

Package ‘ergm’

April 17, 2009

Version 2.2-1

Date 2009-03-08

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

Author Mark S. Handcock <handcock@stat.washington.edu>, David R. Hunter <dhunter@stat.psu.edu>, Carter T. Butts <butts@uci.edu>, Steven M. Goodreau <goodreau@u.washington.edu>, Pavel N. Krivitsky <pavel@stat.washington.edu>, Martina Morris <morris@u.washington.edu>

Maintainer Mark S. Handcock <handcock@stat.washington.edu>

Depends network

Suggests coda, KernSmooth, sna

Description An integrated set of tools to analyze and simulate networks based on exponential-family random graph models (ERGM). “ergm” is a part of the “statnet” suite of packages for network analysis. For a list of functions type: help(package=’ergm’)

License file LICENSE

URL <http://statnet.org>

Repository CRAN

Date/Publication 2009-03-08 16:10:33

R topics documented:

ergm-package	2
ergmuserterms-package	4
anova.ergm	5
as.network.numeric	7
coef.ergm	8
control.ergm	9
control.gof	12
control.simulate	13

edgelist.ergm	15
ergm	16
ergm-terms	24
faux.magnolia.high	37
faux.mesa.high	39
flobusiness	40
flomarriage	41
florentine	42
g4	43
Getting.Started	44
gof	46
mcmc.diagnostics.ergm	48
molecule	51
network.update	51
plot.ergm	52
plot.gofobject	54
print.ergm	56
samplk	57
sampson	58
simulate.ergm	59
summary.ergm	62
summary.gofobject	63
summary.statistics	64
Index	66

ergm-package	<i>Fit, Simulate and Diagnose Exponential-Family Models for Networks</i>
--------------	--

Description

`ergm` is a collection of functions to plot, fit, diagnose, and simulate from random graph models. For a list of functions type: `help(package='ergm')`

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 the original authors are to 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*
<http://statnetproject.org>.

All programs derived from this package must cite it. For complete citation information, use `citation(package="ergm")`.

Details

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 R. The `ergm` package allows maximum likelihood estimates of exponential random network models to be calculated using Markov Chain Monte Carlo. The package also provides tools for plotting networks, simulating networks and assessing model goodness-of-fit.

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

Author(s)

Mark S. Handcock (handcock@stat.washington.edu),
David R. Hunter (dhunter@stat.psu.edu),
Carter T. Butts (buttsc@uci.edu),
Steven M. Goodreau (goodreau@u.washington.edu), and
Martina Morris (morrism@u.washington.edu)
Maintainer: Mark S. Handcock (handcock@stat.washington.edu)

References

- Admiraal R, Handcock MS (2007). **networksis**: Simulate bipartite graphs with fixed marginals through sequential importance sampling. Statnet Project, Seattle, WA. Version 1, <http://statnetproject.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). <http://www.jstatsoft.org/v24/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 (2006). **netperm**: Permutation Models for Relational Data. Version 0.2, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2007). **sna**: Tools for Social Network Analysis. Version 1.5, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2008). **network**: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). <http://www.jstatsoft.org/v24/i02/>.

- Butts CT, with help~from David~Hunter, Handcock MS (2007). **network**: Classes for Relational Data. Version 1.2, <http://erzuli.ss.uci.edu/R.stuff>.
- 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). <http://www.jstatsoft.org/v24/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. www.csss.washington.edu/Papers/wp39.pdf
- Handcock MS (2003b). **degrenet**: Models for Skewed Count Distributions Relevant to Networks. Statnet Project, Seattle, WA. Version 1.0, <http://statnetproject.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 2, <http://statnetproject.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, <http://statnetproject.org>.
- Hunter, D. R. and Handcock, M. S. (2006) Inference in curved exponential family models for networks, *Journal of Computational and Graphical Statistics*, 15: 565-583
- Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). <http://www.jstatsoft.org/v24/i03/>.
- Krivitsky PN, Handcock MS (2007). **latentnet**: Latent position and cluster models for statistical networks. Seattle, WA. Version 2, <http://statnetproject.org>.
- Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). <http://www.jstatsoft.org/v24/i04/>.
- Strauss, D., and Ikeda, M.(1990). Pseudolikelihood estimation for social networks *Journal of the American Statistical Association*, 85, 204-212.

ergmuserterms-package

Add Statistics Terms for the 'ergm' Package

Description

The `ergm` package is capable of fitting a wide range of exponential random network models, in which the probability of a given network, y , on a set of nodes is $\exp(\theta \cdot g(y)) / c(\theta)$, where $g(y)$ is a vector of network statistics, θ is a parameter vector of the same length and $c(\theta)$ is the normalizing constant for the distribution. The `ergm` function fits these models when they are expressed via an

R `formula` object, of the form $y \sim \langle \text{model terms} \rangle$, where y is a network object or a matrix that can be coerced to a network object. To create a network object in R, use the `network()` function, then add nodal attributes to it using the `%v%` operator if necessary.

The `ergm` package contains a wide range of terms. For the details on the possible `<model terms>`, see [ergm-terms](#).

This package can be modified by users to add user-defined terms to `ergm` models. The terms can be used throughout the `ergm` package and behave identically to the supplied terms.

Details

The `ergmuserterms` package is available from the `statnet` website (<http://statnetproject.org>).

The code contains some simple examples and templates. These include:

`m2star` *Mixed 2-stars, a.k.a. 2-paths*. This option can only be specified with a directed network; for undirected graphs see `kstar(2)`. This option adds one statistic to the model, equal to the number of mixed-2-stars in the network, defined as a pair of edges $\{(i \rightarrow j), (j \rightarrow k)\}$.

`testme` *A clone of Edges*. This is included for purposes of an example. This option adds one graph statistic equal to the number of edges in the graph. For undirected graphs, `edges` is isomorphic to `kstar(1)`; for directed networks, `edges` is isomorphic to both `ostar(1)` and `istar(1)`.

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 graph statistics using MCMC. This sample is then returned to R, which then approximates the MLE.

See Also

`ergm`, `network`, `ergm-terms`

Examples

```
## Not run:
library(ergmuserterms)
data(sampson)
monk.fit <- ergm(samplike~m2star)
summary(monk.fit)

monk.fit <- ergm(samplike ~ m2star + testme)
summary(monk.fit)
## End(Not run)
```

 anova.ergm

 ANOVA for Linear Model Fits

Description

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

Usage

```
## S3 method for class 'ergm':
anova(object, ...)
## S3 method for class 'ergmlist':
anova(object, ..., scale = 0, test = "F")
```

Arguments

object, ...	objects of class <code>ergm</code> , usually, a result of a call to <code>ergm</code> .
test	a character string specifying the test statistic to be used. Can be one of "F", "Chisq" or "Cp", with partial matching allowed, or NULL for no test.
scale	numeric. An estimate of the noise variance σ^2 . If zero this will be estimated from the largest model considered.

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.

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 R'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](#).

Examples

```
data(molecule)
molecule %v% "atomic type" <- c(1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3)
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))
anova(fit0, fit1)
anova(fit0, fit1, fit2)
```

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 'network.numeric':
as(x, directed = TRUE,
    hyper = FALSE, loops = FALSE, multiple = FALSE, bipartite = FALSE,
    ignore.eval = TRUE, names.eval = NULL,
    edge.check = FALSE,
    density=NULL, theta0=NULL, numedges=NULL, ...)
```

Arguments

x	count; the number of nodes in the network. If <code>bipartite=TRUE</code> , it is the number of events 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.

<code>names.eval</code>	optionally, the name of the attribute in which edge values should be stored. Currently ignored.
<code>edge.check</code>	logical; perform consistency checks on new edges?
<code>density</code>	numeric; the probability of a tie for Bernoulli networks. If neither <code>density</code> nor <code>theta0</code> are 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.)
<code>theta0</code>	numeric; the log-odds of a tie for Bernoulli networks. It is only used if <code>density</code> is not specified.
<code>numedges</code>	count; if present, sample the Bernoulli network conditional on this number of edges (rather than independently with the specified probability).
<code>...</code>	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`

Author(s)

Carter T. Butts (butts@uci.edu) and Mark S. Handcock (handcock@stat.washington.edu)

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 10 events and 5 actors and density 0.1
g<-network(5, bipartite=10, density=0.1)
```

`coef.ergm`*Extract Model Coefficients*

Description

`coef` is a Method which extracts model coefficients from objects returned by the `ergm` function. `coefficients` is an *alias* for it.

Usage

```
## S3 method for class 'ergm':  
coef(object, ...)  
## S3 method for class 'ergm':  
coefficients(object, ...)
```

Arguments

`object` an object for which the extraction of model coefficients is meaningful.
`...` other arguments.

Value

Coefficients extracted from the model object `object`.

See Also

[fitted.values](#) and [residuals](#) for related methods; [glm](#), [lm](#) for model fitting.

Examples

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

`control.ergm`*Auxiliary for Controlling ERGM Fitting*

Description

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

Usage

```
control.ergm(prop.weights = "default", prop.args = NULL,
             nr.maxit = 100,
             calc.mcmc.se = TRUE, hessian = FALSE, compress = TRUE,
             SAN.burnin=NULL,
             maxNumDyadTypes = 1e+06, maxedges = 20000, maxchanges = 1e+06,
             maxMPLEsamplesize = 1e+05, MPLEtype=c("glm", "penalized"), trace = 0,
             steplength = 0.5, drop = TRUE, force.mcmc = FALSE, check.degeneracy=FA
             0.05, metric = c("Likelihood", "raw"), method = c("BFGS", "Nelder-Mead
             trustregion = 20, initial.loglik = NULL, initial.network = NULL,
             style = c("Newton-Raphson", "Robbins-Monro",
             "Stochastic-Approximation"), phase1_n = NULL, initial_gain = NULL,
             nsubphases = "maxit", niterations = NULL, phase3_n = NULL,
             RobMon.phase1n_base = 7, RobMon.phase2n_base = 7, RobMon.phase2sub
             = 4, RobMon.init_gain = 0.1, RobMon.phase3n = 500, dyninterval =
             1000, packagenames="ergm", parallel = 0, returnMCMCstats = TRUE)
```

Arguments

`prop.weights` Specifies the method to allocate probabilities of being proposed to dyads. Defaults to "default", which picks a reasonable default for the specified constraint. Possible values are "TNT", "random", and "nonobserved", though not all values may be used with all possible constraints (in the `ergm` function).

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

`nr.maxit` count; The maximum number of iterations in the Newton-Raphson optimization. Defaults to 100. `maxit` gives the total number of likelihood function evaluations.

`calc.mcmc.se` logical; should the contribution to the standard errors of the estimator incurred by the MCMC sampling be computed. Default is TRUE.

`hessian` logical; Should the Hessian matrix of the likelihood function be computed. Default is TRUE.

`compress` logical; Should the matrix of sample statistics returned be compressed to the set of unique statistics with a column of frequencies post-pended. This also uses a compression algorithm in the computation of the maximum pseudo-likelihood estimate that will dramatically speed it for large networks. Default is FALSE.

`SAN.burnin` Burnin used for calling SAN routine. If NULL, `burnin` is used.

`maxNumDyadTypes` count; The maximum number of unique pseudolikelihood change statistics to be allowed if `compress=TRUE`. It is only relevant in that case. Default is 10000.

`maxedges` Maximum number of edges for which to allocate space.

`maxchanges` Maximum number of changes in dynamic network simulation for which to allocate space.

`maxMPLEsamplesize` count; the sample size to use for endogenous sampling in the pseudolikelihood computation. Default is 10^{11} .

<code>MPLetype</code>	one of "glm" or "penalized"; method to use for psuedolikelihood. "glm" is the usual formal logistic regression. "penalized" uses the bias-reduced method of Firth (1993) as originally implemented by Meinhard Ploner, Daniela Dunkler, Harry Southworth, and Georg Heinze in the "logistf" package. Default is "glm".
<code>trace</code>	non-negative integer; If positive, tracing information on the progress of the optimization is produced. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing. (To understand exactly what these do see the source code for <code>optim</code> : higher levels give more detail.)
<code>steplength</code>	Multiplier for step length, to make fitting more stable at the cost of efficiency.
<code>drop</code>	logical; Should the degenerate terms in the model be dropped from the fit? If statistics occur on the extreme of their range they correspond to infinite parameter estimates. Default is TRUE.
<code>force.mcmc</code>	logical; should MCMC maximum likelihood be used? Only relevant for dyadic independent networks, in which the MLE could be found using MPL instead.
<code>check.degeneracy</code>	Logical: Should the diagnostics include a check for model degeneracy?
<code>mcmc.precision</code>	vector; upper bounds on the precision of the standard errors induced by the MCMC algorithm. Defaults to 0.05.
<code>metric</code>	character; The name of the optimization metric to use. Defaults to "Likelihood".
<code>method</code>	character; The name of the optimization method to use. See <code>optim</code> for the options. The default method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm). It is attributed to Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.
<code>trustregion</code>	numeric; The maximum amount the algorithm will allow the approximated likelihood to be increased at a given iteration. Defaults to 20. See Boer, Huisman, Snijders, and Zeggelink (2003) for details.
<code>initial.loglik</code>	Initial value of loglikelihood, if known.
<code>initial.network</code>	Initial network for MCMC, if different from observed network.
<code>style</code>	character; The style of maximum likelihood estimation to use. The default is optimization of an MCMC estimate of the log-likelihood. An alternative is to use a form of stochastic approximation ("Robbins-Monro"). 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.
<code>phase1_n</code>	count; The 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 Boer, Huisman, Snijders, and Zeggelink (2003) for details.
<code>initial_gain</code>	numeric; The initial gain to Phase 2 of the stochastic approximation algorithm. Defaults to 0.1. See Boer, Huisman, Snijders, and Zeggelink (2003) for details.
<code>nsubphases</code>	count; The number of sub-phases in Phase 2 of the stochastic approximation algorithm. Defaults to <code>maxit</code> . See Boer, Huisman, Snijders, and Zeggelink (2003) for details.

<code>niterations</code>	count; The 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 Boer, Huisman, Snijders, and Zeggelink (2003) for details.
<code>phase3_n</code>	count; The sample size for the MCMC sample in Phase 3 of the stochastic approximation algorithm. Defaults to 1000. See Boer, Huisman, Snijders, and Zeggelink (2003) for details.
<code>RobMon.phase1n_base</code>	Robbins-Monro control parameter
<code>RobMon.phase2n_base</code>	Robbins-Monro control parameter
<code>RobMon.phase2sub</code>	Robbins-Monro control parameter
<code>RobMon.init_gain</code>	Robbins-Monro control parameter
<code>RobMon.phase3n</code>	Robbins-Monro control parameter
<code>returnMCMCstats</code>	logical; If this is <code>TRUE</code> (the default) the matrix of change statistics from the MCMC run is returned as component <code>sample</code> . This matrix is actually an object of class <code>mcmc</code> and can be used directly in the <code>CODA</code> package to assess MCMC convergence.
<code>dyninterval</code>	Number of Metropolis-Hastings proposal for each phase in the dynamic network simulation.
<code>packagenames</code>	Names of packages in which changestatsitics are found.
<code>parallel</code>	Number of threads in which to run the sampling.

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

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

See Also

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

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

Description

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

Usage

```
control.gof.formula(prop.weights = "default", prop.args = NULL, drop = TRUE, summarizestats = FALSE, maxchanges = 1e+06)
```

```
control.gof.ergm(prop.weights = NULL, prop.args = NULL, drop = TRUE, summarizestats = FALSE, maxchanges = 1e+06)
```

Arguments

`prop.weights` Specifies the method to allocate probabilities of being proposed to dyads. For the `simulate.formula` variant, defaults to "default", which picks a reasonable default for the specified constraint. For `simulate.ergm` variant, defaults to NULL, to reuse the weights with which the given `ergm.object` was fitted. Other possible values are "TNT", "random", and "nonobserved", though not all values may be used with all possible constraints.

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

`drop` logical; Should the degenerate terms in the model be dropped from the fit? If statistics occur on the extreme of their range they correspond to infinite parameter estimates. Default is TRUE.

`summarizestats` logical; Print out a summary of the sufficient statistics of the generated network. This is useful as a diagnostic. Default is FALSE.

`maxchanges` Currently unused.

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.simulate *Auxiliary for Controlling ERGM Simulation*

Description

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

Usage

```
control.simulate(prop.weights = "default", prop.args = NULL,
drop = FALSE, summarizestats = FALSE,
maxchanges = 1e+06,
packagenames="ergm",
parallel=0)
```

```
control.simulate.formula(prop.weights = "default", prop.args = NULL,
drop = FALSE, summarizestats = FALSE,
maxchanges = 1e+06,
packagenames="ergm",
parallel=0)
```

```
control.simulate.ergm(prop.weights = NULL, prop.args = NULL, drop = FALSE,
summarizestats = FALSE,
maxchanges = 1e+06,
packagenames="ergm",
parallel=0)
```

Arguments

prop.weights	Specifies the method to allocate probabilities of being proposed to dyads. For the <code>simulate.formula</code> variant, defaults to "default", which picks a reasonable default for the specified constraint. For <code>simulate.ergm</code> variant, defaults to NULL, to reuse the weights with which the given <code>ergm.object</code> was fitted. Other possible values are "TNT", "random", and "nonobserved", though not all values may be used with all possible constraints.
prop.args	An alternative, direct way of specifying additional arguments to proposal.
drop	logical; Should the degenerate terms in the model be dropped from the fit? If statistics occur on the extreme of their range they correspond to infinite parameter estimates. Default is FALSE.
summarizestats	logical; Print out a summary of the sufficient statistics of the generated network. This is useful as a diagnostic. Default is FALSE.
maxchanges	Currently unused
packagenames	Names of packages in which changestatistics are found.
parallel	Number of threads in which to run the sampling.

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`.

<code>edgelist.ergm</code>	<i>Return edgelist in standard ergm format</i>
----------------------------	--

Description

`edgelist.ergm` returns an edgelist for a network in a format that is expected by many of the routines of the `ergm` package.

Usage

```
## Default S3 method:
edgelist.ergm(x, ...)
## S3 method for class 'network':
edgelist.ergm(x, ...)
## S3 method for class 'matrix':
edgelist.ergm(x, directed=TRUE, check.uniqueness=TRUE,
              check.sorted=TRUE, ...)
```

Arguments

<code>x</code>	an R object. Either a network or a matrix of some sort.
<code>directed</code>	logical: Are the edges directed? If <code>x</code> is a square matrix with <code>NROWS(x)!=2</code> , equal to <code>!all(x==t(x))</code>
<code>check.uniqueness</code>	An edgelist should have unique rows. By default, therefore, uniqueness is checked. However, if the input is known to have unique rows already, then setting this option to <code>FALSE</code> will save computing time
<code>check.sorted</code>	An edgelist should have its rows sorted in dictionary order. By default, therefore, the matrix is checked to see whether it is sorted. However, if the input is known to be sorted already, then setting this option to <code>FALSE</code> will save computing time.
<code>...</code>	Additional arguments, to be passed to lower-level functions in the future.

Details

The function takes a network or an (adjacency or edgelist) matrix, then returns an edgelist (of class "matrix") in standard format. If x is a network, the value returned equals `edgelist.ergm(as.matrix(x), "edgelist"), directed=is.directed(x)`.

The standard format is as follows:

1. The matrix has two columns
2. No row has two identical entries
3. Each row is unique
4. The rows are in dictionary order: They are sorted by the first column, then by the second in case of ties
5. If `directed=TRUE`, the element in the first column is always smaller than the element in the second column (otherwise, the entries in that row are switched before sorting).

Value

A matrix with two rows, in the format described under "Details"

See Also

`as.matrix.network`

ergm

Exponential Family Random Graph Models

Description

`ergm` is used to fit linear exponential random network models, in which the probability of a given network, y , on a set of nodes is $\exp(\theta \cdot g(y)) / c(\theta)$, where $g(y)$ is a vector of network statistics, θ is a parameter vector of the same length and $c(\theta)$ is the normalizing constant for the distribution. `ergm` can return either a maximum pseudo-likelihood estimate or an approximate maximum likelihood estimator based on a Monte Carlo scheme.

Usage

```
ergm(formula, theta0="MPLE",
      MPLEonly=FALSE, MLEestimate=!MPLEonly, seed=NULL,
      burnin=10000, MCMCsamplesize=10000, interval=100, maxit=3,
      constraints=~.,
      meanstats = NULL,
      control=control.ergm(),
      verbose=FALSE, ...)
```

Arguments

<code>formula</code>	formula; an R <code>formula</code> object, of the form $y \sim \langle \text{model terms} \rangle$, where y is a <code>network</code> object or a matrix that can be coerced to a <code>network</code> object. For the details on the possible $\langle \text{model terms} \rangle$, see <code>ergm-terms</code> and Morris, Handcock and Hunter (2008). To create a <code>network</code> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> operator if necessary.
<code>theta0</code>	vector; the parameter value used to generate the MCMC sample and as a starting value for the estimation. By default the MPLE is used (<code>theta0="MPLE"</code>).
<code>MPLEonly</code>	logical; TRUE if the maximum pseudo-likelihood estimate is to be computed and returned. Note that <code>MPLEonly=TRUE</code> will render moot most other parameters in this list.
<code>MLEestimate</code>	logical; TRUE if only the Monte Carlo maximum likelihood estimate is to be computed and returned.
<code>burnin</code>	count; the number of proposals before any MCMC sampling is done. It typically is set to a fairly large number.
<code>MCMCsamplesize</code>	count; the number of network statistics, randomly drawn from a given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm.
<code>interval</code>	count; the number of proposals between sampled statistics.
<code>maxit</code>	count; the 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.
<code>constraints</code>	<p>A one-sided formula specifying one or more constraints on the support of the distribution of the networks being modeled, using syntax similar to the <code>formula</code> argument. Multiple constraints may be given, separated by “+” operators. Together with the model terms in the formula, the constraints define the distribution of networks being modeled.</p> <p>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 <code>prop.args</code> argument to <code>control.ergm</code>.</p> <p>The default is <code>~.</code>, for an unconstrained model.</p> <p>The constraint terms currently implemented are</p> <ul style="list-style-type: none"> . or NULL A placeholder for no constraints: all networks of a particular size and type have non-zero probability. Cannot be combined with other constraints. bd(<code>attribs</code>, <code>maxout</code>, <code>maxin</code>, <code>minout</code>, <code>minin</code>) Constrain maximum and minimum vertex degree. See “Placing Bounds on Degrees” section for more information. degrees and nodedegrees Preserve the degree of each vertex of the given network: only networks whose vertex degrees are the same as those in the network passed in the model formula have non-zero probability. degreedist Preserve the degree distribution of the given network: only networks whose degree distributions are the same as those in the network passed in the model formula have non-zero probability.

	indegreedist and outdegreedist Preserve the (respectively) indegree or outdegree distribution of the given network.
	edges Preserve the edge count of the given network: only networks having the same number of edges as the network passed in the model formula have non-zero probability.
	observed Preserve the observed dyads of the given network.
	Not all combinations of the above are supported.
meanstats	vector; the mean-value parameter value for the model. If this is given the algorithm finds the natural parameter value corresponding to this mean-value parameter. If it is missing then mean-value parameter used are the observed statistics of the network in the formula.
control	A list of control parameters for algorithm tuning. Constructed using <code>control.ergm</code> .
seed	integer; random number integer seed. Defaults to NULL to use whatever the state of the random number generator is at the time of the call.
verbose	logical; if this is TRUE, the program will print out additional information, including goodness of fit statistics.
...	Additional arguments, to be passed to lower-level functions in the future.

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, that is used in the maximum likelihood estimation routine.
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, <code>sample</code> is roughly a random sample from the distribution of network statistics specified by the model with the parameter equal to <code>MCMCtheta</code> . If <code>MPLEonly=TRUE</code> then <code>MCMCtheta</code> 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.
samplesize	The size of the MCMC sample
failure	Logical: Did the MCMC estimation fail?
mc.se	MCMC standard error estimates
newnetwork	The final network at the end of the MCMC simulation

<code>burnin</code>	If included, the burnin used for the MCMC simulation
<code>interval</code>	If included, the interval used for the MCMC simulation
<code>network</code>	Original network
<code>theta.original</code>	The first value of <code>theta0</code>
<code>mplefit</code>	The MPLE fit as a <code>glm</code> object.
<code>null.deviance</code>	Deviance of the null model.
<code>mle.lik</code>	The approximate log-likelihood for the MLE. The value is only approximate because it is estimated based on the MCMC random sample.
<code>etamap</code>	The set of functions mapping the true parameter <code>theta</code> to the canonical parameter <code>eta</code> (irrelevant except in a curved exponential family model)
<code>degeneracy.value</code>	Score calculated to assess the degree of degeneracy in the model.
<code>degeneracy.type</code>	Supporting output for <code>degeneracy.value</code> . Mainly for internal use.
<code>formula</code>	The original <code>formula</code> entered into the <code>ergm</code> function.
<code>constraints</code>	Constraints used by original <code>ergm</code> call
<code>prop.weights</code>	MCMC proposal weights used by original <code>ergm</code> call (part of the <code>control.ergm</code> function output).
<code>offset</code>	vector of logical telling which model parameters are to be set at a fixed value (i.e., not estimated).
<code>drop</code>	list of terms that were dropped due to extreme values of the corresponding statistics on the observed network.

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

Model Terms

The `ergm` function allows the user to explore a large number of potential models for their network data. The terms currently supported by the program, and a brief description of each is given in the documentation `ergm-terms`. In the `formula` for the model, the model terms are various function-like calls, some of which require arguments, separated by `+` signs. For a more detailed understanding of the model terms, see and Morris, Handcock and Hunter (2008).

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 default mechanism for proposing new networks for the MCMC sample space is the Metropolis-Hastings algorithm, which simply chooses a dyad at random and proposes to toggle that edge; each possible dyad is equally likely. The `proposaltype` option allow many more complex proposals to be specified. We have developed and implemented a wide range of algorithms. These are described in the documentation for `proposaltype`. For example, we have included proposal functions that condition on maintaining the absolute degree distribution for the observed network. Each proposal network will have exactly the same number of nodes with each degree as does the original network; this means that if the proposal network removes an edge between a node of degree 3 and a node of degree 5, it must also add an edge between a node of degree 2 and a node of degree 4. Note that one or both of the latter nodes may be the same as the former nodes.

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

Placing Bounds on Degrees:

There are many times when one may wish to condition on the number of inedges or outedges possessed by a node, either as a consequence of some intrinsic property of that node (e.g., to control for activity or popularity processes), to account for known outliers of some kind, and thus we wish to limit its indegree, an intrinsic property of the sampling scheme whence came our data (e.g., the survey asked everyone to name only three friends total) or as a function of the attributes of the nodes to which a node has edges (e.g., we specify that nodes designated “male” have a maximum number of outdegrees to nodes designated “female”). To accomplish this we use the `constraints` term `bd`.

Let’s consider the simple cases first. Suppose you want to condition on the total number of degrees regardless of attributes. That is, if you had a survey that asked respondents to name three alters and no more, then you might want to limit your maximal outdegree to three without regard to any of the alters’ attributes. The argument is then:

```
constraints=~bd(maxout=3)
```

Similar calls are used to restrict the number of indegrees (`maxin`), the minimum number of outdegrees (`minout`), and the minimum number of indegrees (`minin`).

You can also set ego specific limits. For example:

```
constraints=bd(maxout=rep(c(3,4),c(36,35)))
```

limits the first 36 to 3 and the other 35 to 4 outdegrees.

Multiple restrictions can be combined. `bd` is very flexible. In general, the `bd` term can contain up to five arguments:

```
bd(attrs=attrs,
    maxout=maxout,
    maxin=maxin,
    minout=minout,
    minin=minin)
```

Omitted arguments are unrestricted, and arguments of length 1 are replicated out to all nodes (as above). If an individual entry in `maxout`,..., `minin` is `NA` then no restriction of that kind is applied to that actor.

In general, `attribs` is a matrix of the attributes on which we are conditioning. The dimensions of `attribs` are `n_nodes` rows by `attrcount` columns, where `attrcount` is the number of distinct attribute values on which we want to condition (i.e., a separate column is required for “male” and “female” if we want to condition on the number of ties to both “male” and “female” partners). The value of `attribs[n, i]`, therefore, is `TRUE` if node `n` has attribute value `i`, and `FALSE` otherwise. (Note that, since each column represents only a single value of a single attribute, the values of this matrix are all Boolean (`TRUE` or `FALSE`)). It is important to note that `attribs` is a matrix of nodal attributes, not alter attributes.

So, for instance, if we wanted to construct an `attribs` matrix with two columns, one each for male and female attribute values (we are conditioning on these values of the attribute “sex”), and the attribute `sex` is represented in `ads.sex` as an `n_node`-long vector of 0s and 1s (men and women), then our code would look as follows:

```
# male column: bit vector, TRUE for males
attrsex1 <- (ads.sex == 0)
# female column: bit vector, TRUE for females
attrsex2 <- (ads.sex == 1)
# now create attribs matrix
attribs <- matrix(ncol=2, nrow=71, data=c(attrsex1, attrsex2))
```

`maxout` is a matrix of alter attributes, with the same dimensions as the `attribs` matrix. `maxout` is `n_nodes` rows by `attrcount` columns. The value of `maxout[n, i]`, therefore, is the maximum number of outdegrees permitted from node `n` to nodes with the attribute `i` (where a `NA` means there is no maximum).

For example: if we wanted to create a `maxout` matrix to work with our `attribs` matrix above, with a maximum from every node of five outdegrees to males and five outdegrees to females, our code would look like this:

```
# every node has maximum of 5 outdegrees to male alters
maxoutsex1 <- c(rep(5, 71))
# every node has maximum of 5 outdegrees to female alters
maxoutsex2 <- c(rep(5, 71))
# now create maxout matrix
maxout <- cbind(maxoutsex1, maxoutsex2)
```

The `maxin`, `minout`, and `minin` matrices are constructed exactly like the `maxout` matrix, except for the maximum allowed indegree, the minimum allowed outdegree, and the minimum allowed indegree, respectively. Note that in an undirected network, we only look at the outdegree matrices; `maxin` and `minin` will both be ignored in this case.

```
attribs[n][0] = 1 # just the ego values
maxout[n][0] = minout[n][0] = observed outdegree of n in network
maxin[n][0] = minin[n][0] = observed indegree of n in network
```

References

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

- Admiraal R, Handcock MS (2007). **networksis**: Simulate bipartite graphs with fixed marginals through sequential importance sampling. Statnet Project, Seattle, WA. Version 1. <http://statnetproject.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). <http://www.jstatsoft.org/v24/i07/>.
- 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 (2006). **netperm**: Permutation Models for Relational Data. Version 0.2, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2007). **sna**: Tools for Social Network Analysis. Version 1.5, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2008). **network**: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). <http://www.jstatsoft.org/v24/i02/>.
- Butts CT, with help~from David~Hunter, Handcock MS (2007). **network**: Classes for Relational Data. Version 1.2, <http://erzuli.ss.uci.edu/R.stuff>.
- Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). A **statnet** Tutorial. *Journal of Statistical Software*, 24(8). <http://www.jstatsoft.org/v24/i08/>.
- 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. www.csss.washington.edu/Papers/wp39.pdf
- Handcock MS (2003b). **degreenet**: Models for Skewed Count Distributions Relevant to Networks. Statnet Project, Seattle, WA. Version 1.0, <http://statnetproject.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 2, <http://statnetproject.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, <http://statnetproject.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). <http://www.jstatsoft.org/v24/i03/>.
- Krivitsky PN, Handcock MS (2007). **latentnet**: Latent position and cluster models for statistical networks. Seattle, WA. Version 2, <http://statnetproject.org>.
- Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). <http://www.jstatsoft.org/v24/i04/>.

See Also

network, %v%, %n%, ergm-terms, summary.ergm, print.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="wealth", 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(1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3)
#
# 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"),
  MCMCsamplesize=10000)
summary(gest)

# compare it to differential homophily by atomic type
gest <- ergm(molecule ~ edges + kstar(2) + triangle
  + nodematch("atomic type",diff=TRUE),
  MCMCsamplesize=10000)
summary(gest)

```

Description

The function `ergm` is used to fit linear exponential random graph models, in which the probability of a given network, y , on a set of nodes is $\exp\{\theta \cdot g(y)\} / c(\theta)$, where $g(y)$ is a vector of network statistics for y , θ is a parameter vector of the same length and $c(\theta)$ is the normalizing constant for the distribution.

The network statistics $g(y)$ are entered as terms in the function call to `ergm`.

This page describes the possible terms (and hence network statistics).

Specifying models

Terms to `ergm` are specified by a formula to represent the network and network statistics. This is done via a formula, that is, an R formula object, of the form $y \sim \langle \text{term 1} \rangle + \langle \text{term 2} \rangle \dots$, where y is a network object or a matrix that can be coerced to a network object, and $\langle \text{term 1} \rangle$, $\langle \text{term 2} \rangle$, etc, are each terms chosen from the list given below. To create a network object in R, use the `network` function, then add nodal attributes to it using the `%v%` operator if necessary.

Possible terms to represent network statistics

The `ergm` function allows the user to explore a large number of potential models for their network data. What follows is a list of model terms currently available by the program, and a brief description of each. In the formula for the model, the model terms are various function-like calls, some of which require arguments, separated by + signs.

Additional terms can be coded up by users via the `statnetuserterms` package.

The terms currently available are:

- absdiff(attrname, pow=1)** *Absolute difference*: The `attrname` argument is a character string giving the name of a quantitative attribute in the network's vertex attribute list. This term adds one network statistic to the model equaling the sum of $\text{abs}(\text{attrname}[i] - \text{attrname}[j])^{\text{pow}}$ for all edges (i,j) in the network.
- absdiffcat(attrname, base=NULL)** *Categorical absolute difference*: The `attrname` argument is a character string giving the name of a quantitative attribute in the network's vertex attribute list. This term adds one statistic for every possible nonzero distinct value of $\text{abs}(\text{attrname}[i] - \text{attrname}[j])$ in the network; the value of each such statistic is the number of edges in the network with the corresponding absolute difference. The optional `base` argument is a vector indicating which nonzero differences, in order from smallest to largest, should be omitted from the model (i.e., treated like the zero-difference category). The `base` argument, if used, should contain indices, not differences themselves. For instance, if the possible values of $\text{abs}(\text{attrname}[i] - \text{attrname}[j])$ are 0, 0.5, 3, 3.5, and 10, then to omit 0.5 and 10 one should set `base=c(1, 4)`. Note that this term should generally be used only when the quantitative attribute has a limited number of possible values; an example is the "Grade" attribute of the `faux.mesa.high` or `faux.magnolia.high` datasets.
- altkstar(lambda, fixed=FALSE)** *Alternating k-star*: This term adds one network statistic to the model equal to a weighted alternating sequence of k-star statistics with weight parameter `lambda`. 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. The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with undirected networks.
- asymmetric(attrname=NULL, diff=FALSE, keep=NULL)** *Asymmetric dyads*: 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. This term can only be used with directed networks. If the optional `attrname` argument is used, only asymmetric pairs that match on the named vertex attribute are counted. The optional modifiers `diff` and `keep` are used in the same way as for the `nodematch` term; refer to this term for details and an example.
- blconcurrent(by=NULL)** *Concurrent node count for the first mode in a bipartite (aka two-mode) network*: 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. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list; it functions

just like the `by` argument of the `bldegree` term. This term can only be used with undirected bipartite networks.

bldegree(d, by=NULL) *Degree for the first mode in a bipartite (aka two-mode) network:*

The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the i th such statistic equals the number of nodes of degree `d[i]` in the first mode of a bipartite network, i.e. with exactly `d[i]` edges. The first mode of a bipartite network object is sometimes known as the "actor" mode. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then each node's degree is tabulated only with other nodes having the same value of the `by` attribute. This term can only be used with undirected bipartite networks.

blfactor(attrname, base=1) *Factor attribute effect for the first mode in a bipartite (aka two-mode) network :*

The `attrname` argument is a character string giving the name of a categorical attribute in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` 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. 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`. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, `base=1`, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the "fruit" factor has levels "orange", "apple", "banana", and "pear", then to add just two terms, one for "apple" and one for "pear", then set "banana" and "orange" to the base (remember to sort the values first) by using `nodefactor("fruit", base=2:3)`. This term can only be used with undirected bipartite networks.

blstar(k, attrname=NULL) *k-Stars for the first mode in a bipartite (aka two-mode) network:*

The `k` argument is a vector of distinct integers. 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 $\{O_1, \dots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \dots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this 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. Note that `blstar(1)` is equal to `b2star(1)` and to `edges`.

blstarmix(k, attrname, base=NULL, diff=TRUE) *Mixing matrix for k-stars centered on the first mode of a bipartite network:*

Only a single value of k is allowed. 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 `attrname`. 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 called `attrname`). A different statistic is created for each value of `attrname` seen in a `b1` node, even if no k -stars are observed with this value. Whether a different statistic is created for each value seen in a `b2` node depends on the value of the `diff` argument: 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. The `base` term may be used to control which of the possible terms are left out of the model: By default, all terms are included, but if `base` is set to a vector of indices then the corresponding terms (in the order they would be created when `base=NULL`) are left out.

- b1twostar(b1attrname, b2attrname, base=NULL)** *Two-star census for central nodes centered on the first mode of a bipartite network:* This term takes two nodal attribute names, one for `b1` nodes (actors in some contexts) and one for `b2` nodes (events in some contexts). Only `b1attrname` is required; if `b2attrname` is not passed, it is assumed to be the same as `b1attrname`. Assuming that there are n_1 values of `b1attrname` among the `b1` nodes and n_2 values of `b2attrname` among the `b2` nodes, then the total number of distinct categories of two stars according to these two attributes is $n_1(n_2)(n_2 + 1)/2$. This model term creates a distinct statistic counting each of these categories. The `base` term may be used to leave some of these categories out; when passed as a vector of integer indices (in the order the statistics would be created when `base=NULL`), the corresponding terms will be left out.
- b2concurrent(by=NULL)** *Concurrent node count for the second mode in a bipartite (aka two-mode) network:* 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. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list; it functions just like the `by` argument of the `b2degree` term. This term can only be used with undirected bipartite networks.
- b2degree(d, by=NULL)** *Degree for the second mode in a bipartite (aka two-mode) network:* The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the i th such statistic equals the number of nodes of degree `d[i]` in the second mode of a bipartite network, i.e. with exactly `d[i]` edges. The second mode of a bipartite network object is sometimes known as the "event" mode. The optional term `by` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then each node's degree is tabulated only with other nodes having the same value of the `by` attribute. This term can only be used with undirected bipartite networks.
- b2factor(attrname, base=1)** *Factor attribute effect for the second mode in a bipartite (aka two-mode) network :* The `attrname` argument is a character string giving the name of a categorical attribute in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` 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. 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`. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, `base=1`, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the "fruit" factor has levels "orange", "apple", "banana", and "pear", then to add just two terms, one for "apple" and one for "pear", then set "banana" and "orange" to the base (remember to sort the values first) by using `nodefactor("fruit", base=2:3)`. This term can only be used with undirected bipartite networks.

- b2star(k, attrname=NULL)** *k-Stars for the second mode in a bipartite (aka two-mode) network*: The `k` argument is a vector of distinct integers. 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 $\{O_1, \dots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \dots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified 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. This term can only be used for undirected bipartite networks. Note that `b2star(1)` is equal to `b1star(1)` and to `edges`.
- b2starmix(k, attrname, base=NULL, diff=TRUE)** *Mixing matrix for k-stars centered on the second mode of a bipartite network*: This term is exactly the same as `b1starmix` except that the roles of `b1` and `b2` are reversed.
- b2twestar(b1attrname, b2attrname, base=NULL)** *Two-star census for central nodes centered on the second mode of a bipartite network*: This term is exactly the same as `b1twestar` except that the roles of `b1` and `b2` are reversed.
- balance** *Balanced triads*: 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 even number of ties (i.e., 0 and 2).
- concurrent(by=NULL)** *Concurrent node count*: This term adds one network statistic to the model, equal to the number of nodes in the network with degree 2 or higher. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list; it functions just like the `by` argument of the `degree` term. This term can only be used with undirected networks.
- ctriple(attrname=NULL)** *Cyclic triples*: 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 $\{(i \rightarrow j), (j \rightarrow k), (k \rightarrow i)\}$. Note that for all directed networks, `triangle` is equal to `ttriple+ctriple`, so at most two of these three terms can be in a model. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is over the number of cyclic triples where all three nodes have the same value of the attribute. This term can only be used with directed networks.
- cycle(k)** *Cycles*: The `k` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `k`; the i th such statistic equals the number of cycles in the network with length exactly `k[i]`. The cycle statistic applies to both directed and undirected networks. For directed networks, it counts directed cycles of length k , as opposed to undirected cycles in the undirected case. The directed cycle terms of lengths 2 and 3 are equivalent to `mutual` and `ctriple` (respectively). The undirected cycle term of length 3 is equivalent to `triangle`, and there is no undirected cycle term of length 2.
- degree(d, by=NULL)** *Degree*: The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the i th such statistic equals the number of nodes in the network of degree `d[i]`, i.e. with exactly `d[i]` edges. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then each node's degree is tabulated only with other

nodes having the same value of the `by` attribute. This term can only be used with undirected networks; for directed networks see `idegree` and `odegree`.

density *Density*: 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 $n(n-1)$.

dsp(d) *Dyadwise shared partners*: The `d` argument is a vector of distinct integers. 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. For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the dyad).

dyadcov(x, attrname=NULL) *Dyadic covariate*: If the network is directed, `x` is either a (symmetric) matrix of covariates, one for each possible dyad (i, j) , or an undirected 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). 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 `edgcov` and `dyadcov` terms are equivalent for undirected networks.

edgcov(x, attrname=NULL) *Edge covariate*: The `x` argument is either a square matrix of covariates, one for each possible edge in the network, 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). This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The `edgcov` term applies to both directed and undirected networks. For undirected networks the covariates are also assumed to be undirected. The `edgcov` and `dyadcov` terms are equivalent for undirected networks.

edges *Edges*: This term adds one network statistic equal to the number of edges 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)`.

esp(d) *Edgewise shared partners*: 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. For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the edge and in the same direction).

gwbldegree(decay, fixed=FALSE) *Geometrically weighted degree distribution for the first mode in a bipartite (aka two-mode) network*: This term adds one network statistic to the model equal to the weighted degree distribution with weight parameter `decay`, 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 statistic is based on the version given as equation (14) in Hunter (2007). See the "Remark" in section 3 of that paper to see why it is used rather than

the version given in Snijders et al. (2006). The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with undirected bipartite networks.

gwb2degree(decay, fixed=FALSE) *Geometrically weighted degree distribution for the second mode in a bipartite (aka two-mode) network:* This term adds one network statistic to the model equal to the weighted degree distribution with weight parameter `decay`, for nodes in the second mode of a bipartite network. The second mode of a bipartite network object is sometimes known as the "event" mode. This statistic is based on the version given as equation (14) in Hunter (2007). See the "Remark" in section 3 of that paper to see why it is used rather than the version given in Snijders et al. (2006). The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with undirected bipartite networks.

gwdegree(decay, fixed=FALSE) *Geometrically weighted degree distribution:* This term adds one network statistic to the model equal to the weighted degree distribution with weight parameter `decay`. This is the version given as equation (14) in Hunter (2007). See the "Remark" in section 3 of that paper to see why it is used rather than the version given in Snijders et al. (2006). The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with undirected networks.

gwdsp(alpha, fixed=FALSE) *Geometrically weighted dyadwise shared partner distribution:* This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with weight parameter `alpha` > 0 . The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can be used with directed and undirected networks. For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the dyad).

gwesp(alpha, fixed=FALSE) *Geometrically weighted edgewise shared partner distribution:* This term is just like `gwdsp` except it adds a statistic equal to the geometrically weighted *edgewise* (not *dyadwise*) shared partner distribution with weight parameter `alpha`. The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can be used with directed and undirected networks. For directed networks the geometric weighting is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the edge and in the same direction).

gwdegree(decay, fixed=FALSE) *Geometrically weighted in-degree distribution:* This term adds one network statistic to the model equal to the weighted in-degree distribution with weight parameter `decay`. The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with directed networks.

gwmsp(alpha, fixed=FALSE) *Geometrically weighted nonedgewise shared partner distribution*: 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 `alpha`. The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can be used with directed and undirected networks. For directed networks the geometric weighting is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the non-edge and in the same direction).

gdegree(decay, fixed=FALSE) *Geometrically weighted out-degree distribution*: This term adds one network statistic to the model equal to the weighted out-degree distribution with weight parameter `decay`. The optional argument `fixed` indicates whether the scale parameter `lambda` 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. This term can only be used with directed networks.

hamming(x, cov, attrname=NULL) *Hamming distance*: 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`. (If no argument is given, `x` is taken to be the observed network, i.e., the network on the left side of the \sim in the formula that defines the ERGM.) 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. The argument `cov` is either a matrix of edgewise weights or a network; if the latter, the optional argument `attrname` provides the name of the edge attribute to use for weight values.

hammingmix(attrname, x, base=0) *Hamming distance within mixing*: This term adds one statistic to the model for every possible pairing of attribute values of the network. Each such statistic is the Hamming distance (i.e., the number of differences) between the appropriate subset of dyads in the network and the corresponding subset in `x`. The ordering of the attribute values is alphabetical. The option `base` gives the index of statistics to be omitted from the tabulation. For example `base=2` will omit the second statistic, making it the de facto reference category. This term can only be used with directed networks.

idegree(d, by=NULL) *In-degree*: The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the 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. The optional term `by` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then each node's degree is tabulated only with other nodes having the same value of the `by` attribute. This term can only be used with directed networks; for undirected networks see `degree`.

intransitive *Intransitive triads*: This term adds one statistic to the model, equal to the number of triads in the network that are intransitive. 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. This term can only be used with directed networks.

isolates *Isolates*: 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.

istar(k, attrname=NULL) *In-stars*: The `k` argument is a vector of distinct integers. 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 $\{O_1, \dots, O_k\}$ such that the ties $(O_j \rightarrow N)$ exist for $j = 1, \dots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this 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`.

kstar(k, attrname=NULL) *k-Stars*: The `k` argument is a vector of distinct integers. 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 $\{O_1, \dots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \dots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. 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`.

localtriangle(x) *Triangles within neighborhoods*: 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 $(i \rightarrow j), (j \rightarrow k)$ and either $(k \rightarrow i)$ or $(k \leftarrow i)$ where again all nodes are within the same neighborhood. The argument `x` is a network or an 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`.

m2star *Mixed 2-stars, a.k.a 2-paths*: 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`.

match(attrname, diff=FALSE, keep=NULL) *Uniform homophily and differential homophily*: This is an alias for `nodematch(attrname, diff=FALSE)`.

meandeg *Mean vertex degree*: 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`.

mutual(attrname=NULL, diff=FALSE, keep=NULL) *Mutuality*: This term adds one network statistic to the model equal to the number of pairs of actors i and j for which $(i \rightarrow j)$ and $(j \rightarrow i)$ both exist. This term can only be used with directed networks. If the optional `attrname` argument is used, only mutual pairs that match on the named vertex attribute are counted. The optional modifiers `diff` and `keep` are used in the same way as for the `nodematch` term; refer to this term for details and an example.

nearsimmelian *Near simmelian triads*: 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. This term can only be used with directed networks.

nodecov(attrname) *Main effect of a covariate*: The `attrname` argument is a character string giving the name of a numeric (not categorical) attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the sum of `attrname(i)` and `attrname(j)` for all edges (i, j) in the network. For categorical attributes, see `nodefactor`. Note that for directed networks, `nodecov` equals `nodeicov` plus `nodeocov`.

nodefactor(attrname, base=1) *Factor attribute effect*: The `attrname` argument is a character vector giving one or more names of categorical attributes in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` 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. In particular, for edges whose endpoints both have the same attribute values, this value is counted twice. To include all attribute values is usually not a good idea – though this may be accomplished if desired by setting `base=0` – because the sum of all such statistics equals twice the number of edges and hence a linear dependency would arise in any model also including edges. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, `base=1`, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the “fruit” factor has levels “orange”, “apple”, “banana”, and “pear”, then to add just two terms, one for “apple” and one for “pear”, then set “banana” and “orange” to the base (remember to sort the values first) by using `nodefactor("fruit", base=2:3)`. For an analogous term for quantitative vertex attributes, see `nodecov`.

nodeicov(attrname) *Main effect of a covariate for in-edges*: The `attrname` argument is a character string giving the name of a numeric (not categorical) attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the total value of `attrname(j)` for all edges (i, j) in the network. This term may only be used with directed networks. For categorical attributes, see `nodeifactor`.

nodeifactor(attrname, base=1) *Factor attribute effect for in-edges*: The `attrname` argument is a character vector giving one or more names of a categorical attribute in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` 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. To include all attribute values is usually not a good idea – though this may be accomplished if desired by setting `base=0` – 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. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, `base=1`, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the “fruit” factor has levels “orange”, “apple”, “banana”, and “pear”, then to add just two terms, one for “apple” and one for “pear”, then set “banana” and “orange” to the base (remember to sort the values first) by using `nodeifactor("fruit", base=2:3)`. For an analogous term for quantitative vertex attributes, see `nodeicov`.

nodematch(attrname, diff=FALSE, keep=NULL) *Uniform homophily and differential*

homophily: The `attrname` argument is a character vector giving one or more names of attributes in the network's vertex attribute list. When `diff=FALSE`, this term adds one network statistic to the model, which counts the number of edges (i, j) for which `attrname(i) == attrname(j)`. (When multiple names are given, the statistic counts only those on which all the named attributes match.) When `diff=TRUE`, p network statistics are added to the model, where p is the number of unique values of the `attrname` attribute. The k th such statistic counts the number of edges (i, j) for which `attrname(i) == attrname(j) == value(k)`, where `value(k)` is the k th smallest unique value of the `attrname` attribute. If set to `non-NULL`, the optional `keep` argument should be a vector of integers giving the values of k that should be considered for matches; other values are ignored (this works for both `diff=FALSE` and `diff=TRUE`). For instance, to add two statistics, counting the matches for just the 2nd and 4th categories, use `nodematch` with `diff=TRUE` and `keep=c(2, 4)`.

nodemix(attrname, base=NULL) *Nodal attribute mixing*: The `attrname` argument is a character vector giving the names of categorical attributes in the network's vertex attribute list. 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 names are given, a statistic is added for each combination of attribute values for those names.) In other words, this term produces one statistic for every entry in the mixing matrix for the attribute(s). The ordering of the attribute values is alphabetical (for nominal categories) or numerical (for ordered categories). The optional `base` argument is a vector of integers corresponding to the pairings that should not be included. If `base` contains only negative integers, then these integers correspond to the only pairings that should be included. By default (i.e., with `base=NULL` or `base=0`), all pairings are included.

nodecov(attrname) *Main effect of a covariate for out-edges*: The `attrname` argument is a character string giving the name of a numeric (not categorical) attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the total value of `attrname(i)` for all edges (i, j) in the network. This term may only be used with directed networks. For categorical attributes, see `nodeofactor`.

nodeofactor(attrname, base=1) *Factor attribute effect for out-edges*: The `attrname` argument is a character string giving one or more names of categorical attributes in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` 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. To include all attribute values is usually not a good idea – though this may be accomplished if desired by setting `base=0` – 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`. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, `base=1`, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the “fruit” factor has levels “orange”, “apple”, “banana”, and “pear”, then to add just two terms, one for “apple” and one for “pear”, then set “banana” and “orange” to the `base` (remember to sort the values first) by using `nodeofactor("fruit", base=2:3)`. For an analogous term for quantitative vertex attributes, see `nodecov`.

nsp(d) *Nonedgewise shared partners*: 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.

For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the non-edge and in the same direction).

odegree(d, by=NULL) *Out-degree*: The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the 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. The optional argument `by` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then each node's degree is tabulated only with other nodes having the same value of the `by` attribute. This term can only be used with directed networks; for undirected networks see `degree`.

ostar(k, attrname=NULL) *k-Outstars*: The `k` argument is a vector of distinct integers. 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 $\{O_1, \dots, O_k\}$ such that the ties $(N \rightarrow O_j)$ exist for $j = 1, \dots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this 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`. Note that `ostar(1)` is equal to both `istar(1)` and `edges`.

receiver(base=1) *Receiver effect*: 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). The `base` argument allows the user to determine which nodes' statistics should be omitted. The `base` argument can also be a vector of negative indices, to specify which should be added instead of deleted, and `base=0` specifies that all statistics should be included. This term can only be used with directed networks. For undirected networks, see `sociality`.

sender(base=1) *Sender effect*: 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). The `base` argument allows the user to determine which nodes' statistics should be omitted. The `base` argument can also be a vector of negative indices, to specify which should be added instead of deleted, and `base=0` specifies that all statistics should be included. This term can only be used with directed networks. For undirected networks, see `sociality`.

simmelian *Simmelian triads*: 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. This term can only be used with directed networks.

simmelianties *Ties in simmelian triads*: 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

Simmeliants (given the number of Simmelians). This term can only be used with directed networks.

sociality(attrname=NULL, base=1) *Undirected degree*: This term adds one network statistic for each node equal to the number of ties of that node. The optional `attrname` argument is a character string giving the name of an attribute in the network's vertex attribute list that takes categorical values. If provided, this term only counts ties between nodes with the same value of the attribute (an actor-specific version of the `nodematch` term). This term can only be used with undirected networks. For directed networks, see `sender` and `receiver`. By default, `base=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 `sender` and `receiver` terms.

threepath(keep=1:4) *Three-paths*: For an undirected network, this term adds one statistic equal to the number of threepaths, where a threepath is defined as a path of length three that traverses three distinct edges. Note that a threepath need not include four distinct nodes; in particular, a triangle counts as three threepaths. For a directed network, this term adds four statistics (or some subset of these four specified by the `keep` argument), 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 threepath is of the form $i \rightarrow j \rightarrow k \rightarrow l$, and RRL threepath is of the form $i \rightarrow j \rightarrow k \leftarrow l$, etc. Like in the undirected case, there is no requirement that the nodes be distinct in a directed threepath. However, the three edges must all be distinct. Thus, a mutual tie $i \leftrightarrow j$ does not count as a threepath of the form $i \rightarrow j \rightarrow i \leftarrow j$; however, in the subnetwork $i \leftrightarrow j \rightarrow k$, there are two directed threepaths, one LRR ($k \leftarrow j \rightarrow i \leftarrow j$) and one RRR ($j \rightarrow i \rightarrow j \leftarrow k$).

transitive *Transitive triads*: 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.

triadcensus(d) *Triad census*: 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, 030T, 030C, 201, 120D, 120U, 120C, 210, 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. By specifying a numeric vector of integers from 0 to 15 as the `d` argument, the user may specify a set of terms to add other than the default value of 1:15. 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), and the default is to add 1:3, which is to say that the 0 is dropped; however, this too may be controlled by changing the `d` argument to a numeric vector giving a subset of $\{0, 1, 2, 3\}$.

triangle(attrname=NULL) *Triangles*: 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 `ctruple` — thus at most two of these three terms can be in a model. The optional argument `attrname` restricts the count to those triples of nodes with equal values of the vertex attribute specified by `attrname`.

`tripercent(attrname=NULL)` *Triangle percentage*: 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). For the definition of `triangle`, see `triangle`. The optional argument `attrname` restricts the counts (both numerator and denominator) to those triples of nodes with equal values of the vertex attribute specified by `attrname`. 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.

`ttriple(attrname=NULL)` *Transitive triples*: This term adds one statistic to the model, equal to the number of transitive triples in the network, defined as a set of edges $\{(i \rightarrow j), (j \rightarrow k), (i \rightarrow k)\}$. Note that `triangle` equals `ttriple`+`ctruple` for a directed network, so at most two of the three terms can be in a model. The optional argument `attrname` is a character string giving the name of an attribute in the network’s vertex attribute list. If this is specified then the count is over the number of transitive triples where all three nodes have the same value of the attribute. This term can only be used with directed networks.

`twopath` *2-Paths*: 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.

References

- 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.
- 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`, `network`, `%v%`, `%n%`, `sna`, `summary.ergm`, `print.ergm`

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)
```

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 communities 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 <http://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* <http://statnetproject.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), burnin=10000,
interval=1000, MCMCsamplesize=2500, maxit=25,
control=control.ergm(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, burnin=100000000,
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.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 community 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 <http://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* <http://statnetproject.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=T) + nodematch("Race",diff=T)
+ nodematch("Sex",diff=F) + gwdegree(1.0, fixed=T)
+ gwesp(1.0, fixed=T) + gwdisp(1.0, fixed=T)
```

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](#)

flobusiness

Florentine Family Business Ties Data as a "network" object

Description

This is a data set of business ties among Renaissance Florentine families. The data is originally from Padgett (1994) via UCINET and stored as a `network` object.

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 relations are business ties (`flobusiness` - specifically, recorded financial ties such as loans, credits and joint partnerships).

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. Vertex information is provided (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 Strozzi (15).

Usage

```
data(florentine)
```

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](#), [flomarriage](#)

`flomarrriage`*Florentine Family Marriage Ties Data as a "network" object*

Description

This is a data set of marriage ties among Renaissance Florentine families. The data is originally from Padgett (1994) via UCINET and stored as a `network` object.

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 relations are marriage alliances (`flomarrriage` between the families).

As Breiger & Pattison point out, the original data are symmetrically coded. This is perhaps acceptable perhaps for marital ties. Vertex information is provided 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 Strozzi (15).

Usage

```
data(florentine)
```

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

`flobusiness`, `flo`, `network`, `plot.network`, `ergm`

`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.

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 Strozzi (15).

Usage

```
data(florentine)
```

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.

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.

Usage

```
data(g4)
```

Source

Steve Goodreau

See Also

florentine, network, plot.network, ergm

Examples

```
data(g4)
summary(ergm(g4 ~ odegree(3), MPLOnly=TRUE))
summary(ergm(g4 ~ odegree(3), theta0=0))
```

Getting.Started

*Getting Started with "ergm": Fit, simulate and diagnose exponential-family models for networks***Description**

`ergm` is a collection of functions to plot, fit, diagnose, and simulate from random graph models. For a list of functions type: `help(package='ergm')`

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 the original authors are to be cited as given in `citation("ergm")`:

Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2003 *ergm: Fit, simulate and diagnose exponential-family models for networks*
<http://statnetproject.org>.

All published work derived from this package must cite it. For complete citation information, use `citation(package="ergm")`.

Details

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 R. The `ergm` package allows maximum likelihood estimates of exponential random network models to be calculated using Markov Chain Monte Carlo. The package also provides tools for plotting networks, simulating networks and assessing model goodness-of-fit.

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

Author(s)

Mark S. Handcock (handcock@stat.washington.edu),
David R. Hunter (dhunter@stat.psu.edu),
Carter T. Butts (butts@uci.edu),
Steven M. Goodreau (goodreau@u.washington.edu), and
Martina Morris (morrism@u.washington.edu)
Maintainer: Mark S. Handcock (handcock@stat.washington.edu)

References

- Admiraal R, Handcock MS (2007). **networks**: Simulate bipartite graphs with fixed marginals through sequential importance sampling. Statnet Project, Seattle, WA. Version 1, <http://statnetproject.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). <http://www.jstatsoft.org/v24/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 (2006). **netperm**: Permutation Models for Relational Data. Version 0.2, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2007). **sna**: Tools for Social Network Analysis. Version 1.5, <http://erzuli.ss.uci.edu/R.stuff>.
- Butts CT (2008). **network**: A Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2). <http://www.jstatsoft.org/v24/i02/>.

- Butts CT, with help~from David~Hunter, Handcock MS (2007). **network**: Classes for Relational Data. Version 1.2, <http://erzuli.ss.uci.edu/R.stuff>.
- 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). <http://www.jstatsoft.org/v24/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. www.csss.washington.edu/Papers/wp39.pdf
- Handcock MS (2003b). **degreenet**: Models for Skewed Count Distributions Relevant to Networks. Statnet Project, Seattle, WA. Version 1.0, <http://statnetproject.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 2, <http://statnetproject.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, <http://statnetproject.org>.
- Hunter, D. R. and Handcock, M. S. (2006) Inference in curved exponential family models for networks, *Journal of Computational and Graphical Statistics*, 15: 565-583.
- Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). <http://www.jstatsoft.org/v24/i03/>.
- Krivitsky PN, Handcock MS (2007). **latentnet**: Latent position and cluster models for statistical networks. Seattle, WA. Version 2, <http://statnetproject.org>.
- Morris M, Handcock MS, Hunter DR (2008). Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects. *Journal of Statistical Software*, 24(4). <http://www.jstatsoft.org/v24/i04/>.
- Strauss, D., and Ikeda, M.(1990). Pseudolikelihood estimation for social networks. *Journal of the American Statistical Association*, 85, 204-212.

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

```
## Default S3 method:
gof(object,...)
## S3 method for class 'formula':
gof(formula, ..., theta0=NULL,
     nsim=100, burnin=10000, interval=1000,
     GOF=~degree+espartners+distance,
     constraints=~.,
     control=control.gof.formula(),
     seed=NULL,
     verbose=FALSE)
## S3 method for class 'ergm':
gof(object, ...,
     nsim=100,
     GOF=~degree+espartners+distance,
     burnin=10000, interval=1000,
     constraints=NULL,
     control=control.gof.ergm(),
     seed=NULL,
     theta0=NULL, verbose=FALSE)
```

Arguments

object	an R object. Either a formula or an ergm object. See documentation for ergm .
formula	formula; An R formula object, of the form $y \sim \langle \text{model terms} \rangle$, where y is a network object or a matrix that can be coerced to a network object. This specifies the model to simulate from. For the details on the possible $\langle \text{model terms} \rangle$, see ergm-terms . To create a network object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> operator if necessary.
theta0	When given either a formula or an object of class <code>ergm</code> , <code>theta0</code> are the parameters from which the sample is drawn. By default set to a vector of 0.
nsim	The number of simulations to use for the MCMC p -values. This is the size of the sample of graphs to be randomly drawn from the distribution specified by the object on the set of all graphs.
GOF	formula; an R formula object, of the form $\sim \langle \text{model terms} \rangle$ 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 <code>formula</code> , and typically are not. Examples are the degree distribution ("degree"), minimum geodesic distances ("dist"), and shared partner distributions ("espartners" and "dspartners"). For the details on the possible $\langle \text{model terms} \rangle$, see ergm-terms .
burnin	Number of proposed edge toggles before any MCMC sampling is done. If the model is correct this can be 0. Currently, there is no support for any check of the Markov chain mixing, so <code>burnin</code> should be set to a fairly large number.
interval	Number of proposed edge toggles between sampled statistics. The program prints a warning if too few proposed toggles are being accepted (if the number of proposed toggles between sampled observations ever equals an integral multiple of $100 \cdot (1 + \text{the number of toggles accepted})$).

<code>constraints</code>	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 <code>ergm</code> for more information. For <code>gof.formula</code> , defaults to unconstrained. For <code>gof.ergm</code> , defaults to the constraints with which <code>object</code> was fitted.
<code>control</code>	A list to control parameters, constructed using <code>control.gof.formula</code> or <code>control.gof.ergm</code> (which have different defaults).
<code>seed</code>	integer; random number integer seed. Defaults to <code>NULL</code> to use whatever the state of the random number generator is at the time of the call.
<code>verbose</code>	Provide verbose information on the progress of the simulation.
<code>...</code>	Additional arguments, to be passed to lower-level functions in the future.

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.

A plot of the summary measures is plotted. More information can be found by looking at the documentation of `ergm`.

Value

`gof`, `gof.ergm`, and `gof.formula` return an object of class `gofobject`. This is a list of the tables of statistics and p -values. This is typically plotted using `plot.gofobject`.

See Also

`ergm`, `network`, `simulate.ergm`, `summary.ergm`, `plot.gofobject`

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), theta0=c(-1.6339, 0.0049))
plot(gofflo2)
```

```
mcmc.diagnostics.ergm
```

Conduct MCMC diagnostics on an ergm fit

Description

This function creates simple diagnostic plots for the MCMC sampled statistics produced from a fit. It also prints the Raftery-Lewis diagnostics, indicates if they are sufficient, and suggests the run length required.

Usage

```
## S3 method for class 'ergm':
mcmc.diagnostics (object, sample = "sample", smooth = TRUE,
                  r = 0.0125, digits = 6, maxplot = 1000, verbose = TRUE,
                  center = TRUE, main = "Summary of MCMC samples", xlab =
                  "Iterations", ylab = "", ...)
```

Arguments

object	An object. See documentation for ergm .
sample	The component of object on which the diagnosis is based. The two usuals ones are <code>thetasample</code> from the auxiliary sample of the natural parameter and <code>sample</code> the (default) sample of the sufficient statistics from the model.
smooth	Draw a smooth line through trace plots
r	Percentile of the distribution to estimate
digits	Number of digits to print
maxplot	Maximum number of statistics to plot
verbose	If this is TRUE, print out more information about the MCMC runs including lag correlations.
center	logical; should the samples be centered on the observed statistics.
main	Figure title for the diagnostic plots.
xlab	X-axis label for diagnostic plots
ylab	Y-axis label for diagnostic plots
...	Additional arguments, to be passed to lower-level functions in the future.

Details

The plots produced are a trace of the sampled output and a density estimate for each variable in the chain.

The Raftery-Lewis diagnostic is a run length control diagnostic based on a criterion of accuracy of estimation of the quantile q . It is intended for use on a short pilot run of a Markov chain. The number of iterations required to estimate the quantile q to within an accuracy of $\pm r$ with probability p is calculated. Separate calculations are performed for each variable within each chain.

In fact, an `object` contains the matrix of statistics from the MCMC run as component `$sample`. This matrix is actually an object of class `mcmc` and can be used directly in the `CODA` package to assess MCMC convergence. *Hence all MCMC diagnostic methods available in `cod`a are available directly.* See the examples and <http://www.mrc-bsu.cam.ac.uk/bugs/classic/coda04/readme.shtml>.

More information can be found by looking at the documentation of [ergm](#).

Value

`mcmc.diagnostics.ergm` returns a matrix of Raftery-Lewis diagnostics.

Details of output

- M** The number of `burn in` iterations to be discarded (total over all chains).
- N** The number of iterations after `burn in` required to estimate the quantile q to within an accuracy of $\pm r$ with probability p (total over all chains). The overall number of iterations required ($M + N$).
- Total** Overall number of iterations required ($M + N$).
- Nmin** The minimum required sample size for a chain with no correlation between consecutive samples. Positive autocorrelation will increase the required sample size above this minimum value.
- I** An estimate (the `dependence factor`) of the extent to which auto-correlation inflates the required sample size. Values of `I` larger than 5 indicate strong autocorrelation which may be due to a poor choice of starting value, high posterior correlations, or `stickiness` of the MCMC algorithm.

Author(s)

Mark S. Handcock, handcock@stat.washington.edu based on the `cod`a package and also ideas from `mcgibbsit` by Gregory R. Warnes gregory_r_warnes@groton.pfizer.com. It is based on the the R function `raftery.diag` in `cod`a. `raftery.diag`, in turn, is based on the FORTRAN program `gibbsit` written by Steven Lewis which is available from the Statlib archive.

References

- Warnes, G.W. (2000). Multi-Chain and Parallel Algorithms for Markov Chain Monte Carlo. Dissertation, Department of Biostatistics, University of Washington,
- Raftery, A.E. and Lewis, S.M. (1992). One long run with diagnostics: Implementation strategies for Markov chain Monte Carlo. *Statistical Science*, 7, 493-497.

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

See Also

[ergm](#), [network](#), [coda](#), [mcgibbsit](#), [summary.ergm](#)

Examples

```
#
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$sample, ask=FALSE)
ergm.raftery.diag(gest$sample, r=0.1)
#
# A full range of diagnostics are available
# using codamenu()
#
```

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 - reminiscent of a chemical molecule. It is stored as a [network](#) object.

Usage

```
data(molecule)
```

See Also

[florentine](#), [sampson](#), [network](#), [plot.network](#), [ergm](#)

`network.update` *Replaces the sociomatrix in a network object*

Description

Replaces the sociomatrix in a network object with the sociomatrix specified by `newmatrix`. See [ergm](#) for more information.

Usage

```
network.update(nw, newmatrix, matrix.type=NULL)
```

Arguments

`nw` a [network](#) object. See documentation for the [network](#) package.

`newmatrix` Either an adjacency matrix (a matrix of zeros and ones indicating the presence of a tie from *i* to *j*) or an edgelist (a two-column matrix listing origin and destination node numbers for each edge; note that in an undirected matrix, the first column should be the smaller of the two numbers).

`matrix.type` One of "adjacency" or "edgelist" telling which type of matrix `newmatrix` is. Default is to use the [which.matrix.type](#) function.

Value

`network.update` returns a [network](#) 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
network.update(flomarriage, rand.mat)
# Try this with an edgelist
rand.mat <- as.matrix.network.edgelist(flomarriage)[1:5,]
network.update(flomarriage, rand.mat)
```

plot.ergm

*Plotting Method for class ergm***Description**

`plot.ergm` is the plotting method for `ergm` objects. It plots the MCMC diagnostics via the `mcmc.diagnostics` function. See `ergm` for more information on how to fit these models.

Usage

```
## S3 method for class 'ergm':
plot(x, ..., mle=FALSE, comp.mat = NULL,
      label = NULL, label.col = "black",
      xlab, ylab, main, label.cex = 0.8, edge.lwd = 1,
      edge.col=1, al = 0.1,
      contours=0, density=FALSE, only.subdens = FALSE,
      drawarrows=FALSE,
      contour.color=1, plotnetwork=FALSE, pie = FALSE, piesize=0.07,
      vertex.col=1, vertex.pch=19, vertex.cex=2,
      mycol=c("black", "red", "green", "blue", "cyan",
              "magenta", "orange", "yellow", "purple"),
      mypch=15:19, mycex=2:10)
```

Arguments

<code>x</code>	an R object of class <code>ergm</code> . See documentation for <code>ergm</code> .
<code>mle</code>	Plots the network using the MLE of the positions for latent models.
<code>pie</code>	For latent clustering models, each node is drawn as a pie chart representing the probabilities of cluster membership.
<code>piesize</code>	The size of the pie charts.
<code>contours</code>	For latent models, plots a contours by contours array of the network with one contour per network corresponding to the posterior distribution of each of the nodes.
<code>contour.color</code>	Color of the contour lines.
<code>density</code>	If <code>density=TRUE</code> , plots the density of the posterior position of the nodes. If <code>density=c(nr,nc)</code> , plots a nr by nc array of density estimates for each cluster.
<code>only.subdens</code>	If <code>density=c(nr,nc)</code> , only plots the densities of the clusters, not the overall density.
<code>drawarrows</code>	If <code>density=TRUE</code> , draws the ties on the density plot.
<code>plotnetwork</code>	If <code>density=c(nr,nc)</code> , a plot of the network is also shown.
<code>comp.mat</code>	For latent models, the positions are Procrustes transformed to look like <code>comp.mat</code> .

label	A vector of the same length as the number of nodes containing the labels of the nodes.
label.col	The color to be used for plotting the labels.
label.cex	The size of the node labels.
xlab	Title for the x axis.
ylab	Title for the y axis.
main	The main title for the network.
edge.lwd	The line width for the arrows between nodes.
edge.col	The color of the arrows between nodes.
al	The length of the arrow heads.
vertex.col	The color of the nodes as defined by <code>mycol</code> . Can be specified as an attribute of the network used in the model.
vertex.pch	The plotting character of the nodes as defined by <code>mypch</code> . Can be specified as an attribute of the network used in the model. By default it is 15 - a red square.
vertex.cex	The size of the nodes as defined by <code>mycex</code> . Can be specified as an attribute of the network used in the model.
mycol	Vector of colors to be used. Defaults to: <code>c("black","red","green","blue","cyan","magenta","orange","yellow","purple")</code>
mypch	Vector of plotting characters to be used. Defaults to:
mycex	Vector of character expansion values.
...	Other optional arguments to be used by the plot function.

Details

Plots the results of an `ergm` fit.

More information can be found by looking at the documentation of [ergm](#).

Value

NULL

See Also

`ergm`, `network`, `plot.network`, `plot`, `add.contours`

Examples

```
## Not run:
#
# The example assumes you have the 'latentnet' package installed.
#
# Using Sampson's Monk data, lets fit a
# simple latent position model
#
data(sampson)
```

```

#
# Get the group labels
#
samp.labs <- substr(get.vertex.attribute(samplike,"group"),1,1)
#
samp.fit <- ergm(samplike ~ latent(k=2), burnin=10000,
                MCMCsamplesize=2000, interval=30)
#
# See if we have convergence in the MCMC
mcmc.diagnostics(samp.fit)
#
# Plot the fit
#
plot(samp.fit,label=samp.labs, vertex.col="group")
#
# Using Sampson's Monk data, lets fit a latent clustering model
#
samp.fit <- ergm(samplike ~ latentcluster(k=2, ngroups=3), burnin=10000,
                MCMCsamplesize=2000, interval=30)
#
# See if we have convergence in the MCMC
mcmc.diagnostics(samp.fit)
#
# Lets look at the goodness of fit:
#
plot(samp.fit,label=samp.labs, vertex.col="group")
plot(samp.fit,pie=TRUE,label=samp.labs)
plot(samp.fit,density=c(2,2))
plot(samp.fit,contours=5,contour.color="red")
plot(samp.fit,density=TRUE,drawarrows=TRUE)
add.contours(samp.fit,nlevels=8,lwd=2)
points(samp.fit$Z.mkl,pch=19,col=samp.fit$class)
## End(Not run)

```

plot.gofobject

Plot Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model

Description

`plot.gofobject` 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.

Usage

```

## S3 method for class 'gofobject':
plot(x, ...,
      cex.axis=0.7, plotlogodds=FALSE,
      main = "Goodness-of-fit diagnostics",

```

```
normalize.reachability=FALSE,
verbose=FALSE)
```

Arguments

<code>x</code>	an object of class <code>gofobject</code> , typically produced by the <code>gof.ergm</code> or <code>gof.formula</code> functions. See the documentation for these.
<code>cex.axis</code>	Character expansion of the axis labels relative to that for the plot.
<code>plotlogodds</code>	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.
<code>main</code>	Title for the goodness-of-fit plots.
<code>normalize.reachability</code>	Should the reachability proportion be normalized to make it more comparable with the other geodesic distance proportions.
<code>verbose</code>	Provide verbose information on the progress of the plotting.
<code>...</code>	Additional arguments, to be passed to the plot function.

Details

`gof.ergm` produces a sample of networks randomly drawn from the specified model. This function produces a plot of the summary measures.

Value

none

See Also

`gof.ergm`, `gof.formula`, `ergm`, `network`, `simulate.ergm`

Examples

```
#
data(florentine)
#
# test the gof.ergm function
#
gest <- ergm(flomarriage ~ edges + kstar(2))
gest
summary(gest)

#
# Plot the probabilities first
#
gofflo <- gof(gest)
gofflo
plot(gofflo)
#
```

```
# And now the odds
#
plot(gofflo, plotlogodds=TRUE)
#
# Use the formula version
#
gof(flomarriage ~ edges + kstar(2), theta0=c(-1.6339, 0.0049))
```

print.ergm

Exponential Random Graph Models

Description

`print.ergm` is the method used to print an `ergm` object created by the `ergm` function.

Usage

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

Arguments

<code>x</code>	An <code>ergm</code> object. See documentation for <code>ergm</code> .
<code>digits</code>	Significant digits for coefficients
<code>...</code>	Additional arguments, to be passed to lower-level functions in the future.

Details

Automatically called when an object of class `ergm` is printed. Currently, `print.ergm` summarizes the number of Newton-Raphson iterations required, the size of the MCMC sample, the theta vector governing the selection of the sample, and the Monte Carlo MLE.

Value

The value returned is the `ergm` object itself.

See Also

network, ergm

Examples

```
data(florentine)

x <- ergm(flomarriage ~ density)
class(x)
x
```

 samplk

Longitudinal networks of positive affection within a monastery as a "network" object

Description

Sampson (1969) recorded the social interactions among a group of monks while resident as an experimenter on vision, and collected numerous sociometric rankings. During his stay, a political "crisis in the cloister" resulted in the expulsion of four monks (Nos. 2, 3, 17, and 18) and the voluntary departure of several others - most immediately, Nos. 1, 7, 14, 15, and 16. (In the end, only 5, 6, 9, and 11 remained). Of particular interest is the data on positive affect relations ("liking"), in which each monk was asked if they had positive relations to each of the other monks.

The data were gathered at three times to capture changes in group sentiment over time: `samplk1`, `samplk2`, and `samplk3`. They represent three time points in the period during which a new cohort entered the monastery near the end of the study but before the major conflict began. Each member ranked only his top three choices on "liking." (Some subjects offered tied ranks for their top four choices). A tie from monk A to monk B exists if A nominated B as one of his three best friends at that that time point.

`samplk3` is a data set of Hoff, Raftery and Handcock (2002).

See also the data set `sampson` containing the time-aggregated graph `samplike`. It is the cumulative tie for "liking" over the three periods. For this, a tie from monk A to monk B exists if A nominated B as one of his three best friends at any of the three time points.

All graphs are stored as `network` objects.

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), and Fienberg, Meyer, and Wasserman (1981), Hoff, Raftery, and Handcock (2002), etc. This is only a small piece of the data collected by Sampson.

Usage

```
data(samplk)
```

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.

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.

See Also

`sampson`, `florentine`, `network`, `plot.network`, `ergm`

sampson

Cumulative network of positive affection within a monastery as a “network” object

Description

Sampson (1969) recorded the social interactions among a group of monks while resident as an experimenter on vision, and collected numerous sociometric rankings. During his stay, a political “crisis in the cloister” resulted in the expulsion of four monks (Nos. 2, 3, 17, and 18) and the voluntary departure of several others - most immediately, Nos. 1, 7, 14, 15, and 16. (In the end, only 5, 6, 9, and 11 remained). Of particular interest is the data on positive affect relations (“liking”), in which each monk was asked if they had positive relations to each of the other monks.

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 entered the monastery near the end of the study but before the major conflict began. Each member ranked only his top three choices on “liking.” (Some subjects offered tied ranks for their top four choices). A tie from monk A to monk B exists if A nominated B as one of his three best friends at that that time point.

`samplike` is the time-aggregated graph. It is the cumulative tie for “liking” over the three periods. For this, a tie from monk A to monk B exists if A nominated B as one of his three best friends at any of the three time points.

All graphs are stored as `network` objects.

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), and Fienberg, Meyer, and Wasserman (1981), Hoff, Raftery, and Handcock (2002), etc. This is only a small piece of the data collected by Sampson.

Usage

```
data(sampson)
```

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.

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.

See Also

florentine, network, plot.network, ergm

simulate.ergm	<i>Draw from the distribution of an Exponential Family Random Graph Model</i>
---------------	---

Description

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

Usage

```
## S3 method for class 'formula':
simulate(object, nsim=1, seed=NULL, ..., theta0,
         burnin=1000, interval=1000,
         basis=NULL,
         statonly=FALSE,
         sequential=TRUE,
         constraints = ~.,
         control = control.simulate.formula(),
         verbose=FALSE)

## S3 method for class 'ergm':
simulate(object, nsim=1, seed=NULL, ..., theta0=NULL,
         burnin=1000, interval=1000,
         statonly=FALSE,
         sequential=TRUE,
         constraints = NULL,
         control = control.simulate.ergm(),
         verbose=FALSE)
```

Arguments

object	an R object. Either a <code>formula</code> or an <code>ergm</code> object. The <code>formula</code> should be of the form <code>y ~ <model terms></code> , where <code>y</code> is a network object or a matrix that can be coerced to a <code>network</code> object. For the details on the possible <code><model terms></code> , see <code>ergm-terms</code> . To create a <code>network</code> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> 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	Random number integer seed. The default is <code>sample(10000000, size=1)</code> .
theta0	For Bernoulli networks this is the log-odds of a tie, however it is only used if <code>prob</code> is not specified. When given either a <code>formula</code> or an object of class <code>ergm</code> , <code>theta0</code> are the parameters from which the sample is drawn.
burnin	The number of proposed proposals before any MCMC sampling is done. Currently, there is no support for any check of the Markov chain mixing, so <code>burnin</code> should be set to a fairly large number.

<code>interval</code>	The number of proposals between sampled statistics. The program prints a warning if too few proposals are being accepted (if the number of proposals between sampled observations ever equals an integral multiple of 100(1+the number of proposals accepted)).
<code>basis</code>	An optional <code>network</code> object to start the MCMC algorithm from. This overrides the left-hand-side of the <code>formula</code> . If neither a left-hand-side nor a <code>basis</code> is present, an error results because the characteristics of the network (e.g., size and directedness) must be specified.
<code>constraints</code>	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 <code>ergm</code> for more information. For <code>simulate.formula</code> , defaults to no constraints. For <code>simulate.ergm</code> , defaults to using the same constraints as those with which <code>object</code> was fitted.
<code>control</code>	A list of control parameters for algorithm tuning. Constructed using <code>control.simulate.ergm</code> or <code>control.simulate.formula</code> , which have different defaults.
<code>statsonly</code>	If TRUE, return only the network statistics (not the network(s) themselves)
<code>sequential</code>	Should the returned draws use the prior draw as the starting network or always use the initially passed network? For random draws the results should be similar (stochastically), but the <code>sequential=TRUE</code> option is useful for dynamic draws.
<code>verbose</code>	If this is TRUE, we will print out more information as we run the program, including (currently) some goodness of fit statistics.
<code>...</code>	further arguments passed to or used by methods.

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 `theta0` 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 `theta0`.

Note that the first network is sampled after `burnin + interval` 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 `nsim==1`, `simulate.ergm` returns an object of class `network`. If `nsim>1`, it returns an object of class `network.series` that is a list with the following elements:

<code>formula</code>	The <code>formula</code> used to generate the sample.
<code>networks</code>	A list of the generated networks.
<code>stats</code>	The $n \times p$ matrix of network change statistics, where n is the sample size and p is the number of network change statistics specified in the model.

See Also

ergm, network, print.network

Examples

```

#
# Let's draw from a Bernoulli model with 16 nodes
# and density 0.5 (i.e., theta0 = c(0,0))
#
g.sim <- simulate(network(16) ~ edges + mutual)
#
# What are the statistics like?
#
summary(g.sim ~ edges + mutual)
#
# Now simulate a network with higher mutuality
#
g.sim <- simulate(network(16) ~ edges + mutual, theta0=c(0,2))
#
# How do the statistics like?
#
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 5 realizations
# of a edges and 2-star network
#
g.sim <- simulate(~edges+kstar(2),nsim=5,theta0=c(-1.8,0.03),
                 basis=g.use,
                 burnin=100000,interval=1000)

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
#
g.sim <- simulate(gest,nsim=100,burnin=1000,interval=1000)
g.sim

```

summary.ergm *Summarizing ERGM Model Fits*

Description

`summary` method for class "ergm".

Usage

```
## S3 method for class 'ergm':
summary(object, ...,
         digits = max(3, getOption("digits") - 3),
         correlation = FALSE, covariance = FALSE, eps = 1e-04)
```

Arguments

<code>object</code>	an object of class "ergm", usually, a result of a call to <code>ergm</code> .
<code>digits</code>	Significant digits for coefficients
<code>correlation</code>	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
<code>covariance</code>	logical; if TRUE, the covariance matrix of the estimated parameters is returned and printed.
<code>eps</code>	number; indicates the smallest p-value. See <code>printCoefmat</code> .
<code>...</code>	further arguments passed to or from other methods.

Details

`summary.ergm` tries to be smart about formatting the coefficients, standard errors, etc.

Value

The function `summary.ergm` computes and returns a list of summary statistics of the fitted `ergm` model given in `object`.

See Also

`network`, `ergm`, `print.ergm`. The model fitting function `ergm`, `summary`.

Function `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.gofobject *Summaries the Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model*

Description

`summary.gofobject` 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.

Usage

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

Arguments

`object` an object of class `gofobject`, typically produced by the `gof.ergm` or `gof.formula` functions. See the documentation for these.

`...` Additional arguments, to be passed to the plot function.

Details

`gof.ergm` produces a sample of networks randomly drawn from the specified model. This function produces a print out the summary measures.

Value

none

See Also

`gof.ergm`, `gof.formula`, `ergm`, `network`, `simulate.ergm`

Examples

```
#  
data(florentine)  
#  
# test the gof.ergm function  
#  
gest <- ergm(flomarriage ~ edges + kstar(2))  
gest  
summary(gest)  
  
#  
# Plot the probabilities first  
#
```

```
gofflo <- gof(gest)
gofflo
summary(gofflo)
```

summary.statistics *Calculation of network or graph statistics*

Description

Used to calculate the specified statistics for an observed network if its argument is a formula for an [ergm](#). See [ergm-terms](#) for more information on the statistics that may be specified.

Usage

```
## S3 method for class 'formula':
summary.statistics(object, ..., drop=FALSE, basis=NULL)
## S3 method for class 'ergm':
summary.statistics(object, ..., drop=FALSE, basis=NULL)
```

Arguments

object	an R object. It is either an R formula object (see above) or an ergm model object. In the latter case, <code>summary.statistics</code> is called for the <code>object\$formula</code> object. In the former case, <code>object</code> is of the form <code>y ~ <model terms></code> , where <code>y</code> is a network object or a matrix that can be coerced to a network object. For the details on the possible <code><model terms></code> , see ergm-terms . To create a network object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> operator if necessary.
drop	logical: Should terms whose observed statistics are extreme among the set of all possible network statistics (which result in nonexistent MLEs) be dropped?
basis	An optional network object relative to which the global statistics should be calculated.
...	further arguments passed to or used by methods.

Details

If `object` is of class [formula](#), then [summary](#) may be used in lieu of `summary.statistics` because `summary.formula` calls the `summary.statistics` function. The function actually cumulates the change statistics when removing edges from the observed network one by one until the empty network results. Since each model term has a prespecified value (zero by default) for the corresponding statistic(s) on an empty network, these change statistics give the absolute statistics on the original network.

Value

A vector of statistics measured on the network.

See Also

ergm, network, ergm-terms

Examples

```
#  
# Lets look at the Florentine marriage data  
#  
data(florentine)  
#  
# test the summary.statistics 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
```

Index

*Topic **classes**

as.network.numeric, 6

*Topic **datasets**

faux.magnolia.high, 37

faux.mesa.high, 38

flobusiness, 40

flomarriage, 41

florentine, 42

g4, 43

molecule, 50

samplk, 57

sampson, 58

*Topic **graphs**

as.network.numeric, 6

ergmuserterms-package, 4

plot.gofobject, 54

summary.gofobject, 63

*Topic **models**

anova.ergm, 5

coef.ergm, 8

control.ergm, 9

control.gof, 12

control.simulate, 13

edgelist.ergm, 14

ergm, 15

ergm-package, 2

ergm-terms, 23

Getting.Started, 43

gof, 45

mcmc.diagnostics.ergm, 48

network.update, 51

plot.ergm, 52

print.ergm, 56

simulate.ergm, 59

summary.ergm, 62

summary.statistics, 64

*Topic **package**

ergm-package, 2

Getting.Started, 43

*Topic **regression**

anova.ergm, 5

coef.ergm, 8

summary.ergm, 62

absdiff (*ergm-terms*), 23

absdiffcat (*ergm-terms*), 23

altkstar (*ergm-terms*), 23

anova, 6

anova.ergm, 5

anova.ergm.list, 6

anova.ergm.list (*anova.ergm*), 5

as.matrix.network, 15

as.network.numeric, 6, 6

asymmetric (*ergm-terms*), 23

b1concurrent (*ergm-terms*), 23

b1degree (*ergm-terms*), 23

b1factor (*ergm-terms*), 23

b1star (*ergm-terms*), 23

b1starmix (*ergm-terms*), 23

b1twostar (*ergm-terms*), 23

b2concurrent (*ergm-terms*), 23

b2degree (*ergm-terms*), 23

b2factor (*ergm-terms*), 23

b2star (*ergm-terms*), 23

b2starmix (*ergm-terms*), 23

b2twostar (*ergm-terms*), 23

balance (*ergm-terms*), 23

coda, 49, 50

coef, 62

coef.ergm, 8

coefficients.ergm (*coef.ergm*), 8

concurrent (*ergm-terms*), 23

control.ergm, 9, 13, 14, 16–18

control.gof, 12, 12, 14

control.gof.ergm, 47

control.gof.formula, 47

control.simulate, 12, 13, 13

- control.simulate.ergm, 60
- control.simulate.formula, 60
- ctriple (ergm-terms), 23
- cycle (ergm-terms), 23
- degree (ergm-terms), 23
- density (ergm-terms), 23
- dsp (ergm-terms), 23
- dyadcov (ergm-terms), 23
- edgecov (ergm-terms), 23
- edgelist.ergm, 14, 14
- edges (ergm-terms), 23
- ergm, 2, 4–6, 8, 9, 11–14, 15, 15, 17–19, 23, 24, 38–40, 43–54, 56, 59, 60, 62–64
- ergm-terms, 4, 16, 18, 46, 59, 64
- ergm-package, 2
- ergm-terms, 23
- ergm.object, 12, 13
- ergm.terms (ergm-terms), 23
- ergmuserterms (ergmuserterms-package), 4
- ergmuserterms-package, 4
- esp (ergm-terms), 23
- faux.magnolia.high, 24, 37, 40
- faux.mesa.high, 24, 38, 38
- fauxhigh (faux.mesa.high), 38
- fitted.values, 8
- flobusiness, 40, 40, 42
- flomarriage, 41, 41, 42
- florentine, 42
- formula, 4, 16, 18, 59, 60, 64
- g4, 43
- Getting.Started, 43
- glm, 8
- gof, 12–14, 45, 45, 47
- gof.ergm, 47, 55, 63
- gof.formula, 47, 55, 63
- gwbldegree (ergm-terms), 23
- gwb2degree (ergm-terms), 23
- gwdegree (ergm-terms), 23
- gwdsp (ergm-terms), 23
- gwesp (ergm-terms), 23
- gwidegree (ergm-terms), 23
- gwensp (ergm-terms), 23
- gwodegree (ergm-terms), 23
- hamming (ergm-terms), 23
- hammingmix (ergm-terms), 23
- idegree (ergm-terms), 23
- intransitive (ergm-terms), 23
- isolates (ergm-terms), 23
- istar (ergm-terms), 23
- kstar (ergm-terms), 23
- lm, 8
- localtriangle (ergm-terms), 23
- m2star (ergm-terms), 23
- mcgibbsit, 49, 50
- mcmc.diagnostics, 52
- mcmc.diagnostics (mcmc.diagnostics.ergm), 48
- mcmc.diagnostics.ergm, 48, 49
- meandeg (ergm-terms), 23
- molecule, 50
- mutual (ergm-terms), 23
- nearsimmelian (ergm-terms), 23
- network, 2, 6, 7, 16, 24, 37–44, 50, 51, 57–60, 64
- network.update, 51, 51
- nodecov (ergm-terms), 23
- nodefactor (ergm-terms), 23
- nodeicov (ergm-terms), 23
- nodeifactor (ergm-terms), 23
- nodematch (ergm-terms), 23
- nodemix (ergm-terms), 23
- nodecov (ergm-terms), 23
- nodeofactor (ergm-terms), 23
- nsp (ergm-terms), 23
- odegree (ergm-terms), 23
- optim, 10
- ostar (ergm-terms), 23
- plot.ergm, 52, 52
- plot.gofobject, 47, 54, 54
- plot.network, 38, 40
- print.ergm, 18, 56, 56
- print.gofobject (summary.gofobject), 63
- printCoefmat, 62
- receiver (ergm-terms), 23
- residuals, 8

samplike (*sampson*), 58
samplk, 57
samplk1 (*samplk*), 57
samplk2 (*samplk*), 57
samplk3 (*samplk*), 57
sampsom, 57, 58
sender (*ergm-terms*), 23
simmelian (*ergm-terms*), 23
simmelianties (*ergm-terms*), 23
simulate, 14, 59
simulate.ergm, 12–14, 59
simulate.formula, 12–14
simulate.formula (*simulate.ergm*),
59
smalldiff (*ergm-terms*), 23
sna, 31, 35
sociality (*ergm-terms*), 23
summary, 62, 64
summary (*summary.statistics*), 64
summary.ergm, 18, 50, 62, 62
summary.gofobject, 63, 63
summary.statistics, 64
summary.statistics.default
(*summary.statistics*), 64
summary.statistics.ergm
(*summary.statistics*), 64
summary.statistics.formula
(*summary.statistics*), 64
summary.statistics.matrix
(*summary.statistics*), 64
summary.statistics.network
(*summary.statistics*), 64

terms-ergm (*ergm-terms*), 23
terms.ergm (*ergm-terms*), 23
threepath (*ergm-terms*), 23
transitive (*ergm-terms*), 23
triad.classify, 31, 35
triadcensus (*ergm-terms*), 23
triangle (*ergm-terms*), 23
tripercent (*ergm-terms*), 23
ttriple (*ergm-terms*), 23
twopath (*ergm-terms*), 23

which.matrix.type, 51