network	bipartite categorical nodal attribute undirected

`b2starmix(k, attr, base, diff)` (bin)	ergm	Mixing matrix for k-stars centered on the second mode of a bipartite network	bipartite categorical nodal attribute undirected
`b2twostar(b1attr, b2attr, base, b1levels, b2levels, levels2)` (bin)	ergm	Two-star census for central nodes centered on the second mode of a bipartite network	bipartite categorical nodal attribute undirected
`balance` (bin)	ergm	Balanced triads	directed triad-related undirected
`coincidence(levels, active)` (bin)	ergm	Coincident node count for the second mode in a bipartite (aka two-mode) network	bipartite undirected
`concurrent(by, levels)` (bin)	ergm	Concurrent node count	categorical nodal attribute undirected
`concurrentties(by, levels)` (bin)	ergm	Concurrent tie count	categorical nodal attribute undirected
`ctriple(attr, diff, levels)` (bin) `ctriad` (bin)	ergm	Cyclic triples	categorical nodal attribute directed triad-related
`cycle(k, semi)` (bin)	ergm	k-Cycle Census	directed undirected
`cyclicalties(attr, levels)` (bin) `cyclicalties(threshold)` (val)	ergm	Cyclical ties	directed undirected

`cyclicalweights(twopath, combine, affect)` (val)	ergm	Cyclical weights	directed nonnegative undirected
`ddsp(d, type)` (bin)	ergm	Directed dyadwise shared partners	directed
`degcor` (bin)	ergm	Degree Correlation	undirected
`degcrossprod` (bin)	ergm	Degree Cross-Product	undirected
`degrange(from, to, by, homophily, levels)` (bin)	ergm	Degree range	categorical nodal attribute undirected
`degree(d, by, homophily, levels)` (bin)	ergm	Degree	categorical nodal attribute frequently-used undirected
`degree1.5` (bin)	ergm	Degree to the 3/2 power	undirected
`density` (bin)	ergm	Density	directed dyad-independent undirected
`desp(d, type)` (bin)	ergm	Directed edgewise shared partners	directed
`dgwdsp(decay, fixed, cutoff, type)` (bin)	ergm	Geometrically weighted dyadwise shared partner distribution	directed
`dgwesp(decay, fixed, cutoff, type)` (bin)	ergm	Geometrically weighted edgewise shared partner distribution	directed
`dgwnsp(decay, fixed, cutoff, type)` (bin)	ergm	Geometrically weighted non-edgewise shared partner distribution	directed
`diff(attr, pow, dir, sign.action)` (bin) `diff(attr, pow, dir, sign.action, form)` (val)	ergm	Difference	bipartite directed dyad-independent frequently-used quantitative nodal attribute undirected
`dnsp(d, type)` (bin)	ergm	Directed non-edgewise shared partners	directed

dsp(d) (bin)	ergm	Dyadwise shared partners	directed undirected
dyadcov(x, attrname) (bin)	ergm	Dyadic covariate	categorical nodal attribute directed dyad-independent undirected
edgecov(x, attrname) (bin) edgecov(x, attrname, form) (val)	ergm	Edge covariate	directed dyad-independent frequently-used undirected
edges (bin) nonzero (val) edges (val)	ergm	Number of edges in the network	directed dyad-independent undirected
equalto(value, tolerance) (val)	ergm	Number of dyads with values equal to a specific value (within tolerance)	directed dyad-independent undirected
esp(d) (bin)	ergm	Edgewise shared partners	directed undirected
greaterthan(threshold) (val)	ergm	Number of dyads with values strictly greater than a threshold	directed dyad-independent undirected
gwb1degree(decay, fixed, attr, cutoff, levels) (bin)	ergm	Geometrically weighted degree distribution for the first mode in a bipartite network	bipartite curved undirected
gwb1dsp(decay, fixed, cutoff) (bin)	ergm	Geometrically weighted dyadwise shared partner distribution for dyads in the first bipartition	bipartite curved undirected

`gwb2degree(decay, fixed,` `attr, cutoff, levels)` (bin)	ergm	Geometrically weighted degree distribution for the second mode in a bipartite network	bipartite curved undirected
`gwb2dsp(decay, fixed,` `cutoff)` (bin)	ergm	Geometrically weighted dyadwise shared partner distribution for dyads in the second bipartition	bipartite curved undirected
`gwdegree(decay, fixed,` `attr, cutoff, levels)` (bin)	ergm	Geometrically weighted degree distribution	curved frequentlyused undirected
`gwdsp(decay, fixed,` `cutoff)` (bin)	ergm	Geometrically weighted dyadwise shared partner distribution	curved directed undirected
`gwesp(decay, fixed,` `cutoff)` (bin)	ergm	Geometrically weighted edgewise shared partner distribution	curved directed frequentlyused undirected
`gwidegree(decay, fixed,` `attr, cutoff, levels)` (bin)	ergm	Geometrically weighted in-degree distribution	curved directed
`gwnsp(decay, fixed,` `cutoff)` (bin)	ergm	Geometrically weighted nonedgewise shared partner distribution	curved directed undirected
`gwodegree(decay, fixed,` `attr, cutoff, levels)` (bin)	ergm	Geometrically weighted out-degree distribution	curved directed
`hamming(x, cov, attrname)` (bin)	ergm	Hamming distance	directed dyadindependent undirected
`idegrange(from, to, by,` `homophily, levels)` (bin)	ergm	In-degree range	categorical nodal attribute directed
`idegree(d, by, homophily,` `levels)` (bin)	ergm	In-degree	categorical nodal attribute directed frequentlyused
`idegree1.5` (bin)	ergm	In-degree to the 3/2 power	directed

`ininterval(lower, upper, open)` (val)	ergm	Number of dyads whose values are in an interval	directed dyad-independent undi-rected
`intransitive` (bin)	ergm	Intransitive triads	directed triad-related
`isolatededges` (bin)	ergm	Isolated edges	bipartite undi-rected
`isolates` (bin)	ergm	Isolates	directed frequently-used undi-rected
`istar(k, attr, levels)` (bin)	ergm	In-stars	categorical nodal at-tribute directed
`kstar(k, attr, levels)` (bin)	ergm	k-Stars	categorical nodal at-tribute undi-rected
`localtriangle(x)` (bin)	ergm	Triangles within neighborhoods	directed triad-related undi-rected
`m2star` (bin)	ergm	Mixed 2-stars, a.k.a 2-paths	directed
`meandeg` (bin)	ergm	Mean vertex degree	directed dyad-independent undi-rected
`mm(attrs, levels, levels2)` (bin) `mm(attrs, levels, levels2, form)` (val)	ergm	Mixing matrix cells and margins	categorical nodal at-tribute directed dyad-independent frequently-used undi-rected

`mutual(same, by, diff, keep, levels)` (bin) `mutual(form, threshold)` (val)	ergm	Mutuality	directed frequently-used
`nearsimmelian` (bin)	ergm	Near simmelian triads	directed triad-related
`nodecov(attr)` (bin) `nodemain` (bin) `nodecov(attr, form)` (val) `nodemain(attr, form)` (val)	ergm	Main effect of a covariate	directed dyad-independent frequently-used quantitative nodal attribute undirected
`nodecovar(center, transform)` (val)	ergm	Covariance of undirected dyad values incident on each actor	directed
`nodefactor(attr, base, levels)` (bin) `nodefactor(attr, base, levels, form)` (val)	ergm	Factor attribute effect	categorical nodal attribute directed dyad-independent frequently-used undirected
`nodeicov(attr)` (bin) `nodeicov(attr, form)` (val)	ergm	Main effect of a covariate for in-edges	directed frequently-used quantitative nodal attribute
`nodeicovar(center, transform)` (val)	ergm	Covariance of in-dyad values incident on each actor	directed
`nodeifactor(attr, base, levels)` (bin) `nodeifactor(attr, base, levels, form)` (val)	ergm	Factor attribute effect for in-edges	categorical nodal attribute directed dyad-independent frequently-used

`nodematch(attr, diff, keep, levels)` (bin) `nodematch(attr, diff, keep, levels, form)` (val) `match(attr, diff, keep, levels, form)` (val)	ergm	Uniform homophily and differential homophily	categorical nodal attribute directed dyad-independent frequently-used undirected
`nodemix(attr, base, b1levels, b2levels, levels, levels2)` (bin) `nodemix(attr, base, b1levels, b2levels, levels, levels2, form)` (val)	ergm	Nodal attribute mixing	categorical nodal attribute directed dyad-independent frequently-used undirected
`nodeocov(attr)` (bin) `nodeocov(attr, form)` (val)	ergm	Main effect of a covariate for out-edges	directed dyad-independent quantitative nodal attribute
`nodeocovar(center, transform)` (val)	ergm	Covariance of out-dyad values incident on each actor	directed
`nodeofactor(attr, base, levels)` (bin) `nodeofactor(attr, base, levels, form)` (val)	ergm	Factor attribute effect for out-edges	categorical nodal attribute directed dyad-independent
`nsp(d)` (bin)	ergm	Nonedgewise shared partners	directed undirected
`odegrange(from, to, by, homophily, levels)` (bin)	ergm	Out-degree range	categorical nodal attribute directed
`odegree(d, by, homophily, levels)` (bin)	ergm	Out-degree	categorical nodal attribute directed frequently-used

`odegree1.5` (bin)	ergm	Out-degree to the 3/2 power	directed
`opentriad` (bin)	ergm	Open triads	triad-related undirected
`ostar(k, attr, levels)` (bin)	ergm	k-Outstars	categorical nodal attribute directed
`receiver(base, nodes)` (bin) `receiver(base, nodes, form)` (val)	ergm	Receiver effect	directed dyad-independent
`sender(base, nodes)` (bin) `sender(base, nodes, form)` (val)	ergm	Sender effect	directed dyad-independent
`simmelian` (bin)	ergm	Simmelian triads	directed triad-related
`simmelianties` (bin)	ergm	Ties in simmelian triads	directed triad-related
`smalldiff(attr, cutoff)` (bin)	ergm	Number of ties between actors with similar attribute values	directed dyad-independent quantitative nodal attribute undirected
`smallerthan(threshold)` (val)	ergm	Number of dyads with values strictly smaller than a threshold	directed dyad-independent undirected
`sociality(attr, base, levels, nodes)` (bin) `sociality(attr, base, levels, nodes, form)` (val)	ergm	Undirected degree	categorical nodal attribute dyad-independent undirected
`sum(pow)` (val)	ergm	Sum of dyad values (optionally taken to a power)	directed undirected

threepath (bin) threetrail(keep, levels) (bin)	ergm	Three-trails	directed triad- related undi- rected
transitive (bin)	ergm	Transitive triads	directed triad- related
transitiveties(attr, levels) (bin)	ergm	Transitive ties	categorical nodal at- tribute directed triad- related undi- rected
transitiveweights(twopath, combine, affect) (val)	ergm	Transitive weights	directed nonneg- ative triad- related undi- rected
triadcensus(levels) (bin)	ergm	Triad census	directed triad- related undi- rected
triangle(attr, diff, levels) (bin) triangles(attr, diff, levels) (bin)	ergm	Triangles	categorical nodal at- tribute directed frequently- used triad- related undi- rected
tripercent(attr, diff, levels) (bin)	ergm	Triangle percentage	categorical nodal at- tribute triad- related undi- rected

`ttriple(attr, diff, levels)` (bin) `ttriad` (bin)	ergm	Transitive triples	categorical nodal at-tribute directed triad-related
`twopath` (bin)	ergm	2-Paths	directed undi-rected

Term index (operator):

Term	Package	Description	Concepts
`B(formula, form)` (val)	ergm	Wrap binary terms for use in valued models	operator
`Curve(formula, params, map, gradient, minpar, maxpar, cov)` (bin) `Parametrise(formula, params, map, gradient, minpar, maxpar, cov)` (bin) `Parametrize(formula, params, map, gradient, minpar, maxpar, cov)` (bin) `Curve(formula, params, map, gradient, minpar, maxpar, cov)` (val) `Parametrise(formula, params, map, gradient, minpar, maxpar, cov)` (val) `Parametrize(formula, params, map, gradient, minpar, maxpar, cov)` (val)	ergm	Impose a curved structure on term parameters	operator
`Exp(formula)` (bin) `Exp(formula)` (val)	ergm	Exponentiate a network's statistic	operator
`F(formula, filter)` (bin)	ergm	Filtering on arbitrary one-term model	operator
`Label(formula, label, pos)` (bin) `Label(formula, label, pos)` (val)	ergm	Modify terms' coefficient names	operator
`Log(formula, log0)` (bin) `Log(formula, log0)` (val)	ergm	Take a natural logarithm of a network's statistic	operator
`NodematchFilter(formula, attrname)` (bin)	ergm	Filtering on nodematch	operator
`Offset(formula, coef, which)` (bin)	ergm	Terms with fixed coefficients	operator

Prod(formulas, label) (bin) Prod(formulas, label) (val)	ergm	A product (or an arbitrary power combination) of one or more formulas	operator
S(formula, attrs) (bin)	ergm	Evaluation on an induced subgraph	operator
Sum(formulas, label) (bin) Sum(formulas, label) (val)	ergm	A sum (or an arbitrary linear combination) of one or more formulas	operator
Symmetrize(formula, rule) (bin)	ergm	Evaluation on symmetrized (undirected) network	directed operator

Frequently-used terms:

Term	bin	bip	dir	dyad-indep	op	val	undir
b1cov	o	o		o		o	o
b1degree	o	o					o
b1factor	o	o		o		o	o
b1nodematch	o	o		o			o
b2concurrent	o	o					o
b2cov	o	o		o		o	o
b2degree	o	o					o
b2factor	o	o		o		o	o
b2nodematch	o	o		o			o
degree	o						o
diff	o	o	o	o		o	o
edgecov	o		o	o		o	o
gwdegree	o						o
gwesp	o		o				o
idegree	o		o				
isolates	o		o				o
mm	o		o	o		o	o
mutual	o		o			o	
nodecov	o		o	o		o	o
nodefactor	o		o	o		o	o
nodeicov	o		o			o	
nodeifactor	o		o	o		o	
nodematch	o		o	o		o	o
nodemix	o		o	o		o	o
odegree	o		o				
triangle	o		o				o

Operator terms:

Term	bin	bip	dir	dyad-indep	val	undir
B					o	
Curve	o				o	

Exp	o		o
F	o		
Label	o		o
Log	o		o
NodematchFilter	o		
Offset	o		
Prod	o		o
S	o		
Sum	o		o
Symmetrize	o	o	

All terms:

Term	op	val	bin	dir	dyad-indep	quant nodal attr	undir	cat nodal attr	curved	triad rel	bip	freq	nneg
B	o	o											
Curve	o	o	o										
Exp	o	o	o										
F	o		o										
Label	o	o	o										
Log	o	o	o										
NodematchFilter	o		o										
Offset	o		o										
Prod	o	o	o										
S	o		o										
Sum	o	o	o										
Symmetrize	o		o	o									
absdiff		o	o	o	o	o	o						
absdiffcat		o	o	o	o		o	o					
altkstar		o				o	o	o					
asymmetric		o	o	o						o			
atleast		o		o	o	o							
atmost		o		o	o	o							
attrcov			o	o	o	o							
b1concurrent			o			o	o				o		
b1cov		o	o		o	o	o				o	o	
b1degrange			o			o					o		
b1degree			o			o	o				o	o	
b1dsp			o			o					o		
b1factor		o	o		o	o	o				o	o	
b1mindegree			o			o					o		
b1nodematch			o		o	o	o				o	o	
b1sociality		o	o		o	o					o		
b1star			o			o	o				o		
b1starmix			o			o	o				o		

b1twostar		o				o	o			o			
b2concurrent		o				o				o	o		
b2cov	o	o		o	o	o				o	o		
b2degrange		o				o				o			
b2degree		o				o	o			o	o		
b2dsp		o				o				o			
b2factor	o	o		o		o	o			o	o		
b2mindegree		o				o				o			
b2nodematch		o		o		o	o			o	o		
b2sociality	o	o		o		o				o			
b2star		o				o	o			o			
b2starmix		o				o	o			o			
b2twostar		o				o	o			o			
balance		o	o			o			o				
coincidence		o				o				o			
concurrent		o				o	o						
concurrentties		o				o	o						
ctriple		o	o			o			o				
cycle		o	o			o							
cyclicalties	o	o	o			o							
cyclicalweights	o		o			o							o
ddsp		o	o										
degcor		o				o							
degcrossprod		o				o							
degrange		o				o	o						
degree		o				o	o				o		
degree1.5		o				o							
density		o	o	o		o							
desp		o	o										
dgwdsp		o	o										
dgwesp		o	o										
dgwnsp		o	o										
diff	o	o	o	o	o	o				o	o		
dnsp		o	o										
dsp		o	o			o							
dyadcov		o	o	o		o	o						
edgecov	o	o	o	o		o					o		
edges	o	o	o	o		o							
equalto	o		o	o		o							
esp		o	o			o							
greaterthan	o		o	o		o							
gwb1degree		o				o		o		o			
gwb1dsp		o				o		o		o			
gwb2degree		o				o		o		o			
gwb2dsp		o				o		o		o			
gwdegree		o				o		o				o	

gwdsp		o	o			o	o		
gwesp		o	o			o	o		o
gwidegree		o	o				o		
gwnsp		o	o			o	o		
gwodegree		o	o				o		
hamming		o	o	o		o			
idegrange		o	o				o		
idegree		o	o				o		o
idegree1.5		o	o						
ininterval	o		o	o		o			
intransitive		o	o					o	
isolatededges		o				o			o
isolates		o	o			o			o
istar		o	o				o		
kstar		o				o	o		
localtriangle		o	o			o		o	
m2star		o	o						
meandeg		o	o	o		o			
mm	o	o	o	o		o	o		o
mutual	o	o	o						o
nearsimmelian		o	o					o	
nodecov	o	o	o	o	o	o			o
nodecovar	o		o						
nodefactor	o	o	o	o		o	o		o
nodeicov	o	o	o		o				o
nodeicovar	o		o						
nodeifactor	o	o	o	o		o			o
nodematch	o	o	o	o		o	o		o
nodemix	o	o	o	o		o	o		o
nodeocov	o	o	o	o	o				
nodeocovar	o		o						
nodeofactor	o	o	o	o		o			
nsp		o	o			o			
odegrange		o	o				o		
odegree		o	o				o		o
odegree1.5		o	o						
opentriad		o				o		o	
ostar		o	o				o		
receiver	o	o	o	o					
sender	o	o	o	o					
simmelian		o	o					o	
simmelianties		o	o					o	
smalldiff		o	o	o	o	o			
smallerthan	o		o	o		o			
sociality	o	o		o		o	o		
sum	o		o			o			

threetrail		o	o	o		o		
transitive		o	o			o		
transitiveties		o	o	o	o	o		
transitiveweights	o		o	o		o		o
triadcensus		o	o	o		o		
triangle		o	o	o	o	o	o	
tripercent		o		o	o	o		
ttriple		o	o		o	o		
twopath		o	o	o				

Terms by keywords:

operator B, Curve, Exp, F, Label, Log, NodematchFilter, Offset, Prod, S, Sum, Symmetrize

valued B, Curve, Exp, Label, Log, Prod, Sum, absdiff, absdiffcat, atleast, atmost, b1cov, b1factor, b1sociality, b2cov, b2factor, b2sociality, cyclicalties, cyclicalweights, diff, edgecov, edges, equalto, greaterthan, ininterval, mm, mutual, nodecov, nodecovar, nodefactor, nodeicov, nodeicovar, nodeifactor, nodematch, nodemix, nodeocov, nodeocovar, nodeofactor, receiver, sender, smallerthan, sociality, sum, transitiveweights

binary Curve, Exp, F, Label, Log, NodematchFilter, Offset, Prod, S, Sum, Symmetrize, absdiff, absdiffcat, altkstar, asymmetric, attrcov, b1concurrent, b1cov, b1degrange, b1degree, b1dsp, b1factor, b1mindegree, b1nodematch, b1sociality, b1star, b1starmix, b1twostar, b2concurrent, b2cov, b2degrange, b2degree, b2dsp, b2factor, b2mindegree, b2nodematch, b2sociality, b2star, b2starmix, b2twostar, balance, coincidence, concurrent, concurrentties, ctriple, cycle, cyclicalties, ddsp, degcor, degcrossprod, degrange, degree, degree1.5, density, desp, dgwdsp, dgwesp, dgwnsp, diff, dnsp, dsp, dyadcov, edgecov, edges, esp, gwb1degree, gwb1dsp, gwb2degree, gwb2dsp, gwdegree, gwdsp, gwesp, gwidegree, gwnsp, gwodegree, hamming, idegrange, idegree, idegree1.5, intransitive, isolatededges, isolates, istar, kstar, localtriangle, m2star, meandeg, mm, mutual, nearsimmelian, nodecov, nodefactor, nodeicov, nodeifactor, nodematch, nodemix, nodeocov, nodeofactor, nsp, odegrange, odegree, odegree1.5, opentriad, ostar, receiver, sender, simmelian, simmelianties, smalldiff, sociality, threetrail, transitive, transitiveties, triadcensus, triangle, tripercent, ttriple, twopath

directed Symmetrize, absdiff, absdiffcat, asymmetric, atleast, atmost, attrcov, balance, ctriple, cycle, cyclicalties, cyclicalweights, ddsp, density, desp, dgwdsp, dgwesp, dgwnsp, diff, dnsp, dsp, dyadcov, edgecov, edges, equalto, esp, greaterthan, gwdsp, gwesp, gwidegree, gwnsp, gwodegree, hamming, idegrange, idegree, idegree1.5, ininterval, intransitive, isolates, istar, localtriangle, m2star, meandeg, mm, mutual, nearsimmelian, nodecov, nodecovar, nodefactor, nodeicov, nodeicovar, nodeifactor, nodematch, nodemix, nodeocov, nodeocovar, nodeofactor, nsp, odegrange, odegree, odegree1.5, ostar, receiver, sender, simmelian, simmelianties, smalldiff, smallerthan, sum, threetrail, transitive, transitiveties, transitiveweights, triadcensus, triangle, ttriple, twopath

dyad-independent absdiff, absdiffcat, asymmetric, atleast, atmost, attrcov, b1cov, b1factor, b1nodematch, b1sociality, b2cov, b2factor, b2nodematch, b2sociality, density, diff, dyadcov, edgecov, edges, equalto, greaterthan, hamming, ininterval, meandeg, mm, nodecov, nodefactor, nodeifactor, nodematch, nodemix, nodeocov, nodeofactor, receiver, sender, smalldiff, smallerthan, sociality

quantitative nodal attribute absdiff, b1cov, b2cov, diff, nodecov, nodeicov, nodeocov, smalldiff

undirected absdiff, absdiffcat, altkstar, atleast, atmost, attrcov, b1concurrent, b1cov, b1degrange, b1degree, b1dsp, b1factor, b1mindegree, b1nodematch, b1sociality, b1star, b1starmix, b1twostar,

b2concurrent, b2cov, b2degrange, b2degree, b2dsp, b2factor, b2mindegree, b2nodematch, b2sociality, b2star, b2starmix, b2twostar, balance, coincidence, concurrent, concurrentties, cycle, cyclicalties, cyclicalweights, degcor, degcrossprod, degrange, degree, degree1.5, density, diff, dsp, dyadcov, edgecov, edges, equalto, esp, greaterthan, gwb1degree, gwb1dsp, gwb2degree, gwb2dsp, gwdegree, gwdsp, gwesp, gwnsp, hamming, ininterval, isolatededges, isolates, kstar, localtriangle, meandeg, mm, nodecov, nodefactor, nodematch, nodemix, nsp, opentriad, smalldiff, smallerthan, sociality, sum, threetrail, transitiveties, transitiveweights, triadcensus, triangle, tripercent, twopath

categorical nodal attribute absdiffcat, altkstar, b1concurrent, b1degree, b1factor, b1nodematch, b1star, b1starmix, b1twostar, b2degree, b2factor, b2nodematch, b2star, b2starmix, b2twostar, concurrent, concurrentties, ctriple, degrange, degree, dyadcov, idegrange, idegree, istar, kstar, mm, nodefactor, nodeifactor, nodematch, nodemix, nodeofactor, odegrange, odegree, ostar, sociality, transitiveties, triangle, tripercent, ttriple

curved altkstar, gwb1degree, gwb1dsp, gwb2degree, gwb2dsp, gwdegree, gwdsp, gwesp, gwidegree, gwnsp, gwodegree

triad-related asymmetric, balance, ctriple, intransitive, localtriangle, nearsimmelian, opentriad, simmelian, simmelianties, threetrail, transitive, transitiveties, transitiveweights, triadcensus, triangle, tripercent, ttriple

bipartite b1concurrent, b1cov, b1degrange, b1degree, b1dsp, b1factor, b1mindegree, b1nodematch, b1sociality, b1star, b1starmix, b1twostar, b2concurrent, b2cov, b2degrange, b2degree, b2dsp, b2factor, b2mindegree, b2nodematch, b2sociality, b2star, b2starmix, b2twostar, coincidence, diff, gwb1degree, gwb1dsp, gwb2degree, gwb2dsp, isolatededges

frequently-used b1cov, b1degree, b1factor, b1nodematch, b2concurrent, b2cov, b2degree, b2factor, b2nodematch, degree, diff, edgecov, gwdegree, gwesp, idegree, isolates, mm, mutual, nodecov, nodefactor, nodeicov, nodeifactor, nodematch, nodemix, odegree, triangle

nonnegative cyclicalweights, transitiveweights

References

- Krivitsky P. N., Hunter D. R., Morris M., Klumb C. (2021). "ergm 4.0: New features and improvements." arXiv:2106.04997. https://arxiv.org/abs/2106.04997

- Bomiriya, R. P, Bansal, S., and Hunter, D. R. (2014). Modeling Homophily in ERGMs for Bipartite Networks. Submitted.

- Butts, CT. (2008). "A Relational Event Framework for Social Action." *Sociological Methodology,* 38(1).

- Davis, J.A. and Leinhardt, S. (1972). The Structure of Positive Interpersonal Relations in Small Groups. In J. Berger (Ed.), *Sociological Theories in Progress, Volume 2*, 218–251. Boston: Houghton Mifflin.

- Holland, P. W. and S. Leinhardt (1981). An exponential family of probability distributions for directed graphs. *Journal of the American Statistical Association*, 76: 33–50.

- 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.

- Hunter, D. R. (2007). Curved exponential family models for social networks. *Social Networks*, 29: 216–230.

- Krackhardt, D. and Handcock, M. S. (2007). Heider versus Simmel: Emergent Features in Dynamic Structures. *Lecture Notes in Computer Science*, 4503, 14–27.

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

- Robins, G; Pattison, P; and Wang, P. (2009). "Closure, Connectivity, and Degree Distributions: Exponential Random Graph (p*) Models for Directed Social Networks." *Social Networks,* 31:105-117.

- Snijders T. A. B., G. G. van de Bunt, and C. E. G. Steglich. Introduction to Stochastic Actor-Based Models for Network Dynamics. *Social Networks*, 2010, 32(1), 44-60. doi:10.1016/j.socnet.2009.02.004

- Morris M, Handcock MS, and Hunter DR. Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 2008, 24(4), 1-24. doi:10.18637/jss.v024.i04

- Snijders, T. A. B., P. E. Pattison, G. L. Robins, and M. S. Handcock (2006). New specifications for exponential random graph models, *Sociological Methodology*, 36(1): 99-153.

See Also

ergm package, search.ergmTerms, ergm, network, %v%, %n%

Examples

```
## Not run:
ergm(flomarriage ~ kstar(1:2) + absdiff("wealth") + triangle)

ergm(molecule ~ edges + kstar(2:3) + triangle
                        + nodematch("atomic type",diff=TRUE)
                        + triangle + absdiff("atomic type"))

## End(Not run)
```

---

ergm_MCMC_sample   *Internal Function to Sample Networks and Network Statistics*

Description

This is an internal function, not normally called directly by the user. The ergm_MCMC_sample function samples networks and network statistics using an MCMC algorithm via MCMC_wrapper and is capable of running in multiple threads using ergm_MCMC_slave.

The ergm_MCMC_slave function calls the actual C routine and does minimal preprocessing.

Usage

```
ergm_MCMC_sample(
  state,
  control,
  theta = NULL,
  verbose = FALSE,
```

```
  ...,
  eta = ergm.eta(theta, (if (is.ergm_state(state)) as.ergm_model(state) else
    as.ergm_model(state[[1]]))$etamap)
)

ergm_MCMC_slave(
  state,
  eta,
  control,
  verbose,
  ...,
  burnin = NULL,
  samplesize = NULL,
  interval = NULL
)
```

Arguments

state	an [ergm_state](#) representing the sampler state, containing information about the network, the model, the proposal, and (optionally) initial statistics, or a list thereof.
control	A list of control parameters for algorithm tuning, typically constructed with [control.ergm()](#), [control.simulate.ergm()](#), etc., which have different defaults. Their documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()](#) (StatNet ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
theta	the (possibly curved) parameters of the model.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
...	additional aruguments.
eta	the natural parameters of the model; by default constructed from theta.
burnin, samplesize, interval	
	MCMC paramters that can be used to temporarily override those in the control list.

Value

ergm_MCMC_sample returns a list containing:

stats	an [mcmc.list](#) with sampled statistics.
networks	a list of final sampled networks, one for each thread.
status	status code, propagated from ergm_MCMC_slave().
final.interval	adaptively determined MCMC interval.

sampnetworks	If `control$MCMC.save_networks` is set and is `TRUE`, a list of lists of `ergm_states` corresponding to the sampled networks.

`ergm_MCMC_slave` returns the MCMC sample as a list of the following:

s	the matrix of statistics.
state	an [ergm_state](#) object for the new network.
status	success or failure code: `0` is success, `1` for too many edges, and `2` for a Metropolis-Hastings proposal failing, `-1` for [ergm_model](#) or [ergm_proposal](#) not passed and missing from the cache.

Note

`ergm_MCMC_sample` and `ergm_MCMC_slave` replace `ergm.getMCMCsample` and `ergm.mcmcslave` respectively. They differ slightly in their argument names and in their return formats. For example, `ergm_MCMC_sample` expects `ergm_state` rather than network/model/proposal, and `theta` or `eta` rather than `eta0`; and it does not return `statsmatrix` or `newnetwork` elements. Rather, if parallel processing is not in effect, `stats` is an [mcmc.list](#) with one chain and `networks` is a list with one element.

Note that unless `stats` is a part of the `ergm_state`, the returned stats will be relative to the original network, i.e., the calling function must shift the statistics if required.

At this time, repeated calls to `ergm_MCMC_sample` will not produce the same sequence of networks as a single long call, even with the same starting seeds. This is because the network sampling algorithms rely on the internal state of the network representation in C, which may not be reconstructed exactly the same way when "resuming". This behaviour may change in the future.

Examples

```
# This example illustrates constructing "ingredients" for calling
# ergm_MCMC_sample() from calls to simulate.ergm(). One can also
# construct an ergm_state object directly from ergm_model(),
# ergm_proposal(), etc., but the approach shown here is likely to
# be the least error-prone and the most robust to future API
# changes.
#
# The regular simulate() call hierarchy is
#
# simulate_formula.network(formula) ->
#   simulate.ergm_model(ergm_model) ->
#     simulate.ergm_state_full(ergm_state)
#
# They take an argument, return.args=, that will interrupt the call
# and have it return its arguments. We can use it to obtain
# low-level inputs robustly.

data(florentine)
control <- control.simulate(MCMC.burnin = 2, MCMC.interval = 1)


# FYI: Obtain input for simulate.ergm_model():
```

```
sim.mod <- simulate(flomarriage~absdiff("wealth"), constraints=~edges,
                    coef = NULL, nsim=3, control=control,
                    return.args="ergm_model")
names(sim.mod)
str(sim.mod$object,1) # ergm_model

# Obtain input for simulate.ergm_state_full():
sim.state <- simulate(flomarriage~absdiff("wealth"), constraints=~edges,
                      coef = NULL, nsim=3, control=control,
                      return.args="ergm_state")
names(sim.state)
str(sim.state$object, 1) # ergm_state

# This control parameter would be set by nsim in the regular
# simulate() call:
control$MCMC.samplesize <- 3

# Capture intermediate networks; can also be left NULL for just the
# statistics:
control$MCMC.save_networks <- TRUE

# Simulate starting from this state:
out <- ergm_MCMC_sample(sim.state$object, control, theta = -1, verbose=6)
names(out)
out$stats # Sampled statistics
str(out$networks, 1) # Updated ergm_state (one per thread)
# List (an element per thread) of lists of captured ergm_states,
# one for each sampled network:
str(out$sampnetworks, 2)
lapply(out$sampnetworks[[1]], as.network) # Converted to networks.

# One more, picking up where the previous sampler left off, but see Note:
control$MCMC.samplesize <- 1
str(ergm_MCMC_sample(out$networks, control, theta = -1, verbose=6), 2)
```

---

ergm_plot.mcmc.list *Plot MCMC list using* lattice *package graphics*

Description

Plot MCMC list using `lattice` package graphics

Usage

```
ergm_plot.mcmc.list(x, main = NULL, vars.per.page = 3, ...)
```

Arguments

x	an `mcmc.list` object containing the mcmc diagnostic samples.
main	character, main plot heading title.
vars.per.page	Number of rows (one variable per row) per plotting page. Ignored if `latticeExtra` package is not installed.
...	additional arguments, currently unused.

Note

This is not a method at this time.

---

ergm_state_cache   *A rudimentary cache for large objects*

Description

This cache is intended to store large, infrequently changing data structures such as `ergm_model`s and `ergm_proposal`s on worker nodes.

Usage

```
ergm_state_cache(
  comm = c("pass", "all", "clear", "insert", "get", "check", "list"),
  hash,
  object
)
```

Arguments

comm	a character string giving the desired function; see the default argument above for permitted values and Details for meanings; partial matching is supported.
hash	a character string, typically a `digest()` of the object.
object	the object to be stored.
	Supported tasks are, respectively, to do nothing (the default), return all entries (mainly useful for testing), clear the cache, insert into cache, retrieve an object by hash, check if a hash is present, or list hashes defined.
	Deleting an entry can be accomplished by inserting a `NULL` for that hash.
	Cache is limited to a hard-coded size (currently 4). This should accommodate an `ergm_model` and an `ergm_proposal` for unconstrained and constrained MCMC. When additional objects are stored, the oldest object is purged and garbage-collected.

Note

If called via, say, clusterMap(cl, ergm_state_cache, ...) the function will not accomplish anything. This is because `parallel` package will serialise the ergm_state_cache() function object, send it to the remote node, evaluate it there, and fetch the return value. This will leave the environment of the worker's ergm_state_cache() unchanged. To actually evaluate it on the worker nodes, it is recommended to wrap it in an empty function whose environment is set to globalenv(). See Examples below.

Examples

```
## Not run:
# Wrap ergm_state_cache() and call it explicitly from ergm:
call_ergm_state_cache <- function(...) ergm::ergm_state_cache(...)

# Reset the function's environment so that it does not get sent to
# worker nodes (who have their own instance of ergm namespace
# loaded).
environment(call_ergm_state_cache) <- globalenv()

# Now, call the the wrapper function, with ... below replaced by
# lists of desired arguments.
clusterMap(cl, call_ergm_state_cache, ...)

## End(Not run)
```

---

ergm_symmetrize   *Return a symmetrized version of a binary network*

Description

Return a symmetrized version of a binary network

Usage

```
ergm_symmetrize(x, rule = c("weak", "strong", "upper", "lower"), ...)

## Default S3 method:
ergm_symmetrize(x, rule = c("weak", "strong", "upper", "lower"), ...)

## S3 method for class 'network'
ergm_symmetrize(x, rule = c("weak", "strong", "upper", "lower"), ...)
```

Arguments

x	an object representing a network.
rule	a string specifying how the network is to be symmetrized; see sna::symmetrize() for details; for the network method, it can also be a function or a list; see Details.
...	additional arguments to sna::symmetrize().

Details

The `network` method requires more flexibility, in order to specify how the edge attributes are handled. Therefore, `rule` can be one of the following types:

a character vector The string is interpreted as in `sna::symmetrize()`. For edge attributes, `"weak"` takes the maximum value and `"strong"` takes the minimum value" for ordered attributes, and drops the unordered.

a function The function is evaluated on a `data.frame` constructed by joining (via `merge()`) the edge `tibble` with all attributes and NA indicators with itself reversing tail and head columns, and appending original columns with `".th"` and the reversed columns with `".ht"`. It is then evaluated for each attribute in turn, given two arguments: the data frame and the name of the attribute.

a list The list must have exactly one unnamed element, and the remaining elements must be named with the names of edge attributes. The elements of the list are interpreted as above, allowing each edge attribute to be handled differently. Unnamed arguments are dropped.

Methods (by class)

- `default`: The default method, passing the input on to `sna::symmetrize()`.
- `network`: A method for `network` objects, which preserves network and vertex attributes, and handles edge attributes.

Note

This was originally exported as a generic to overwrite `sna::symmetrize()`. By developer's request, it has been renamed; eventually, `sna` or `network` packages will export the generic instead.

Examples

```
data(sampson)
samplike[1,2] <- NA
samplike[4.1] <- NA
sm <- as.matrix(samplike)

tst <- function(x,y){
  mapply(identical, x, y)
}

stopifnot(all(tst(as.logical(as.matrix(ergm_symmetrize(samplike, "weak"))), sm | t(sm))),
        all(tst(as.logical(as.matrix(ergm_symmetrize(samplike, "strong"))), sm & t(sm))),
          all(tst(c(as.matrix(ergm_symmetrize(samplike, "upper"))),
                  sm[cbind(c(pmin(row(sm),col(sm))),c(pmax(row(sm),col(sm))))])),
          all(tst(c(as.matrix(ergm_symmetrize(samplike, "lower"))),
                  sm[cbind(c(pmax(row(sm),col(sm))),c(pmin(row(sm),col(sm))))]))))
```

---

esp-ergmTerm   *Edgewise shared partners*

Description

This is just like the dsp term, except this term adds one network statistic to the model for each element in d where the i th such statistic equals the number of edges (rather than dyads) in the network with exactly d[i] shared partners. This term can be used with directed and undirected networks.

Usage

```
# binary: esp(d)
```

Arguments

d   a vector of distinct integers

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) edge i -> j in a directed graph, the number of shared partners counted by esp is the number of nodes k that have edges i -> k -> j. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see desp.

See Also

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

Exp-ergmTerm   *Exponentiate a network's statistic*

Description

Evaluate the terms specified in formula and exponentiates them with base e .

Usage

```
# binary: Exp(formula)
```

```
# valued: Exp(formula)
```

Arguments

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

See Also

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

F-ergmTerm   *Filtering on arbitrary one-term model*

Description

Evaluates the given formula on a network constructed by taking y and removing any edges for which $f_{i,j}(y_{i,j}) = 0$.

Usage

```
# binary: F(formula, filter)
```

Arguments

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

filter   must contain one binary ergm term, with the following properties:

- dyadic independence;
- dyadwise contribution of 0 for a 0-valued dyad.

Formally, this means that it is expressable as

$$g(y) = \sum_{i,j} f_{i,j}(y_{i,j}),$$

where for all i, j, and y, $f_{i,j}(y_{i,j})$ for which $f_{i,j}(0) = 0$. For convenience, the term in specified can be a part of a simple logical or comparison operation: (e.g., ~!nodematch("A") or ~abs("X")>3), which filters on $f_{i,j}(y_{i,j}) \bigcirc 0$ instead.

See Also

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

---

faux.desert.high *Faux desert High School as a network object*

Description

This data set represents a simulation of a directed in-school friendship network. The network is named faux.desert.high.

Usage

```
data(faux.desert.high)
```

Format

faux.desert.high is a [network](#) object with 107 vertices (students, in this case) and 439 directed edges (friendship nominations). To obtain additional summary information about it, type summary(faux.desert.high).

The vertex attributes are Grade, Sex, and Race. The Grade attribute has values 7 through 12, indicating each student's grade in school. The Race attribute is based on the answers to two questions, one on Hispanic identity and one on race, and takes six possible values: White (non-Hisp.), Black (non-Hisp.), Hispanic, Asian (non-Hisp.), Native American, and Other (non-Hisp.)

Licenses and Citation

If the source of the data set does not specified otherwise, this data set is protected by the Creative Commons License https://creativecommons.org/licenses/by-nc-nd/2.5/.

When publishing results obtained using this data set, the original authors (Resnick et al, 1997) should be cited. In addition this package should be cited as:

Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2003 *statnet: Software tools for the Statistical Modeling of Network Data* https://statnet.org.

Source

The data set is simulation based upon an ergm model fit to data from one school community from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

The school in question (a single school with 7th through 12th grades) was selected from the Add Health "structure files." Documentation on these files can be found here: https://addhealth.cpc.unc.edu/documentation/codebooks/.

The stucture file contains directed out-ties representing each instance of a student who named another student as a friend. Students could nominate up to 5 male and 5 female friends. Note that registered students who did not take the AddHealth survey or who were not listed by name on the schools' student roster are not included in the stucture files. In addition, we removed any students with missing values for race, grade or sex.

The following [ergm](#) model was fit to the original data (with code updated for modern syntax):

```
 desert.fit <- ergm(original.net ~ edges + mutual +
absdiff("grade") + nodefactor("race", base=5) + nodefactor("grade", base=3)
+ nodefactor("sex") + nodematch("race", diff = TRUE) + nodematch("grade",
diff = TRUE) + nodematch("sex", diff = FALSE) + idegree(0:1) + odegree(0:1)
+ gwesp(0.1,fixed=T), constraints = ~bd(maxout=10), control =
control.ergm(MCMLE.steplength = .25, MCMC.burnin = 100000, MCMC.interval =
10000, MCMC.samplesize = 2500, MCMLE.maxit = 100), verbose=T)
```

Then the faux.desert.high dataset was created by simulating a single network from the above model fit:

```
 faux.desert.high <- simulate(desert.fit, nsim=1,
                 control=snctrl(MCMC.burnin=1e+8),
                 constraints = ~edges)
```

References

Resnick M.D., Bearman, P.S., Blum R.W. et al. (1997). *Protecting adolescents from harm. Findings from the National Longitudinal Study on Adolescent Health*, *Journal of the American Medical Association*, 278: 823-32.

See Also

[network](#), [plot.network](#), [ergm](#), [faux.desert.high](#), [faux.mesa.high](#), [faux.magnolia.high](#)

---

faux.dixon.high   *Faux dixon High School as a network object*

Description

This data set represents a simulation of a directed in-school friendship network. The network is named faux.dixon.high.

Usage

```
data(faux.dixon.high)
```

Format

faux.dixon.high is a [network](#) object with 248 vertices (students, in this case) and 1197 directed edges (friendship nominations). To obtain additional summary information about it, type summary(faux.dixon.high).

The vertex attributes are Grade, Sex, and Race. The Grade attribute has values 7 through 12, indicating each student's grade in school. The Race attribute is based on the answers to two questions, one on Hispanic identity and one on race, and takes six possible values: White (non-Hisp.), Black (non-Hisp.), Hispanic, Asian (non-Hisp.), Native American, and Other (non-Hisp.)

Licenses and Citation

If the source of the data set does not specified otherwise, this data set is protected by the Creative Commons License https://creativecommons.org/licenses/by-nc-nd/2.5/.

When publishing results obtained using this data set, the original authors (Resnick et al, 1997) should be cited. In addition this package should be cited as:

Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2003 *statnet: Software tools for the Statistical Modeling of Network Data* https://statnet.org.

Source

The data set is simulation based upon an ergm model fit to data from one school community from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

The school in question (a single school with 7th through 12th grades) was selected from the Add Health "structure files." Documentation on these files can be found here: https://addhealth.cpc.unc.edu/documentation/codebooks/.

The stucture file contains directed out-ties representing each instance of a student who named another student as a friend. Students could nominate up to 5 male and 5 female friends. Note that registered students who did not take the AddHealth survey or who were not listed by name on the schools' student roster are not included in the stucture files. In addition, we removed any students with missing values for race, grade or sex.

The following ergm model was fit to the original data (with code updated for modern syntax):

```
 dixon.fit <- ergm(original.net ~ edges + mutual +
absdiff("grade") + nodefactor("race", base=5) + nodefactor("grade", base=3)
+ nodefactor("sex") + nodematch("race", diff = TRUE) + nodematch("grade",
diff = TRUE) + nodematch("sex", diff = FALSE) + idegree(0:1) + odegree(0:1)
+ gwesp(0.1,fixed=T), constraints = ~bd(maxout=10), control =
control.ergm(MCMLE.steplength = .25, MCMC.burnin = 100000, MCMC.interval =
10000, MCMC.samplesize = 2500, MCMLE.maxit = 100), verbose=T)
```

Then the faux.dixon.high dataset was created by simulating a single network from the above model fit:

```
 faux.dixon.high <- simulate(dixon.fit, nsim=1, burnin=1e+8,
constraint = "edges")
```

References

Resnick M.D., Bearman, P.S., Blum R.W. et al. (1997). *Protecting adolescents from harm. Findings from the National Longitudinal Study on Adolescent Health*, *Journal of the American Medical Association*, 278: 823-32.

See Also

network, plot.network, ergm, faux.desert.high, faux.mesa.high, faux.magnolia.high

---

faux.magnolia.high   *Goodreau's Faux Magnolia High School as a network object*

Description

This data set represents a simulation of an in-school friendship network. The network is named faux.magnolia.high because the school commnunities on which it is based are large and located in the southern US.

Usage

```
data(faux.magnolia.high)
```

Format

faux.magnolia.high is a [network] object with 1461 vertices (students, in this case) and 974 undirected edges (mutual friendships). To obtain additional summary information about it, type summary(faux.magnolia.high).

The vertex attributes are Grade, Sex, and Race. The Grade attribute has values 7 through 12, indicating each student's grade in school. The Race attribute is based on the answers to two questions, one on Hispanic identity and one on race, and takes six possible values: White (non-Hisp.), Black (non-Hisp.), Hispanic, Asian (non-Hisp.), Native American, and Other (non-Hisp.)

Licenses and Citation

If the source of the data set does not specified otherwise, this data set is protected by the Creative Commons License <https://creativecommons.org/licenses/by-nc-nd/2.5/>.

When publishing results obtained using this data set, the original authors (Resnick et al, 1997) should be cited. In addition this package should be cited as:

Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2003 *statnet: Software tools for the Statistical Modeling of Network Data* <https://statnet.org>.

Source

The data set is based upon a model fit to data from two school communities from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

The two schools in question (a junior and senior high school in the same community) were combined into a single network dataset. Students who did not take the AddHealth survey or who were not listed on the schools' student rosters were eliminated, then an undirected link was established between any two individuals who both named each other as a friend. All missing race, grade, and sex values were replaced by a random draw with weights determined by the size of the attribute classes in the school.

The following [ergm] model was fit to the original data:

```
 magnolia.fit <- ergm (magnolia ~ edges +
nodematch("Grade",diff=T) + nodematch("Race",diff=T) +
nodematch("Sex",diff=F) + absdiff("Grade") + gwesp(0.25,fixed=T),
control=control.ergm(MCMC.burnin=10000, MCMC.interval=1000, MCMLE.maxit=25,
                     MCMC.samplesize=2500, MCMLE.steplength=0.25))
```

Then the faux.magnolia.high dataset was created by simulating a single network from the above model fit:

```
 faux.magnolia.high <- simulate (magnolia.fit, nsim=1,
                 control = snctrl(MCMC.burnin=100000000), constraints = ~edges)
```

References

Resnick M.D., Bearman, P.S., Blum R.W. et al. (1997). *Protecting adolescents from harm. Findings from the National Longitudinal Study on Adolescent Health*, *Journal of the American Medical Association*, 278: 823-32.

See Also

[network](), [plot.network](), [ergm](), [faux.mesa.high]()

---

faux.mesa.high *Goodreau's Faux Mesa High School as a network object*

Description

This data set (formerly called "fauxhigh") represents a simulation of an in-school friendship network. The network is named faux.mesa.high because the school commnunity on which it is based is in the rural western US, with a student body that is largely Hispanic and Native American.

Usage

```
data(faux.mesa.high)
```

Format

faux.mesa.high is a [network]() object with 205 vertices (students, in this case) and 203 undirected edges (mutual friendships). To obtain additional summary information about it, type summary(faux.mesa.high).

The vertex attributes are Grade, Sex, and Race. The Grade attribute has values 7 through 12, indicating each student's grade in school. The Race attribute is based on the answers to two questions, one on Hispanic identity and one on race, and takes six possible values: White (non-Hisp.), Black (non-Hisp.), Hispanic, Asian (non-Hisp.), Native American, and Other (non-Hisp.)

Licenses and Citation

If the source of the data set does not specified otherwise, this data set is protected by the Creative Commons License <https://creativecommons.org/licenses/by-nc-nd/2.5/>.

When publishing results obtained using this data set, the original authors (Resnick et al, 1997) should be cited. In addition this package should be cited as:

Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2003 *statnet: Software tools for the Statistical Modeling of Network Data* <https://statnet.org>.

Source

The data set is based upon a model fit to data from one school community from the AddHealth Study, Wave I (Resnick et al., 1997). It was constructed as follows:

A vector representing the sex of each student in the school was randomly re-ordered. The same was done with the students' response to questions on race and grade. These three attribute vectors were permuted independently. Missing values for each were randomly assigned with weights determined by the size of the attribute classes in the school.

The following ergm formula was used to fit a model to the original data:

```
 ~ edges + nodefactor("Grade") + nodefactor("Race") +
nodefactor("Sex") + nodematch("Grade",diff=TRUE) +
nodematch("Race",diff=TRUE) + nodematch("Sex",diff=FALSE) +
gwdegree(1.0,fixed=TRUE) + gwesp(1.0,fixed=TRUE) + gwdsp(1.0,fixed=TRUE)
```

The resulting model fit was then applied to a network with actors possessing the permuted attributes and with the same number of edges as in the original data.

The processes for handling missing data and defining the race attribute are described in Hunter, Goodreau \& Handcock (2008).

References

Hunter D.R., Goodreau S.M. and Handcock M.S. (2008). *Goodness of Fit of Social Network Models*, *Journal of the American Statistical Association*.

Resnick M.D., Bearman, P.S., Blum R.W. et al. (1997). *Protecting adolescents from harm. Findings from the National Longitudinal Study on Adolescent Health*, *Journal of the American Medical Association*, 278: 823-32.

See Also

network, plot.network, ergm, faux.magnolia.high

---

fix.curved *Convert a curved ERGM into a corresponding "fixed" ERGM.*

Description

The generic `fix.curved` converts an [ergm](#) object or formula of a model with curved terms to the variant in which the curved parameters are fixed. Note that each term has to be treated as a special case.

Usage

```
fix.curved(object, ...)

## S3 method for class 'ergm'
fix.curved(object, ...)

## S3 method for class 'formula'
fix.curved(object, theta, ...)
```

Arguments

`object`	An [ergm](#) object or an ERGM formula. The curved terms of the given formula (or the formula used in the fit) must have all of their arguments passed by name.
`...`	Unused at this time.
`theta`	Curved model parameter configuration.

Details

Some ERGM terms such as [gwesp](#) and [gwdegree](#) have two forms: a curved form, for which their decay or similar parameters are to be estimated, and whose canonical statistics is a vector of the term's components ([esp(1)](#), [esp(2)](#), ... and [degree(1)](#), [degree(2)](#), ..., respectively) and a "fixed" form where the decay or similar parameters are fixed, and whose canonical statistic is just the term itself. It is often desirable to fit a model estimating the curved parameters but simulate the "fixed" statistic.

This function thus takes in a fit or a formula and performs this mapping, returning a "fixed" model and parameter specification. It only works for curved ERGM terms included with the [ergm](#) package. It does not work with curved terms not included in ergm.

Value

A list with the following components:

`formula`	The "fixed" formula.
`theta`	The "fixed" parameter vector.

See Also

[ergm](#), [simulate.ergm](#)

Examples

```
data(sampson)
gest<-ergm(samplike~edges+gwesp(),
           control=control.ergm(MCMLE.maxit=2))
summary(gest)
# A statistic for esp(1),...,esp(16)
simulate(gest,output="stats")

tmp<-fix.curved(gest)
tmp
# A gwesp() statistic only
simulate(tmp$formula, coef=tmp$theta, output="stats")
```

---

fixallbut-ergmConstraint

Preserve the dyad status in all but the given edges

Description

Preserve the dyad status in all but `free.dyads`.

Usage

```
# fixallbut(free.dyads)
```

Arguments

free.dyads   edgelist or network. Networks will be converted to the corresponding edgelist.

See Also

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

---

fixedas-ergmConstraint

Preserve and preclude edges

Description

Preserve the edges in 'present' and preclude the edges in 'absent'.

Usage

```
# fixedas(present, absent)
```

Arguments

`present, absent`

                edgelist or network

See Also

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

---

florentine	*Florentine Family Marriage and Business Ties Data as a "network" object*

Description

This is a data set of marriage and business ties among Renaissance Florentine families. The data is originally from Padgett (1994) via UCINET and stored as a [network](#) object.

Usage

```
data(florentine)
```

Details

Breiger \& Pattison (1986), in their discussion of local role analysis, use a subset of data on the social relations among Renaissance Florentine families (person aggregates) collected by John Padgett from historical documents. The two relations are business ties (`flobusiness` - specifically, recorded financial ties such as loans, credits and joint partnerships) and marriage alliances (`flomarriage`).

As Breiger \& Pattison point out, the original data are symmetrically coded. This is acceptable perhaps for marital ties, but is unfortunate for the financial ties (which are almost certainly directed). To remedy this, the financial ties can be recoded as directed relations using some external measure of power - for instance, a measure of wealth. Both graphs provide vertex information on (1) `wealth` each family's net wealth in 1427 (in thousands of lira); (2) `priorates` the number of priorates (seats on the civic council) held between 1282- 1344; and (3) `totalties` the total number of business or marriage ties in the total dataset of 116 families (see Breiger \& Pattison (1986), p 239).

Substantively, the data include families who were locked in a struggle for political control of the city of Florence around 1430. Two factions were dominant in this struggle: one revolved around the infamous Medicis (9), the other around the powerful Strozzis (15).

Source

Padgett, John F. 1994. Marriage and Elite Structure in Renaissance Florence, 1282-1500. Paper delivered to the Social Science History Association.

References

Wasserman, S. and Faust, K. (1994) *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge, England.

Breiger R. and Pattison P. (1986). *Cumulated social roles: The duality of persons and their algebras*, Social Networks, 8, 215-256.

See Also

flo, network, plot.network, ergm

---

g4   *Goodreau's four node network as a "network" object*

Description

This is an example thought of by Steve Goodreau. It is a directed network of four nodes and five ties stored as a network object.

Usage

```
data(g4)
```

Details

It is interesting because the maximum likelihood estimator of the model with out degree 3 in it exists, but the maximum psuedolikelihood estimator does not.

Source

Steve Goodreau

See Also

florentine, network, plot.network, ergm

Examples

```
data(g4)
summary(ergm(g4 ~ odegree(3), estimate="MPLE"))
summary(ergm(g4 ~ odegree(3), control=control.ergm(init=0)))
```

geweke.diag.mv	*Multivariate version of* coda*'s* `coda::geweke.diag()`.

Description

Rather than comparing each mean independently, compares them jointly. Note that it returns an `htest` object, not a `geweke.diag` object.

Usage

```
geweke.diag.mv(x, frac1 = 0.1, frac2 = 0.5, split.mcmc.list = FALSE, ...)
```

Arguments

x	an `mcmc`, `mcmc.list`, or just a matrix with observations in rows and variables in columns.
frac1, frac2	the fraction at the start and, respectively, at the end of the sample to compare.
split.mcmc.list	
	when given an `mcmc.list`, whether to test each chain individually.
...	additional arguments, passed on to `approx.hotelling.diff.test()`, which passes them to `spectrum0.mvar()`, etc.; in particular, `order.max=` can be used to limit the order of the AR model used to estimate the effective sample size.

Value

An object of class `htest`, inheriting from that returned by `approx.hotelling.diff.test()`, but with p-value considered to be 0 on insufficient sample size.

Note

If `approx.hotelling.diff.test()` returns an error, then assume that burn-in is insufficient.

See Also

`coda::geweke.diag()`, `approx.hotelling.diff.test()`

| gof | *Conduct Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model* |

Description

[gof](#) calculates p-values for geodesic distance, degree, and reachability summaries to diagnose the goodness-of-fit of exponential family random graph models. See [ergm](#) for more information on these models.

Usage

```
gof(object, ...)

## S3 method for class 'ergm'
gof(
  object,
  ...,
  coef = coefficients(object),
  GOF = NULL,
  constraints = object$constraints,
  control = control.gof.ergm(),
  verbose = FALSE
)

## S3 method for class 'formula'
gof(
  object,
  ...,
  coef = NULL,
  GOF = NULL,
  constraints = ~.,
  basis = eval_lhs.formula(object),
  control = NULL,
  unconditional = TRUE,
  verbose = FALSE
)

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

## S3 method for class 'gof'
plot(
  x,
  ...,
  cex.axis = 0.7,
  plotlogodds = FALSE,
```

```
  main = "Goodness-of-fit diagnostics",
  normalize.reachability = FALSE,
  verbose = FALSE
)
```

Arguments

object	Either a formula or an [ergm](#) object. See documentation for [ergm](#).
...	Additional arguments, to be passed to lower-level functions.
coef	When given either a formula or an object of class ergm, coef are the parameters from which the sample is drawn. By default set to a vector of 0.
GOF	formula; an formula object, of the form ~ <model terms> specifying the statistics to use to diagnosis the goodness-of-fit of the model. They do not need to be in the model formula specified in formula, and typically are not. Currently supported terms are the degree distribution ("degree" for undirected graphs, "idegree" and/or "odegree" for directed graphs, and "b1degree" and "b2degree" for bipartite undirected graphs), geodesic distances ("distance"), shared partner distributions ("espartners" and "dspartners"), the triad census ("triadcensus"), and the terms of the original model ("model"). The default formula for undirected networks is ~ degree + espartners + distance + model, and the default formula for directed networks is ~ idegree + odegree + espartners + distance + model. By default a "model" term is added to the formula. It is a very useful overall validity check and a reminder of the statistical variation in the estimates of the mean value parameters. To omit the "model" term, add "- model" to the formula.
constraints	A one-sided formula specifying one or more constraints on the support of the distribution of the networks being modeled. See the help for similarly-named argument in [ergm](#) for more information. For gof.formula, defaults to unconstrained. For gof.ergm, defaults to the constraints with which object was fitted.
control	A list of control parameters for algorithm tuning, typically constructed with [control.gof.formula()](#) or [control.gof.ergm()](#), which have different defaults. Their documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()](#) (StatNet ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
basis	a value (usually a [network](#)) to override the LHS of the formula.
unconditional	logical; if TRUE, the simulation is unconditional on the observed dyads. if not TRUE, the simulation is conditional on the observed dyads. This is primarily used internally when the network has missing data and a conditional GoF is produced.
x	an object of class gof for printing or plotting.
cex.axis	Character expansion of the axis labels relative to that for the plot.

plotlogodds   Plot the odds of a dyad having given characteristics (e.g., reachability, minimum
                   geodesic distance, shared partners). This is an alternative to the probability of a
                   dyad having the same property.

main   Title for the goodness-of-fit plots.

normalize.reachability
                   Should the reachability proportion be normalized to make it more comparable
                   with the other geodesic distance proportions.

Details

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

For GOF = ~model, the model's observed sufficient statistics are plotted as quantiles of the simulated
sample. In a good fit, the observed statistics should be near the sample median (0.5).

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

Value

`gof`, `gof.ergm`, and `gof.formula` return an object of class gof.ergm, which inherits from class
gof. This is a list of the tables of statistics and p-values. This is typically plotted using `plot.gof`.

Methods (by class)

- ergm: Perform simulation to evaluate goodness-of-fit for a specific `ergm()` fit.

- formula: Perform simulation to evaluate goodness-of-fit for a model configuration specified
  by a `formula`, coefficient, constraints, and other settings.

- gof: `print.gof` summaries the diagnostics such as the degree distribution, geodesic distances,
  shared partner distributions, and reachability for the goodness-of-fit of exponential family
  random graph models. See `ergm` for more information on these models. (summary.gof is a
  deprecated alias that may be repurposed in the future.)

- gof: `plot.gof` plots diagnostics such as the degree distribution, geodesic distances, shared
  partner distributions, and reachability for the goodness-of-fit of exponential family random
  graph models. See `ergm` for more information on these models.

Note

For gof.ergm and gof.formula, default behavior depends on the directedness of the network in-
volved; if undirected then degree, espartners, and distance are used as default properties to examine.
If the network in question is directed, "degree" in the above is replaced by idegree and odegree.

See Also

`ergm()`, `network()`, `simulate.ergm()`, `summary.ergm()`

Examples

```
data(florentine)
gest <- ergm(flomarriage ~ edges + kstar(2))
gest
summary(gest)

# test the gof.ergm function
gofflo <- gof(gest)
gofflo

# Plot all three on the same page
# with nice margins
par(mfrow=c(1,3))
par(oma=c(0.5,2,1,0.5))
plot(gofflo)

# And now the log-odds
plot(gofflo, plotlogodds=TRUE)

# Use the formula version of gof
gofflo2 <-gof(flomarriage ~ edges + kstar(2), coef=c(-1.6339, 0.0049))
plot(gofflo2)
```

---

greaterthan-ergmTerm   *Number of dyads with values strictly greater than a threshold*

Description

Adds the number of statistics equal to the length of threshold equaling to the number of dyads whose values exceed the corresponding element of threshold .

Usage

```
# valued: greaterthan(threshold=0)
```

Arguments

threshold   a vector of numerical values

See Also

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

---

gwb1degree-ergmTerm *Geometrically weighted degree distribution for the first mode in a bipartite network*

Description

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

This term can only be used with undirected bipartite networks.

Usage

```
# binary: gwb1degree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL)
```

Arguments

decay   nonnegative decay parameter for the first mode degree frequencies; required if `fixed=TRUE` and ignored with a warning otherwise.

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

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

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

levels   TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

gwb1dsp-ergmTerm   *Geometrically weighted dyadwise shared partner distribution for*
                        dyads in the first bipartition

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise
shared partner distribution for dyads in the first bipartition with decay parameter decay parameter,
which should be non-negative. This term can only be used with bipartite networks.

Usage

```
# binary: gwb1dsp(decay=0, fixed=FALSE, cutoff=30)
```

Arguments

decay   nonnegative decay parameter for the shared partner counts; required if fixed=TRUE
                and ignored with a warning otherwise.

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

cutoff   This optional argument sets the number of underlying b1dsp terms to use in
                computing the statistics when fixed=FALSE, in order to reduce the computa-
                tional burden. Its default value can also be controlled by the gw.cutoff term
                option control parameter. (See ?control.ergm.)

See Also

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

---

gwb2degree-ergmTerm   *Geometrically weighted degree distribution for the second mode in a*
                        bipartite network

Description

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

Usage

```
# binary: gwb2degree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL)
```

Arguments

decay	nonnegative decay parameter for the second mode degree frequencies; required if `fixed=TRUE` and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE` , which means the scale parameter is not fixed and thus the model is a curved exponential family.
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
cutoff	This optional argument sets the number of underlying degree terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See ?control.ergm.)
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

gwb2dsp-ergmTerm	*Geometrically weighted dyadwise shared partner distribution for dyads in the second bipartition*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution for dyads in the second bipartition with decay parameter decay parameter, which should be non-negative. This term can only be used with bipartite networks.

Usage

```
# binary: gwb2dsp(decay=0, fixed=FALSE, cutoff=30)
```

Arguments

decay	nonnegative decay parameter for the shared partner counts; required if `fixed=TRUE` and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE` , which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying b2dsp terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See ?control.ergm.)

See Also

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

---

gwdegree-ergmTerm   *Geometrically weighted degree distribution*

Description

This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the decay parameter, which should be non-negative.

Usage

```
# binary: gwdegree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL)
```

Arguments

decay	nonnegative decay parameter for the degree frequencies; required if fixed=TRUE and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is FALSE , which means the scale parameter is not fixed and thus the model is a curved exponential family.
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
cutoff	This optional argument sets the number of underlying degree terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)
levels	TODO (See Specifying Vertex attributes and Levels ([?nodal_attributes](nodal_attributes)) for details.)

See Also

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

---

gwdsp-ergmTerm   *Geometrically weighted dyadwise shared partner distribution*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter decay parameter, which should be non-negative. This term can be used with directed and undirected networks.

Usage

```
# binary: gwdsp(decay, fixed=FALSE, cutoff=30)
```

Arguments

decay   nonnegative decay parameter for the shared partner or directed 2-path count; required if `fixed=TRUE` and ignored with a warning otherwise.

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

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

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) dyad (i,j) in a directed graph, the number of shared partners counted by gwdsp is the number of nodes k that have edges i -> k -> j. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see dgwdsp.

The decay parameter was called alpha prior to ergm 3.7.

See Also

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

---

gwesp-ergmTerm   *Geometrically weighted edgewise shared partner distribution*

Description

This term is just like gwdsp except it adds a statistic equal to the geometrically weighted edgewise (not dyadwise) shared partner distribution with decay parameter decay parameter, which should be non-negative. This term can be used with directed and undirected networks.

Usage

```
# binary: gwesp(decay, fixed=FALSE, cutoff=30)
```

Arguments

decay   nonnegative decay parameter for the shared partner or directed 2-path count;
             required if fixed=TRUE and ignored with a warning otherwise.

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

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

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) edge i -> j in a directed graph, the number of shared partners counted by gwesp is the number of nodes k that have edges i -> k -> j. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see dgwesp.

The decay parameter was called alpha prior to ergm 3.7.

See Also

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

---

gwidegree-ergmTerm   *Geometrically weighted in-degree distribution*

Description

This term adds one network statistic to the model equal to the weighted in-degree distribution with
decay parameter decay parameter, which should be non-negative. This term can only be used with
directed networks.

Usage

```
# binary: gwidegree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL)
```

Arguments

decay   nonnegative decay parameter for the indegree frequencies; required if `fixed=TRUE`
               and ignored with a warning otherwise.

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

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
               for details.)

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

levels   TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for
               details.)

See Also

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

---

gwnsp-ergmTerm   *Geometrically weighted nonedgewise shared partner distribution*

Description

This term is just like gwesp and gwdsp except it adds a statistic equal to the geometrically weighted
nonedgewise (that is, over dyads that do not have an edge) shared partner distribution with weight
parameter decay parameter, which should be non-negative. This term can be used with directed and
undirected networks.

Usage

```
# binary: gwnsp(decay, fixed=FALSE, cutoff=30)
```

Arguments

decay   nonnegative decay parameter for the shared partner or directed 2-path count; required if `fixed=TRUE` and ignored with a warning otherwise.

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

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

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) non-edge `(i,j)` in a directed graph, the number of shared partners counted by `gwnsp` is the number of nodes k that have edges `i -> k -> j`. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see `dgwnsp`.

The `decay` parameter was called `alpha` prior to `ergm` 3.7.

See Also

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

---

   gwodegree-ergmTerm   *Geometrically weighted out-degree distribution*

Description

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

Usage

```
# binary: gwodegree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL)
```

Arguments

decay   nonnegative decay parameter for the outdegree frequencies; required if `fixed=TRUE` and ignored with a warning otherwise.

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

attr   quantitative attribute (see Specifying Vertex attributes and Levels (`?nodal_attributes`) for details.)

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

levels   TODO (See Specifying Vertex attributes and Levels ([`?nodal_attributes`](#)) for details.)

See Also

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

---

hamming-ergmConstraint

Preserve the hamming distance to the given network (BROKEN: Do NOT Use)

Description

This constraint is currently broken. Do not use.

Usage

```
# hamming
```

See Also

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

---

hamming-ergmTerm   *Hamming distance*

Description

This term adds one statistic to the model equal to the weighted or unweighted Hamming distance of the network from the network specified by x . Unweighted Hamming distance is defined as the total number of pairs (i, j) (ordered or unordered, depending on whether the network is directed or undirected) on which the two networks differ. If the optional argument cov is specified, then the weighted Hamming distance is computed instead, where each pair (i, j) contributes a pre-specified weight toward the distance when the two networks differ on that pair.

Usage

```
# binary: hamming(x, cov, attrname=NULL)
```

Arguments

x	defaults to be the observed network, i.e., the network on the left side of the $\sim$ in the formula that defines the ERGM.
cov	either a matrix of edgewise weights or a network
attrname	option argument that provides the name of the edge attribute to use for weight values when a network is specified in cov

See Also

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

HammingConstantEdges-ergmProposal
TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	edges hamming		0	random	c

HammingTNT-ergmProposal
                    TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	hamming sparse		0	random	c

idegrange-ergmTerm   *In-degree range*

Description

This term adds one network statistic to the model for each element of from (or to); the i th such statistic equals the number of nodes in the network of in-degree greater than or equal to from[i] but strictly less than to[i] , i.e. with in-edge count in semiopen interval [from,to) .

This term can only be used with directed networks; for undirected networks (bipartite and not) see degrange . For degrees of specific modes of bipartite networks, see b1degrange and b2degrange . For in-degrees, see idegrange .

Usage

```
# binary: idegrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
```

Arguments

from, to	vectors of distinct integers. If one of the vectors have length 1, it is recycled to the length of the other. Otherwise, it must have the same length.
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

idegree-ergmTerm   *In-degree*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of nodes in the network of in-degree d[i] , i.e. the number of nodes with exactly d[i] in-edges. This term can only be used with directed networks; for undirected networks see degree .

Usage

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

Arguments

d	a vector of distinct integers
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels ([?nodal_attributes](#)) for details.)

See Also

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

---

idegree1.5-ergmTerm   *In-degree to the 3/2 power*

Description

This term adds one network statistic to the model equaling the sum over the actors of each actor's indegree taken to the 3/2 power (or, equivalently, multiplied by its square root). This term is analogous to the term of Snijders et al. (2010), equation (12). This term can only be used with directed networks.

Usage

```
# binary: idegree1.5
```

See Also

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

---

idegreedist-ergmConstraint

Preserve the indegree distribution

Description

Preserve the indegree distribution of the given network.

Usage

```
# idegreedist
```

See Also

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

---

idegrees-ergmConstraint

Preserve indegree for directed networks

Description

For directed networks, preserve the indegree of each vertex of the given network, while allowing outdegree to vary

Usage

```
# idegrees
```

See Also

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

---

ininterval-ergmTerm   *Number of dyads whose values are in an interval*

Description

Adds one statistic equaling to the number of dyads whose values are between lower and upper .

Usage

```
# valued: ininterval(lower=-Inf, upper=+Inf, open=c(TRUE,TRUE))
```

Arguments

lower   defaults to -Inf

upper   defaults to +Inf

open   a logical vector of length 2 that controls whether the interval is open (exclusive) on the lower and on the upper end, respectively. open can also be specified as one of "[]" , "(]" , "[)" , and "()" .

See Also

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

InitErgmTerm.threepath

Three-trails

Description

For an undirected network, this term adds one statistic equal to the number of 3-trails, where a 3-trail is defined as a trail of length three that traverses three distinct edges. Note that a 3-trail need not include four distinct nodes; in particular, a triangle counts as three 3-trails. For a directed network, this term adds four statistics (or some subset of these four), one for each of the four distinct types of directed three-paths. If the nodes of the path are written from left to right such that the middle edge points to the right (R), then the four types are RRR, RRL, LRR, and LRL. That is, an RRR 3-trail is of the form $i \to j \to k \to l$, and RRL 3-trail is of the form $i \to j \to k \leftarrow l$, etc. Like in the undirected case, there is no requirement that the nodes be distinct in a directed 3-trail. However, the three edges must all be distinct. Thus, a mutual tie $i \leftrightarrow j$ does not count as a 3-trail of the form $i \to j \to i \leftarrow j$; however, in the subnetwork $i \leftrightarrow j \to k$, there are two directed 3-trails, one LRR ($k \leftarrow j \to i \leftarrow j$) and one RRR ($j \to i \to j \leftarrow k$).

Usage

```
# binary: threepath
```

```
# binary: threetrail(keep=NULL, levels=NULL)
```

Arguments

keep   deprecated

levels   specify a subset of the four statistics for directed networks. (See Specifying
                Vertex attributes and Levels ([?nodal_attributes](#)) for details.)

Note

The argument keep is retained for backwards compatibility and may be removed in a future version.
When both keep and levels are passed, levels overrides keep.

This term used to be (inaccurately) called threepath . That name has been deprecated and may be
removed in a future version.

See Also

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

---

intransitive-ergmTerm   *Intransitive triads*

Description

This term adds one statistic to the model, equal to the number of triads in the network that are intran-
sitive. The intransitive triads are those of type 111D , 201 , 111U , 021C , or 030C in the categorization
of Davis and Leinhardt (1972). For details on the 16 possible triad types, see triad.classify in
the sna package. Note the distinction from the ctriple term.

Usage

```
# binary: intransitive
```

Note

This term can only be used with directed networks.

See Also

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

---

is.curved   *Testing for curved exponential family*

Description

These functions test whether an ERGM fit, formula, or some other object represents a curved exponential family.

The method for NULL always returns FALSE by convention.

Usage

```
is.curved(object, ...)

## S3 method for class '`NULL`'
is.curved(object, ...)

## S3 method for class 'formula'
is.curved(object, response = NULL, basis = NULL, ...)

## S3 method for class 'ergm'
is.curved(object, ...)
```

Arguments

object	An [ergm](#) object or an ERGM formula.
...	Arguments passed on to lower-level functions.
response	Either a character string, a formula, or NULL (the default), to specify the response attributes and whether the ERGM is binary or valued. Interpreted as follows:

> NULL Model simple presence or absence, via a binary ERGM.
>
> **character string** The name of the edge attribute whose value is to be modeled. Type of ERGM will be determined by whether the attribute is [logical](#) (TRUE/FALSE) for binary or [numeric](#) for valued.
>
> **a formula** must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is evaluated in the formula's environment with the network's edge attributes accessible as variables. The optional NAME specifies the name of the edge attribute into which the results should be stored, with the default being a concise version of EXPR. Normally, the type of ERGM is determined by whether the result of evaluating EXPR is logical or numeric, but the optional TYPE can be used to override by specifying a scalar of the type involved (e.g., TRUE for binary and 1 for valued).

basis	See [ergm()](#).

Details

Curvature is checked by testing if all model parameters are canonical.

Value

TRUE if the object represents a curved exponential family; FALSE otherwise.

---

is.dyad.independent   *Testing for dyad-independence*

Description

These functions test whether an ERGM fit, a formula, or some other object represents a dyad-independent model.

The method for NULL always returns TRUE by convention.

Usage

```
is.dyad.independent(object, ...)

## S3 method for class '`NULL`'
is.dyad.independent(object, ...)

## S3 method for class 'formula'
is.dyad.independent(object, response = NULL, basis = NULL, ...)

## S3 method for class 'ergm_conlist'
is.dyad.independent(object, object.obs = NULL, ...)

## S3 method for class 'ergm'
is.dyad.independent(object, ...)
```

Arguments

object	The object to be tested for dyadic independence.
...	Unused at this time.
response	Either a character string, a formula, or NULL (the default), to specify the response attributes and whether the ERGM is binary or valued. Interpreted as follows:

        NULL Model simple presence or absence, via a binary ERGM.

        character string The name of the edge attribute whose value is to be modeled. Type of ERGM will be determined by whether the attribute is [logical](#) (TRUE/FALSE) for binary or [numeric](#) for valued.

        a formula must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is evaluated in the formula's environment with the network's edge attributes accessible as variables. The optional NAME specifies the name of the edge attribute into which the results should be stored, with the default being a concise version of EXPR. Normally, the type of ERGM is determined by whether the result of evaluating EXPR is logical or numeric, but the optional TYPE can be used to override by specifying a scalar of the type involved (e.g., TRUE for binary and 1 for valued).

basis	See [ergm]().
object.obs	For the [ergm_conlist]() method, the observed data constraint.

Details

Dyad independence is determined by checking if all of the constituent parts of the object (formula, ergm terms, constraints, etc.) are flagged as dyad-independent.

Value

TRUE if the model implied by the object is dyad-independent; FALSE otherwise.

---

is.valued	*Function to check whether an ERGM fit or some aspect of it is valued*

Description

Function to check whether an ERGM fit or some aspect of it is valued

Usage

```
is.valued(object, ...)

## S3 method for class 'ergm_state'
is.valued(object, ...)

## S3 method for class 'edgelist'
is.valued(object, ...)

## S3 method for class 'ergm'
is.valued(object, ...)

## S3 method for class 'network'
is.valued(object, ...)
```

Arguments

object	the object to be tested.
...	additional arguments for methods, currently unused.

Methods (by class)

- ergm_state: a method for [ergm_state]() objects.
- edgelist: a method for [edgelist]() objects.
- ergm: a method for [ergm]() objects.
- network: a method for [network]() objects.

---

isolatededges-ergmTerm

Isolated edges

Description

This term adds one statistic to the model equal to the number of isolated edges in the network, i.e., the number of edges each of whose endpoints has degree 1. This term can only be used with undirected networks.

Usage

```
# binary: isolatededges
```

See Also

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

---

isolates-ergmTerm   *Isolates*

Description

This term adds one statistic to the model equal to the number of isolates in the network. For an undirected network, an isolate is defined to be any node with degree zero. For a directed network, an isolate is any node with both in-degree and out-degree equal to zero.

Usage

```
# binary: isolates
```

See Also

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

---

istar-ergmTerm   *In-stars*

Description

This term adds one network statistic to the model for each element in k . The i th such statistic counts the number of distinct k[i] -instars in the network, where a k -instar is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $(O_j \rightarrow N)$ exist for $j = 1, \ldots, k$. If attr is specified then the count is over the number of k -instars where all nodes have the same value of the attribute. This term can only be used for directed networks; for undirected networks see kstar . Note that istar(1) is equal to both ostar(1) and edges .

Usage

```
# binary: istar(k, attr=NULL, levels=NULL)
```

Arguments

k	a vector of distinct integers
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

kapferer   *Kapferer's tailor shop data*

Description

This well-known social network dataset, collected by Bruce Kapferer in Zambia from June 1965 to August 1965, involves interactions among workers in a tailor shop as observed by Kapferer himself.

Usage

```
data(kapferer)
```

Format

Two network objects, kapferer and kapferer2. The kapferer dataset contains only the 39 individuals who were present at both data-collection time periods. However, these data only reflect data collected during the first period. The individuals' names are included as a nodal covariate called names.

Details

An interaction is defined by Kapferer as "continuous uninterrupted social activity involving the participation of at least two persons"; only transactions that were relatively frequent are recorded. All of the interactions in this particular dataset are "sociational", as opposed to "instrumental". Kapferer explains the difference (p. 164) as follows:

"I have classed as transactions which were sociational in content those where the activity was markedly convivial such as general conversation, the sharing of gossip and the enjoyment of a drink together. Examples of instrumental transactions are the lending or giving of money, assistance at times of personal crisis and help at work."

Kapferer also observed and recorded instrumental transactions, many of which are unilateral (directed) rather than reciprocal (undirected), though those transactions are not recorded here. In addition, there was a second period of data collection, from September 1965 to January 1966, but these data are also not recorded here. All data are given in Kapferer's 1972 book on pp. 176-179.

During the first time period, there were 43 individuals working in this particular tailor shop; however, the better-known dataset includes only those 39 individuals who were present during both time collection periods. (Missing are the workers named Lenard, Peter, Lazarus, and Laurent.) Thus, we give two separate network datasets here: kapferer is the well-known 39-individual dataset, whereas kapferer2 is the full 43-individual dataset.

Source

Original source: Kapferer, Bruce (1972), Strategy and Transaction in an African Factory, Manchester University Press.

---

kstar-ergmTerm   *k-Stars*

Description

This term adds one network statistic to the model for each element in k . The i th such statistic counts the number of distinct k[i] -stars in the network, where a k -star is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \ldots, k$. If this is specified then the count is over the number of k -stars where all nodes have the same value of the attribute. This term can only be used for undirected networks; for directed networks, see istar , ostar , twopath and m2star . Note that kstar(1) is equal to edges .

Usage

```
# binary: kstar(k, attr=NULL, levels=NULL)
```

Arguments

k	a vector of distinct integers
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

Label-ergmTerm   *Modify terms' coefficient names*

Description

This operator evaluates `formula` without modification, but modifies its coefficient and/or parameter names based on `label` and `pos` .

Usage

```
# binary: Label(formula, label, pos)
```

```
# valued: Label(formula, label, pos)
```

Arguments

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

label   either a character vector specifying the label for the terms or a function through
            which term names are mapped (or a `as_mapper` -style formula).

pos   controls how `label` modifies the term names: one of `"prepend"` , `"replace"` ,
            `"append"` , or `"("` , with the latter wrapping the term names in parentheses like
            a function call with name specified by `label` .

See Also

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

---

localtriangle-ergmTerm
                    Triangles within neighborhoods

Description

This term adds one statistic to the model equal to the number of triangles in the network between nodes "close to" each other. For an undirected network, a local triangle is defined to be any set of three edges between nodal pairs $\{(i, j), (j, k), (k, i)\}$ that are in the same neighborhood. For a directed network, a triangle is defined as any set of three edges $(i \to j), (j \to k)$ and either $(k \to i)$ or $(k \leftarrow i)$ where again all nodes are within the same neighborhood.

Usage

```
# binary: localtriangle(x)
```

Arguments

x   an undirected network or an symmetric adjacency matrix that specifies whether
                               the two nodes are in the same neighborhood. Note that triangle , with or
                               without an argument, is a special case of localtriangle .

See Also

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

Log-ergmTerm   *Take a natural logarithm of a network's statistic*

Description

Evaluate the terms specified in formula and takes a natural (base e) logarithm of them. Since an
ERGM statistic must be finite, log0 specifies the value to be substituted for log(0) . The default
value seems reasonable for most purposes.

Usage

```
# binary: Log(formula, log0=-1/sqrt(.Machine$double.eps))

# valued: Log(formula, log0=-1/sqrt(.Machine$double.eps))
```

Arguments

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

log0   the value to be substituted for log(0)

See Also

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

---

logLik.ergm   *A* [logLik](#) *method for* [ergm](#) *fits.*

Description

A function to return the log-likelihood associated with an [ergm](#) fit, evaluating it if necessary. If the
log-likelihood was not computed for object, produces an error unless eval.loglik=TRUE.

Usage

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

## S3 method for class 'ergm'
deviance(object, ...)

## S3 method for class 'ergm'
AIC(object, ..., k = 2)

## S3 method for class 'ergm'
BIC(object, ...)
```

Arguments

object	An [ergm](#) fit, returned by [ergm](#).
add	Logical: If TRUE, instead of returning the log-likelihood, return object with log-likelihood value (and the null 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, typically constructed with [control.logLik.ergm()](#). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()](#) (Stat-Net ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
...	Other arguments to the likelihood functions.
k	see help for [AIC()](#).

Value

The form of the output of logLik.ergm depends on add: add=FALSE (the default), a [logLik](#) object. If add=TRUE (the default), an [ergm](#) object with the log-likelihood set.

As of version 3.1, all likelihoods for which logLikNull is not implemented are computed relative to the reference measure. (I.e., a null model, with no terms, is defined to have likelihood of 0, and all other models are defined relative to that.)

Functions

- deviance.ergm: A `deviance()` method.
- AIC.ergm: An `AIC()` method.
- BIC.ergm: A `BIC()` method.

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

Examples

```
# See help(ergm) for a description of this model. The likelihood will
# not be evaluated.
data(florentine)
## Not run:
# The default maximum number of iterations is currently 20. We'll only
# use 2 here for speed's sake.
gest <- ergm(flomarriage ~ kstar(1:2) + absdiff("wealth") + triangle, eval.loglik=FALSE)

gest <- ergm(flomarriage ~ kstar(1:2) + absdiff("wealth") + triangle, eval.loglik=FALSE,
             control=control.ergm(MCMLE.maxit=2))
# Log-likelihood is not evaluated, so no deviance, AIC, or BIC:
summary(gest)
# Evaluate the log-likelihood and attach it to the object.

# The default number of bridges is currently 20. We'll only use 3 here
# for speed's sake.
gest.logLik <- logLik(gest, add=TRUE)

gest.logLik <- logLik(gest, add=TRUE, control=control.logLik.ergm(bridge.nsteps=3))
# Deviances, AIC, and BIC are now shown:
summary(gest.logLik)
# Null model likelihood can also be evaluated, but not for all constraints:
logLikNull(gest) # == network.dyadcount(flomarriage)*log(1/2)

## End(Not run)
```

---

logLikNull   *Calculate the null model likelihood*

Description

Calculate the null model likelihood

Usage

```
logLikNull(object, ...)

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

Arguments

object	a fitted model.
...	further arguments to lower-level functions.
	logLikNull computes, when possible the log-probability of the data under the null model (reference distribution).
control	A list of control parameters for algorithm tuning, typically constructed with control.logLik.ergm(). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility snctrl() (Stat-Net ConTRoL) also provides argument completion for the available control functions and limited argument name checking.

Value

logLikNull returns an object of type logLik if it is able to compute the null model probability, and NA otherwise.

Methods (by class)

- ergm: A method for ergm fits; currently only implemented for binary ERGMs with dyad-independent sample-space constraints.

---

m2star-ergmTerm   *Mixed 2-stars, a.k.a 2-paths*

Description

This term adds one statistic to the model, equal to the number of mixed 2-stars in the network, where a mixed 2-star is a pair of distinct edges $(i{\rightarrow}j), (j{\rightarrow}k)$. A mixed 2-star is sometimes called a 2-path because it is a directed path of length 2 from i to k via j . However, in the case of a 2-path the focus is usually on the endpoints i and k , whereas for a mixed 2-star the focus is usually on the midpoint j . This term can only be used with directed networks; for undirected networks see kstar(2) . See also twopath .

Usage

```
# binary: m2star
```

See Also

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

---

mcmc.diagnostics   *Conduct MCMC diagnostics on a model fit*

Description

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

Usage

```
mcmc.diagnostics(object, ...)

## S3 method for class 'ergm'
mcmc.diagnostics(
  object,
  center = TRUE,
  esteq = TRUE,
  vars.per.page = 3,
  which = c("plots", "texts", "summary", "autocorrelation", "crosscorrelation",
    "burnin"),
  ...
)
```

Arguments

object	A model fit object to be diagnosed.
...	Additional arguments, to be passed to plotting functions.
center	Logical: If TRUE, center the samples on the observed statistics.
esteq	Logical: If TRUE, for statistics corresponding to curved ERGM terms, summarize the curved statistics by their negated estimating function values (evaluated at the MLE of any curved parameters) (i.e., $\eta'_I(\hat{\theta}) \cdot (g_I(Y) - g_I(y))$ for I being indices of the canonical parameters in question), rather than the canonical (sufficient) vectors of the curved statistics relative to the observed $(g_I(Y) - g_I(y))$.
vars.per.page	Number of rows (one variable per row) per plotting page. Ignored if latticeExtra package is not installed.
which	A character vector specifying which diagnostics to plot and/or print. Defaults to all of the below if meaningful:
	"plots" Traceplots and density plots of sample values for all statistic or estimating function elements.
	"texts" Shorthand for the following text diagnostics.
	"summary" Summary of network statistic or estimating function elements as produced by coda::summary.mcmc.list().
	"autocorrelation" Autocorrelation of each of the network statistic or estimating function elements.

> "crosscorrelation" Cross-correlations between each pair of the network statis-
> tic or estimating function elements.
>
> "burnin" Burn-in diagnostics, in particular, the Geweke test.
>
> Partial matching is supported. (E.g., which=c("auto","cross") will print au-
> tocorrelation and cross-correlations.)

Details

A pair of plots are produced for each statistic:a trace of the sampled output statistic values on the
left and density estimate for each variable in the MCMC chain on the right. Diagnostics printed to
the console include correlations and convergence diagnostics.

For [ergm()](#) specifically, recent changes in the estimation algorithm mean that these plots can no
longer be used to ensure that the mean statistics from the model match the observed network statis-
tics. For that functionality, please use the GOF command: gof(object, GOF=~model).

In fact, an ergm output 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 https://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 [ergm](#).

Methods (by class)

- ergm:

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. Spiegel-
halter 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](#), network package, coda package, [summary.ergm](#)

Examples

```
## Not run:
#
data(florentine)
#
# test the mcmc.diagnostics function
#
gest <- ergm(flomarriage ~ edges + kstar(2))
summary(gest)
```

```
#
# Plot the probabilities first
#
mcmc.diagnostics(gest)
#
# Use coda directly
#
library(coda)
#
plot(gest$sample, ask=FALSE)
#
# A full range of diagnostics is available
# using codamenu()
#

## End(Not run)
```

---

meandeg-ergmTerm   *Mean vertex degree*

Description

This term adds one network statistic to the model equal to the average degree of a node. Note that this term is a constant multiple of both edges and density .

Usage

```
# binary: meandeg
```

See Also

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

---

mm-ergmTerm   *Mixing matrix cells and margins*

Description

attrs is the rows of the mixing matrix and whose RHS gives that for its columns. A one-sided formula (e.g., ~A) is symmetrized (e.g., A~A). A two-sided formula with a dot on one side calculates the margins of the mixing matrix, analogously to nodefactor , with A~. calculating the row/sender/b1 margins and .~A calculating the column/receiver/b2 margins.

Usage

```
# binary: mm(attrs, levels=NULL, levels2=-1)

# valued: mm(attrs, levels=NULL, levels2=-1, form="sum")
```

Arguments

attrs   a two-sided formula whose LHS gives the attribute or attribute function (see
               Specifying Vertex attributes and Levels (?nodal_attributes) for details.) for
               the rows of the mixing matrix and whose RHS gives for its columns. A one-
               sided formula (e.g., ~A) is symmetrized (e.g., A~A)

levels   subset of rows and columns to be used. (See Specifying Vertex attributes and
               Levels (?nodal_attributes) for details.)

levels2   which specific cells of the matrix to include

form   character how to aggregate tie values in a valued ERGM

See Also

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

---

molecule	*Synthetic network with 20 nodes and 28 edges*

Description

This is a synthetic network of 20 nodes that is used as an example within the [ergm](#) documentation.
It has an interesting elongated shape

- reminencent of a chemical molecule. It is stored as a [network](#) object.

Usage

```
data(molecule)
```

See Also

florentine, sampson, network, plot.network, ergm

mutual-ergmTerm   *Mutuality*

Description

In binary ERGMs, equal to the number of pairs of actors i and j for which $(i{\rightarrow}j)$ and $(j{\rightarrow}i)$ both exist. For valued ERGMs, equal to $\sum_{i<j} m(y_{i,j}, y_{j,i})$, where m is determined by form argument: "min" for $\min(y_{i,j}, y_{j,i})$, "nabsdiff" for $-|y_{i,j}, y_{j,i}|$, "product" for $y_{i,j}y_{j,i}$, and "geometric" for $\sqrt{y_{i,j}}\sqrt{y_{j,i}}$. See Krivitsky (2012) for a discussion of these statistics. form="threshold" simply computes the binary mutuality after thresholding at threshold.

This term can only be used with directed networks.

Usage

```
# binary: mutual(same=NULL, by=NULL, diff=FALSE, keep=NULL, levels=NULL)

# valued: mutual(form="min",threshold=0)
```

Arguments

same   if the optional argument is passed (see Specifying Vertex attributes and Levels
                (?nodal_attributes) for details), only mutual pairs that match on the attribute
                are counted; separate counts for each unique matching value can be obtained
                by using diff=TRUE with same. Only one of same or by may be used. If both
                parameters are used, by is ignored. This paramer is affected by diff.

by   if the optional argument is passed (see Specifying Vertex attributes and Lev-
                els (?nodal_attributes) for details), then each node is counted separately for
                each mutual pair in which it occurs and the counts are tabulated by unique val-
                ues of the attribute. This means that the sum of the mutual statistics when by is
                used will equal twice the standard mutual statistic. Only one of same or by may
                be used. If both parameters are used, by is ignored. This paramer is not affected
                by diff.

keep   deprecated

levels   which statistics should be kept whenever the mutual term would ordinarily re-
                sult in multiple statistics. (See Specifying Vertex attributes and Levels (?nodal_attributes)
                for details.)

form   character how to aggregate tie values in a valued ERGM

Note

The argument keep is retained for backwards compatibility and may be removed in a future version. When both keep and levels are passed, levels overrides keep.

See Also

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

---

nearsimmelian-ergmTerm

Near simmelian triads

Description

This term adds one statistic to the model equal to the number of near Simmelian triads, as defined by Krackhardt and Handcock (2007). This is a sub-graph of size three which is exactly one tie short of being complete.

Usage

```
# binary: nearsimmelian
```

Note

This term can only be used with directed networks.

See Also

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

---

network.list	*A convenience container for a list of* [network](#) *objects, output by* [simulate.ergm](#) *among others.*

Description

A convenience container for a list of [network](#) objects, output by [simulate.ergm](#) among others.

Usage

```
network.list(object, ...)

## S3 method for class 'network.list'
print(x, stats.print = FALSE, ...)

## S3 method for class 'network.list'
summary(
  object,
  stats.print = TRUE,
  net.print = FALSE,
  net.summary = FALSE,
  ...
)
```

Arguments

`object, x`	a `list` of networks or a `network.list` object.
`...`	for `network.list`, additional attributes to be set on the network list; for others, arguments passed down to lower-level functions.
`stats.print`	Logical: If TRUE, print network statistics.
`net.print`	Logical: If TRUE, print network overviews.
`net.summary`	Logical: If TRUE, print network summaries.

Methods (by generic)

- print: A [print()](#) method for network lists.

- summary: A [summary()](#) method for network lists.

See Also

[simulate.ergm](#)

Examples

```
# Draw from a Bernoulli model with 16 nodes
# and tie probability 0.1
#
g.use <- network(16, density=0.1, directed=FALSE)
#
# Starting from this network let's draw 3 realizations
# of a model with edges and 2-star terms
#
g.sim <- simulate(~edges+kstar(2), nsim=3, coef=c(-1.8, 0.03),
                basis=g.use, control=control.simulate(
                   MCMC.burnin=100000,
                   MCMC.interval=1000))
print(g.sim)
summary(g.sim)
```

---

nodal_attributes	*Specifying nodal attributes and their levels*

Description

This document describes the ways to specify nodal attributes or functions of nodal attributes and which levels for categorical factors to include. For the helper functions to facilitate this, see [nodal_attributes-API](#).

Usage

```
LARGEST(l, a)

SMALLEST(l, a)

COLLAPSE_SMALLEST(object, n, into)
```

Arguments

`object, l, a, n, into`

> `COLLAPSE_SMALLEST`, `LARGEST`, and `SMALLEST` are technically functions but they are generally not called in a standard fashion but rather as a part of an vertex attribute specification or a level specification as described below. The above usage examples are needed to pass R's package checking without warnings; please disregard them, and refer to the sections and examples below instead.

Specifying nodal attributes

Term nodal attribute arguments, typically called `attr`, `attrs`, `by`, or `on` are interpreted as follows:

a character string Extract the vertex attribute with this name.

a character vector of length > 1 Extract the vertex attributes and paste them together, separated by dots if the term expects categorical attributes and (typically) combine into a covariate matrix if it expects quantitative attributes.

a function The function is called on the LHS network and additional arguments to [ergm_get_vattr()](), expected to return a vector or matrix of appropriate dimension. (Shorter vectors and matrix columns will be recycled as needed.)

a formula The expression on the RHS of the formula is evaluated in an environment of the vertex attributes of the network, expected to return a vector or matrix of appropriate dimension. (Shorter vectors and matrix columns will be recycled as needed.) Within this expression, the network itself accessible as either `.` or `.nw`. For example, `nodecov(~abs(Grade-mean(Grade))/network.size(.))` would return the absolute difference of each actor's "Grade" attribute from its network-wide mean, divided by the network size.

an `AsIs` **object created by** `I()` Use as is, checking only for correct length and type.

Any of these arguments may also be wrapped in or piped through `COLLAPSE_SMALLEST(attr, n, into)` or, `attr %>% COLLAPSE_SMALLEST(n, into)`, a convenience function that will transform the attribute by collapsing the smallest `n` categories into one, naming it `into`. Note that `into` must be of the same type (numeric, character, etc.) as the vertex attribute in question.

The name the nodal attribute receives in the statistic can be overridden by setting a an [attr()]-style attribute `"name"`.

Specifying categorical attribute levels and their ordering

For categorical attributes, to select which levels are of interest and their ordering, use the argument `levels`. Selection of nodes (from the appropriate vector of nodal indices) is likewise handled as the selection of levels, using the argument `nodes`. These arguments are interpreted as follows:

an expression wrapped in `I()` Use the given list of levels as is.

a numeric or logical vector Used for indexing of a list of all possible levels (typically, unique values of the attribute) in default older (typically lexicographic), i.e., `sort(unique(attr))[levels]`. In particular, `levels=TRUE` will retain all levels. Negative values exclude. Another special value is LARGEST, which will refer to the most frequent category, so, say, to set such a category as the baseline, pass `levels=-LARGEST`. In addition, `LARGEST(n)` will refer to the `n` largest categories. SMALLEST works analogously. Note that if there are ties in frequencies, they will be broken arbitrarily. To specify numeric or logical levels literally, wrap in `I()`.

`NULL` Retain all possible levels; usually equivalent to passing TRUE.

a character vector Use as is.

a function The function is called on the list of unique values of the attribute, the values of the attribute themselves, and the network itself, depending on its arity. Its return value is interpreted as above.

a formula The expression on the RHS of the formula is evaluated in an environment in which the network itself is accessible as `.nw`, the list of unique values of the attribute as `.` or as `.levels`, and the attribute vector itself as `.attr`. Its return value is interpreted as above.

a matrix For mixing effects (i.e., `level2=` arguments), a matrix can be used to select elements of the mixing matrix, either by specifying a logical (`TRUE` and `FALSE`) matrix of the same dimension as the mixing matrix to select the corresponding cells or a two-column numeric matrix indicating giving the coordinates of cells to be used.

Note that `levels`, `nodes`, and others often have a default that is sensible for the term in question.

Examples

```
library(magrittr) # for %>%

data(faux.mesa.high)

# Activity by grade with a baseline grade excluded:
summary(faux.mesa.high~nodefactor(~Grade))
# Name overrides:
summary(faux.mesa.high~nodefactor("Form"~Grade)) # Only for terms that don't use the LHS.
summary(faux.mesa.high~nodefactor(~structure(Grade,name="Form")))
# Retain all levels:
summary(faux.mesa.high~nodefactor(~Grade, levels=TRUE)) # or levels=NULL
# Use the largest grade as baseline (also Grade 7):
summary(faux.mesa.high~nodefactor(~Grade, levels=-LARGEST))
# Activity by grade with no baseline smallest two grades (11 and
# 12) collapsed into a new category, labelled 0:
table(faux.mesa.high %v% "Grade")
summary(faux.mesa.high~nodefactor((~Grade) %>% COLLAPSE_SMALLEST(2, 0),
                                   levels=TRUE))

# Mixing between lower and upper grades:
summary(faux.mesa.high~mm(~Grade>=10))
# Mixing between grades 7 and 8 only:
summary(faux.mesa.high~mm("Grade", levels=I(c(7,8))))
# or
```

```
summary(faux.mesa.high~mm("Grade", levels=1:2))
# or using levels2 (see ? mm) to filter the combinations of levels,
summary(faux.mesa.high~mm("Grade",
         levels2=~sapply(.levels,
                         function(l)
                           l[[1]]%in%c(7,8) && l[[2]]%in%c(7,8))))

# Here are some less complex ways to specify levels2. This is the
# full list of combinations of sexes in an undirected network:
summary(faux.mesa.high~mm("Sex", levels2=TRUE))
# Select only the second combination:
summary(faux.mesa.high~mm("Sex", levels2=2))
# Equivalently,
summary(faux.mesa.high~mm("Sex", levels2=-c(1,3)))
# or
summary(faux.mesa.high~mm("Sex", levels2=c(FALSE,TRUE,FALSE)))
# Select all *but* the second one:
summary(faux.mesa.high~mm("Sex", levels2=-2))
# Select via a mixing matrix: (Network is undirected and
# attributes are the same on both sides, so we can use either M or
# its transpose.)
(M <- matrix(c(FALSE,TRUE,FALSE,FALSE),2,2))
summary(faux.mesa.high~mm("Sex", levels2=M)+mm("Sex", levels2=t(M)))
# Select via an index of a cell:
idx <- cbind(1,2)
summary(faux.mesa.high~mm("Sex", levels2=idx))

# mm() term allows two-sided attribute formulas with different attributes:
summary(faux.mesa.high~mm(Grade~Race, levels2=TRUE))
# It is possible to have collapsing functions in the formula; note
# the parentheses around "~Race": this is because a formula
# operator (~) has lower precedence than pipe (|>):
summary(faux.mesa.high~mm(Grade~(~Race) %>% COLLAPSE_SMALLEST(3,"BWO"), levels2=TRUE))

# Some terms, such as nodecov(), accept matrices of nodal
# covariates. An certain R quirk means that columns whose
# expressions are not typical variable names have their names
# dropped and need to be adjusted. Consider, for example, the
# linear and quadratic effects of grade:
Grade <- faux.mesa.high %v% "Grade"
colnames(cbind(Grade, Grade^2)) # Second column name missing.
colnames(cbind(Grade, Grade2=Grade^2)) # Can be set manually,
colnames(cbind(Grade, `Grade^2`=Grade^2)) # even to non-variable-names.
colnames(cbind(Grade, Grade^2, deparse.level=2)) # Alternatively, deparse.level=2 forces naming.
rm(Grade)

# Therefore, the nodal attribute names are set as follows:
summary(faux.mesa.high~nodecov(~cbind(Grade, Grade^2))) # column names dropped with a warning
summary(faux.mesa.high~nodecov(~cbind(Grade, Grade2=Grade^2))) # column names set manually
summary(faux.mesa.high~nodecov(~cbind(Grade, Grade^2, deparse.level=2))) # using deparse.level=2

# Activity by grade with a random covariate. Note that setting an attribute "name" gives it a name:
randomcov <- structure(I(rbinom(network.size(faux.mesa.high),1,0.5)), name="random")
```

```
summary(faux.mesa.high~nodefactor(I(randomcov)))
```

---

nodecov-ergmTerm   *Main effect of a covariate*

Description

This term adds a single network statistic for each quantitative attribute or matrix column to the model equaling the sum of attr(i) and attr(j) for all edges (i, j) in the network. For categorical attributes, see nodefactor . Note that for directed networks, nodecov equals nodeicov plus nodeocov .

Usage

```
# binary: nodecov(attr)

# binary: nodemain

# valued: nodecov(attr, form="sum")

# valued: nodemain(attr, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See [ergm-options](#) for how to invoke the old behaviour.

See Also

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

---

nodecovar-ergmTerm   *Covariance of undirected dyad values incident on each actor*

Description

This term adds one statistic equal to $\sum_{i,j<k} y_{i,j} y_{i,k}/(n-2)$. This can be viewed as a valued analog of the `star(2)` statistic.

Usage

```
# valued: nodecovar(center, transform)
```

Arguments

center   If `center=TRUE` , the $y_{\cdot,\cdot}$ s are centered by their mean over the whole network before the calculation. Note that this makes the model non-local, but it may alleviate multimodailty.

transform   If `transform="sqrt"` , $y_{\cdot,\cdot}$ s are repaced by their square roots before the calculation. This makes sense for counts in particular. If `center=TRUE` as well, they are centered by the mean of the square roots.

Note

Note that this term replaces `nodesqrtcovar` , which has been deprecated in favor of `nodecovar(transform="sqrt")` .

See Also

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

---

nodefactor-ergmTerm   *Factor attribute effect*

Description

This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attr` attribute (or each combination of the attributes given). Each of these statistics gives the number of times a node with that attribute or those attributes appears in an edge in the network.

Usage

```
# binary: nodefactor(attr, base=1, levels=-1)

# valued: nodefactor(attr, base=1, levels=-1, form="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
base	deprecated
levels	this optional argument controls which levels of the attribute attributes and Levels (?nodal_attributes) for details.)
form	character how to aggregate tie values in a valued ERGM

Note

To include all attribute values is usually not a good idea, because the sum of all such statistics equals the number of edges and hence a linear dependency would arise in any model also including edges. The default, levels=-1, is therefore to omit the first (in lexicographic order) attribute level. To include all levels, pass either levels=TRUE (i.e., keep all levels) or levels=NULL (i.e., do not filter levels).

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

See Also

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

---

nodeicov-ergmTerm   *Main effect of a covariate for in-edges*

Description

This term adds a single network statistic for each quantitative attribute or matrix column to the model equaling the total value of attr(j) for all edges (i, j) in the network. This term may only be used with directed networks. For categorical attributes, see nodeifactor .

Usage

```
# binary: nodeicov(attr)

# valued: nodeicov(attr, form="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
form	character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See ergm-options for how to invoke the old behaviour.

See Also

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

---

nodeicovar-ergmTerm   *Covariance of in-dyad values incident on each actor*

Description

This term adds one statistic equal to $\sum_{i,j,k} y_{j,i} y_{k,i}/(n-2)$. This can be viewed as a valued analog of the istar(2) statistic.

Usage

```
# valued: nodeicovar(center, transform)
```

Arguments

center   If center=TRUE , the $y_{\cdot,\cdot}$ s are centered by their mean over the whole network before the calculation. Note that this makes the model non-local, but it may alleviate multimodailty.

transform   If transform="sqrt" , $y_{\cdot,\cdot}$ s are repaced by their square roots before the calculation. This makes sense for counts in particular. If center=TRUE as well, they are centered by the mean of the square roots.

Note

Note that this term replaces nodeisqrtcovar , which has been deprecated in favor of nodeicovar(transform="sqrt") .

See Also

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

---

nodeifactor-ergmTerm   *Factor attribute effect for in-edges*

Description

This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attr attribute (or each combination of the attributes given). Each of these statistics gives the number of times a node with that attribute or those attributes appears as the terminal node of a directed tie.

For an analogous term for quantitative vertex attributes, see nodeicov .

Usage

```
# binary: nodeifactor(attr, base=1, levels=-1)

# valued: nodeifactor(attr, base=1, levels=-1, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                for details.)

base   deprecated

levels   this optional argument controls which levels of the attribute attributes and Levels
                (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

To include all attribute values is usually not a good idea, because the sum of all such statistics equals
the number of edges and hence a linear dependency would arise in any model also including edges.
The default, levels=-1, is therefore to omit the first (in lexicographic order) attribute level. To
include all levels, pass either levels=TRUE (i.e., keep all levels) or levels=NULL (i.e., do not filter
levels).

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and levels are passed, levels overrides base.

See Also

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

---

nodematch-ergmTerm   *Uniform homophily and differential homophily*

Description

When diff=FALSE , this term adds one network statistic to the model, which counts the number of
edges (i, j) for which attr(i)==attr(j) . This is also called "uniform homophily," because each
group is assumed to have the same propensity for within-group ties. When multiple attribute names
are given, the statistic counts only ties for which all of the attributes match. When diff=TRUE ,
p network statistics are added to the model, where p is the number of unique values of the attr
attribute. The k th such statistic counts the number of edges (i, j) for which attr(i) == attr(j)
== value(k) , where value(k) is the k th smallest unique value of the attr attribute. This is
also called "differential homophily," because each group is allowed to have a unique propensity
for within-group ties. Note that a statistical test of uniform vs. differential homophily should be
conducted using the ANOVA function.

By default, matches on all levels k are counted. This works for both diff=TRUE and diff=FALSE .

Usage

```
# binary: nodematch(attr, diff=FALSE, keep=NULL, levels=NULL)

# valued: nodematch(attr, diff=FALSE, keep=NULL, levels=NULL, form="sum")

# valued: match(attr, diff=FALSE, keep=NULL, levels=NULL, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
                for details.)

diff   specify if the term has uniform or differential homophily

keep   deprecated

levels   this optional argument controls which levels of the attribute attributes and Levels
                (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

The argument keep is retained for backwards compatibility and may be removed in a future version. When both keep and levels are passed, levels overrides keep.

See Also

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

NodematchFilter-ergmTerm

Filtering on nodematch

Description

Evaluates the terms specified in formula on a network constructed by taking y and removing any edges for which attrname(i)!=attrname(j) .

Usage

```
# binary: NodematchFilter(formula, attrname)
```

Arguments

formula   formula to be evaluated

attrname   a character vector giving one or more names of attributes in the network's vertex
                attribute list.

See Also

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

---

nodemix-ergmTerm   *Nodal attribute mixing*

Description

By default, this term adds one network statistic to the model for each possible pairing of attribute values. The statistic equals the number of edges in the network in which the nodes have that pairing of values. (When multiple attributes are specified, a statistic is added for each combination of attribute values for those attributes.) In other words, this term produces one statistic for every entry in the mixing matrix for the attribute(s). By default, the ordering of the attribute values is lexicographic: alphabetical (for nominal categories) or numerical (for ordered categories).

Usage

```
# binary: nodemix(attr, base=NULL, b1levels=NULL, b2levels=NULL, levels=NULL, levels2=-1)

# valued: nodemix(attr, base=NULL, b1levels=NULL, b2levels=NULL, levels=NULL,
#                  levels2=-1, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

base   deprecated

b1levels, b2levels, levels

                control what statistics are included in the model and the order in which they appear. `levels` applies to unipartite networks; `b1levels` and `b2levels` apply to bipartite networks (see Specifying Vertex attributes and Levels (?nodal_attributes) for details)

levels2   similar to the other levels arguments above and applies to all networks. Optionally allows a factor or character matrix to be specified to group certain levels. Level combinations corresponding to NA are excluded. Combinations specified by the same character or level will be grouped together and summarised by the same statistic. If an empty string is specified, the level combinations will be ungrouped. Only the upper triangle needs to be specified for undirected networks. For example, `levels2=matrix(c('A', '', NA, 'A'), 2, 2, byrow=TRUE)` on an undirected matrix will group homophilous ties while leaving ties between 1 and 2 ungrouped.

form   character how to aggregate tie values in a valued ERGM

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels2 are passed, levels2 overrides base.

See Also

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

---

nodeocov-ergmTerm   *Main effect of a covariate for out-edges*

Description

This term adds a single network statistic for each quantitative attribute or matrix column to the model equaling the total value of attr(i) for all edges (i, j) in the network. This term may only be used with directed networks. For categorical attributes, see nodeofactor .

Usage

```
# binary: nodeocov(attr)

# valued: nodeocov(attr, form="sum")
```

Arguments

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

form   character how to aggregate tie values in a valued ERGM

Note

ergm versions 3.9.4 and earlier used different arguments for this term. See [ergm-options](#) for how to invoke the old behaviour.

See Also

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

---

nodeocovar-ergmTerm   *Covariance of out-dyad values incident on each actor*

Description

This term adds one statistic equal to $\sum_{i,j,k} y_{i,j} y_{i,k} / (n-2)$. This can be viewed as a valued analog of the ostar(2) statistic.

Usage

```
# valued: nodeocovar(center, transform)
```

Arguments

center	whether the $y_{.,.}$ s are centered by their mean over the whole network before the calculation. Note that this makes the model non-local, but it may alleviate multimodailty.
transform	if transform="sqrt" , $y_{.,.}$ s are repaced by their square roots before the calculation. This makes sense for counts in particular. If center=TRUE as well, they are centered by the mean of the square roots.

Note

Note that this term replaces nodeosqrtcovar , which has been deprecated in favor of nodeocovar(transform="sqrt") .

See Also

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

---

nodeofactor-ergmTerm   *Factor attribute effect for out-edges*

Description

This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the attr attribute (or each combination of the attributes given). Each of these statistics gives the number of times a node with that attribute or those attributes appears as the node of origin of a directed tie.

Usage

```
# binary: nodeofactor(attr, base=1, levels=-1)

# valued: nodeofactor(attr, base=1, levels=-1, form="sum")
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
base	deprecated
levels	this optional argument controls which levels of the attribute attributes and Levels ([?nodal_attributes](#)) for details.)
form	character how to aggregate tie values in a valued ERGM

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and levels are passed, levels overrides base.

To include all attribute values is usually not a good idea, because the sum of all such statistics equals the number of edges and hence a linear dependency would arise in any model also including edges. The default, levels=-1, is therefore to omit the first (in lexicographic order) attribute level. To include all levels, pass either levels=TRUE (i.e., keep all levels) or levels=NULL (i.e., do not filter levels).

This term can only be used with directed networks.

See Also

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

---

nparam   *Length of the parameter vector associated with an object or with its terms.*

Description

This is a generic that returns the number of parameters associated with a model or a model fit.

Usage

```
nparam(object, ...)

## Default S3 method:
nparam(object, ...)

## S3 method for class 'ergm'
nparam(object, offset = NA, ...)
```

Arguments

object	An object for which number of parameters is defined.
...	Additional arguments to methods.
offset	If NA (the default), all model terms are counted; if TRUE, only offset terms are counted; and if FALSE, offset terms are skipped.

Methods (by class)

- default: By default, the length of the [coef()](#) vector is returned.
- ergm: A method to return the number of parameters of an [ergm](#) fit.

---

nsp-ergmTerm   *Nonedgewise shared partners*

Description

This is just like the dsp and esp terms, except this term adds one network statistic to the model for each element in d where the i th such statistic equals the number of non-edges (that is, dyads that do not have an edge) in the network with exactly d[i] shared partners. This term can be used with directed and undirected networks.

Usage

```
# binary: nsp(d)
```

Arguments

d   a vector of distinct integers

Note

For directed networks, only outgoing two-path ("OTP") shared partners are counted. In other words, for a (directed) non-edge (i,j) in a directed graph, the number of shared partners counted by nsp is the number of nodes k that have edges i -> k -> j. (These may also be called homogeneous shared partners.) To count other types of shared partners instead, see dnsp.

See Also

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

---

observed-ergmConstraint
                           Preserve the observed dyads of the given network

---

Description

Preserve the observed dyads of the given network.

Usage

```
# observed
```

See Also

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

---

odegrange-ergmTerm   *Out-degree range*

Description

This term adds one network statistic to the model for each element of from (or to); the i th such statistic equals the number of nodes in the network of out-degree greater than or equal to from[i] but strictly less than to[i] , i.e. with out-edge count in semiopen interval [from,to) .

This term can only be used with directed networks; for undirected networks (bipartite and not) see degrange . For degrees of specific modes of bipartite networks, see b1degrange and b2degrange . For in-degrees, see idegrange .

Usage

```
# binary: odegrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
```

Arguments

from, to	vectors of distinct integers. If one of the vectors have length 1, it is recycled to the length of the other. Otherwise, it must have the same length.
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

odegree-ergmTerm   *Out-degree*

Description

This term adds one network statistic to the model for each element in d ; the i th such statistic equals the number of nodes in the network of out-degree d[i] , i.e. the number of nodes with exactly d[i] out-edges. This term can only be used with directed networks; for undirected networks see degree .

Usage

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

Arguments

d	a vector of distinct integers
by, homophily	the optional argument by specifies a vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If this is specified and homophily is TRUE , then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the by attribute. If by is specified and homophily is FALSE (the default), then separate degree range statistics are calculated for nodes having each separate value of the attribute.
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

odegree1.5-ergmTerm   *Out-degree to the 3/2 power*

Description

This term adds one network statistic to the model equaling the sum over the actors of each actor's outdegree taken to the 3/2 power (or, equivalently, multiplied by its square root). This term is analogous to the term of Snijders et al. (2010), equation (12). This term can only be used with directed networks.

Usage

```
# binary: odegree1.5
```

See Also

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

odegreedist-ergmConstraint

Preserve the outdegree distribution

Description

Preserve the outdegree distribution of the given network.

Usage

```
# odegreedist
```

See Also

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

odegrees-ergmConstraint

Preserve outdegree for directed networks

Description

For directed networks, preserve the outdegree of each vertex of the given network, while allowing indegree to vary

Usage

```
# odegrees
```

See Also

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

Offset-ergmTerm   *Terms with fixed coefficients*

Description

This operator is analogous to the offset() wrapper, but the coefficients are specified within the term and the curved ERGM mechanism is used internally.

Usage

```
# binary: Offset(formula, coef, which)
```

Arguments

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

coef   coefficients to the formula

which   used to specify which of the parameters in the formula are fixed. It can be a logical vector (recycled as needed), a numeric vector of indices of parameters to be fixed, or a character vector of parameter names.

See Also

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

---

opentriad-ergmTerm   *Open triads*

Description

This term adds one statistic to the model equal to the number of 2-stars minus three times the number of triangles in the network. It is currently only implemented for undirected networks.

Usage

```
# binary: opentriad
```

See Also

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

---

ostar-ergmTerm   *k-Outstars*

Description

This term adds one network statistic to the model for each element in k . The i th such statistic counts the number of distinct k[i] -outstars in the network, where a k -outstar is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $(N{\to}O_j)$ exist for $j = 1, \ldots, k$. If attr is specified then the count is the number of k -outstars where all nodes have the same value of the attribute. This term can only be used with directed networks; for undirected networks see kstar .

Usage

```
# binary: ostar(k, attr=NULL, levels=NULL)
```

Arguments

k	a vector of distinct integers
attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
levels	TODO (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

Note

ostar(1) is equal to both istar(1) and edges .

See Also

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

---

param_names   *Names of the parameters associated with an object.*

Description

This is a generic that returns a vector giving the names of the parameters associated with a model or a model fit.

Usage

```
param_names(object, ...)

## Default S3 method:
param_names(object, ...)
```

Arguments

`object`	An object for which parameter names are defined.
`...`	Additional arguments to methods.

Methods (by class)

- default: By default, the names of the [`coef()`](coef()) vector is returned.

---

predict.formula   *ERGM-based tie probabilities*

Description

Calculate model-predicted **conditional** and **unconditional** tie probabilities for dyads in the given network. Conditional probabilities of a dyad given the state of all the remaining dyads in the graph are computed exactly. Unconditional probabilities are computed through simulating networks using the given model. Currently there are two methods implemented:

- Method for formula objects requires (1) an ERGM model formula with an existing network object on the left hand side and model terms on the right hand side, and (2) a vector of corresponding parameter values.

- Method for ergm objects, as returned by [`ergm()`](ergm()), takes both the formula and parameter values from the fitted model object.

Both methods can limit calculations to specific set of dyads of interest.

Usage

```
## S3 method for class 'formula'
predict(
  object,
  theta,
  conditional = TRUE,
  type = c("response", "link"),
  nsim = 100,
  output = c("data.frame", "matrix"),
  ...
)

## S3 method for class 'ergm'
predict(object, ...)
```

Arguments

`object`	a formula or a fitted ERGM model object
`theta`	numeric vector of ERGM model parameter values
`conditional`	logical whether to compute conditional or unconditional predicted probabilities
`type`	character element, one of `"response"` (default) or `"link"` - whether the returned predictions are on the probability scale or on the scale of linear predictor. This is similar to `type` argument of `predict.glm()`.
`nsim`	integer, number of simulated networks used for computing unconditional probabilities. Defaults to 100.
`output`	character, type of object returned. Defaults to `"data.frame"`. See section Value below.
`...`	other arguments passed to/from other methods. For the `predict.formula` method, if `conditional=TRUE` arguments are passed to `ergmMPLE()`. If `conditional=FALSE` arguments are passed to `simulate_formula()`.

Value

Type of object returned depends on the argument `output`. If `output="data.frame"` the function will return a data frame with columns:

- `tail, head` – indices of nodes identifying a dyad
- `p` – predicted conditional tie probability

If `output="matrix"` the function will return an "adjacency matrix" with the predicted probabilities. Diagonal values are 0s.

Examples

```
# A three-node empty directed network
net <- network.initialize(3, directed=TRUE)

# In homogeneous Bernoulli model with odds of a tie of 1/5 all ties are
# equally likely
predict(net ~ edges, log(1/5))

# Let's add a tie so that `net` has 1 tie out of possible 6 (so odds of 1/5)
net[1,2] <- 1

# Fit the model
fit <- ergm(net ~ edges)

# The p's should be identical
predict(fit)
```

Prod-ergmTerm   *A product (or an arbitrary power combination) of one or more formu-*
                              las

Description

This operator evaluates a list of formulas whose corresponnding RHS statistics will be multiplied
elementwise. They are required to be nonnegative.

Usage

```
# binary: Prod(formulas, label)

# valued: Prod(formulas, label)
```

Arguments

formulas   a list (constructed using [list()](#) or [c()](#) of [ergm()](#)-style formulas whose RHS
                    gives the statistics to be evaluated, or a single formula.

                    If a formula in the list has an LHS, it is interpreted as follows:

                        • a numeric scalar: Network statistics of this formula will be exponentiated
                          by this.
                        • a numeric vector: Corresponding network statistics of this formula will be
                          exponentiated by this.
                        • a numeric matrix: Vector of network statistics will be exponentiated by this
                          using the same pattern as matrix multiplication.
                        • a character string: One of several predefined multiplicative combinations.
                          Currently supported presets are as follows:
                            – "prod": Network statistics of this formula will be multiplied together;
                              equivalent to matrix(1,1,p) , where p is the length of the network
                              statistic vector.
                            – "geomean": Network statistics of this formula will be geometrically
                              averaged; equivalent to matrix(1/p,1,p) , where p is the length of
                              the network statistic vector.

label   used to specify the names of the elements of the resulting term product vector. If
                    label is a character vector of length 1, it will be recycled with indices appended.
                    If a function is specified, formulas parameter names are extracted and their list
                    of character vectors is passed label.

Details

Note that each formula must either produce the same number of statistics or be mapped through a
matrix to produce the same number of statistics.

A single formula is also permitted. This can be useful if one wishes to, say, scale or multiply
together the statistics returned by a formula.

Offsets are ignored unless there is only one formula and the transformation only scales the statistics (i.e., the effective transformation matrix is diagonal).

Curved models are supported, subject to some limitations. In particular, the first model's etamap will be used, overwriting the others. If label is not of length 1, it should have an attr -style attribute "curved" specifying the names for the curved parameters.

Note

The current implementation piggybacks on the Log , Exp , and Sum operators, essentially Exp(~Sum(~Log(formula), label)) . This may result in loss of precision, particularly for extremely large or small statistics. The implementation may change in the future.

See Also

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

---

randomtoggle-ergmProposal

Propose a randomly selected dyad to toggle

Description

Propose a randomly selected dyad to toggle

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli		.dyads bd	-2	random	c

---

rank_test.ergm   *A lack-of-fit test for ERGMs*

Description

A simple test reporting the sample quantile of the observed network's probability in the distribution under the MLE. This is a conservative p-value for the null hypothesis of the observed network being a draw from the distribution of interest.

Usage

```
rank_test.ergm(x, plot = FALSE)
```

Arguments

x	an [ergm()](#) object.
plot	if TRUE, plot the empirical distribution.

Value

The sample quantile of the observed network's probability among the predicted.

---

receiver-ergmTerm   *Receiver effect*

Description

This term adds one network statistic for each node equal to the number of in-ties for that node. This measures the popularity of the node. The term for the first node is omitted by default because of linear dependence that arises if this term is used together with edges , but its coefficient can be computed as the negative of the sum of the coefficients of all the other actors. That is, the average coefficient is zero, following the Holland-Leinhardt parametrization of the p_1 model (Holland and Leinhardt, 1981). This term can only be used with directed networks. For undirected networks, see sociality .

Usage

```
# binary: receiver(base=1, nodes=-1)

# valued: receiver(base=1, nodes=-1, form="sum")
```

Arguments

base	deprecated
nodes	specify which nodes' statistics should be included or excluded (see Specifying Vertex attributes and Levels (?nodal_attributes) for details)
form	character how to aggregate tie values in a valued ERGM

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and nodes are passed, nodes overrides base.

See Also

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

S-ergmTerm   *Evaluation on an induced subgraph*

Description

This operator takes a two-sided forumla `attrs` whose LHS gives the attribute or attribute function
for which tails and heads will be used to construct the induced subgraph. They must evaluate either
to a logical vector equal in length to the number of tails (for LHS) and heads (for RHS) indicating
which nodes are to be used to induce the subgraph or a numeric vector giving their indices.

Usage

```
# binary: S(formula, attrs)
```

Arguments

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

attrs   a two-sided formula to be used. A one-sided formula (e.g., ~A) is symmetrized
               (e.g., A~A).

Details

As with indexing vectors, the logical vector will be recycled to the size of the network or the size
of the appropriate bipartition, and negative indices will deselect vertices.

When the two sets are identical, the induced subgraph retains the directedness of the original graph.
Otherwise, an undirected bipartite graph is induced.

See Also

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

---

samplk   *Longitudinal networks of positive affection within a monastery as a*
                              "network" object

Description

Three [network](#) objects containing the "liking" nominations of Sampson's (1969) monks at the three
time points.

Usage

```
data(samplk)
```

Details

Sampson (1969) recorded the social interactions among a group of monks while he was a resident as an experimenter at the cloister. During his stay, a political "crisis in the cloister" resulted in the expulsion of four monks– namely, the three "outcasts," Brothers Elias, Simplicius, Basil, and the leader of the "young Turks," Brother Gregory. Not long after Brother Gregory departed, all but one of the "young Turks" left voluntarily: Brothers John Bosco, Albert, Boniface, Hugh, and Mark. Then, all three of the "waverers" also left: First, Brothers Amand and Victor, then later Brother Romuald. Eventually, Brother Peter and Brother Winfrid also left, leaving only four of the original group.

Of particular interest are the data on positive affect relations ("liking," using the terminology later adopted by White et al. (1976)), in which each monk was asked if he had positive relations to each of the other monks. Each monk ranked only his top three choices (or four, in the case of ties) on "liking". Here, we consider a directed edge from monk A to monk B to exist if A nominated B among these top choices.

The data were gathered at three times to capture changes in group sentiment over time. They represent three time points in the period during which a new cohort had entered the monastery near the end of the study but before the major conflict began. These three time points are labeled T2, T3, and T4 in Tables D5 through D16 in the appendices of Sampson's 1969 dissertation. and the corresponding network data sets are named samplk1, samplk2, and samplk3, respectively.

See also the data set sampson containing the time-aggregated graph samplike.

samplk3 is a data set of Hoff, Raftery and Handcock (2002).

The data sets are stored as network objects with three vertex attributes:

group Groups of novices as classified by Sampson, that is, "Loyal", "Outcasts", and "Turks", but with a fourth group called the "Waverers" by White et al. (1975) that comprises two of the original Loyal opposition and one of the original Outcasts. See the samplike data set for the original classifications of these three waverers.

cloisterville An indicator of attendance in the minor seminary of "Cloisterville" before coming to the monastery.

vertex.names The given names of the novices. NB: These names have been corrected as of ergm version 3.6.1.

This data set is standard in the social network analysis literature, having been modeled by Holland and Leinhardt (1981), Reitz (1982), Holland, Laskey and Leinhardt (1983), Fienberg, Meyer, and Wasserman (1981), and Hoff, Raftery, and Handcock (2002), among others. This is only a small piece of the data collected by Sampson.

This data set was updated for version 2.5 (March 2012) to add the cloisterville variable and refine the names. This information is from de Nooy, Mrvar, and Batagelj (2005). The original vertex names were: Romul_10, Bonaven_5, Ambrose_9, Berth_6, Peter_4, Louis_11, Victor_8, Winf_12, John_1, Greg_2, Hugh_14, Boni_15, Mark_7, Albert_16, Amand_13, Basil_3, Elias_17, Simp_18. The numbers indicate the ordering used in the original dissertation of Sampson (1969).

Mislabeling in Versions Prior to 3.6.1

In ergm versions 3.6.0 and earlier, The adjacency matrices of the samplike, samplk1, samplk2, and samplk3 networks reflected the original Sampson (1969) ordering of the names even though the

vertex labels used the name order of de Nooy, Mrvar, and Batagelj (2005). That is, in `ergm` version 3.6.0 and earlier, the vertices were mislabeled. The correct order is the same one given in Tables D5, D9, and D13 of Sampson (1969): John Bosco, Gregory, Basil, Peter, Bonaventure, Berthold, Mark, Victor, Ambrose, Romauld (Sampson uses both spellings "Romauld" and "Ramauld" in the dissertation), Louis, Winfrid, Amand, Hugh, Boniface, Albert, Elias, Simplicius. By contrast, the order given in `ergm` version 3.6.0 and earlier is: Ramuald, Bonaventure, Ambrose, Berthold, Peter, Louis, Victor, Winfrid, John Bosco, Gregory, Hugh, Boniface, Mark, Albert, Amand, Basil, Elias, Simplicius.

Source

Sampson, S.~F. (1968), *A novitiate in a period of change: An experimental and case study of relationships,* Unpublished Ph.D. dissertation, Department of Sociology, Cornell University.

http://vlado.fmf.uni-lj.si/pub/networks/data/esna/sampson.htm

References

White, H.C., Boorman, S.A. and Breiger, R.L. (1976). *Social structure from multiple networks. I. Blockmodels of roles and positions.* American Journal of Sociology, 81(4), 730-780.

Wouter de Nooy, Andrej Mrvar, Vladimir Batagelj (2005) *Exploratory Social Network Analysis with Pajek*, Cambridge: Cambridge University Press

See Also

sampson, florentine, network, plot.network, ergm

| sampson | *Cumulative network of positive affection within a monastery as a "network" object* |

Description

A `network` object containing the cumulative "liking" nominations of Sampson's (1969) monks over the three time points.

Usage

```
data(sampson)
```

Details

Sampson (1969) recorded the social interactions among a group of monks while he was a resident as an experimenter at the cloister. During his stay, a political "crisis in the cloister" resulted in the expulsion of four monks– namely, the three "outcasts," Brothers Elias, Simplicius, Basil, and the leader of the "young Turks," Brother Gregory. Not long after Brother Gregory departed, all but one of the "young Turks" left voluntarily: Brothers John Bosco, Albert, Boniface, Hugh, and Mark. Then, all three of the "waverers" also left: First, Brothers Amand and Victor, then later Brother

Romuald. Eventually, Brother Peter and Brother Winfrid also left, leaving only four of the original group.

Of particular interest are the data on positive affect relations ("liking," using the terminology later adopted by White et al. (1976)), in which each monk was asked if he had positive relations to each of the other monks. Each monk ranked only his top three choices (or four, in the case of ties) on "liking". Here, we consider a directed edge from monk A to monk B to exist if A nominated B among these top choices.

The data were gathered at three times to capture changes in group sentiment over time. They represent three time points in the period during which a new cohort had entered the monastery near the end of the study but before the major conflict began. These three time points are labeled T2, T3, and T4 in Tables D5 through D16 in the appendices of Sampson's 1969 dissertation. The `samplike` data set is the time-aggregated network. Thus, a tie from monk A to monk B exists if A nominated B as one of his three (or four, in case of ties) best friends at any of the three time points.

See also the data sets `samplk1`, `samplk2`, and `samplk3`, containing the networks at each of the three individual time points.

The data set is stored as a `network` object with three vertex attributes:

group Groups of novices as classified by Sampson: "Loyal", "Outcasts", and "Turks".

cloisterville An indicator of attendance in the minor seminary of "Cloisterville" before coming to the monastery.

vertex.names The given names of the novices. NB: These names have been corrected as of `ergm` version 3.6.1; see details below.

In addition, the data set has an edge attribute, `nominations`, giving the number of times (out of 3) that monk A nominated monk B.

This data set is standard in the social network analysis literature, having been modeled by Holland and Leinhardt (1981), Reitz (1982), Holland, Laskey and Leinhardt (1983), Fienberg, Meyer, and Wasserman (1981), and Hoff, Raftery, and Handcock (2002), among others. This is only a small piece of the data collected by Sampson.

This data set was updated for version 2.5 (March 2012) to add the `cloisterville` variable and refine the names. This information is from de Nooy, Mrvar, and Batagelj (2005). The original vertex names were: Romul_10, Bonaven_5, Ambrose_9, Berth_6, Peter_4, Louis_11, Victor_8, Winf_12, John_1, Greg_2, Hugh_14, Boni_15, Mark_7, Albert_16, Amand_13, Basil_3, Elias_17, Simp_18. The numbers indicate the ordering used in the original dissertation of Sampson (1969).

Mislabeling in Versions Prior to 3.6.1

In `ergm` version 3.6.0 and earlier, The adjacency matrices of the `samplike`, `samplk1`, `samplk2`, and `samplk3` networks reflected the original Sampson (1969) ordering of the names even though the vertex labels used the name order of de Nooy, Mrvar, and Batagelj (2005). That is, in `ergm` version 3.6.0 and earlier, the vertices were mislabeled. The correct order is the same one given in Tables D5, D9, and D13 of Sampson (1969): John Bosco, Gregory, Basil, Peter, Bonaventure, Berthold, Mark, Victor, Ambrose, Romauld (Sampson uses both spellings "Romauld" and "Ramauld" in the dissertation), Louis, Winfrid, Amand, Hugh, Boniface, Albert, Elias, Simplicius. By contrast, the order given in `ergm` version 3.6.0 and earlier is: Ramuald, Bonaventure, Ambrose, Berthold, Peter, Louis, Victor, Winfrid, John Bosco, Gregory, Hugh, Boniface, Mark, Albert, Amand, Basil, Elias, Simplicius.

Source

Sampson, S.~F. (1968), *A novitiate in a period of change: An experimental and case study of relationships,* Unpublished Ph.D. dissertation, Department of Sociology, Cornell University.

http://vlado.fmf.uni-lj.si/pub/networks/data/esna/sampson.htm

References

White, H.C., Boorman, S.A. and Breiger, R.L. (1976). *Social structure from multiple networks. I. Blockmodels of roles and positions.* American Journal of Sociology, 81(4), 730-780.

Wouter de Nooy, Andrej Mrvar, Vladimir Batagelj (2005) *Exploratory Social Network Analysis with Pajek*, Cambridge: Cambridge University Press

See Also

florentine, network, plot.network, ergm

---

san	*Use Simulated Annealing to attempt to match a network to a vector of mean statistics*

Description

This function attempts to find a network or networks whose statistics match those passed in via the `target.stats` vector.

Usage

```
san(object, ...)

## S3 method for class 'formula'
san(
  object,
  response = NULL,
  reference = ~Bernoulli,
  constraints = ~.,
  target.stats = NULL,
  nsim = NULL,
  basis = NULL,
  output = c("network", "edgelist", "ergm_state"),
  only.last = TRUE,
  control = control.san(),
  verbose = FALSE,
  offset.coef = NULL,
  ...
)
```

```
## S3 method for class 'ergm_model'
san(
  object,
  reference = ~Bernoulli,
  constraints = ~.,
  target.stats = NULL,
  nsim = NULL,
  basis = NULL,
  output = c("network", "edgelist", "ergm_state"),
  only.last = TRUE,
  control = control.san(),
  verbose = FALSE,
  offset.coef = NULL,
  ...
)
```

Arguments

object
: Either a [formula](#) or an [ergm](#) object. The [formula](#) should be of the form y ~
  <model terms>, where y is a network object or a matrix that can be coerced to a
  [network](#) object. For the details on the possible <model terms>, see [ergmTerm](#).
  To create a [network](#) object in , use the network() function, then add nodal
  attributes to it using the %v% operator if necessary.

...
: Further arguments passed to other functions.

response
: Either a character string, a formula, or NULL (the default), to specify the response
  attributes and whether the ERGM is binary or valued. Interpreted as follows:

  NULL Model simple presence or absence, via a binary ERGM.

  character string The name of the edge attribute whose value is to be modeled.
  Type of ERGM will be determined by whether the attribute is [logical](#)
  (TRUE/FALSE) for binary or [numeric](#) for valued.

  a formula must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is
  evaluated in the formula's environment with the network's edge attributes
  accessible as variables. The optional NAME specifies the name of the edge
  attribute into which the results should be stored, with the default being a
  concise version of EXPR. Normally, the type of ERGM is determined by
  whether the result of evaluating EXPR is logical or numeric, but the optional
  TYPE can be used to override by specifying a scalar of the type involved
  (e.g., TRUE for binary and 1 for valued).

reference
: A one-sided formula specifying the reference measure $(h(y))$ to be used. See
  help for [ERGM reference measures](#) implemented in the **[ergm](#)** package.

constraints
: A one-sided formula specifying one or more constraints on the support of the
  distribution of the networks being simulated. See the documentation for a simi-
  lar argument for [ergm](#) and see [list of implemented constraints](#) for more informa-
  tion. For simulate.formula, defaults to no constraints. For simulate.ergm,
  defaults to using the same constraints as those with which object was fitted.

target.stats
: A vector of the same length as the number of non-offset statistics implied by the
  formula, which is either object itself in the case of san.formula or object$formula

	in the case of san.ergm.
nsim	Number of networks to generate. Deprecated: just use [replicate()]().
basis	If not NULL, a network object used to start the Markov chain. If NULL, this is taken to be the network named in the formula.
output	Character, one of "network" (default), "edgelist", or "ergm_state": determines the output format. Partial matching is performed.
only.last	if TRUE, only return the last network generated; otherwise, return a [network.list]() with nsim networks.
control	A list of control parameters for algorithm tuning, typically constructed with [control.san()](). Its documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()]() (StatNet ConTRoL) also provides argument completion for the available control functions and limited argument name checking.
verbose	A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.
offset.coef	A vector of offset coefficients; these must be passed in by the user. Note that these should be the same set of coefficients one would pass to ergm via its offset.coef argument.
formula	(By default, the formula is taken from the ergm object. If a different formula object is wanted, specify it here.

Details

Acceptance probabilities for proposed toggles are computed as we now describe. There are two contributions: one from targeted statistics and one from offsets.

For the targeted statistics, a matrix of weights W is determined on each san iteration as follows. On the first iteration, the matrix W is the n by n identity matrix (n = number of target statistics), divided by n. On subsequent iterations: if control$SAN.invcov.diag = FALSE (the default), then the matrix W is the inverse of the covariance matrix of the targeted statistics, divided by the sum of its (the inverse's) diagonal; if control$SAN.invcov.diag = TRUE, then W is the inverse of the diagonal (regarded as a matrix) of the covariance matrix of the targeted statistics, divided by the sum of its (the inverse's) diagonal. In either of these two cases, the covariance matrix is computed based on proposals (not acceptances) made on the previous iteration, and the normalization for W is such that sum(diag(W)) = 1. The component of the acceptance probability coming from the targeted statistics is then computed for a given W as exp([y.Wy - x.Wx]/T) where T is the temperature, y the column vector of differences network statistics - target statistics computed before the current proposal is made, x the column vector of differences network statistics - target statistics computed assuming the current proposal is accepted, and . the dot product. If control$SAN.maxit > 1, then on the ith iteration, the temperature T takes the value control$SAN.tau * (1/i - 1/control$SAN.maxit)/(1 - 1/control$SAN.maxit); if control$SAN.maxit = 1, then the temperature T takes the value 0. Thus, T steps down from control$SAN.tau to 0 and is always 0 on the final iteration.

Offsets also contribute to the acceptance probability, as follows. If eta are the canonical offsets and Delta the corresponding change statistics for a given proposal, then the offset contribution to the acceptance probability is simply exp(eta.Delta) where . denotes the dot product. By default, finite

offsets are ignored, but this behavior can be changed by setting control$SAN.ignore.finite.offsets = FALSE.

The overall acceptance probability is the product of the targeted statistics contribution and the offset contribution (with the product capped at one).

Value

A network or list of networks that hopefully have network statistics close to the target.stats vector. Additionally, attr()-style attributes formula and stats are included.

Methods (by class)

- formula: Sufficient statistics are specified by a formula.
- ergm_model: A lower-level function that expects a pre-initialized ergm_model.

Examples

```
# initialize x to a random undirected network with 50 nodes and a density of 0.1
x <- network(50, density = 0.05, directed = FALSE)

# try to find a network on 50 nodes with 300 edges, 150 triangles,
# and 1250 4-cycles, starting from the network x
y <- san(x ~ edges + triangles + cycle(4), target.stats = c(300, 150, 1250))

# check results
summary(y ~ edges + triangles + cycle(4))

# initialize x to a random directed network with 50 nodes
x <- network(50)

# add vertex attributes
x %v% 'give' <- runif(50, 0, 1)
x %v% 'take' <- runif(50, 0, 1)

# try to find a set of 100 directed edges making the outward sum of
# 'give' and the inward sum of 'take' both equal to 62.5, so in
# edges (i,j) the node i tends to have above average 'give' and j
# tends to have above average 'take'
y <- san(x ~ edges + nodeocov('give') + nodeicov('take'), target.stats = c(100, 62.5, 62.5))

# check results
summary(y ~ edges + nodeocov('give') + nodeicov('take'))


# initialize x to a random undirected network with 50 nodes
x <- network(50, directed = FALSE)

# add a vertex attribute
x %v% 'popularity' <- runif(50, 0, 1)

# try to find a set of 100 edges making the total sum of
```

```
# popularity(i) and popularity(j) over all edges (i,j) equal to
# 125, so nodes with higher popularity are more likely to be
# connected to other nodes
y <- san(x ~ edges + nodecov('popularity'), target.stats = c(100, 125))

# check results
summary(y ~ edges + nodecov('popularity'))

# creates a network with denser "core" spreading out to sparser
# "periphery"
plot(y)
```

search.ergmConstraints

Search the ERGM Constraint

Description

Searches through the `ergm.constraints` help page and prints out a list of constraints appropriate for the specified network's structural constraints, optionally restricting by additional keywords and search term matches.

Usage

```
search.ergmConstraints(search, keywords, name, packages)
```

Arguments

search	optional character search term to search for in the text of the constraint descriptions. Only matching constraints will be returned. Matching is case insensitive.
keywords	optional character vector of keyword tags to use to restrict the results (i.e. 'curved', 'triad-related')
name	optional character name of a specific constraint to return
packages	optional character vector indicating the subset of packages in which to search

Details

Uses `grep` internally to match the search terms against the constraint description, so search is currently matched as a single phrase. Keyword tags will only return a match if all of the specified tags are included in the constraint

Value

prints out the name and short description of matching constraints, and invisibly returns them as a list. If name is specified, prints out the full definition for the named constraint.

Author(s)

skyebend@uw.edu

See Also

See also `ergm.constraints` for the complete documentation

Examples

```
# find all of the constraint that mention degrees
search.ergmConstraints('degree')

# search on multiple keywords
search.ergmConstraints(keywords=c('directed','dyad-independent'))

# print out the content for a specific constraint
search.ergmConstraints(name='b1degrees')

# request the bipartite keyword in the ergm package
search.ergmConstraints(keywords='directed', packages='ergm')
```

---

search.ergmProposals   *Search the ERGM Proposals*

Description

Searches through the `ergm.proposals` help page and prints out a list of proposals appropriate for the specified network's structural constraints, optionally restricting by additional keywords and search term matches.

Usage

```
search.ergmProposals(search, name, reference, constraints, packages)
```

Arguments

search	optional character search term to search for in the text of the term descriptions. Only matching terms will be returned. Matching is case insensitive.
name	optional character name of a specific proposal to return
reference	optional character to limit proposals to only those with a matching reference
constraints	optional character vector to limit proposals to only those with matching hard or soft constraints
packages	optional character vector indicating the subset of packages in which to search

Details

Uses `grep` internally to match the search terms against the proposal description, so `search` is currently matched as a single phrase. Keyword tags will only return a match if all of the specified tags are included in the proposal

Value

prints out the name and short description of matching terms, and invisibly returns them as a list. If `name` is specified, prints out the full definition for the named term.

See Also

See also `ergm.proposals` for the complete documentation

Examples

```
# find all of the proposals that mention triangles
search.ergmProposals('MH algorithm')

# print out the content for a specific proposals
search.ergmProposals(name='randomtoggle')

# find all proposals with required or optional constraints
search.ergmProposals(constraints='.dyads')

# find all proposals with references
search.ergmProposals(reference='Bernoulli')

# request proposals that mention triangle in the ergm package
search.ergmProposals('MH algorithm', packages='ergm')
```

---

search.ergmReferences   *Search the ERGM References*

Description

Searches through the `ergm.references` help page and prints out a list of terms appropriate for the specified network's structural constraints, optionally restricting by additional keywords and search term matches.

Usage

```
search.ergmReferences(search, keywords, name, packages)
```

Arguments

search	optional character search term to search for in the text of the term descriptions. Only matching terms will be returned. Matching is case insensitive.
keywords	optional character vector of keyword tags to use to restrict the results (i.e. 'curved', 'triad-related')
name	optional character name of a specific term to return
packages	optional character vector indicating the subset of packages in which to search

Details

Uses [grep](#) internally to match the search terms against the term description, so search is currently matched as a single phrase. Keyword tags will only return a match if all of the specified tags are included in the term.

Value

prints out the name and short description of matching terms, and invisibly returns them as a list. If name is specified, prints out the full definition for the named term.

Author(s)

skyebend@uw.edu

See Also

See also [ergm.references](#) for the complete documentation

Examples

```
# find all of the references that mention dyad
search.ergmReferences('dyad')

# search on multiple keywords
search.ergmReferences(keywords=c('binary','discrete'))

# print out the content for a specific reference
search.ergmReferences(name='Bernoulli')

# request the bipartite keyword in the ergm package
search.ergmReferences(keywords='binary', packages='ergm')
```

---

search.ergmTerms *Search the ERGM terms*

Description

Searches through the `ergm.terms` help page and prints out a list of terms appropriate for the specified network's structural constraints, optionally restricting by additional keywords and search term matches.

Usage

```
search.ergmTerms(search, net, keywords, name, packages)
```

Arguments

search	optional character search term to search for in the text of the term descriptions. Only matching terms will be returned. Matching is case insensitive.
net	a network object that the term would be applied to, used as template to determine directedness, bipartite, etc
keywords	optional character vector of keyword tags to use to restrict the results (i.e. 'curved', 'triad-related')
name	optional character name of a specific term to return
packages	optional character vector indicating the subset of packages in which to search

Details

Uses `grep` internally to match the search terms against the term description, so `search` is currently matched as a single phrase. Keyword tags will only return a match if all of the specified tags are included in the term.

Value

prints out the name and short description of matching terms, and invisibly returns them as a list. If `name` is specified, prints out the full definition for the named term.

Author(s)

skyebend@uw.edu

See Also

See also `ergm.terms` for the complete documentation

Examples

```
# find all of the terms that mention triangles
search.ergmTerms('triangle')

# two ways to search for bipartite terms:

# search using a bipartite net as a template
myNet<-network.initialize(5,bipartite=3)
search.ergmTerms(net=myNet)

# or request the bipartite keyword
search.ergmTerms(keywords='bipartite')

# search on multiple keywords
search.ergmTerms(keywords=c('bipartite','dyad-independent'))

# print out the content for a specific term
search.ergmTerms(name='b2factor')

# request the bipartite keyword in the ergm package
search.ergmTerms(keywords='bipartite', packages='ergm')
```

---

sender-ergmTerm   *Sender effect*

Description

This term adds one network statistic for each node equal to the number of out-ties for that node.
This measures the activity of the node. The term for the first node is omitted by default because
of linear dependence that arises if this term is used together with edges , but its coefficient can be
computed as the negative of the sum of the coefficients of all the other actors. That is, the average
coefficient is zero, following the Holland-Leinhardt parametrization of the p_1 model (Holland
and Leinhardt, 1981).

For undirected networks, see sociality .

Usage

```
# binary: sender(base=1, nodes=-1)

# valued: sender(base=1, nodes=-1, form="sum")
```

Arguments

base	deprecated
nodes	specify which nodes' statistics should be included or excluded (see Specifying Vertex attributes and Levels (?nodal_attributes) for details)
form	character how to aggregate tie values in a valued ERGM

Note

The argument base is retained for backwards compatibility and may be removed in a future version. When both base and nodes are passed, nodes overrides base.

This term can only be used with directed networks.

See Also

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

---

  simmelian-ergmTerm   *Simmelian triads*

Description

This term adds one statistic to the model equal to the number of Simmelian triads, as defined by Krackhardt and Handcock (2007). This is a complete sub-graph of size three.

Usage

```
# binary: simmelian
```

Note

This term can only be used with directed networks.

See Also

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

---

  simmelianties-ergmTerm

                        Ties in simmelian triads

Description

This term adds one statistic to the model equal to the number of ties in the network that are associated with Simmelian triads, as defined by Krackhardt and Handcock (2007). Each Simmelian has six ties in it but, because Simmelians can overlap in terms of nodes (and associated ties), the total number of ties in these Simmelians is less than six times the number of Simmelians. Hence this is a measure of the clustering of Simmelians (given the number of Simmelians).

Usage

```
# binary: simmelianties
```

Note

This term can only be used with directed networks.

See Also

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

---

simulate.ergm	*Draw from the distribution of an Exponential Family Random Graph Model*

Description

[simulate](#) is used to draw from exponential family random network models. See [ergm](#) for more information on these models.

The method for [ergm](#) objects inherits the model, the coefficients, the response attribute, the reference, the constraints, and most simulation parameters from the model fit, unless overridden by passing them explicitly. Unless overridden, the simulation is initialized with a random draw from the fitted model, saved by [ergm()](#).

Usage

```
## S3 method for class 'formula_lhs_network'
simulate(object, nsim = 1, seed = NULL, ...)

simulate_formula(object, ..., basis = eval_lhs.formula(object))

## S3 method for class 'network'
simulate_formula(
  object,
  nsim = 1,
  seed = NULL,
  coef,
  response = NULL,
  reference = ~Bernoulli,
  constraints = ~.,
  observational = FALSE,
  monitor = NULL,
  statsonly = FALSE,
  esteq = FALSE,
  output = c("network", "stats", "edgelist", "ergm_state"),
  simplify = TRUE,
  sequential = TRUE,
  control = control.simulate.formula(),
  verbose = FALSE,
  ...,
```

```
    basis = ergm.getnetwork(object),
    do.sim = NULL,
    return.args = NULL
)

## S3 method for class 'ergm_state'
simulate_formula(
  object,
  nsim = 1,
  seed = NULL,
  coef,
  response = NULL,
  reference = ~Bernoulli,
  constraints = ~.,
  observational = FALSE,
  monitor = NULL,
  statsonly = FALSE,
  esteq = FALSE,
  output = c("network", "stats", "edgelist", "ergm_state"),
  simplify = TRUE,
  sequential = TRUE,
  control = control.simulate.formula(),
  verbose = FALSE,
  ...,
  basis = ergm.getnetwork(object),
  do.sim = NULL,
  return.args = NULL
)

## S3 method for class 'ergm_model'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  coef,
 reference = if (is(constraints, "ergm_proposal")) NULL else trim_env(~Bernoulli),
  constraints = trim_env(~.),
  observational = FALSE,
  monitor = NULL,
  basis = NULL,
  esteq = FALSE,
  output = c("network", "stats", "edgelist", "ergm_state"),
  simplify = TRUE,
  sequential = TRUE,
  control = control.simulate.formula(),
  verbose = FALSE,
  ...,
  do.sim = NULL,
```

```
    return.args = NULL
)

## S3 method for class 'ergm_state_full'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  coef,
  esteq = FALSE,
  output = c("network", "stats", "edgelist", "ergm_state"),
  simplify = TRUE,
  sequential = TRUE,
  control = control.simulate.formula(),
  verbose = FALSE,
  ...,
  return.args = NULL
)

## S3 method for class 'ergm'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  coef = coefficients(object),
  response = object$network %ergmlhs% "response",
  reference = object$reference,
  constraints = list(object$constraints, object$obs.constraints),
  observational = FALSE,
  monitor = NULL,
  basis = object$newnetwork,
  statsonly = FALSE,
  esteq = FALSE,
  output = c("network", "stats", "edgelist", "ergm_state"),
  simplify = TRUE,
  sequential = TRUE,
  control = control.simulate.ergm(),
  verbose = FALSE,
  ...
)
```

Arguments

object   Either a [formula](#) or an [ergm](#) object. The [formula](#) should be of the form y ~
                <model terms>, where y is a network object or a matrix that can be coerced to a
                [network](#) object. For the details on the possible <model terms>, see [ergmTerm](#).
                To create a [network](#) object in , use the network() function, then add nodal
                attributes to it using the %v% operator if necessary.

nsim	Number of networks to be randomly drawn from the given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm.
seed	Seed value (integer) for the random number generator. See `set.seed`.
...	Further arguments passed to or used by methods.
basis	a value (usually a `network`) to override the LHS of the formula.
coef	Vector of parameter values for the model from which the sample is to be drawn. If `object` is of class ergm, the default value is the vector of estimated coefficients. Can be set to NULL to bypass, but only if `return.args` below is used.
response	Either a character string, a formula, or NULL (the default), to specify the response attributes and whether the ERGM is binary or valued. Interpreted as follows:

NULL Model simple presence or absence, via a binary ERGM.

character string The name of the edge attribute whose value is to be modeled. Type of ERGM will be determined by whether the attribute is `logical` (TRUE/FALSE) for binary or `numeric` for valued.

a formula must be of the form NAME~EXPR|TYPE (with | being literal). EXPR is evaluated in the formula's environment with the network's edge attributes accessible as variables. The optional NAME specifies the name of the edge attribute into which the results should be stored, with the default being a concise version of EXPR. Normally, the type of ERGM is determined by whether the result of evaluating EXPR is logical or numeric, but the optional TYPE can be used to override by specifying a scalar of the type involved (e.g., TRUE for binary and 1 for valued).

reference	A one-sided formula specifying the reference measure ($h(y)$) to be used. See help for ERGM reference measures implemented in the **ergm** package.
constraints	A one-sided formula specifying one or more constraints on the support of the distribution of the networks being simulated. See the documentation for a similar argument for ergm and see list of implemented constraints for more information. For simulate.formula, defaults to no constraints. For simulate.ergm, defaults to using the same constraints as those with which `object` was fitted.
observational	Inherit observational constraints rather than model constraints.
monitor	A one-sided formula specifying one or more terms whose value is to be monitored. These terms are appended to the model, along with a coefficient of 0, so their statistics are returned. An `ergm_model` objectcan be passed as well.
statsonly	Logical: If TRUE, return only the network statistics, not the network(s) themselves. Deprecated in favor of output=.
esteq	Logical: If TRUE, compute the sample estimating equations of an ERGM: if the model is non-curved, all non-offset statistics are returned either way, but if the model is curved, the score estimating function values (3.1) by Hunter and Handcock (2006) are returned instead.
output	Normally character, one of "network" (default), "stats", "edgelist", or "ergm_state": determines the output format. Partial matching is performed.
	Alternatively, a function with prototype function(ergm_state, chain, iter, ...) that is called for each returned network, and its return value, rather than the network itself, is stored. This can be used to, for example, store the simulated

networks to disk without storing them in memory or compute network statistics not implemented using the ERGM API, without having to store the networks themselves.

simplify   Logical: If TRUE the output is "simplified": sampled networks are returned in a single list, statistics from multiple parallel chains are stacked, etc.. This makes it consistent with behavior prior to ergm 3.10.

sequential   Logical: If FALSE, each of the nsim simulated Markov chains begins at the initial network. If TRUE, the end of one simulation is used as the start of the next. Irrelevant when nsim=1.

control   A list of control parameters for algorithm tuning, typically constructed with [control.simulate.ergm()](#) or [control.simulate.formula()](#), which have different defaults. Their documentation gives the the list of recognized control parameters and their meaning. The more generic utility [snctrl()](#) (StatNet Control) also provides argument completion for the available control functions and limited argument name checking.

verbose   A logical or an integer to control the amount of progress and diagnostic information to be printed. FALSE/0 produces minimal output, wit higher values producing more detail. Note that very high values (5+) may significantly slow down processing.

do.sim   Logical; a deprecated interface superseded by return.args, that saves the inputs to the next level of the function.

return.args   Character; if not NULL, the simulate method for that particular class will, instead of proceeding for simulation, instead return its arguments as a list that can be passed as a second argument to [do.call()](#) or a lower-level function such as [ergm_MCMC_sample()](#). This can be useful if, for example, one wants to run several simulations with varying coefficients and does not want to reinitialize the model and the proposal every time. Valid inputs at this time are "formula", "ergm_model", and one of the "ergm_state" classes, for the three respective stopping points.

Details

A sample of networks is randomly drawn from the specified model. The model is specified by the first argument of the function. If the first argument is a [formula](#) then this defines the model. If the first argument is the output of a call to [ergm](#) then the model used for that call is the one fit – and unless coef is specified, the sample is from the MLE of the parameters. If neither of those are given as the first argument then a Bernoulli network is generated with the probability of ties defined by prob or coef.

Note that the first network is sampled after burnin steps, and any subsequent networks are sampled each interval steps after the first.

More information can be found by looking at the documentation of [ergm](#).

Value

If output=="stats" an [mcmc](#) object containing the simulated network statistics. If control$parallel>0, an [mcmc.list](#) object. If simplify=TRUE (the default), these would then be "stacked" and converted

to a standard matrix. A logical vector indicating whether or not the term had come from the monitor= formula is stored in attr()-style attribute "monitored".

Otherwise, a representation of the simulated network is returned, in the form specified by output. In addition to a network representation or a list thereof, they have the following attr-style attributes:

formula The formula used to generate the sample.

stats An mcmc or mcmc.list object as above.

control Control parameters used to generate the sample.

constraints Constraints used to generate the sample.

reference The reference measure for the sample.

monitor The monitoring formula.

response The edge attribute used as a response.

The following are the permitted network formats:

"network" If nsim==1, an object of class network. If nsim>1, it returns an object of class network.list (a list of networks) with the above-listed additional attributes.

"edgelist" An edgelist representation of the network, or a list thereof, depending on nsim.

"ergm_state" A semi-internal representation of a network consisting of a network object emptied of edges, with an attached edgelist matrix, or a list thereof, depending on nsim.

If simplify==FALSE, the networks are returned as a nested list, with outer list being the parallel chain (including 1 for no parallelism) and inner list being the samples within that chains (including 1, if one network per chain). If TRUE, they are concatenated, and if a total of one network had been simulated, the network itself will be returned.

Functions

- simulate.ergm_state_full: a low-level function to simulate from an ergm_state object.

Note

simulate.ergm_model() is a lower-level interface, providing a simulate() method for ergm_model class. The basis argument is required; monitor, if passed, must be an ergm_model as well; and constraints can be an ergm_proposal object instead.

See Also

ergm, network, ergm_MCMC_sample() for a demonstration of return.args=.

Examples

```
#
# Let's draw from a Bernoulli model with 16 nodes
# and density 0.5 (i.e., coef = c(0,0))
#
g.sim <- simulate(network(16) ~ edges + mutual, coef=c(0, 0))
#
# What are the statistics like?
```

```
#
summary(g.sim ~ edges + mutual)
#
# Now simulate a network with higher mutuality
#
g.sim <- simulate(network(16) ~ edges + mutual, coef=c(0,2))
#
# How do the statistics look?
#
summary(g.sim ~ edges + mutual)
#
# Let's draw from a Bernoulli model with 16 nodes
# and tie probability 0.1
#
g.use <- network(16,density=0.1,directed=FALSE)
#
# Starting from this network let's draw 3 realizations
# of a edges and 2-star network
#
g.sim <- simulate(~edges+kstar(2), nsim=3, coef=c(-1.8,0.03),
                  basis=g.use, control=control.simulate(
                    MCMC.burnin=1000,
                    MCMC.interval=100))
g.sim
summary(g.sim)
#
# attach the Florentine Marriage data
#
data(florentine)
#
# fit an edges and 2-star model using the ergm function
#
gest <- ergm(flomarriage ~ edges + kstar(2))
summary(gest)
#
# Draw from the fitted model (statistics only), and observe the number
# of triangles as well.
#
g.sim <- simulate(gest, nsim=10,
             monitor=~triangles, output="stats",
             control=control.simulate.ergm(MCMC.burnin=1000, MCMC.interval=100))
g.sim

# Custom output: store the edgecount (computed in R), iteration index, and chain index.
output.f <- function(x, iter, chain, ...){
  list(nedges = network.edgecount(as.network(x)),
       chain = chain, iter = iter)
}
g.sim <- simulate(gest, nsim=3,
             output=output.f, simplify=FALSE,
             control=control.simulate.ergm(MCMC.burnin=1000, MCMC.interval=100))
unclass(g.sim)
```

---

simulate.formula	*A* simulate *Method for* formula *objects that dispatches based on the Left-Hand Side*

Description

This method evaluates the left-hand side (LHS) of the given formula and dispatches it to an appropriate method based on the result by setting an nonce class name on the formula.

Usage

```
## S3 method for class 'formula'
simulate(object, nsim = 1, seed = NULL, ..., basis, newdata, data)

## S3 method for class 'formula_lhs'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	a one- or two-sided [formula](#).
nsim, seed	number of realisations to simulate and the random seed to use; see [simulate()](#).
...	additional arguments to methods.
basis	if given, overrides the LHS of the formula for the purposes of dispatching.
newdata, data	if passed, the object's LHS is evaluated in this environment; at most one of the two may be passed.

The dispatching works as follows:

1. If basis is not passed, and the formula has an LHS the expression on the LHS of the formula in the object is evaluated in the environment newdata or data (if given), in any case enclosed by the environment of object. Otherwise, basis is used.

2. The result is set as an attribute ".Basis" on object. If there is no basis or LHS, it is not set.

3. The class vector of object has c("formula_lhs_*CLASS*", "formula_lhs") prepended to it, where *CLASS* is the class of the LHS value or basis. If LHS or basis has multiple classes, they are all prepended; if there is no LHS or basis, c("formula_lhs_", "formula_lhs") is.

4. [simulate()](#) generic is evaluated on the new object, with all arguments passed on, excluding basis; if newdata or data are missing, they too are not passed on. The evaluation takes place in the parent's environment.

A "method" to receive a formula whose LHS evaluates to *CLASS* can therefore be implemented by a function simulate.formula_lhs_\var{CLASS}(). This function can expect a [formula](#) object, with additional attribute .Basis giving the evaluated LHS (so that it does not need to be evaluated again).

Functions

- `simulate.formula_lhs`: A function to catch the situation when there is no method implemented for the class to which the LHS evaluates.

See Also

[simulate.ergm()](#) family of functions, which uses this interface.

---

smalldiff-ergmTerm   *Number of ties between actors with similar attribute values*

Description

This term adds one statistic, having as its value the number of edges in the network for which the incident actors' attribute values differ less than `cutoff` ; that is, number of edges between i to j such that `abs(attr[i]-attr[j])<cutoff` .

Usage

```
# binary: smalldiff(attr, cutoff)
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.)
maximum	difference in attribute values for ties to be considered

See Also

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

---

smallerthan-ergmTerm   *Number of dyads with values strictly smaller than a threshold*

Description

Adds the number of statistics equal to the length of `threshold` equaling to the number of dyads whose values are exceeded by the corresponding element of `threshold` .

Usage

```
# valued: smallerthan(threshold=0)
```

Arguments

threshold	vector of numerical values

See Also

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

---

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.boo
   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.sampsize.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, RM.phase1n_base, RM.phase2n_base,
   RM.phase2sub, RM.init_gain, RM.phase3n, RM.interval, RM.burnin, RM.samplesize,
   Step.maxit, Step.gridsize, Step.interval, Step.burnin, Step.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()

---

sociality-ergmTerm   *Undirected degree*

Description

This term adds one network statistic for each node equal to the number of ties of that node. For
directed networks, see sender and receiver .

Usage

```
# binary: sociality(attr=NULL, base=1, levels=NULL, nodes=-1)

# valued: sociality(attr=NULL, base=1, levels=NULL, nodes=-1, form="sum")
```

Arguments

attr, levels	this optional argument is deprecated and will be replaced with a more elegant implementation in a future release. In the meantime, it specifies a categorical vertex attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details). If provided, this term only counts ties between nodes with the same value of the attribute (an actor-specific version of the nodematch term), restricted to be one of the values specified by (also deprecated) levels if levels is not NULL .
base	deprecated
nodes	By default, nodes=-1 means that the statistic for the first node will be omitted, but this argument may be changed to control which statistics are included just as for the nodes argument of sender and receiver terms.
form	character how to aggregate tie values in a valued ERGM

Note

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and levels are passed, levels overrides base.

The argument base is retained for backwards compatibility and may be removed in a future version.
When both base and nodes are passed, nodes overrides base.

This term can only be used with undirected networks.

See Also

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

sparse-ergmHint	*Sparse network*

Description

The network is sparse. This typically results in a Tie-Non-Tie (TNT) proposal regime.

Usage

```
# sparse
```

See Also

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

spectrum0.mvar	*Multivariate version of* coda*'s* [spectrum0.ar()](#).

Description

Its return value, divided by nrow(cbind(x)), is the estimated variance-covariance matrix of the sampling distribution of the mean of x if x is a multivatriate time series with AR(p) structure, with p determined by AIC.

Usage

```
spectrum0.mvar(
  x,
  order.max = NULL,
  aic = is.null(order.max),
  tol = .Machine$double.eps^0.5,
  ...
)
```

Arguments

x	a matrix with observations in rows and variables in columns.
order.max	maximum (or fixed) order for the AR model.
aic	use AIC to select the order (up to order.max).
tol	drop components until the reciprocal condition number of the transformed variance-covariance matrix is greater than this.
...	additional arguments to [ar()](#).

Value

A square matrix with dimension equalling to the number of columns of x, with an additional attribute `"infl"` giving the factor by which the effective sample size is reduced due to autocorrelation, according to the Vats, Flegal, and Jones (2015) estimate for ESS.

Note

`ar()` fails if `crossprod(x)` is singular, which is remedied by mapping the variables onto the principal components of x, dropping redundant dimentions.

StdNormal-ergmProposal
TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
StdNormal			0	random	c

StdNormal-ergmReference
Standard Normal reference

Description

Specifies each dyad's baseline distribution to be the normal distribution with mean 0 and variance 1.

Usage

```
# StdNormal
```

See Also

[ergmReference](#) for index of reference distributions currently visible to the package.

---

strat-ergmHint *Stratified dyads*

Description

The dyads in the network are stratified according to an attribute combination. This typically results in stratifying proposals by mixing type on a vertex attribute.

Usage

```
# strat(attr=NULL, pmat=NULL, empirical=FALSE)
```

Details

The user may pass a vertex attribute attr as an argument (the default for attr gives every vertex the same attribute value), and may also pass a matrix of weights pmat (the default for pmat gives equal weight to each mixing type). See Specifying Vertex Attributes and Levels for details on specifying vertex attributes. The matrix pmat , if specified, must have the same dimensions as a mixing matrix for the network and attribute under consideration, and the correspondence between rows and columns of pmat and values of attr is the same as for a mixing matrix.

The interpretation is that pmat[i,j]/sum(pmat) is the probability of proposing a toggle for mixing type (i,j) . (For undirected, unipartite networks, pmat is first symmetrized, and then entries below the diagonal are set to zero. Only entries on or above the diagonal of the symmetrized pmat are considered when making proposals. This accounts for the convention that mixing is undirected in an undirected, unipartite network: a tail of type i and a head of type j has the same mixing type as a tail of type j and a head of type i .)

As an alternative way of specifying pmat , the user may pass empirical=TRUE to use the mixing matrix of the network beginning the MCMC chain as pmat . In order for this to work, that network should have a reasonable (in particular, nonempty) edge set.

While some mixing types may be assigned zero proposal probability (either with a direct specification of pmat or with empirical=TRUE), this will not be recognized as a constraint by all components of ergm , and should be used with caution.

See Also

ergmHint for index of constraints and hints currently visible to the package.

---

sum-ergmTerm *Sum of dyad values (optionally taken to a power)*

Description

This term adds one statistic equal to the sum of dyad values taken to the power pow.

Usage

```
# valued: sum(pow=1)
```

Arguments

pow   power of dyad values. Defaults to 1.

See Also

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

Sum-operator-ergmTerm   *A sum (or an arbitrary linear combination) of one or more formulas*

Description

This operator sums up the RHS statistics of the input formulas elementwise.

Usage

```
# binary: Sum(formulas, label)

# valued: Sum(formulas, label)
```

Arguments

formulas   a list (constructed using [list()](#) or [c()](#)) of [ergm()](#)-style formulas whose RHS
                 gives the statistics to be evaluated, or a single formula.

                 If a formula in the list has an LHS, it is interpreted as follows:

                 - a numeric scalar: Network statistics of this formula will be multiplied by
                   this.
                 - a numeric vector: Corresponding network statistics of this formula will be
                   multiplied by this.
                 - a numeric matrix: Vector of network statistics will be pre-multiplied by
                   this.
                 - a character string: One of several predefined linear combinations. Currently
                   supported presets are as follows:
                     - "sum" Network statistics of this formula will be summed up; equivalent
                       to matrix(1,1,p), where p is the length of the network statistic vector.
                     - "mean" Network statistics of this formula will be averaged; equivalent
                       to matrix(1/p,1,p), where p is the length of the network statistic
                       vector.

label   used to specify the names of the elements of the resulting term sum vector. If
                 label is a character vector of length 1, it will be recycled with indices appended.
                 If a function is specified, formulas parameter names are extracted and their list
                 of character vectors is passed label.

Details

Note that each formula must either produce the same number of statistics or be mapped through a matrix to produce the same number of statistics.

A single formula is also permitted. This can be useful if one wishes to, say, scale or sum up the statistics returned by a formula.

Offsets are ignored unless there is only one formula and the transformation only scales the statistics (i.e., the effective transformation matrix is diagonal).

Curved models are supported, subject to some limitations. In particular, the first model's etamap will be used, overwriting the others. If label is not of length 1, it should have an attr -style attribute "curved" specifying the names for the curved parameters.

See Also

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

---

summary.ergm	*Summarizing ERGM Model Fits*

Description

base::summary() method for ergm() fits.

Usage

```
## S3 method for class 'ergm'
summary(
  object,
  ...,
  correlation = FALSE,
  covariance = FALSE,
  total.variation = TRUE
)

## S3 method for class 'summary.ergm'
print(
  x,
  digits = max(3, getOption("digits") - 3),
  correlation = x$correlation,
  covariance = x$covariance,
  signif.stars = getOption("show.signif.stars"),
  eps.Pvalue = 1e-04,
  print.formula = FALSE,
  print.fitinfo = TRUE,
  print.coefmat = TRUE,
  print.message = TRUE,
  print.deviances = TRUE,
```

```
    print.drop = TRUE,
    print.offset = TRUE,
    print.call = TRUE,
    ...
)
```

Arguments

object	an object of class "ergm", usually, a result of a call to [ergm()]().
...	For [summary.ergm()]() additional arguments are passed to [logLik.ergm()](). For [print.summary.ergm()](), to [stats::printCoefmat]().
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
covariance	logical; if TRUE, the covariance matrix of the estimated parameters is returned and printed.
total.variation	
	logical; if TRUE, the standard errors reported in the Std. Error column are based on the sum of the likelihood variation and the MCMC variation. If FALSE only the likelihood variation is used. The p-values are based on this source of variation.
x	object of class summary.ergm returned by [summary.ergm()]().
digits	significant digits for coefficients
signif.stars	whether to print dots and stars to signify statistical significance. See [print.summary.lm()]().
eps.Pvalue	p-values below this level will be printed as "<eps.Pvalue".
print.formula, print.fitinfo, print.coefmat, print.message, print.deviances, print.drop, print.offset,	
	which components of the fit summary to print.

Details

[summary.ergm()]() tries to be smart about formatting the coefficients, standard errors, etc.

The default printout of the summary object contains the call, number of iterations used, null and residual deviances, and the values of AIC and BIC (and their MCMC standard errors, if applicable). The coefficient table contains the following columns:

- Estimate, Std. Error - parameter estimates and their standard errors
- MCMC % - if total.variation=TRUE (default) the percentage of standard error attributable to MCMC estimation process rounded to an integer. See also [vcov.ergm()]() and its sources argument.
- z value, Pr(>|z|) - z-test and p-values

Value

The returned object is a list of class "ergm.summary" with the following elements:

formula	ERGM model formula
call	R call used to fit the model

`correlation, covariance`

        whether to print correlation/covariance matrices of the estimated parameters

`pseudolikelihood`

        was the model estimated with MPLE

`independence`   is the model dyad-independent

`control`   the [`control.ergm()`] object used

`samplesize`   MCMC sample size

`message`   optional message on the validity of the standard error estimates

`null.lik.0`   It is `TRUE` of the null model likelihood has not been calculated. See [`logLikNull()`]

`devtext, devtable`

        Deviance type and table

`aic, bic`   values of AIC and BIC

`coefficients`   matrices with model parameters and associated statistics

`asycov`   asymptotic covariance matrix

`asyse`   asymptotic standard error matrix

`offset, drop, estimate, iterations, mle.lik, null.lik`

        see documentation of the object returned by [`ergm()`]

See Also

The model fitting function [`ergm()`], [`print.ergm()`], and [`base::summary()`]. Function [`stats::coef()`] will extract the matrix of coefficients with standard errors, t-statistics and p-values.

Examples

```
data(florentine)

x <- ergm(flomarriage ~ density)
summary(x)
```

---

summary.formula   *Calculation of network or graph statistics or other attributes specified on a formula*

Description

Most generally, this function computes those summaries of the object on the LHS of the formula that are specified by its RHS. In particular, if given a network as its LHS and [`ergmTerm`] on its RHS, it computes the sufficient statistics associated with those terms.

Usage

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

Arguments

`object`	A formula having as its LHS a `network` object or a matrix that can be coerced to a `network` object, a `network.list`, or other types to be summarized using a formula. (See 'methods('summary_formula') for the possible LHS types.
`...`	further arguments passed to or used by methods.

Details

In practice, `summary.formula()` is a thin wrapper around the `summary_formula()` generic, which dispatches methods based on the class of the LHS of the formula.

Value

A vector of statistics specified in RHS of the formula.

See Also

`ergm()`, `network()`, `ergmTerm`

Examples

```
#
# Lets look at the Florentine marriage data
#
data(florentine)
#
# test the summary_formula function
#
summary(flomarriage ~ edges + kstar(2))
m <- as.matrix(flomarriage)
summary(m ~ edges)  # twice as large as it should be
summary(m ~ edges, directed=FALSE) # Now it's correct
```

Symmetrize-ergmTerm   *Evaluation on symmetrized (undirected) network*

Description

Evaluates the terms in `formula` on an undirected network constructed by symmetrizing the LHS network using one of four rules:

1. "weak" A tie (i, j) is present in the constructed network if the LHS network has either tie (i, j) or (j, i) (or both).

2. "strong" A tie (i, j) is present in the constructed network if the LHS network has both tie (i, j) and tie (j, i) .

3. "upper" A tie (i, j) is present in the constructed network if the LHS network has tie $(\min(i, j), \max(i, j))$: the upper triangle of the LHS network.

4. "lower" A tie (i, j) is present in the constructed network if the LHS network has tie $(\max(i,j), \min(i,j))$: the lower triangle of the LHS network.

Usage

```
# binary: Symmetrize(formula, rule="weak")
```

Arguments

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

rule   one of "weak", "strong", "upper", "lower"

See Also

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

TNT-ergmProposal   *Default MH algorithm*

Description

Stratifies the population of dyads edge status: those having ties and those having no ties (hence T/NT). This is useful for improving performance in sparse networks, because it gives at least 50\

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Bernoulli	sparse	.dyads bd	-1	TNT	c

---

transitive-ergmTerm   *Transitive triads*

Description

This term adds one statistic to the model, equal to the number of triads in the network that are transitive. The transitive triads are those of type `120D` , `030T` , `120U` , or `300` in the categorization of Davis and Leinhardt (1972). For details on the 16 possible triad types, see `?triad.classify` in the sna package. Note the distinction from the `ttriple` term. This term can only be used with directed networks.

Usage

```
# binary: transitive
```

See Also

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

---

transitiveties-ergmTerm

Transitive ties

Description

This term adds one statistic, equal to the number of ties $i \rightarrow j$ such that there exists a two-path from i to j . (Related to the `ttriple` term.)

Usage

```
# binary: transitiveties(attr=NULL, levels=NULL)
```

Arguments

attr	quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) If set, all three nodes involved (i , j , and the node on the two-path) must match on this attribute in order for $i \rightarrow j$ to be counted.
levels	TODO (See Specifying Vertex attributes and Levels ([?nodal_attributes](#)) for details.)

See Also

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

---

transitiveweights-ergmTerm

Transitive weights

Description

This statistic implements the transitive weights statistic defined by Krivitsky (2012), Equation 13. For each of these options, the first (and the default) is more stable but also more conservative, while the second is more sensitive but more likely to induce a multimodal distribution of networks.

Usage

```
# valued: transitiveweights(twopath="min", combine="max", affect="min")
```

Arguments

twopath	the minimum of the constituent dyads (`"min"`) or their geometric mean (`"geomean"`)
combine	the maximum of the 2-path strengths (`"max"`) or their sum (`"sum"`)
affect	the minimum of the focus dyad and the combined strength of the two paths (`"min"`) or their geometric mean (`"geomean"`)

See Also

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

---

triadcensus-ergmTerm   *Triad census*

Description

For a directed network, this term adds one network statistic for each of an arbitrary subset of the 16 possible types of triads categorized by Davis and Leinhardt (1972) as `003`, `012`, `102`, `021D`, `021U`, `021C`, `111D`, `111U`, `0` and `300` . Note that at least one category should be dropped; otherwise a linear dependency will exist among the 16 statistics, since they must sum to the total number of three-node sets. By default, the category `003` , which is the category of completely empty three-node sets, is dropped. This is considered category zero, and the others are numbered 1 through 15 in the order given above. Each statistic is the count of the corresponding triad type in the network. For details on the 16 types, see `?triad.classify` in the `sna` package, on which this code is based. For an undirected network, the triad census is over the four types defined by the number of ties (i.e., 0, 1, 2, and 3).

Usage

```
# binary: triadcensus(levels)
```

Arguments

levels	For directed networks, specify a set of terms to add other than the default value of `1:15`. attributes and Levels ([?nodal_attributes](#)) for details.)

See Also

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

---

triangle-ergmTerm   *Triangles*

Description

By default, this term adds one statistic to the model equal to the number of triangles in the network. For an undirected network, a triangle is defined to be any set $\{(i,j),(j,k),(k,i)\}$ of three edges. For a directed network, a triangle is defined as any set of three edges $(i{\rightarrow}j)$ and $(j{\rightarrow}k)$ and either $(k{\rightarrow}i)$ or $(k{\leftarrow}i)$. The former case is called a "transitive triple" and the latter is called a "cyclic triple", so in the case of a directed network, triangle equals ttriple plus ctriple — thus at most two of these three terms can be in a model.

Usage

```
# binary: triangle(attr=NULL, diff=FALSE, levels=NULL)

# binary: triangles(attr=NULL, diff=FALSE, levels=NULL)
```

Arguments

attr, diff   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes) for details.) If attr is specified and diff is FALSE , then the count is restricted to those triples of nodes with equal values of the vertex attribute specified by attr . If attr is specified and diff is TRUE , then one statistic is added for each value of attr , equal to the number of triangles where all three nodes have that value of the attribute.

levels   add one statistic for each value specified if diff is TRUE. (See Specifying Vertex attributes and Levels (?nodal_attributes) for details.)

See Also

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

---

tripercent-ergmTerm   *Triangle percentage*

Description

By default, this term adds one statistic to the model equal to 100 times the ratio of the number of triangles in the network to the sum of the number of triangles and the number of 2-stars not in triangles (the latter is considered a potential but incomplete triangle). In case the denominator equals zero, the statistic is defined to be zero. For the definition of triangle, see triangle . This is often called the mean correlation coefficient. This term can only be used with undirected networks; for directed networks, it is difficult to define the numerator and denominator in a consistent and meaningful way.

Usage

```
# binary: tripercent(attr=NULL, diff=FALSE, levels=NULL)
```

Arguments

attr, diff   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
              for details.) If attr is specified and diff is FALSE , then the counts are restricted
              to those triples of nodes with equal values of the vertex attribute specified by
              attr . If attr is specified and diff is TRUE , then one statistic is added for each
              value of attr , equal to the number of triangles where all three nodes have that
              value of the attribute.

levels   add one statistic for each value specified if diff is TRUE attributes and Levels
              (?nodal_attributes) for details.)

See Also

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

---

ttriple-ergmTerm   *Transitive triples*

Description

By default, this term adds one statistic to the model, equal to the number of transitive triples
in the network, defined as a set of edges $\{(i{\to}j), j{\to}k), (i{\to}k)\}$. Note that triangle equals
ttriple+ctriple for a directed network, so at most two of the three terms can be in a model.

Usage

```
# binary: ttriple(attr=NULL, diff=FALSE, levels=NULL)
```

```
# binary: ttriad
```

Arguments

attr, diff   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
              for details.) If attr is specified and diff is FALSE , then the count is over the
              number of transitive triples where all three nodes have the same value of the
              attribute. If attr is specified and diff is TRUE , then one statistic is added for
              each value of attr , equal to the number of triangles where all three nodes have
              that value of the attribute.

levels   add one statistic for each value specified if diff is TRUE. (See Specifying Vertex
              attributes and Levels (?nodal_attributes) for details.)

attr   quantitative attribute (see Specifying Vertex attributes and Levels (?nodal_attributes)
              for details.)

Note

This term can only be used with directed networks.

See Also

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

---

twopath-ergmTerm	*2-Paths*

Description

This term adds one statistic to the model, equal to the number of 2-paths in the network. For a directed network this is defined as a pair of edges $(i{\rightarrow}j), (j{\rightarrow}k)$, where i and j must be distinct. That is, it is a directed path of length 2 from i to k via j. For directed networks a 2-path is also a mixed 2-star but the interpretation is usually different; see m2star . For undirected networks a twopath is defined as a pair of edges $\{i, j\}, \{j, k\}$. That is, it is an undirected path of length 2 from i to k via j, also known as a 2-star.

Usage

```
# binary: twopath
```

See Also

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

Unif-ergmProposal	*TODO*

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Unif			0	random	c

Unif-ergmReference   *Continuous Uniform reference*

Description

Specifies each dyad's baseline distribution to be continuous uniform between a and b: $h(y) = 1$, with the support being [a, b].

Usage

```
# Unif(a,b)
```

Arguments

a, b   minimum and maximum to the baseline discrete uniform distribution, both inclusive. Both values must be finite.

See Also

ergmReference for index of reference distributions currently visible to the package.

UnifNonObserved-ergmProposal
                           TODO

Description

TODO

Details

Reference	Enforced	May Enforce	Priority	Weight	Class
Unif	observed		0	random	c

---

update.network   *Update the edges in a network based on a matrix*

Description

Replaces the edges in a network object with the edges corresponding to the sociomatrix or edge list specified by new.

Usage

```
## S3 method for class 'network'
update(object, ...)

update_network(object, new, ...)

## S3 method for class 'matrix_edgelist'
update_network(object, new, attrname = if (ncol(new) > 2) names(new)[3], ...)

## S3 method for class 'data.frame'
update_network(object, new, attrname = if (ncol(new) > 2) names(new)[3], ...)

## S3 method for class 'matrix'
update_network(object, new, matrix.type = NULL, attrname = NULL, ...)

## S3 method for class 'ergm_state'
update_network(object, new, ...)
```

Arguments

object	a network object.
...	Additional arguments; currently unused.
new	Either an adjacency matrix (a matrix of values indicating the presence and/or the value of a tie from i to j) or an edge list (a two-column matrix listing origin and destination node numbers for each edge, with an optional third column for the value of the edge).
attrname	For a network with edge weights gives the name of the edge attribute whose names to set.
matrix.type	One of "adjacency" or "edgelist" telling which type of matrix new is. Default is to use the which.matrix.type function.

Value

A new network object with the edges specified by new and network and vertex attributes copied from the input network object. Input network is not modified.

Functions

- update_network: dispatcher for network update based on the type of updating information.
- update_network.matrix_edgelist: a method for updating a network based on a matrix-form edgelist
- update_network.data.frame: a method for updating a network based on an edgelist
- update_network.matrix: a method for updating a network based on a matrix
- update_network.ergm_state: a method for updating a network based on an ergm_state object.

See Also

ergm(), network

Examples

```
#
data(florentine)
#
# test the network.update function
#
# Create a Bernoulli network
rand.net <- network(network.size(flomarriage))
# store the sociomatrix
rand.mat <- rand.net[,]
# Update the network
update(flomarriage, rand.mat, matrix.type="adjacency")
# Try this with an edgelist
rand.mat <- as.matrix.network.edgelist(flomarriage)[1:5,]
update(flomarriage, rand.mat, matrix.type="edgelist")
```

---

wtd.median   *Weighted Median*

Description

Compute weighted median.

Usage

```
wtd.median(x, na.rm = FALSE, weight = FALSE)
```

Arguments

x	Vector of data, same length as weight
na.rm	Logical: Should NAs be stripped before computation proceeds?
weight	Vector of weights

Details

Uses a simple algorithm based on sorting.

Value

Returns an empirical .5 quantile from a weighted sample.

Index