

# Package ‘ergm’

January 2, 2012

**Version** 2.4-3

**Date** 2011-09-19

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

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

**Maintainer** David R. Hunter <dhunter@stat.psu.edu>

**Depends** network (>= 1.6), nlme, trust

**Suggests** coda, KernSmooth, sna, Rglpk, robustbase, Matrix

**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** GPL-3 + file LICENSE

**URL** <http://statnet.org>

**Repository** CRAN

**Date/Publication** 2011-09-19 13:46:59

## R topics documented:

ergm-package . . . . .	3
ergmuserterms-package . . . . .	5
anova.ergm . . . . .	6
as.network.numeric . . . . .	7
coef.ergm . . . . .	9
control.ergm . . . . .	10
control.gof . . . . .	14
control.san . . . . .	15

control.simulate . . . . .	16
ecoli . . . . .	17
edgelist.ergm . . . . .	18
enformulate.curved . . . . .	19
ergm . . . . .	20
ergm-terms . . . . .	28
ergm.allstats . . . . .	42
ergm.bridge.dindstart.llk . . . . .	44
ergm.bridge.llr . . . . .	45
ergm.exact . . . . .	46
ergmMPLE . . . . .	48
faux.magnolia.high . . . . .	49
faux.mesa.high . . . . .	51
fix.curved . . . . .	52
flobusiness . . . . .	53
flomarriage . . . . .	54
florentine . . . . .	55
g4 . . . . .	56
Getting.Started . . . . .	57
gof . . . . .	59
is.dyad.independent . . . . .	61
is.inCH . . . . .	62
kapferer . . . . .	63
logLik.ergm . . . . .	64
mcmc.diagnostics.ergm . . . . .	65
molecule . . . . .	67
network.update . . . . .	68
plot.ergm . . . . .	69
plot.gofobject . . . . .	71
print.ergm . . . . .	73
samplk . . . . .	74
sampson . . . . .	75
san . . . . .	76
simulate.ergm . . . . .	78
summary.ergm . . . . .	80
summary.gofobject . . . . .	82
summary.statistics . . . . .	83
wtd.median . . . . .	84
<b>Index</b>	<b>85</b>

## 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.ucla.edu](mailto:handcock@stat.ucla.edu)>,  
David R. Hunter <[dhunter@stat.psu.edu](mailto:dhunter@stat.psu.edu)>,  
Carter T. Butts <[buttsc@uci.edu](mailto:buttsc@uci.edu)>,  
Steven M. Goodreau <[goodreau@u.washington.edu](mailto:goodreau@u.washington.edu)>,  
Pavel N. Krivitsky <[pavel@stat.cmu.edu](mailto:pavel@stat.cmu.edu)>, and  
Martina Morris <[morrism@u.washington.edu](mailto:morrism@u.washington.edu)>  
Maintainer: David R. Hunter <[dhunter@stat.psu.edu](mailto:dhunter@stat.psu.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](http://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.

---

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,  $\theta$  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 `%%` 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:

- `m2starMixed 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  $\{(i \rightarrow j), (j \rightarrow k)\}$ .
- `testmeA 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, ..., eval.loglik = FALSE)
## S3 method for class 'ergmlist'
anova(object, ..., eval.loglik = FALSE, scale = 0, test = "F")
```

## Arguments

object, ...	objects of class <code>ergm</code> , usually, a result of a call to <code>ergm</code> .
eval.loglik	a logical specifying whether the log-likelihood will be evaluated if missing.
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 $\sigma^2$ . 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  $\sigma^2$  times the residual degrees of freedom.

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

### Value

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

### Warning

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

### See Also

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

### Examples

```
data(molecule)
molecule %v% "atomic type" <- c(1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,3)
fit0 <- ergm(molecule ~ edges)
anova(fit0)
fit1 <- ergm(molecule ~ edges + nodefactor("atomic type"))
anova(fit1)
fit2 <- ergm(molecule ~ edges + nodefactor("atomic type") + gwesp(0.5,
  fixed=TRUE), eval.loglik=TRUE) # Note the eval.loglik argument.
anova(fit0, fit1)
anova(fit0, fit1, fit2)
```

---

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

---

### Description

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

### Usage

```
## S3 method for class 'numeric'
as.network(x, directed = TRUE,
  hyper = FALSE, loops = FALSE, multiple = FALSE, bipartite = FALSE,
  ignore.eval = TRUE, names.eval = NULL,
  edge.check = FALSE,
  density=NULL, 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.
names.eval	optionally, the name of the attribute in which edge values should be stored. Currently ignored.
edge.check	logical; perform consistency checks on new edges?
density	numeric; the probability of a tie for Bernoulli networks. If neither density nor <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.)
theta0	numeric; the log-odds of a tie for Bernoulli networks. It is only used if density is not specified.
numedges	count; if present, sample the Bernoulli network conditional on this number of edges (rather than independently with the specified probability).
...	additional arguments

**Details**

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

**Value**

An object of class `network`

**References**

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

**See Also**

`network`

## Examples

```
#Draw a random directed network with 25 nodes
g<-network(25)
#Draw a random undirected network with density 0.1
g<-network(25, directed=FALSE, density=0.1)
#Draw a random bipartite network with 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,
             prop.weights.diss = "default", prop.args.diss = NULL,
             nr.maxit = 100,
             calc.mcmc.se = TRUE, hessian = TRUE, compress = TRUE,
             SAN.burnin=NULL,
             maxNumDyadTypes = 1e+06, maxedges = 20000, maxchanges = 1e+06,
             maxMPLEsamplesize = 1e+05, MPLEtype=c("glm", "penalized"),
             nr.reltol=sqrt(.Machine$double.eps), trace = 0,
             steplength = 0.5, sequential=TRUE,
             drop = TRUE, force.mcmc = FALSE, check.degeneracy=FALSE,
             mcmc.precision = 0.05,
             metric = c("lognormal", "Median.Likelihood",
                       "EF.Likelihood", "naive"),
             method = c("BFGS", "Nelder-Mead"),
             trustregion = 20, initial.loglik = NULL, loglik.nsteps = 20,
             initial.network = NULL,
             style = c("Newton-Raphson", "Robbins-Monro",
                      "Stochastic-Approximation", "Stepping", "PILA"),
             style.dyn = c("Robbins-Monro", "SPSA", "SPSA2"),
             phase1_n = NULL, initial_gain = NULL,
             nsubphases = "maxit", niterations = NULL, phase3_n = NULL,
             RobMon.phase1n_base = 7, RobMon.phase2n_base = 100, RobMon.phase2sub
             = 7, RobMon.init_gain = 0.5, RobMon.phase3n = 500,
             stepMCMCsize=100, gridsize=100, dyninterval=1000,
             packagenames="ergm", parallel = 0, returnMCMCstats = TRUE)
```

**Arguments**

**prop.weights** Specifies the proposal distribution used in the MCMC Metropolis-Hastings algorithm. Possible choices are "TNT" or "random"; the "default" is one of these two, depending on the constraints in place (as defined by the constraints argument of the [ergm](#) function), though not all weights may be used with all constraints. The TNT (tie / no tie) option puts roughly equal weight on selecting a dyad with or without a tie as a candidate for toggling, whereas the random option puts equal weight on all possible dyads (though the interpretation of random may change according to the constraints in place). When no constraints are in

place, the default is TNT, which appears to improve Markov chain mixing particularly for networks with a low edge density, as is typical of many realistic social networks.

prop.args	An alternative, direct way of specifying additional arguments to proposal.
prop.weights.diss	As prop.weights, for dissolution model.
prop.args.diss	As prop.args, for dissolution model.
nr.maxit	count; The maximum number of iterations in the Newton-Raphson optimization. Defaults to 1000. 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 $10^{11}$ .
MPLetype	one of "glm" or "penalized"; method to use for pseudolikelihood. "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".
nr.reltol	Relative convergence tolerance, passed to optimization routines like optim. See the reltol control argument for the optim function.
trace	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.)
steplength	Multiplier for step length, to make fitting more stable at the cost of efficiency.
sequential	Should the next iteration of the fit use the last network sampled as the starting network or always use the initially passed network? The results should be similar (stochastically), but the sequential=TRUE option if meanstats is far from the passed network.

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.
force.mcmc	logical; should MCMC maximum likelihood be used? Only relevant for dyadic independent networks, in which the MLE could be found using MPLLE instead.
check.degeneracy	Logical: Should the diagnostics include a check for model degeneracy?
mcmc.precision	vector; upper bounds on the precision of the standard errors induced by the MCMC algorithm. Defaults to 0.05.
metric	character; The name of the optimization metric to use. Defaults to "Median.Likelihood". See Hummel et al (2010) for an explanation of "lognormal" and "naive".
method	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.
trustregion	numeric; The maximum amount the algorithm will allow the approximated likelihood to be increased at a given iteration. Defaults to 20. See Snijders (2002) for details.
initial.loglik	Initial value of loglikelihood, if known.
loglik.nsteps	Number of bridges to use to evaluate dyad-dependent log-likelihood if <code>eval.loglik=TRUE</code> .
initial.network	Initial network for MCMC, if different from observed network.
style	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"). Another alternative is a partial stepping algorithm ("Stepping") as in Hummel et al. (2010). 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.
style.dyn	character; The style of method of moments estimation to use. The default is a form of stochastic approximation ("Robbins-Monro"), but it should only be used if it is known a priori that the derivative of each element of the equilibrium expected values of statistics of interest with respect to the corresponding formation phase parameter is positive. The other option, ("SPSA") is less precise but does not make this assumption. "SPSA2" is multithreaded, but runs slower for some reason.
phase1_n	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 Snijders (2002) for details.
initial_gain	numeric; The initial gain to Phase 2 of the stochastic approximation algorithm. Defaults to 0.1. See Snijders (2002) for details.
nsubphases	count; The number of sub-phases in Phase 2 of the stochastic approximation algorithm. Defaults to <code>maxit</code> . See Snijders (2002) for details.

niterations	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 Snijders (2002) for details.
phase3_n	count; The sample size for the MCMC sample in Phase 3 of the stochastic approximation algorithm. Defaults to 1000. See Snijders (2002) for details.
RobMon.phase1n_base	Robbins-Monro control parameter
RobMon.phase2n_base	Robbins-Monro control parameter
RobMon.phase2sub	Robbins-Monro control parameter
RobMon.init_gain	Robbins-Monro control parameter
RobMon.phase3n	Robbins-Monro control parameter
returnMCMCstats	logical; If this is TRUE (the default) the matrix of change statistics from the MCMC run is returned 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.
stepMCMCsize	MCMC sample size for the preliminary steps of the "Stepping" style of optimization. This is usually chosen to be smaller than the final MCMC sample size (which equals MCMCsamplesize).
gridsize	Integer $N$ such that the "Stepping" style of optimization chooses a step length equal to the largest possible multiple of $1/N$ ; see Hummel et al. (2010) for details.
dyninterval	Number of Metropolis-Hastings proposal for each phase in the dynamic network simulation.
packagenames	Names of packages in which changestats are found.
parallel	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

- Snijders, T.A.B. (2002), Markov Chain Monte Carlo Estimation of Exponential Random Graph Models. *Journal of Social Structure*. Available from <http://www.cmu.edu/joss/content/articles/volume3/Snijders.pdf>.
- Firth (1993), Bias Reduction in Maximum Likelihood Estimates. *Biometrika*, 80: 27-38.

- Hunter, D. R. and M. S. Handcock (2006), Inference in curved exponential family models for networks. *Journal of Computational and Graphical Statistics*, 15: 565-583.
- Hummel, R. M., Hunter, D. R., and Handcock, M. S. (2010), A Steplength Algorithm for Fitting ERGMs, Penn State Department of Statistics Technical Report.

### 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
```

### Arguments

prop.weights	Specifies the method to allocate probabilities of being proposed to dyads. For the <a href="#">simulate.formula</a> variant, defaults to "default", which picks a reasonable default for the specified constraint. For <a href="#">simulate.ergm</a> variant, defaults to NULL, to reuse the weights with which the given <a href="#">ergm.object</a> 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.san

*Auxiliary for Controlling SAN*


---

**Description**

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

**Usage**

```
control.san(prop.weights = "default", prop.args = NULL,
            drop = FALSE, maxedges=20000, maxchanges=1000000,
            packagenames="ergm", network.output="network",
            parallel = 0)
```

**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. 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.
maxedges	maximum number of edges expected in network
packagenames	Names of packages in which changestatistics are found. Currently unused as ergm is presumed.
network.output	character: R class with which to output networks. The options are "network" (default) and "edgelist.compressed" (which saves space but only supports networks without vertex attributes)
maxchanges	Currently unused
parallel	Number of threads in which to run the sampling. Currently unimplemented

**Details**

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

**Value**

A list with arguments as components.

**See Also**[san](#)

---

`control.simulate`*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, maxedges=20000,
maxchanges = 1e+06,
packagenames="ergm",
network.output="network",
parallel=0)
```

```
control.simulate.formula(prop.weights = "default", prop.args = NULL,
drop = FALSE, summarizestats = FALSE, maxedges=20000,
maxchanges = 1e+06,
packagenames="ergm",
network.output="network",
parallel=0)
```

```
control.simulate.ergm(prop.weights = "default", prop.args = NULL, drop = FALSE,
summarizestats = FALSE,
maxchanges = 1e+06, maxedges = 20000,
packagenames="ergm",
network.output="network",
parallel=0)
```

**Arguments**

- |                           |  |
|---------------------------|--|
| <code>prop.weights</code> | Specifies the method to allocate probabilities of being proposed to dyads. For the <a href="#">simulate.formula</a> variant, defaults to "default", which picks a reasonable default for the specified constraint. For <a href="#">simulate.ergm</a> variant, defaults to NULL, to reuse the weights with which the given <a href="#">ergm.object</a> was fitted. Other possible values are "TNT", "random", and "nonobserved", though not all values may be used with all possible constraints. |
| <code>prop.args</code>    | An alternative, direct way of specifying additional arguments to proposal.   |
| <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 FALSE.  |

<code>summarizestats</code>	logical; Print out a summary of the sufficient statistics of the generated network. This is useful as a diagnostic. Default is FALSE.
<code>maxedges</code>	maximum number of edges expected in network
<code>network.output</code>	character; class with which to store the output networks. The options are "network" (default) and "edgelist.compressed" (which saves space but only supports networks without vertex attributes)
<code>maxchanges</code>	Currently unused
<code>packagenames</code>	Names of packages in which changestats 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 [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](#).

---

ecoli

*Two versions of an E. Coli network dataset*

---

### Description

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

### Usage

```
data(ecoli)
```

### Details

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

### Licenses and Citation

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

## Source

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

## References

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

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

---

edgelist.ergm	<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

x	an R object. Either a network or a matrix of some sort.
directed	logical: Are the edges directed? If x is a square matrix with NROWS(x)!=2, equal to !all(x==t(x))
check.uniqueness	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 FALSE will save computing time
check.sorted	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 FALSE will save computing time.
...	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](#)

---

enformulate.curved	<i>Convert a curved ERGM into a form suitable as initial values for the same ergm.</i>
--------------------	--

---

**Description**

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

**Usage**

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

**Arguments**

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

## Details

Because of a current kludge in [ergm](#), output from one run cannot be directly passed as initial values (`theta0`) for the next run if any of the terms are curved. One workaround is to embed the curved parameters into the formula (while keeping `fixed=FALSE`) and remove them from `theta0`.

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

## Value

A list with the following components:

<code>formula</code>	The formula with curved parameter estimates incorporated.
<code>theta</code>	The coefficient vector with curved parameter estimates removed.

## See Also

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

## Examples

```
data(sampson)
gest<-ergm(samplike~edges+gwesp(alpha=.5, fixed=FALSE), maxit=1)
# Error:
gest2<-try(ergm(gest$formula, theta0=coef(gest), maxit=2))
print(gest2)

# Works:
tmp<-enformulate.curved(gest)
tmp
gest2<-try(ergm(tmp$formula, theta0=tmp$theta, maxit=2))
summary(gest2)
```

## 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,  $\theta$  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, MLEstimate=!MPLEonly, seed=NULL,
      burnin=10000, MCMCsamplesize=10000, interval=100, maxit=3,
      constraints=~.,
      meanstats = NULL,
      control=control.ergm(),
      eval.loglik=FALSE,
      verbose=FALSE, ...)
```

**Arguments**

formula	formula; an R <a href="#">formula</a> object, of the form $y \sim \langle \text{model terms} \rangle$ , where $y$ is a <a href="#">network</a> object or a matrix that can be coerced to a <a href="#">network</a> object. For the details on the possible $\langle \text{model terms} \rangle$ , see <a href="#">ergm-terms</a> and Morris, Handcock and Hunter (2008). To create a <a href="#">network</a> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> operator if necessary.
theta0	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> ).
MPLEonly	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. See <a href="#">ergmMPLE</a> to obtain the vector of change statistics, needed to calculate the MPLE.
MLEstimate	logical; TRUE if only the Monte Carlo maximum likelihood estimate is to be computed and returned.
burnin	count; the number of proposals before any MCMC sampling is done. It typically is set to a fairly large number.
MCMCsamplesize	count; the number of network statistics, randomly drawn from a given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm.
interval	count; the number of proposals between sampled statistics.
maxit	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.
constraints	<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 <a href="#">control.ergm</a>.</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"> <li>• <b>or NULL</b> A placeholder for no constraints: all networks of a particular size and type have non-zero probability. Cannot be combined with other constraints.</li> </ul>

	<code>bd(attrs,maxout,maxin,minout,minin)</code> Constrain maximum and minimum vertex degree. See “Placing Bounds on Degrees” section for more information.
	<code>degrees</code> <b>and</b> <code>nodedegrees</code> 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.
	<code>degreedist</code> 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.
	<code>indegreedist</code> <b>and</b> <code>outdegreedist</code> Preserve the (respectively) indegree or out-degree distribution of the given network.
	<code>edges</code> 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.
	<code>observed</code> Preserve the observed dyads of the given network.
	Not all combinations of the above are supported.
<code>meanstats</code>	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, the mean-value parameters used are the observed statistics of the network in the formula.
<code>control</code>	A list of control parameters for algorithm tuning. Constructed using <code>control.ergm</code> .
<code>seed</code>	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.
<code>eval.loglik</code>	For dyad-dependent models, whether to use bridge sampling to evaluate the log-likelihood associated with the fit. Has no effect for dyad-independent models. Defaults to FALSE because bridge sampling takes additional time.
<code>verbose</code>	logical; if this is TRUE, the program will print out additional information, including goodness of fit statistics.
<code>...</code>	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:

<code>coef</code>	The Monte Carlo maximum likelihood estimate of $\theta$ , the vector of coefficients for the model parameters.
<code>sample</code>	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.
<code>iterations</code>	The number of Newton-Raphson iterations required before convergence.
<code>MCMCtheta</code>	The value of $\theta$ 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
burnin	If included, the burnin used for the MCMC simulation
interval	If included, the interval used for the MCMC simulation
network	Original network
theta.original	The first value of theta0
mplefit	The MPLE fit as a glm object.
null.deviance	Deviance of the null model.
mle.lik	The approximate log-likelihood for the MLE. The value is only approximate because it is estimated based on the MCMC random sample.
etamap	The set of functions mapping the true parameter theta to the canonical parameter eta (irrelevant except in a curved exponential family model)
degeneracy.value	Score calculated to assess the degree of degeneracy in the model.
degeneracy.type	Supporting output for degeneracy.value. Mainly for internal use.
formula	The original <a href="#">formula</a> entered into the <a href="#">ergm</a> function.
constraints	Constraints used by original <a href="#">ergm</a> call
prop.weights	MCMC proposal weights used by original <a href="#">ergm</a> call (part of the <a href="#">control.ergm</a> function output).
offset	vector of logical telling which model parameters are to be set at a fixed value (i.e., not estimated).
drop	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 mechanism for proposing new networks for the MCMC sampling scheme, which is a Metropolis-Hastings algorithm, depends on two things: The constraints, which define the set of possible networks that could be proposed in a particular Markov chain step, and the weights placed on these possible steps by the proposal distribution. The former may be controlled using the `constraints` argument described above. The latter may be controlled using the `prop.weights` argument to the `control.ergm` function.

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

### 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 outedges to males and five outedges 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.

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

**See Also**

`network`, `%v%`, `%n%`, [ergm-terms](#), [ergmMPLE](#), [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  $y$ ,  $\theta$  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 `abs(attrname[i]-attrname[j])^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 `abs(attrname[i]-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 `abs(attrname[i]-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.

`b1concurrent(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 `b1degree` term. This term can only be used with undirected bipartite networks.

- `b1degree(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.
- `b1factor(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.
- `b1star(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 `b1star(1)` is equal to `b2star(1)` and to edges.
- `b1starmix(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  $b_2$  nodes (called events in some contexts) are homophilous in the sense that they all share the same value of `attrname`. However, the  $b_1$  node (in some contexts, the actor) at the center of the  $k$ -star does NOT have to have the same value as the  $b_2$  nodes; indeed, the values taken by the  $b_1$  nodes may be completely distinct from those of the  $b_2$  nodes, which allows for the use of this term in cases where there are two separate nodal attributes, one for the  $b_1$  nodes and another for the  $b_2$  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  $b_1$  node, even if no  $k$ -stars are observed with this value. Whether a different statistic is created for each value seen in a  $b_2$  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.

`b2twostar(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 `b1twostar` 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, homophily=FALSE)` *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 and `homophily` is `TRUE`, then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the `by` attribute. If `by` is specified and `homophily` is `FALSE` (the default), then separate degree

statistics are calculated for nodes having each separate value of the attribute. This term can only be used with undirected networks; for directed networks see `idegree` and `odegree`.

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

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

`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 `edgescov` and `dyadcov` terms are equivalent for undirected networks.

`edgescov(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 `edgescov` term applies to both directed and undirected networks. For undirected networks the covariates are also assumed to be undirected. The `edgescov` 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).

- `gwb1degree(decay, fixed=FALSE, cutoff=30)` *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 decay controlled by the decay parameter, 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. The decay parameter is the same as  $\theta_a$  in equation (14) in Hunter (2007). The value supplied for this parameter may be fixed (if `fixed=TRUE`), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden. This term can only be used with undirected bipartite networks.
- `gwb2degree(decay, fixed=FALSE, cutoff=30)` *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 decay controlled by the decay parameter, 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. The decay parameter is the same as  $\theta_e$  in equation (14) in Hunter (2007). The value supplied for this parameter may be fixed (if `fixed=TRUE`), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden. This term can only be used with undirected bipartite networks.
- `gwdegree(decay, fixed=FALSE, cutoff=30)` *Geometrically weighted degree distribution:* This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the decay parameter. The decay parameter is the same as  $\theta_s$  in equation (14) in Hunter (2007). The value supplied for this parameter may be fixed (if `fixed=TRUE`), or it may be used as merely the starting value for the estimation in a curved exponential family model (the default). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden. This term can only be used with undirected networks.
- `gwdsp(alpha, fixed=FALSE, cutoff=30)` *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). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden.
- `gwesp(alpha, fixed=FALSE, cutoff=30)` *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). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden.

`gwidegree(decay, fixed=FALSE, cutoff=30)` *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. The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden.

`gwensp(alpha, fixed=FALSE, cutoff=30)` *Geometrically weighted nonedgewise shared partner distribution:*

This term is just like `gwesp` and `gw dsp` 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). The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden.

`gwodegree(decay, fixed=FALSE, cutoff=30)` *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. The optional argument `cutoff` is only relevant if `fixed=FALSE`. In that case it only uses this number of terms in computing the statistics to reduce the computational burden.

`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, homophily=FALSE)` *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 and `homophily` is `TRUE`, then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the `by` attribute. If `by` is specified and `homophily` is `FALSE` (the default), then separate degree statistics are calculated for nodes having each separate value of the attribute. 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(same=NULL, diff=FALSE, by=NULL, keep=NULL)` *Mutuality*: 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 `same` argument is passed the name of a vertex attribute, only mutual pairs that match on the attribute are counted; separate counts for each unique matching value can be obtained by using `diff=TRUE` with `same`. If `by` is passed the name of a vertex attribute, then each node is counted separately for each mutual pair in which it occurs and the counts are tabulated by unique values of the attribute. This means that the sum of the mutual statistics when `by` is used will equal twice the standard mutual statistic. Only one of `same` or `by` may be used, and only the former is affected by `diff`; if both `same` and `by` are passed, `by` is ignored. Finally, if `keep` is passed a numerical vector, this vector of integers tells which statistics should be kept whenever the mutual term would ordinarily result in multiple statistics.

`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.

`nodeocov(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, homophily=FALSE)` *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 and `homophily` is `TRUE`, then degrees are calculated using the subnetwork consisting of only edges whose endpoints have the same value of the `by` attribute. If `by` is specified and `homophily` is `FALSE` (the default), then separate degree statistics are calculated for nodes having each separate value of the attribute. 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 subgraph 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 Simmelians (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.

`transitiveties(attrname=NULL)` *Transitive ties*: This term adds one statistic, equal to the number of ties  $i \rightarrow j$  such that there exists a two-path from  $i$  to  $j$ . (Related to the `ttriple` term.) When a nodal attribute is passed via `attrname`, all three nodes involved ( $i$ ,  $j$ , and the node on the two-path) must match on this attribute in order for  $i \rightarrow j$  to be counted. 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 `ctriple` — 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). In case the denominator equals zero, the statistic is defined to be zero. 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 is often called the mean correlation coefficient. This term can only be used with undirected networks; for directed networks, it is difficult to define the numerator and denominator in a consistent and meaningful way.

`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`+`ctriple` 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)
```

---

`ergm.allstats`

*Calculate all possible vectors of statistics on a network for an ERGM*

---

## Description

`ergm.allstats` produces a matrix of network statistics for an arbitrary statnet exponential-family random graph model. One possible use for this function is to calculate the exact loglikelihood function for a small network via the `ergm.exact` function.

**Usage**

```
ergm.allstats (formula, zeroobs = TRUE, force = FALSE,
              maxNumChangeStatVectors = 2^16, ...)
```

**Arguments**

formula	an R <code>link{formula}</code> object of the form <code>y ~ &lt;model terms&gt;</code> , where <code>y</code> is a network object or a matrix that can be coerced to a <a href="#">network</a> object. For the details on the possible <code>&lt;model terms&gt;</code> , see <a href="#">ergm-terms</a> . To create a <a href="#">network</a> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%v%</code> operator if necessary.
zeroobs	Logical: Should the vectors be centered so that the network passed in the formula has the zero vector as its statistics?
force	Logical: Should the algorithm be run even if it is determined that the problem may be very large, thus bypassing the warning message that normally terminates the function in such cases?
maxNumChangeStatVectors	Maximum possible number of distinct values of the vector of statistics. It's good to use a power of 2 for this.
...	further arguments; not currently used.

**Details**

The mechanism for doing this is a recursive algorithm, where the number of levels of recursion is equal to the number of possible dyads that can be changed from 0 to 1 and back again. The algorithm starts with the network passed in `formula`, then recursively toggles each edge twice so that every possible network is visited.

`ergm.allstats` should only be used for small networks, since the number of possible networks grows extremely fast with the number of nodes. An error results if it is used on a directed network of more than 6 nodes or an undirected network of more than 8 nodes; use `force=TRUE` to override this error.

**Value**

Returns a list object with these two elements:

weights	integer of counts, one for each row of <code>statmat</code> telling how many networks share the corresponding vector of statistics.
statmat	matrix in which each row is a unique vector of statistics.

**See Also**

[ergm.exact](#)

## Examples

```
# Count by brute force all the edge statistics possible for a 7-node
# undirected network
mynw <- network(matrix(0,7,7),dir=FALSE)
unix.time(a <- ergm.allstats(mynw~edges))

# Summarize results
rbind(t(a$statmat),a$weights)

# Each value of a$weights is equal to 21-choose-k,
# where k is the corresponding statistic (and 21 is
# the number of dyads in an 7-node undirected network).
# Here's a check of that fact:
as.vector(a$weights - choose(21, t(a$statmat)))

# Simple ergm.exact output for this network.
# We know that the loglikelihood for my empty 7-node network
# should simply be -21*log(1+exp(eta)), so we may check that
# the following two values agree:
-21*log(1+exp(.1234))
ergm.exact(.1234, mynw~edges, statmat=a$statmat, weights=a$weights)
```

---

```
ergm.bridge.dindstart.llk
```

*Bridge sampling to estimate log-likelihood of an ERGM, using a dyad-independent ERGM as a starting point.*

---

## Description

This function is a wrapper around [ergm.bridge.llr](#) that uses a dyad-independent ERGM as a starting point for bridge sampling to estimate the log-likelihood for a given dyad-dependent model and parameter configuration. The dyad-independent model may be specified or can be chosen adaptively.

## Usage

```
ergm.bridge.dindstart.llk(object, theta, nsteps = 20, dind = NULL, theta.dind = NULL, basis = NULL, llk)
```

## Arguments

object	A model formula. See <a href="#">ergm</a> for details.
basis	See <a href="#">simulate.ergm</a> .
theta	A vector of coefficients for the configuration of interest.
nsteps	Number of geometric bridges to use.
dind	A one-sided formula with the dyad-independent model to use as a starting point. Defaults to the dyad-independent terms found in the formula object with an overall density term (edges) added if not redundant.

theta.dind	Parameter configuration for the dyad-independent starting point. Defaults to the MLE of dind.
llkonly	Whether only the estimated log-likelihood should be returned. (Defaults to TRUE.)
...	Further arguments to <a href="#">ergm.bridge.llr</a> and <a href="#">simulate.formula.ergm</a> .

**Value**

If llkonly=TRUE, returns the scalar log-likelihood. Otherwise, returns a copy of the list returned by [ergm.bridge.llr](#) in addition to the following components:

llk.dind	The log-likelihood of the dyad-independence model.
llk	The estimated log-likelihood.

**References**

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

**See Also**

[ergm.bridge.llr](#), [simulate.formula.ergm](#)

---

ergm.bridge.llr	<i>A simple implementation of bridge sampling to evaluate log-likelihood-ratio between two ERGM configurations</i>
-----------------	--

---

**Description**

This function uses bridge sampling with geometric spacing to estimate the difference between the log-likelihoods of two parameter configurations for an ERGM via repeated calls to [simulate.formula.ergm](#).

**Usage**

```
ergm.bridge.llr(object, from, to, nsteps = 20,
  sample.size = 10000, burnin = 10000, basis = NULL, verbose = FALSE, llronly = FALSE, ...)
```

**Arguments**

object	A model formula. See <a href="#">ergm</a> for details.
basis	See <a href="#">simulate.ergm</a> .
verbose	How much information to print about the progress.
from	The initial parameter configuration.
to	The final parameter configuration.
nsteps	Number of geometric bridges to use.

sample.size	Total number of MCMC draws to use (to be divided up among the bridges, so each bridge gets <code>sample.size/nsteps</code> draws).
burnin	Total number of steps draws to discard for each bridge (to be divided up among the bridges, so each bridge after the first gets <code>sample.size/sqrt(nsteps)</code> ).
llronly	Whether only the estimated log-ratio should be returned. (Defaults to FALSE.)
...	Further arguments to <a href="#">simulate.formula.ergm</a> .

**Value**

If `llronly=TRUE`, returns the scalar log-likelihood-ratio. Otherwise, returns a list with the following components:

llr	The estimated log-ratio.
llrs	The estimated log-ratios for each of the <code>nsteps</code> bridges.
path	A numeric matrix with <code>nsteps</code> rows, with each row being the respective bridge's parameter configuration.
stats	A numeric matrix with <code>nsteps</code> rows, with each row being the respective bridge's vector of simulated statistics.
Dtheta.Du	The gradient vector of the parameter values with respect to position of the bridge.

**References**

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

**See Also**

[simulate.formula.ergm](#)

---

ergm.exact

*Calculate the exact loglikelihood for an ERGM*

---

**Description**

`ergm.exact` calculates the exact loglikelihood, evaluated at `eta`, for the `statnet` exponential-family random graph model represented by `formula`.

**Usage**

`ergm.exact(eta, formula, statmat=NULL, weights=NULL, ...)`

**Arguments**

eta	vector of canonical parameter values at which the loglikelihood should be evaluated.
formula	an R <code>link{formula}</code> 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 <a href="#">network</a> object. For the details on the possible <code>&lt;model terms&gt;</code> , see <a href="#">ergm-terms</a> . To create a <a href="#">network</a> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%%</code> operator if necessary.
statmat	if NULL, call <a href="#">ergm.allstats</a> to generate all possible graph statistics for the networks in this model.
weights	In case <code>statmat</code> is not NULL, this should be the vector of counts corresponding to the rows of <code>statmat</code> . If <code>statmat</code> is NULL, this is generated by the call to <a href="#">ergm.allstats</a> .
...	further arguments; not currently used.

**Details**

`ergm.exact` should only be used for small networks, since the number of possible networks grows extremely fast with the number of nodes. An error results if it is used on a directed network of more than 6 nodes or an undirected network of more than 8 nodes; use `force=TRUE` to override this error.

In case this function is to be called repeatedly, for instance by an optimization routine, it is preferable to call [ergm.allstats](#) first, then pass `statmat` and `weights` explicitly to avoid repeatedly calculating these objects.

**Value**

Returns the value of the exact loglikelihood, evaluated at `eta`, for the `statnet` exponential-family random graph model represented by `formula`.

**See Also**

[ergm.allstats](#)

**Examples**

```
# Count by brute force all the edge statistics possible for a 7-node
# undirected network
mynw <- network(matrix(0,7,7),dir=FALSE)
unix.time(a <- ergm.allstats(mynw~edges))

# Summarize results
rbind(t(a$statmat),a$weights)

# Each value of a$weights is equal to 21-choose-k,
# where k is the corresponding statistic (and 21 is
# the number of dyads in an 7-node undirected network).
# Here's a check of that fact:
as.vector(a$weights - choose(21, t(a$statmat)))
```

```
# Simple ergm.exact output for this network.
# We know that the loglikelihood for my empty 7-node network
# should simply be -21*log(1+exp(eta)), so we may check that
# the following two values agree:
-21*log(1+exp(.1234))
ergm.exact(.1234, mynw~edges, statmat=a$statmat, weights=a$weights)
```

---

ergmMPLE	<i>ERG M Predictors and response for logistic regression calculation of MPLE</i>
----------	--

---

### Description

Return the predictor matrix, response vector, and vector of weights that can be used to calculate the MPLE for an ERGM.

### Usage

```
ergmMPLE (formula, fitmodel=FALSE, control=control.ergm(),
          verbose=FALSE, ...)
```

### Arguments

formula	An ergm formula. See <a href="#">ergm</a> .
fitmodel	Logical: Should the MPLE be found? If <code>fitmodel==TRUE</code> , then <code>ergmMPLE</code> simply calls the <a href="#">ergm</a> function with the <code>MPLEonly=TRUE</code> option set. If <code>fitmodel==FALSE</code> (the default), then only the response, predictor, and weights are returned; thus, the MPLE may be found by hand or the vector of change statistics may be used in some other way.
control	A list of control parameters for tuning the fitting of an ERGM. Most of these parameters are irrelevant in this context. See <a href="#">control.ergm</a> for details about all of the control parameters.
verbose	Logical; if <code>TRUE</code> , the program will print out some additional information.
...	Additional arguments, to be passed to lower-level functions.

### Details

The MPLE for an ERGM is calculated by first finding the matrix of change statistics. Each row of this matrix is associated with a particular pair (ordered or unordered, depending on whether the network is directed or undirected) of nodes, and the row equals the change in the vector of network statistics (as defined in `formula`) when that pair is toggled from a 0 (no edge) to a 1 (edge), holding all the rest of the network fixed. The MPLE results if we perform a logistic regression in which the predictor matrix is the matrix of change statistics and the response vector is the observed network (i.e., each entry is either 0 or 1, depending on whether the corresponding edge exists or not).

To save space, the algorithm will automatically search for any duplicated rows in the predictor matrix (and corresponding response values). Thus, the `ergmMPLE` function will return three items:

A response vector, a predictor matrix, and a vector of weights, which are really counts that tell how many times each corresponding response, predictor pair is repeated.

Using `fitmodel=FALSE`, note that the result of the fit may be obtained from the `glm` function, as shown in the examples below.

### Value

If `fitmodel==FALSE`, a list with three elements, called "response", "predictor", and "weights". If `fitmodel==TRUE`, an object of class `ergm` that gives the fitted pseudolikelihood model.

### See Also

[ergm](#), [glm](#)

### Examples

```
data(faux.mesa.high)
formula <- faux.mesa.high ~ edges + nodematch("Sex") + nodefactor("Grade")
mplesetup <- ergmMPLC(formula)

# Obtain MPLC coefficients "by hand":
glm(mplesetup$response ~ . - 1, data = data.frame(mplesetup$predictor),
     weights = mplesetup$weights, family="binomial")$coefficients

# Check that the coefficients agree with the output of the ergm function:
ergmMPLC(formula, fitmodel=TRUE)$coef
```

---

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](#)

---

fix.curved

*Convert a curved ERGM into a corresponding “fixed” ERGM.*

---

## Description

The generic `fix.curved` converts an `ergm` object or formula of a model with curved terms to the variant in which the curved parameters are fixed. Note that each term has to be treated as a special case.

## Usage

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

## Arguments

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

## Details

Some ERGM terms such as [gwesp](#) and [gwdegree](#) have two forms: a curved form, for which their decay or similar parameters are to be estimated, and whose canonical statistics is a vector of the term's components ([esp\(1\)](#), [esp\(2\)](#), ... and [degree\(1\)](#), [degree\(2\)](#), ..., respectively) and a "fixed" form where the decay or similar parameters are fixed, and whose canonical statistic is just the term itself. It is often desirable to fit a model estimating the curved parameters but simulate the "fixed" statistic.

This function thus takes in a fit or a formula and performs this mapping, returning a "fixed" model and parameter specification. It only works for curved ERGM terms included with the [ergm](#) package. It does not work with curved terms not included in [ergm](#).

## Value

A list with the following components:

formula	The "fixed" formula.
theta	The "fixed" parameter vector.

## See Also

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

## Examples

```
data(sampson)
gest<-ergm(samplike~edges+gwesp(alpha=.5,fixed=FALSE))
summary(gest)
# A statistic for esp(1),...,esp(16)
simulate(gest,statonly=TRUE)

tmp<-fix.curved(gest)
tmp
# A gwesp() statistic only
simulate(tmp$formula,theta0=tmp$theta,statonly=TRUE)
```

---

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

---

flomarriage

*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 (`flomarriage` 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), MPLExonly=TRUE))
summary(ergm(g4 ~ odegree(3), theta0=0))
```

---

Getting.Started	<i>Getting Started with "ergm": 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 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.ucla.edu](mailto:handcock@stat.ucla.edu)>,  
David R. Hunter <[dhunter@stat.psu.edu](mailto:dhunter@stat.psu.edu)>,  
Carter T. Butts <[buttsc@uci.edu](mailto:buttsc@uci.edu)>,  
Steven M. Goodreau <[goodreau@u.washington.edu](mailto:goodreau@u.washington.edu)>,

Pavel N. Krivitsky <pavel@stat.cmu.edu>, and  
 Martina Morris <morris@u.washington.edu>  
 Maintainer: David R. Hunter <dhunter@stat.psu.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](http://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	<i>Conduct Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model</i>
-----	---

---

## 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=NULL,
     constraints=~.,
     control=control.gof.formula(),
     seed=NULL,
     verbose=FALSE)
## S3 method for class 'ergm'
gof(object, ...,
     nsim=100,
     GOF=NULL,
     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 <a href="#">ergm</a> object. See documentation for <a href="#">ergm</a> .
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 <a href="#">ergm-terms</a> . 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. Currently supported terms are the degree distribution (“degree” for undirected graphs, or “idegree” and/or “odegree” for directed graphs), geodesic distances (“distance”), shared partner distributions (“espartners” and “dpartners”), the triad census (“triadcensus”), and the terms of the original model (“model”).
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 $100 \cdot (1 + \text{the number of toggles accepted})$ ).
constraints	A one-sided formula specifying one or more constraints on the support of the distribution of the networks being modeled. See the help for similarly-named argument in <a href="#">ergm</a> for more information. For <code>gof.formula</code> , defaults to unconstrained. For <code>gof.ergm</code> , defaults to the constraints with which object was fitted.
control	A list to control parameters, constructed using <code>control.gof.formula</code> or <code>control.gof.ergm</code> (which have different defaults).
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	Provide verbose information on the progress of the simulation.
...	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](#).

For `gof.ergm` and `gof.formula`, default behavior depends on the directedness of the network involved; if undirected then degree, espartners, and distance are used as default properties to examine. If the network in question is directed, “degree” in the above is replaced by `idegree` and `odegree`.

### 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)
```

---

is.dyad.independent     *Testing for dyad-independence*

---

### Description

These functions test whether an ERGM fit or formula is dyad-independent.

### Usage

```
## S3 method for class 'ergm'
is.dyad.independent(object, ...)
## S3 method for class 'formula'
is.dyad.independent(object, basis = NULL, constraints = ~., ...)
```

**Arguments**

object            An `ergm` object or an ERGM formula.  
 basis, constraints            See `ergm`.  
 ...                Unused at this time.

**Value**

TRUE if the model fit or one implied by the formula is dyad-independent; FALSE otherwise.

---

is.inCH	<i>Determine whether a vector is in the closure of the convex hull of some sample of vectors</i>
---------	--

---

**Description**

is.inCH returns TRUE if and only if  $p$  is contained in the convex hull of the points given as the rows of  $M$ .

**Usage**

```
is.inCH(p, M)
```

**Arguments**

$p$                 A  $d$ -dimensional vector  
 $M$                 An  $r$  by  $d$  matrix. Each row of  $M$  is a  $d$ -dimensional vector.

**Details**

This function depends on the package `Rglpk` to solve a constrained linear optimization problem in order to determine an answer. The question of whether  $p$  is in a closed convex set  $S$  may be formulated as the question of whether there exists a separating hyperplane between  $p$  and  $S$ , which may in turn be formulated as the question of whether the maximum possible value of a linear function, subject to constraints, has a strictly positive solution.

Note that the answer given could be incorrect simply due to rounding error if the true maximum is close to zero. For this reason, the package `rcdd`, which produces exact rational-number solutions to linear programs, could be used instead of `Rglpk`. However, this approach would require more computing and would therefore be slower.

**Value**

Logical, telling whether  $p$  is in the closed convex hull of the points in  $M$ .

**References**

<http://www.cs.mcgill.ca/~fukuda/soft/polyfaq/node22.html>

---

kapferer

*Kapferer's tailor shop data*

---

## Description

This well-known social network dataset, collected by Bruce Kapferer in Zambia from June 1965 to August 1965, involves interactions among workers in a tailor shop as observed by Kapferer himself. Here, an interaction is defined by Kapferer as "continuous uninterrupted social activity involving the participation of at least two persons"; only transactions that were relatively frequent are recorded. All of the interactions in this particular dataset are "sociational", as opposed to "instrumental". Kapferer explains the difference (p. 164) as follows:

"I have classed as transactions which were sociational in content those where the activity was markedly convivial such as general conversation, the sharing of gossip and the enjoyment of a drink together. Examples of instrumental transactions are the lending or giving of money, assistance at times of personal crisis and help at work."

Kapferer also observed and recorded instrumental transactions, many of which are unilateral (directed) rather than reciprocal (undirected), though those transactions are not recorded here. In addition, there was a second period of data collection, from September 1965 to January 1966, but these data are also not recorded here. All data are given in Kapferer's 1972 book on pp. 176-179.

During the first time period, there were 43 individuals working in this particular tailor shop; however, the better-known dataset includes only those 39 individuals who were present during both time collection periods. (Missing are the workers named Lenard, Peter, Lazarus, and Laurent.) Thus, we give two separate network datasets here: `kapferer` is the well-known 39-individual dataset, whereas `kapferer2` is the full 43-individual dataset.

## Usage

```
data(kapferer)
```

## Format

Two network objects, `kapferer` and `kapferer2`. The `kapferer` dataset contains only the 39 individuals who were present at both data-collection time periods. However, these data only reflect data collected during the first period. The individuals' names are included as a nodal covariate called `names`.

## Source

Original source: Kapferer, Bruce (1972), *Strategy and Transaction in an African Factory*, Manchester University Press.

---

logLik.ergm

*A logLik method for ergm*


---

### Description

A function to return the log-likelihood associated with an [ergm](#) fit, evaluating it if necessary.

### Usage

```
## S3 method for class 'ergm'
logLik(object, nsteps = 20, add = FALSE, force.reeval =
FALSE, eval.loglik = add || force.reeval, ...)
```

### Arguments

object	An <a href="#">ergm</a> fit, returned by <a href="#">ergm</a> .
nsteps	Number of bridges for bridge sampling estimation.
add	Instead of returning the log-likelihood, return object with log-likelihood value set.
eval.loglik	Evaluate the log-likelihood if not set on object.
force.reeval	Whether to reestimate the log-likelihood even if object already has an estimate. Defaults to FALSE.
...	Other arguments to the bridge sampling functions.

### Details

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

### Value

If add=FALSE (the default), a [logLik](#) object. If add=TRUE (the default), an [ergm](#) object with the log-likelihood set.

### References

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

### See Also

[logLik](#), [ergm.bridge.llr](#), [ergm.bridge.dindstart.llk](#)

## Examples

```
# See help(ergm) for a description of this model:
data(florentine)
gest <- ergm(flomarriage ~ kstar(1:2) + absdiff("wealth") + triangle)
# Log-likelihood is not evaluated, so no deviance, AIC, or BIC:
summary(gest)
# Evaluate the log-likelihood and attach it to the object.
gest <- logLik(gest, add=TRUE)
# Deviances, AIC, and BIC now shown:
summary(gest)
```

---

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 = "", curved=TRUE, ...)
```

## Arguments

object	An object. See documentation for <a href="#">ergm</a> .
sample	The component of object on which the diagnosis is based. The two usuals ones are thetasample from the auxiliary sample of the natural parameter and sample 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

curved            If this is TRUE, summarize the curved statistics (evaluated at the MLE of any non-linear parameters), rather than the raw components of the curved statistics.

...                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 coda 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.

### 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.

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

### See Also

[ergm](#), network package, coda package, mcgibbsit package, [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 is 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	<i>Replaces the sociomatrix in a network object</i>
----------------	---

---

**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, output="network")
```

**Arguments**

<code>nw</code>	a <a href="#">network</a> object. See documentation for the <a href="#">network</a> package.
<code>newmatrix</code>	Either an adjacency matrix (a matrix of zeros and ones indicating the presence of a tie from <i>i</i> to <i>j</i> ) 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).
<code>matrix.type</code>	One of "adjacency" or "edgelist" telling which type of matrix <code>newmatrix</code> is. Default is to use the <a href="#">which.matrix.type</a> function.
<code>output</code>	Currently unused.

**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.

drawarrows	If density=TRUE, draws the ties on the density plot.
plotnetwork	If density=c(nr,nc), a plot of the network is also shown.
comp.mat	For latent models, the positions are Procrustes transformed to look like comp.mat.
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 mycol. Can be specified as an attribute of the network used in the model.
vertex.pch	The plotting character of the nodes as defined by mypch. 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 mycex. Can be specified as an attribute of the network used in the model.
mycol	Vector of colors to be used. Defaults to: c("black","red","green","blue","cyan","magenta","orange","yellow","purple")
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)

```

**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 [sampsom](#) 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	<i>Cumulative network of positive affection within a monastery as a "network" object</i>
---------	--

---

**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

---

san	<i>Use Simulated Annealing to attempt to match a network to a vector of mean statistics</i>
-----	---

---

**Description**

This function attempts to find a network or networks whose statistics match those passed in via the meanstats vector.

**Usage**

```
## S3 method for class 'formula'
san(object, nsim=1, seed=NULL, theta0=NULL,
     tau=1, invcov=NULL,
     burnin=10000, interval=10000,
     meanstats=NULL,
     basis=NULL,
     sequential=TRUE,
     constraints = ~.,
     control = control.san(),
     verbose=FALSE, ...)

## S3 method for class 'ergm'
san(object, nsim=1, seed=NULL, theta0=object$coef,
     burnin=10000, interval=10000,
     meanstats=NULL,
     basis=NULL,
     sequential=TRUE,
     constraints = NULL,
     control = control.san(),
     verbose=FALSE, ...)
```

**Arguments**

object	an R object. Either a <a href="#">formula</a> or an <a href="#">ergm</a> object. The <a href="#">formula</a> should be 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 <a href="#">network</a> object. For the details on the possible $\langle \text{model terms} \rangle$ , see <a href="#">ergm-terms</a> . To create a <a href="#">network</a> object in R, use the <code>network()</code> function, then add nodal attributes to it using the <code>%%</code> operator if necessary.
nsim	Number of desired networks.
seed	Random number integer seed.
theta0	Parameter values used for MCMC simulations.
tau	Currently unused.
invcov	Initial inverse covariance matrix used to calculate Mahalanobis distance in determining how far a proposed MCMC move is from the <code>meanstats</code> vector. If NULL, taken to be the covariance matrix returned when fitting the MPLE (if <code>theta0==NULL</code> ) or the identity (otherwise).
burnin	Number of MCMC steps prior to recording first vector of network statistics.
interval	Number of MCMC steps between recordings of network statistics
meanstats	A vector of the same length as the number of terms implied by the formula, which is either object itself in the case of <code>san.formula</code> or <code>object\$formula</code> in the case of <code>san.ergm</code> .
basis	If not NULL, a network that forms the beginning of the Markov chain. If NULL, this is taken to be the network named in the formula.
sequential	Logical: Should the returned draws use the prior draw as the starting network or always use the initially passed network?
constraints	A one-sided formula specifying one or more constraints on the support of the distribution of the networks being simulated. See the documentation for a similar argument for <a href="#">ergm</a> 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 object was fitted.
control	A list of control parameters for algorithm tuning. Constructed using <a href="#">control.san</a> .
verbose	If this is TRUE, we will print out more information as we run the program, including (currently) some goodness of fit statistics.
...	Further arguments passed to or used by methods.

**Value**

A network or list of networks that hopefully have network statistics close to the `meanstats` vector.

---

simulate.ergm      *Draw from the distribution of an Exponential Family Random Graph Model*

---

## Description

`simulate` is used to draw from exponential family random network models 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=object$coef,
         burnin=1000, interval=1000,
         statonly=FALSE,
         sequential=TRUE,
         constraints = NULL,
         control = control.simulate.ergm(),
         verbose=FALSE, ...)
```

## Arguments

<code>object</code>	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 ~ &lt;model terms&gt;</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>&lt;model terms&gt;</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.
<code>nsim</code>	Number of networks to be randomly drawn from the given distribution on the set of all networks, returned by the Metropolis-Hastings algorithm.
<code>seed</code>	Random number integer seed. The default is <code>sample(10000000, size=1)</code> .
<code>theta0</code>	Vector of parameters for the model from which the sample is to be drawn. If object is of class <code>ergm</code> , the default value is the vector of estimated coefficients.
<code>burnin</code>	The number of proposals before any MCMC sampling is done. If <code>NULL</code> in <code>simulate.ergm</code> , the value of <code>object\$burnin</code> is used.
<code>interval</code>	The number of proposals between sampled statistics. If <code>NULL</code> in <code>simulate.ergm</code> , the value of <code>object\$interval</code> is used. 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 $100(1+\text{the number of proposals accepted})$ ).
basis	An optional <a href="#">network</a> object to start the MCMC algorithm from. This overrides the left-hand-side of the formula. If neither a left-hand-side nor a basis is present, an error results because the characteristics of the network (e.g., size and directedness) must be specified.
constraints	A one-sided formula specifying one or more constraints on the support of the distribution of the networks being simulated. See the documentation for a similar argument for <a href="#">ergm</a> 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 object was fitted.
control	A list of control parameters for algorithm tuning. Constructed using <a href="#">control.simulate.ergm</a> or <a href="#">control.simulate.formula</a> , which have different defaults.
statonly	If TRUE, return only the network statistics (not the network(s) themselves)
sequential	If FALSE, each new simulation (of <code>nsim</code> ) is always started at the initial network. If TRUE, the end of one simulation is used as the start of the next. Irrelevant when <code>nsim=1</code> .
verbose	If this is TRUE, we will print out more information as we run the program, including (currently) some goodness of fit statistics.
...	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:

formula	The <a href="#">formula</a> used to generate the sample.
networks	A list of the generated networks.
stats	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

```

## 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

object	an object of class "ergm", usually, a result of a call to <a href="#">ergm</a> .
digits	Significant digits for coefficients
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
covariance	logical; if TRUE, the covariance matrix of the estimated parameters is returned and printed.
eps	number; indicates the smallest p-value. See <a href="#">printCoefmat</a> .
...	Arguments to <a href="#">logLik.ergm</a>

## 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	<i>Summaries the Goodness-of-Fit Diagnostics on a Exponential Family Random Graph Model</i>
-------------------	---

---

## 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 <code>gofobject</code> , typically produced by the <code>gof.ergm</code> or <code>gof.formula</code> 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 <a href="#">formula</a> object (see above) or an <a href="#">ergm</a> 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 ~ &lt;model terms&gt;</code> , where <code>y</code> is a <a href="#">network</a> object or a matrix that can be coerced to a <a href="#">network</a> object. For the details on the possible <code>&lt;model terms&gt;</code> , see <a href="#">ergm-terms</a> . To create a <a href="#">network</a> 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 <a href="#">network</a> 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
```

---

wtd.median

*Weighted Median*

---

**Description**

Compute weighted median.

**Usage**

```
wtd.median (x, na.rm = FALSE, weight=FALSE)
```

**Arguments**

x	Vector of data, same length as weight
na.rm	Logical: Should NAs be stripped before computation proceeds?
weight	Vector of weights

**Details**

Uses a simple algorithm based on sorting.

**Value**

Returns an empirical .5 quantile from a weighted sample.

# Index

- \*Topic **classes**
    - as.network.numeric, 7
  - \*Topic **datasets**
    - ecoli, 17
    - faux.magnolia.high, 49
    - faux.mesa.high, 51
    - flobusiness, 53
    - flomarriage, 54
    - florentine, 55
    - g4, 56
    - kapferer, 63
    - molecule, 67
    - samplk, 74
    - sampson, 75
  - \*Topic **graphs**
    - as.network.numeric, 7
    - ergmuserterms-package, 5
    - plot.gofobject, 71
    - summary.gofobject, 82
  - \*Topic **models**
    - anova.ergm, 6
    - coef.ergm, 9
    - control.ergm, 10
    - control.gof, 14
    - control.san, 15
    - control.simulate, 16
    - edgelist.ergm, 18
    - ergm, 20
    - ergm-package, 3
    - ergm-terms, 28
    - ergm.allstats, 42
    - ergm.exact, 46
    - ergmMPL, 48
    - Getting.Started, 57
    - gof, 59
    - logLik.ergm, 64
    - mcmc.diagnostics.ergm, 65
    - network.update, 68
    - plot.ergm, 69
    - print.ergm, 73
    - san, 76
    - simulate.ergm, 78
    - summary.ergm, 80
    - summary.statistics, 83
  - \*Topic **model**
    - enformulate.curved, 19
    - ergm.bridge.dindstart.llk, 44
    - ergm.bridge.llr, 45
    - fix.curved, 52
    - is.dyad.independent, 61
  - \*Topic **package**
    - ergm-package, 3
    - Getting.Started, 57
  - \*Topic **regression**
    - anova.ergm, 6
    - coef.ergm, 9
    - ergmMPL, 48
    - summary.ergm, 80
  - \*Topic **robust**
    - wtd.median, 84
- 
- absdiff (ergm-terms), 28
  - absdiffcat (ergm-terms), 28
  - altkstar (ergm-terms), 28
  - anova, 7
  - anova.ergm, 6
  - anova.ergmlist, 7
  - anova.ergmlist (anova.ergm), 6
  - as.matrix.network, 19
  - as.network.numeric, 7, 7
  - asymmetric (ergm-terms), 28
  
  - b1concurrent (ergm-terms), 28
  - b1degree (ergm-terms), 28
  - b1factor (ergm-terms), 28
  - b1star (ergm-terms), 28
  - b1starmix (ergm-terms), 28
  - b1twostar (ergm-terms), 28
  - b2concurrent (ergm-terms), 28

- b2degree (ergm-terms), 28
- b2factor (ergm-terms), 28
- b2star (ergm-terms), 28
- b2starmix (ergm-terms), 28
- b2twostar (ergm-terms), 28
- balance (ergm-terms), 28
  
- coef, 81
- coef.ergm, 9
- coefficients.ergm (coef.ergm), 9
- concurrent (ergm-terms), 28
- control.ergm, 10, 15, 17, 21–24, 48
- control.gof, 14, 14, 17
- control.gof.ergm, 60
- control.gof.formula, 60
- control.san, 15, 77
- control.simulate, 14, 15, 16
- control.simulate.ergm, 79
- control.simulate.formula, 79
- ctriple (ergm-terms), 28
- cycle (ergm-terms), 28
  
- degcor (ergm-terms), 28
- degcrossprod (ergm-terms), 28
- degree, 53
- degree (ergm-terms), 28
- density (ergm-terms), 28
- dsp (ergm-terms), 28
- dyadcov (ergm-terms), 28
  
- ecoli, 17
- ecoli1 (ecoli), 17
- ecoli2 (ecoli), 17
- edgescov (ergm-terms), 28
- edgelist.ergm, 18, 18
- edges (ergm-terms), 28
- enformulate.curved, 19
- ergm, 3, 5–7, 9, 10, 13–15, 17, 19, 20, 20, 22–24, 28, 29, 44, 45, 48–53, 57, 59, 60, 62, 64–70, 72, 73, 77–79, 81–83
- ergm-terms, 5, 21, 23, 27, 43, 47, 60, 77, 78, 83
- ergm-package, 3
- ergm-terms, 28
- ergm.allstats, 42, 47
- ergm.bridge.dindstart.llk, 44, 64
- ergm.bridge.llr, 44, 45, 45, 64
- ergm.exact, 42, 43, 46
- ergm.object, 14, 16
  
- ergm.terms (ergm-terms), 28
- ergmMPLE, 21, 27, 48
- ergmuserterms (ergmuserterms-package), 5
- ergmuserterms-package, 5
- esp, 53
- esp (ergm-terms), 28
  
- faux.magnolia.high, 29, 49, 52
- faux.mesa.high, 29, 50, 51
- fauxhigh (faux.mesa.high), 51
- fitted.values, 9
- fix.curved, 52
- flobusiness, 53, 53, 55
- flomarriage, 54, 54, 55
- florentine, 55
- formula, 5, 21, 23, 77–79, 83
  
- g4, 56
- Getting.Started, 57
- glm, 9, 49
- gof, 14, 15, 17, 59, 59, 61
- gof.ergm, 61, 72, 82
- gof.formula, 61, 72, 82
- gwb1degree (ergm-terms), 28
- gwb2degree (ergm-terms), 28
- gwdegree, 53
- gwdegree (ergm-terms), 28
- gwdsp (ergm-terms), 28
- gwesp, 53
- gwesp (ergm-terms), 28
- gwidegree (ergm-terms), 28
- gwensp (ergm-terms), 28
- gwodegree (ergm-terms), 28
  
- hamming (ergm-terms), 28
- hammingmix (ergm-terms), 28
  
- idegree (ergm-terms), 28
- intransitive (ergm-terms), 28
- is.dyad.independent, 61
- is.inCH, 62
- isolates (ergm-terms), 28
- istar (ergm-terms), 28
  
- kapferer, 63
- kapferer2 (kapferer), 63
- kstar (ergm-terms), 28
  
- lm, 9
- localtriangle (ergm-terms), 28

- logLik, 64
- logLik.ergm, 7, 64, 81
  
- m2star (ergm-terms), 28
- mcmc.diagnostics, 69
- mcmc.diagnostics
  - (mcmc.diagnostics.ergm), 65
- mcmc.diagnostics.ergm, 65, 66
- meandeg (ergm-terms), 28
- molecule, 67
- mutual (ergm-terms), 28
  
- nearsimmelian (ergm-terms), 28
- network, 3, 7, 8, 21, 28, 43, 47, 49–57, 67, 68, 74, 75, 77–79, 83
- network.update, 68, 68
- nodecov (ergm-terms), 28
- nodefactor (ergm-terms), 28
- nodeicov (ergm-terms), 28
- nodeifactor (ergm-terms), 28
- nodematch (ergm-terms), 28
- nodemix (ergm-terms), 28
- nodeocov (ergm-terms), 28
- nodeofactor (ergm-terms), 28
- nsp (ergm-terms), 28
  
- odegree (ergm-terms), 28
- optim, 11, 12
- ostar (ergm-terms), 28
  
- plot.ergm, 69, 69
- plot.gofobject, 61, 71, 72
- plot.network, 50, 52
- print.ergm, 23, 27, 73, 73
- print.gofobject (summary.gofobject), 82
- printCoefmat, 81
  
- receiver (ergm-terms), 28
- residuals, 9
  
- samplike (sampson), 75
- samplk, 74
- samplk1 (samplk), 74
- samplk2 (samplk), 74
- samplk3 (samplk), 74
- sampson, 74, 75
- san, 15, 16, 76
- sender (ergm-terms), 28
- simmelian (ergm-terms), 28
- simmelianties (ergm-terms), 28
  
- simulate, 17, 78
- simulate.ergm, 14–17, 20, 44, 45, 53, 78
- simulate.formula, 14, 16, 17
- simulate.formula (simulate.ergm), 78
- simulate.formula.ergm, 45, 46
- smalldiff (ergm-terms), 28
- sna, 36, 41
- sociality (ergm-terms), 28
- summary, 81, 83
- summary (summary.statistics), 83
- summary.ergm, 23, 27, 67, 80, 81
- summary.gofobject, 82, 82
- summary.statistics, 83
  
- tailor (kapferer), 63
- terms-ergm (ergm-terms), 28
- terms.ergm (ergm-terms), 28
- threepath (ergm-terms), 28
- transitive (ergm-terms), 28
- triad.classify, 36, 41
- triadcensus (ergm-terms), 28
- triangle (ergm-terms), 28
- tripercnt (ergm-terms), 28
- ttriple (ergm-terms), 28
- twopath (ergm-terms), 28
  
- which.matrix.type, 68
- wtd.median, 84