Introduction to Egocentric Network Data Analysis with ERGMs using Statnet

Pavel N. Krivitsky and Martina Morris

Contents

Last updated: April 05, 2016

This tutorial is a joint product of the Statnet Development Team:

Mark S. Handcock (University of California, Los Angeles)
Carter T. Butts (University of California, Irvine)
David R. Hunter (Penn State University)
Steven M. Goodreau (University of Washington)
Skye Bender de-Moll (Oakland)
Pavel N. Krivitsky (University of Wollongong) Martina Morris (University of Washington)

For general questions and comments, please refer to the `statnet` wiki and the `statnet` users group and mailing list
http://statnet.org/statnet_users_group.shtml

1. Installation

Open an R session, and set your working directory to the location where you would like to save this work.

To install all of the CRAN packages in the statnet suite:

```
install.packages('statnet')
library(statnet)
```

To install the `ergm.ego`,

```
install.packages('ergm.ego')
```

2. Overview of ergm.ego

The `ergm.ego` package is designed to provide principled estimation of and statistical inference for Exponential-family Random Graph Models ("ERGMs") from egocentrically sampled network data.

In many empirical contexts, it is not feasible to collect a network census or even an adaptive (link-traced) sample. Even when one of these may be possible in practice, egocentrically sampled data are typically cheaper and easier to collect.

Long regarded as the poor country cousin in the network data family, egocentric data contain a remarkable amount of information. With the right statistical methods, such data can be used to explore the properties of the complete networks in which they are embedded. The basic idea here is to combine what is observed, with assumptions, to define a class of models that represent a distribution of networks that are centered on the observed properties. The variation in these networks quantifies some of the uncertainty introduced by the assumptions.

The package comprises:

- a set of utilities to manage the data,
- a set of "ergm terms" that can be used in models,
- and a set of functions for estimation and inference that rely largely on the existing `ergm` package, but include the specific modifications needed in the egocentric data context.

The package is designed to work with the other statnet packages. So, for example, once you have fit a model, you can use the summary and diagnostic functions from `ergm`, simulate to simulate complete network realizations from the model, the network descriptives from `sna` to explore the properities of the network, and you can use other **R** functions and packages as well after converting the network data structure into a data frame.

Putting this all together, you can start with egocentric data, estimate a model, test the coefficients for statistical significance, assess the model goodness of fit, and simulate complete networks of any size from the model. The statistics in your simulated networks will be consistent with the appropriately scaled statistics from your sample for all of the terms that are represented in the model.

3. Background

The full technical details on ERGM estimation and inference from egocentrically sampled data are in a paper that is currently under review. The working paper can be found here. This tutorial provides a brief introduction to the key concepts.

3a. Exponential-family random graph models

ERGMs represent a general class of models based in exponential-family theory for specifying the probability distribution for a set of random graphs or networks. Within this framework, one can—among other tasks—obtain maximum-likehood estimates for the parameters of a specified model for a given data set; test individual models for goodness-of-fit, perform various types of model comparison; and simulate additional networks with the underlying probability distribution implied by that model.

The general form for an ERGM can be written as:

$$P(Y = y; \theta, x) = \frac{\exp(\theta^\top g(y, x))}{\kappa(\theta, x)} \qquad (1)$$

where Y is the random variable for the state of the network (with realization y), $g(y, x)$ is a vector of model statistics for network y, θ is the vector of coefficients for those statistics, and $\kappa(\theta)$ represents the quantity in the numerator summed over all possible networks (typically constrained to be all networks with the same node set as y).

The model terms $g(y, x)$ are functions of network statistics that we hypothesize may be more or less common than what would be expected in a simple random graph (where all ties have the same probability). When

working with egocentrically sampled network data, these statistics must be observed in the sample more details in section 4.2

A key distinction in model terms is *dyad independence* or *dyad dependence*. Dyad independent terms (like nodal homophily terms) imply no dependence between dyads—the presence or absence of a tie may depend on nodal attributes, but not on the state of other ties. Dyad dependent terms (like degree terms, or triad terms), by contrast, imply dependence between dyads. The design of an egocentric sample means that most observable statistics are dyad independent, but there are a few, like degree, that are dyad dependent.

3b. Network Sampling

Network data are distinguished by having two units of analysis: the actors and the links between the actors. This gives rise to a range of sampling designs that can be classified into two groups: link tracing designs (e.g., snowball and respondent driven sampling) and egocentric designs.

3b1. Link-Trace Designs

Link-trace designs have traditionally been used to sample hard-to-reach populations. Sampling begins with a set of seed nodes. The seeds are asked to nominate alters, the alters are then recruited into the sample, asked to nominate their alters, and so on. Each new set of alters is called a *wave* or a *generation*, and the number of waves of sampling is a study design variable. At each wave, a census or a sample of the alters may be elicited, and/or recruited, this too is a study design variable, and alter recruitment may be investigator-driven, or respondent-driven. This gives rise to a wide range of possible link-trace designs. When the decision to elicit a new wave of alters depends on an attribute of the current node (e.g., is this node an injection drug user) the design is called an adaptive sample.

3b2. Egocentric Designs

Egocentric network sampling comprises a range of designs developed specifically for the collection of network data in social science survey research. The design is (ideally) based on a probability sample of respondents ("egos"") who, via interview, are asked to nominate a list of persons ("alters") with whom they have a specific type of relationship ("tie"), and then asked to provide information on the characteristics of the alters and/or the ties. The alters are not recruited or directly observed. Depending on the study design, alters may or may not be uniquely identifiable, and respondents may or may not be asked to provide information on one or more ties among alters (the"alter" matrices). Alters could, in theory, also be present in the data as an ego or as an alter of a different ego; the likelihood of this depends on the sampling fraction.

Egocentric designs sample egos using standard sampling methods, and the sampling of links is implemented through the survey instrument. As a result, these methods are easily integrated into population-based surveys, and, as we show below, inherit many of the inferential benefits.

For the moment `ergm.ego` uses the minimal egocentric network study design, in which alters cannot be uniquely identified and alter matrices are not collected The minimal design is more common, and the data are more widely available, largely because it is less invasive and less time-consuming than designs which include identifiable alter matrices. However, deveopment of estimation where alter–alter matrices are available is being planned.

3c. Existing methods for sampled network data

Model-based

Handcock and Gile (2010): Likelihood inference for partially observed networks, has egocentric data as a special case.

Kosikinen and Robins (2010): Bayesian inference for partially observed networks, has egocentric data as a special case.

Pros:
- Can fit any ERGM that can be identified.
- Can handle link-tracing designs.

Cons:
- Requires alters to be identifiable.
- Cannot take into account sampling weights (unless all attributes that affect sampling weights are part of the model).
- Might not scale.
- Requires knowledge of the *population* distribution of actor attributes used in the model.

Design-based

Krivitsky and Morris (2015) Use design-based estimators for sufficient statistics of the ERGM of interest and then transfer their properties to the ERGM estimate.

Pros:
- Does not require alters to be identifiable.
- Borrows directly from design-based inference methods. (Can easily incorporate sampling weights, stratification, etc.)
- Can fit any ERGM that can be identified (though see below).
- Can be made invariant to network size for some models.

Cons:
- Requires "reimplementation" of the model statistics as "`EgoStats`": currenly does not support alter–alter statistics or directed or bipartite networks.
- Relies on independent sampling form population of interest in some form.
- Cannot be fit to more complex (e.g., RDS) designs.
- Requires knowledge of the *population* distribution of actor attributes used in the model.

4. Theoretical Framework and Definitions

Some notation (sorry)

Population network

N be the population being studied: a very large, but finite, set of actors whose relations are of interest
x_i attribute (e.g., age, sex, race) vector of actor $i \in N$
x_N **(or just x, when there is no ambiguity)** the attributes of actors in N
$\mathbb{Y}(N)$ the set of **dyads** (potential ties) in an undirected network of actors in N
$y \subseteq \mathbb{Y}(N)$ the **population network**: a fixed but unknown network (a set of relationships) of relationships of interest

In particular,

y_{ij} an indicator function of whether a tie between i and j is present in y
$y_i = \{j \in N : y_{ij} = 1\}$ the set of i's network neighbors.

Egocentric sample

e_i the "egocentric" view of network y from the point of view of actor i ("ego"), with the following parts:
$e_i^e \equiv x_i$**:** i's own attributes
$e_i^a \equiv (x_j)_{j \in y_i}$**:** an unordered list of attribute vectors of i's immediate neighbors ("alters"), but **not** their identities (indices in N)

Also, let the kth attribute/covariate observed on ego i and its alters as $e_{i,k}^e \equiv x_{i,k}$ and $e_{i,k}^a \equiv (x_{j,k})_{j \in y_i}$.

Then,

e_N the **egocentric census**, the information retained by the minimal egocentric sampling design
$S \subseteq N$ the set of egos in the sample
e_S the data contained in an egocentric sample

4a. Specifying Egocentric ERGMs

Egocentric ERGMs are specified the same way as plain `ergm`: via terms (e.g. `nodematch`) used to represent predictors on the right-hand size of equations used in:

- calls to `summary` (to obtain measurements of network statistics on a dataset)
- calls to `ergm.ego` (to estimate an ERGM)
- calls to `simulate` (to simulate networks from an ERGM fit)

The terms that can be used in an ERGM depend on the type of network being analyzed (directed or undirected, one-mode or two-mode ("bipartite"), binary or valued edges) and on the statistics that can be observed in the sample.

Even if the whole population is egocentrically observed (i.e., $S = N$, a census), the alters are still not uniquely identifiable. This limits the kinds of network statistics that can be observed, and the ERGM terms that can be fit to such data. We turn to the notion of sufficiency to identify those that can be.

Egocentric Statistics

We call a network statistic $g_k(\cdot, \cdot)$ **egocentric** if it can be expressed as

$$g_k(y, x) \equiv \textstyle\sum_{i \in N} h_k(e_i)$$

for some function $h_k(\cdot)$ of egocentric information associated with a single actor.

The space of egocentric statistics includes **dyadic-independent** statistics that can be expressed in the general form of

$$g_k(y, x) = \sum_{ij \in y} f_k(x_i, x_j)$$

for some symmetric function $f_k(\cdot, \cdot)$ of two actors' attributes; and some **dyadic-dependent** statistics that can be expressed as

$$g_k(y, x) = \sum_{i \in N} f_k(x_i, (x_j)_{j \in y_i})$$

for some function $f_k(\cdot, \cdots)$ of the attributes of an actor and their network neighbors.

What is "egocentric" depends on available data.

Egocentric with basic design

- Homophily

- Covariate effects
- Degree distribution

Egocentric with alter-alter ties
- Triadic closure (transitive/cyclical ties, triangles)
- 4-cycles (possibly)

Egocentric with star sample (full set of alter's ties)
- Degree assortativity

Not Egocentric for other reasons
- Mean degree ($g_k(y,x) = 2|y|/|N|$): e_i doesn't know how big the network is [1]

TABLE 1

Examples of egocentric statistics for undirected networks. $x_{i,k}$ may be a dummy variable indicating i's membership in a particular exogenously defined group. $h_k(e_i)$ that sum over ties are halved because each tie is observed egocentrically twice: once at each end.

Statistic	$g_k(\boldsymbol{y},\boldsymbol{x})$	$h_k(\boldsymbol{e}_i)$
General sum over ties	$\sum_{(i,j)\in\boldsymbol{y}} f_k(\boldsymbol{x}_i,\boldsymbol{x}_j)$	$\frac{1}{2}\sum_{\boldsymbol{z}\in\boldsymbol{e}_i^{\mathrm{a}}} f_k\left(\boldsymbol{e}_i^{\mathrm{e}},\boldsymbol{z}\right)$
Number of ties in the network	$\lvert\boldsymbol{y}\rvert \equiv \sum_{(i,j)\in\boldsymbol{y}} 1$	$\frac{1}{2}\lvert\boldsymbol{e}_i^{\mathrm{a}}\rvert$
weighted by actor covariate $x_{i,k}$	$\sum_{(i,j)\in\boldsymbol{y}}(x_{i,k}+x_{j,k})$	$\frac{1}{2}\left(e_{i,k}^{\mathrm{e}}\lvert\boldsymbol{e}_i^{\mathrm{a}}\rvert+\sum_{z\in\boldsymbol{e}_{i,k}^{\mathrm{a}}} z\right)$
weighted by difference in $x_{i,k}$	$\sum_{(i,j)\in\boldsymbol{y}}\lvert x_{i,k}-x_{j,k}\rvert$	$\frac{1}{2}\sum_{z\in\boldsymbol{e}_{i,k}^{\mathrm{a}}}\lvert e_{i,k}^{\mathrm{e}}-z\rvert$
within groups identified by $x_{i,k}$	$\sum_{(i,j)\in\boldsymbol{y}} 1_{x_{i,k}=x_{j,k}}$	$\frac{1}{2}\sum_{z\in\boldsymbol{e}_{i,k}^{\mathrm{a}}} 1_{e_{i,k}^{\mathrm{e}}=z}$
General sum over actors	$\sum_{i\in N} f_k\left\{\boldsymbol{x}_i,(\boldsymbol{x}_j)_{j\in\boldsymbol{y}_i}\right\}$	$f_k\left(\boldsymbol{e}_i^{\mathrm{e}},\boldsymbol{e}_i^{\mathrm{a}}\right)$
Number of actors with d neighbors	$\sum_{i\in N} 1_{\lvert\boldsymbol{y}_i\rvert=d}$	$1_{\lvert\boldsymbol{e}_i^{\mathrm{a}}\rvert=d}$
weighted by actor covariate $x_{i,k}$	$\sum_{i\in N} x_{i,k}1_{\lvert\boldsymbol{y}_i\rvert=d}$	$x_{i,k}1_{\lvert\boldsymbol{e}_i^{\mathrm{a}}\rvert=d}$

shows some examples of egocentric statistics, and gives their representations in terms of of $h_k(\cdot)$.

4b. Theoretical basis for inference

The framework for inference relies on two basic properties of exponential family models:

- Under the MLE, the expected value of the model's sufficient statistic ($g(y,x)$) under the model is equal to its observed value.
- The MLE is a smooth function of the sufficient statistic, and is defined for "in between" values of the statistics as well (e.g., fractional edges).

Design-based estimation of ERGMs is done in three steps:

1. Estimate the ERGM's sufficient statistics

$$g(y,x) \approx \tilde{g}(e_S) = \frac{|N|}{|S|}\sum_{i\in S} h_k(e_i).$$

- This is an estimate for a population total, so it's consistent, asymptotically normal.

[1] This does not mean that the mean degree itself cannot be estimated from egocentric data, only that our inferential results might not apply.

- We can estimate its variance as we would a variance of a mean.

2. Fit the ERGM by finding $\hat{\theta}$ that corresponds to $\tilde{g}(e_S)$.
   - It exists because of ERGM's properties.
3. Use Delta Method to transfer properties of $\tilde{g}(e_S)$ to $\hat{\theta}$.
   - We can do this, because MLE is a smooth function of $g(y, x)$.

Together, these allow us to use any statistics that can be observed in an egocentric sample as a term in an ERG model, to estimate the model from a complete pseudo-network that has the same (or appropriately scaled) sufficient statistics, and the networks simulated from the fitted model will be centered on the (scaled) observed statistics.

In practice, these statistics may need to be adjusted for network size and some types of observable discrepancies.

4b1. Practical issues

Network Size

The treatment of network size is perhaps the most obvious way that egocentric estimation differs from a standard ERGM estimation on a completely observed network. With a network census, the network size is known; by contrast, with a network sample, we don't typically know the size of the network from which it is drawn. However, if the statistics we observe in the sample scale in a known way with network size, then we can adjust for this in the estimation, and the resulting parameter estimates (with the exception of the edges term) will be "size invariant".

Here we will follow Krivitsky, Handcock, and Morris (2011), who showed that one can obtain a "per capita" size invariant parameterization for dyad-independent statistics by using an offset, approximately equal to $-\log(n)$, where n is the number of nodes in the network. The intuition is that this transforms the density-based parameterization (ties per dyad) that is the natural scale for ERGMs into a mean degree-based parameterization (ties per node):

$$\text{Mean Degree} = \frac{2 \times \text{ties}}{\text{nodes}} = \frac{2T}{n}$$

$$\text{Density} = \frac{\text{ties}}{\text{dyads}} = \frac{T}{\frac{n(n-1)}{2}} = \frac{\text{Mean Degree}}{(n-1)}$$

Once the number of edges is adjusted to preserve the mean degree, Krivitsky et al. show that all of the dyad independent terms are properly scaled. For degree-based terms, we would want, by analogy, for per capita invariance to preserve the degree probability distribution.
Experimental results suggest that the mean-degree preserving offset has this property, but a mathematical proof is elusive.

Observable discrepancies

What we mean by discrepancy is: undirected tie subtotals that are required to balance in theory, but are observed not to balance in the sample. This can happen when ties are broken down by nodal attributes, and the number of ties that group 1 reports to group 2 are not equal to the number that group 2 reports to group 1.

This is another unique feature of egocentrically sampled network data. With a network census, you would have the complete edgelist, with the appropriate nodal attributes for each member of the dyad, so the reports would always balance. For an egocentrically sampled network, and even for an egocentric census, a discrepancy can arise, either from sampling variability, or from measurement error.

In `ergm.ego`, *we assume that any discrepancy is due to sampling variation*, and effectively take the average of the discrepant reports to estimate the number of ties for that ego-alter tie configuration. If you know the source of the discrepancy, or want to make a different assumption you may want to address this before fitting the data in `ergm.ego`.

Constructing the egocentric target statistics

Now that we have a simple way to construct a consistent set of egocentric statistics and implement a size-invariant parameterization, we can set up the target statistics for the ERGM estimation: instead of using $|N|$ we use $|N'|$.

The resulting values are the *target statistics* that we pass to `ergm`.

If we wanted to obtain population estimates from `ergm.ego` for a population with a known size N, we would fit the model with an offset of $\log(N/N') = \log(N) - \log(N')$. In general, however, we do not know N. So we often fit with an offset of $-\log(N')$, which will return again the per capita estimates that can be easily rescaled to any value for simulation purposes.

4b2. Inference: Estimating the standard errors of the ERGM coefficients

The standard errors for coefficients in an `ergm.ego` fit are designed to represent the uncertainty in our estimate. For ERGMs, this uncertainty can be thought of as coming from three possible sources:

- 1. A superpopulation of networks, from which this one network is a single realization: What other networks *could* have been produced by the social process of interest?
- 2. The sampling process of egos: What other samples *could* have been drawn?
- 3. The stochastic MCMC algorithm used to estimate the coefficient: What other MCMC samples *could* we have gotten?

Most treatments of ERGM estimation treat the coefficient θ as a parameter of a superpopulation process of which y is a single realization. The variance of the MLE of θ is then conceived as coming from (1) and (3) above.

In contrast, in `ergm.ego` we treat the network as a fixed, unknown, finite population, so it is not a source of uncertainty. Rather, uncertainty comes from sampling from this network, and from the MCMC algorithm, (2) and (3) above.

This makes `ergm.ego` inference much more like traditional (frequentist) statistical inference: we imagine repeatedly drawing an egocentric sample, and estimating the ERGM on each replicate. The sampling distribution of the estimate reflects how our estimate will vary from sample to sample.

5. The package `ergm.ego`

Since `ergm.ego` is essentially a wrapper around `ergm`, there are relatively few functions in the `ergm.ego` package itself. The functions that are there deal with the specific requirements associated with data management, estimation and inference for egocentrically sampled data.

To get a list of documented functions, type:

```
library(help='ergm.ego')
```

The main `R` objects unique to `ergm.ego` are:

- `egodata` objects for storing the original data (the analog to `network` objects in `ergm`),
- `ergm.ego` objects, which store the model fit results (the analog to `ergm` objects in `ergm`).

Once you simulate from the fit, the resulting objects are just `network` objects.

The set of functions can be divided into groups as follows:

5a. Data construction and manipulation: `egodata` objects and their related functions

The primary data construction function is `as.egodata`. This function will transform data stored in other formats into an `egodata` object that can be used for analysis. The two format conversions currently supported include:

R data frames This is the method used to read in data from an egocentric sample.

- a data frame containing information on egos, which must include a column for unique ego IDs (defaulting to `egoID`)
- a data frame containing information on alters, which also must have the ego ID column for the ego who nominated them
- optionally, a vector of length equal to the number of egos, containing their sampling weights

`network` objects The function will create an egocentric census from an existing network. Mostly used for testing and pedagogical purposes.

The `as.network` method for `egodata` will transform an egodata object into a network object with a target number of nodes and no edges. The nodes will have the attributes of the egos in the dataset, replicating the egos as needed, and taking into account their sampling weights.

Functions for querying 'egodata' objects include:

- `dim` – the number of egos, and the number of columns in the 'egos' table and the 'alters' table.
- `nrow` – the number of egos
- `dimnames` – the `egoIDs`, and the column names in the 'egos' table and the 'alters' table.

Functions for extracting subsets from 'egodata' objects include:

- `head` — first few rows of each component of `egodata`
- `sample` – a `sample` method for `egodata` object with the target number of egos
- `subset` – allows you to define a condition for the subset of egos to choose, and which columns of the `egos` and `alters` tables to select.
- `na.omit` – returns an `egodata` object without NAs in their (or their alter's) columns

5b. Model terms available for ergm.ego

As with ERGM terms, terms in `ergm.ego` are coded as functions in **R**. The possible terms in an `ergm.ego` model are inherently limited to the egocentric ones: statistics that can be inferred from an egocentric sample. In general, these will include variants on nodal covariate terms, mixing terms, and degree distribution terms. The terms actually available in `ergm.ego` are limited in addition to those that have been coded up. At this point this is a small subset (n=11) of the terms currently available for `ergm`, but they have the same names and arguments as their `ergm` counterparts.

For a list of terms currently included in the ergm.ego, type:

```
help('ergm.ego-terms')
```

Dyad independent terms include density and vertex attribute based measures:

- Density: edges
- Vertex attribute effects: nodefactor (for discrete/nominal vars) and nodecov (for continuous)
- Homophily: nodematch (for homophily), nodemix (for general mixing patterns) and absdiff

Dyad dependent terms include degree-based measures:

- Degree: degree, degrange and degreepopularity
- Concurrency: concurrent and concurrentties

5c. ERG Model-related functions

There are three current functions, each mimics the equivalent function in `ergm`

`summary` takes an `ergm` formula with an 'egodata' object on the LHS and `ergm` terms on the RHS, and returns the observed values of the statistics on the RHS. When used on an `ergm.ego` object, this returns a summary of the model fit.

`ergm.ego` the main function used to fit an ERGM to the egodata object. In addition to the arguments that are specific to fitting from egocentrically sampled data, you can pass the usual arguments to `ergm` for controlling the fitting algorithm.

`control.ergm.ego` includes a set of control parameters specifically needed for fitting ERGMs to egocentrically sampled data, as well a method for passing other control parameters to `ergm`.

`vcov` returns the variance–covariance matrix of the estimate for θ.

5d. Simulation related functions

There are two current functions, again mimicking the equivalent function in 'ergm'

`simulate` Simulates complete networks (of any size) from the fitted model. You can either pass the `ergm.ego` object (which contains the results of the model fit) to simulate, or pass a formula as the object, along with a vector of coefficients.

`control.simulate.ergm.ego` Allows you to control how to resolve fractional numbers of vertices when this is produced by the sample weights and the population size selected. Also provides a way to pass controls to SAN and `ergm` simulate.

6. Example Analysis

We will work with the `faux.mesa.high` dataset that is included with the `ergm` package, using the `as.egodata` function to transform it into an egocentrically sampled network. In essence, this creates an egocentrically sampled census from the network. Knowing the complete network will allow us to compare the fits we get from `ergm.ego` and `ergm`.

Preliminaries:

```
library(ergm.ego)
```

Check package versions

```
sessionInfo()
```

Set seed for simulations – this is not necessary, but it ensures that we all get the same results (if we execute the same commands in the same order).

```
set.seed(1)
```

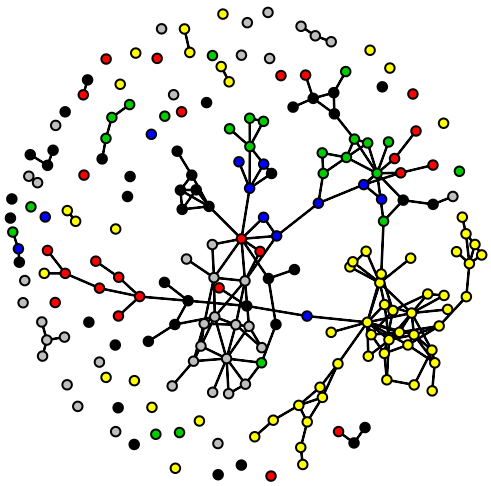
6a. Reading in Data

Start by creating a network object from the faux.mesa.high data:

```
data("faux.mesa.high")
mesa <- faux.mesa.high
```

Let's first take a quick look at the complete network

```
plot(mesa, vertex.col="Grade")
```



Now, let's turn this into an egodata object:

```
mesa.ego <- as.egodata(mesa) # Warning because there were no vertex names in the first place.
```

```
Warning in as.egodata.network(mesa): Non-unique ego IDs; using 1..N.
```

Take a look at this object – there are several ways to do this:

```
str(mesa.ego)
```

```
List of 4
 $ egos    :'data.frame':   205 obs. of  4 variables:
  ..$ vertex.names: int [1:205] 1 2 3 4 5 6 7 8 9 10 ...
  ..$ Grade       : num [1:205] 7 7 11 8 10 10 8 11 9 9 ...
  ..$ Race        : chr [1:205] "Hisp" "Hisp" "NatAm" "Hisp" ...
```

```
  ..$ Sex         : chr [1:205] "F" "F" "M" "M" ...
 $ alters  :'data.frame':   406 obs. of  4 variables:
  ..$ vertex.names: int [1:406] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ Grade       : num [1:406] 7 7 12 7 7 7 7 7 7 7 ...
  ..$ Race        : chr [1:406] "White" "NatAm" "Hisp" "NatAm" ...
  ..$ Sex         : chr [1:406] "F" "F" "F" "F" ...
 $ egoWt   : num [1:205] 1 1 1 1 1 1 1 1 1 1 ...
 $ egoIDcol: chr "vertex.names"
 - attr(*, "class")= chr "egodata"
```

```
summary(mesa.ego)
```

```
         Length Class      Mode
egos       4    data.frame list
alters     4    data.frame list
egoWt    205    -none-     numeric
egoIDcol   1    -none-     character
```

```
head(mesa.ego$egos)
```

```
  vertex.names Grade  Race Sex
1            1     7  Hisp   F
2            2     7  Hisp   F
3            3    11 NatAm   M
4            4     8  Hisp   M
5            5    10 White   F
6            6    10  Hisp   F
```

```
head(mesa.ego$alters)
```

```
  vertex.names Grade  Race Sex
1            1     7 White   F
2            1     7 NatAm   F
3            1    12  Hisp   F
4            1     7 NatAm   F
5            1     7 White   F
6            1     7 NatAm   F
```

Note that the ego table contains the `egoID`, and the nodal attributes Race, Grade and Sex. The alter table also contains the `egoID`, an equivalent set of nodal attributes. The `egoID` column is used by functions that know how to work with `egodata` objects to match alters to the ego that nominated them. The alters do not have unique IDs (because we do not know if they are unique).

So that you get a chance to read in egodata from a .csv file, let's write out the ego and alter tables, and then read them back in to create an egodata object:

```
write(t(mesa.ego$egos), file="mesa.ego.table.csv", ncol=dim(mesa.ego$egos)[2], sep=",")
write(t(mesa.ego$alters), file="mesa.alter.table.csv", ncol=dim(mesa.ego$alters)[2], sep=",")
```

Now read them back in:

```
mesa.egos <- read.csv("mesa.ego.table.csv")
head(mesa.egos)
```

```
  X1 X7  Hisp F
1  2  7  Hisp F
2  3 11 NatAm M
3  4  8  Hisp M
4  5 10 White F
5  6 10  Hisp F
6  7  8 NatAm M
```

```
mesa.alters <- read.csv("mesa.alter.table.csv")
head(mesa.alters)
```

```
  X1 X7 White F
1  1  7 NatAm F
2  1 12  Hisp F
3  1  7 NatAm F
4  1  7 White F
5  1  7 NatAm F
6  1  7 NatAm F
```

And to create an `egodata` object from them, we use the *as.egodata* function:

```
test <- as.egodata(mesa.egos,alters=mesa.alters,egoIDcol="egoID")
head(test$egos)
```

```
  X1 X7  Hisp F
1  2  7  Hisp F
2  3 11 NatAm M
3  4  8  Hisp M
4  5 10 White F
5  6 10  Hisp F
6  7  8 NatAm M
```

```
head(test$alters)
```

```
  X1 X7 White F
1  1  7 NatAm F
2  1 12  Hisp F
3  1  7 NatAm F
4  1  7 White F
5  1  7 NatAm F
6  1  7 NatAm F
```

We will explore some of the other functions available for manipulating the `egodata` object in a later section.

6b. Exploratory data analysis with `ergm.ego`

Prior to model specification, we can explore the data using descriptive statistics observable in the original egocentric sample. In general, the observable statistics are the same as those that `ergm.ego` can estimate.

We can use standard **R** commands to view nodal attribute frequencies:

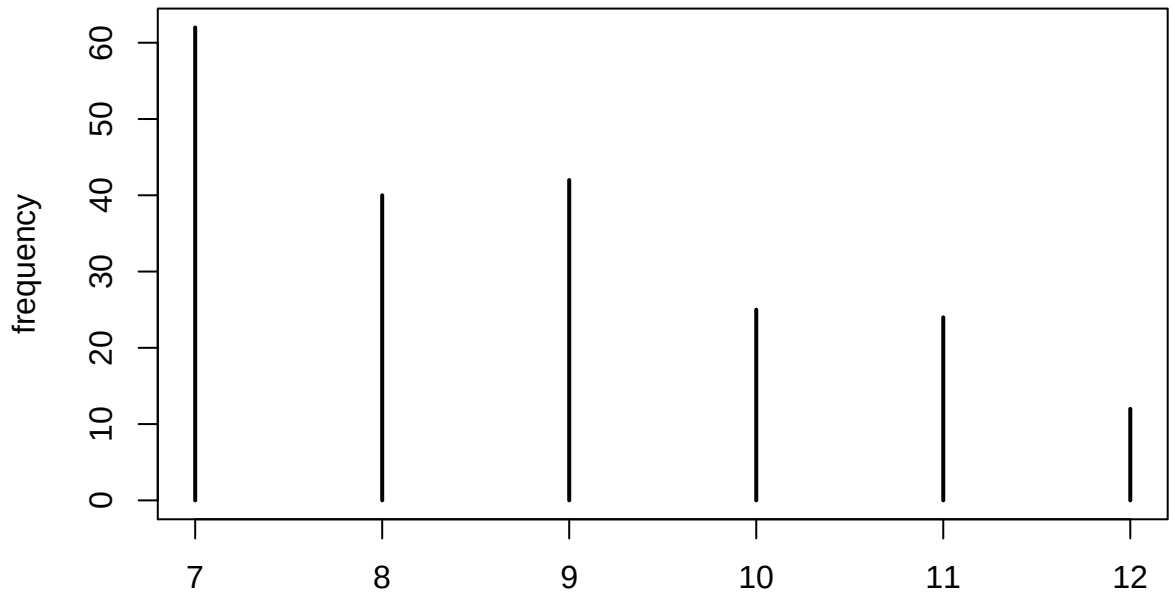
```
table(mesa.ego$egos$Sex, exclude=NULL)
```

```
   F    M <NA>
  99  106    0
```

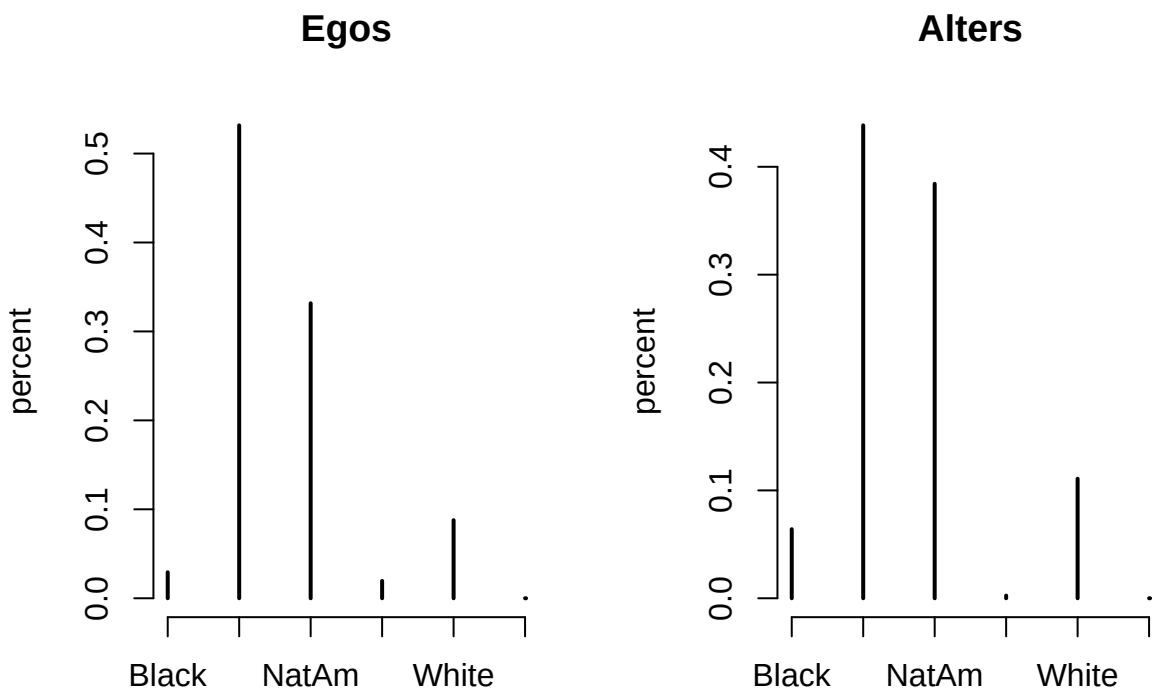
```
table(mesa.ego$egos$Race, exclude=NULL)
```

```
Black  Hisp NatAm Other White  <NA>
    6   109    68     4    18     0
```

```
plot(table(mesa.ego$egos$Grade), ylab="frequency")
```




```
# compare egos and alters...

par(mfrow=c(1,2))
plot(table(mesa.ego$egos$Race, exclude=NULL)/nrow(mesa.ego$egos),
      ylab="percent", main="Egos")
plot(table(mesa.ego$alters$Race, exclude=NULL)/nrow(mesa.ego$alters),
      ylab="percent", main="Alters")
```



To look at the mixing matrix, we need to use a function specific to egodata called `mixingmatrix.egodata`. The function will return either the crosstabulation of the counts of ties by the attribute selected, or the conditional row probabilities.

```
# to get the crosstabulated counts of ties:

mixingmatrix.egodata(mesa.ego,"Grade")
```

```
    alter
ego    7  8  9 10 11 12
  7  150  0  0  1  1  1
  8    0 66  2  4  2  1
  9    0  2 46  7  6  4
  10   1  4  7 18  1  5
  11   1  2  6  1 34  5
  12   1  1  4  5  5 12
```

```
# contrast with the network's mixmatrix:

mixingmatrix(mesa, "Grade")
```

```
Note:  Marginal totals can be misleading
 for undirected mixing matrices.
    7  8  9 10 11 12
7  75  0  0  1  1  1
8   0 33  2  4  2  1
9   0  2 23  7  6  4
10  1  4  7  9  1  5
11  1  2  6  1 17  5
12  1  1  4  5  5  6
```

```
# to get the row conditional probabilities:

mixingmatrix.egodata(mesa.ego, "Grade", rowprob=T)
```

```
    alter
ego      7     8     9    10    11    12
  7  0.980 0.000 0.000 0.007 0.007 0.007
  8  0.000 0.880 0.027 0.053 0.027 0.013
  9  0.000 0.031 0.708 0.108 0.092 0.062
  10 0.028 0.111 0.194 0.500 0.028 0.139
  11 0.020 0.041 0.122 0.020 0.694 0.102
  12 0.036 0.036 0.143 0.179 0.179 0.429
```

```
mixingmatrix.egodata(mesa.ego, "Race", rowprob=T)
```

```
       alter
ego     Black  Hisp NatAm Other White
  Black 0.000 0.308 0.500 0.000 0.192
  Hisp  0.045 0.596 0.230 0.006 0.124
  NatAm 0.083 0.263 0.590 0.000 0.064
  Other 0.000 1.000 0.000 0.000 0.000
  White 0.111 0.489 0.222 0.000 0.178
```

And we can look at the observed number of ties, mean degree, and degree distributions. We construct the edgelist with ego and alter attributes (called "ties"). But note that for the degree distribution we can use the `summary` function with the NHSLS `ergm.ego` object, just as we can in `ergm` with a `network` object.

```
# ties
nrow(mesa.ego$alters)
```

```
[1] 406
```

```
# mean degree
nrow(mesa.ego$egos)/nrow(mesa.ego$alters)
```

```
[1] 0.5049261
```

```
## overall degree distribution
summary(mesa.ego ~ degree(0:20))
```

```
 degree0  degree1  degree2  degree3  degree4  degree5  degree6  degree7
      57       51       30       28       18       10        2        4
 degree8  degree9 degree10 degree11 degree12 degree13 degree14 degree15
       1        2        1        0        0        1        0        0
degree16 degree17 degree18 degree19 degree20
       0        0        0        0        0
```

```
## by sex, and race
summary(mesa.ego ~ degree(0:13, by="Sex"))
```

```
 deg0.SexF  deg1.SexF  deg2.SexF  deg3.SexF  deg4.SexF  deg5.SexF
        23         23         10         17         12          7
 deg6.SexF  deg7.SexF  deg8.SexF  deg9.SexF deg10.SexF deg11.SexF
         1          3          1          0          1          0
deg12.SexF deg13.SexF  deg0.SexM  deg1.SexM  deg2.SexM  deg3.SexM
         0          1         34         28         20         11
 deg4.SexM  deg5.SexM  deg6.SexM  deg7.SexM  deg8.SexM  deg9.SexM
         6          3          1          1          0          2
deg10.SexM deg11.SexM deg12.SexM deg13.SexM
         0          0          0          0
```

```
summary(mesa.ego ~ degree(0:13, by="Race"))
```

```
 deg0.RaceBlack  deg1.RaceBlack  deg2.RaceBlack  deg3.RaceBlack
              1               0               0               2
 deg4.RaceBlack  deg5.RaceBlack  deg6.RaceBlack  deg7.RaceBlack
              0               1               0               1
 deg8.RaceBlack  deg9.RaceBlack deg10.RaceBlack deg11.RaceBlack
              1               0               0               0
deg12.RaceBlack deg13.RaceBlack   deg0.RaceHisp   deg1.RaceHisp
              0               0              33              30
  deg2.RaceHisp   deg3.RaceHisp   deg4.RaceHisp   deg5.RaceHisp
             18              14               8               5
  deg6.RaceHisp   deg7.RaceHisp   deg8.RaceHisp   deg9.RaceHisp
              0               0               0               0
 deg10.RaceHisp  deg11.RaceHisp  deg12.RaceHisp  deg13.RaceHisp
              0               0               0               1
 deg0.RaceNatAm  deg1.RaceNatAm  deg2.RaceNatAm  deg3.RaceNatAm
             18              13               9              10
 deg4.RaceNatAm  deg5.RaceNatAm  deg6.RaceNatAm  deg7.RaceNatAm
              9               3               2               2
 deg8.RaceNatAm  deg9.RaceNatAm deg10.RaceNatAm deg11.RaceNatAm
              0               2               0               0
deg12.RaceNatAm deg13.RaceNatAm  deg0.RaceOther  deg1.RaceOther
              0               0               3               1
 deg2.RaceOther  deg3.RaceOther  deg4.RaceOther  deg5.RaceOther
              0               0               0               0
 deg6.RaceOther  deg7.RaceOther  deg8.RaceOther  deg9.RaceOther
              0               0               0               0
deg10.RaceOther deg11.RaceOther deg12.RaceOther deg13.RaceOther
              0               0               0               0
 deg0.RaceWhite  deg1.RaceWhite  deg2.RaceWhite  deg3.RaceWhite
              2               7               3               2
 deg4.RaceWhite  deg5.RaceWhite  deg6.RaceWhite  deg7.RaceWhite
              1               1               0               1
 deg8.RaceWhite  deg9.RaceWhite deg10.RaceWhite deg11.RaceWhite
              0               0               1               0
deg12.RaceWhite deg13.RaceWhite
              0               0
```

The summary function has an egodata specific argument, `scaleto`, that allows you to scale the summary statistics to a network of arbitrary size. So, for example, we can obtain the degree distribution scaled to a network of size 100,000, or a network that is 100 times larger than the sample.

```
summary(mesa.ego ~ degree(0:10), scaleto=100000)
```

```
   degree0    degree1    degree2    degree3    degree4    degree5
27804.8780 24878.0488 14634.1463 13658.5366  8780.4878  4878.0488
   degree6    degree7    degree8    degree9   degree10
  975.6098  1951.2195   487.8049   975.6098   487.8049
```

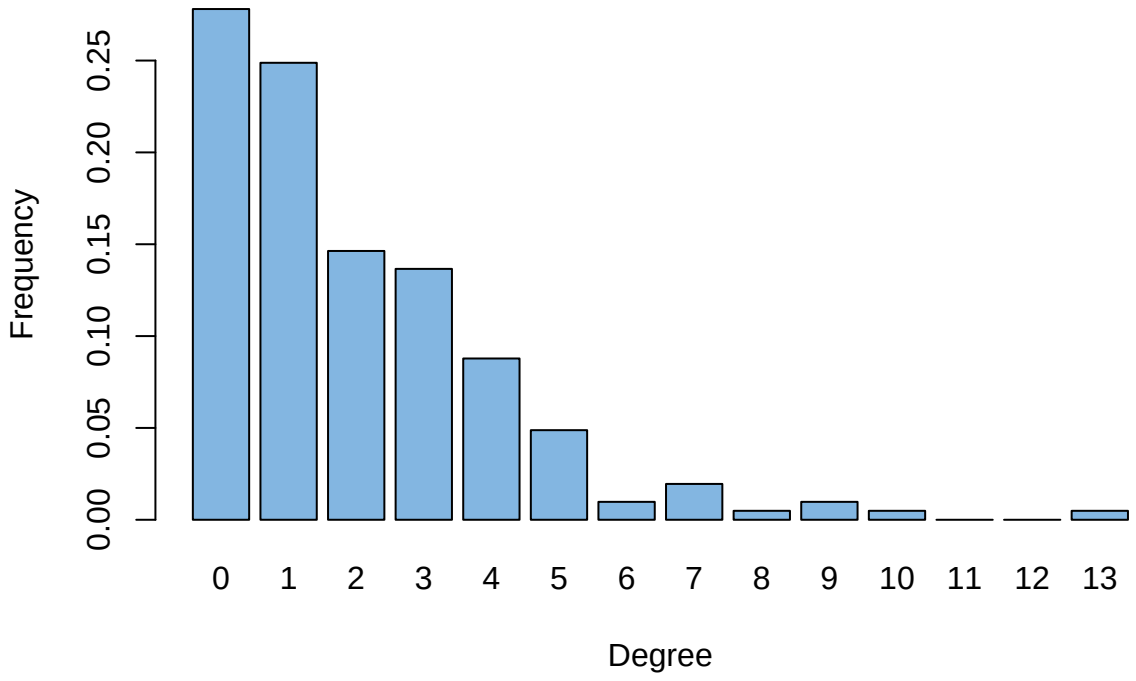
```
summary(mesa.ego ~ degree(0:10), scaleto=nrow(mesa.ego$egos)*100)
```

```
 degree0  degree1  degree2  degree3  degree4  degree5  degree6  degree7
    5700     5100     3000     2800     1800     1000      200      400
 degree8  degree9 degree10
     100      200      100
```

Note that the first scaling results in fractional numbers of nodes at each degree, because the proportion at each degree level does not scale to an integer for this population size. Again, this is not a problem for estimation, but one should be careful with descriptive statistics that expect integer values. The second scaling does result in integer counts because it is a multiple of the sample size.

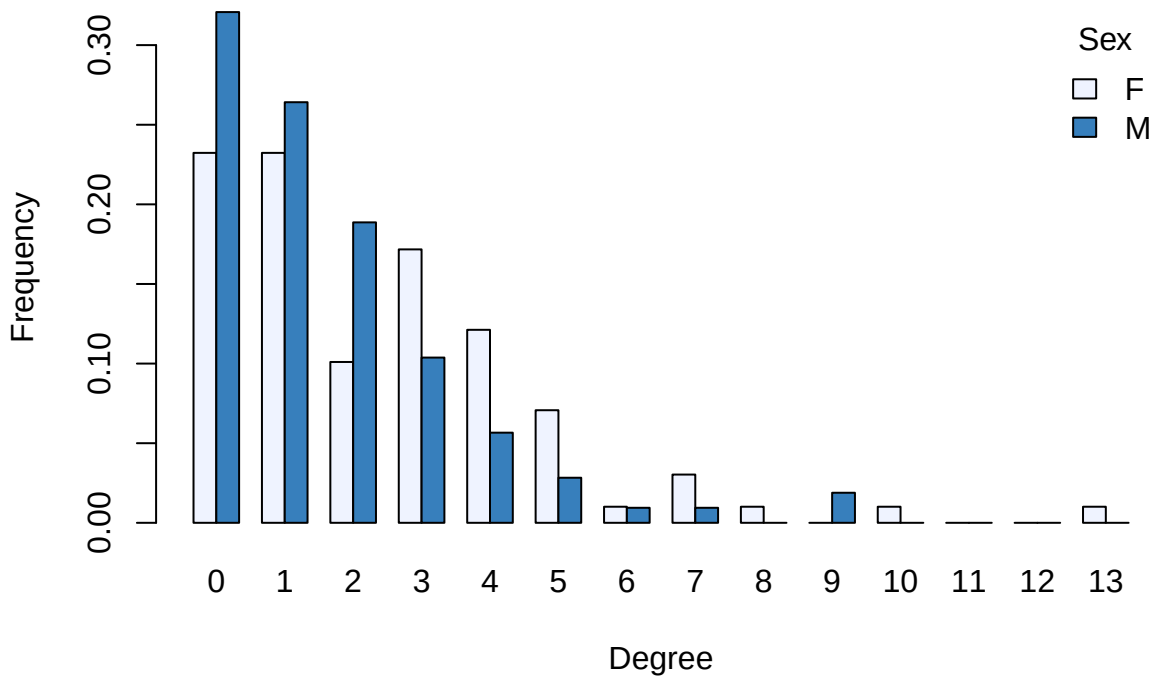
Finally, we can get plot the degree distribution instead, which may provide a summary that is easier to interpret, using another `egodata` specific function: `degreedist.egodata`. As with the `mixingmatrix` function, this can return either the counts or the proportions at each degree.

```
# to get the frequency counts

degreedist.egodata(mesa.ego)
```




```
degreedist.egodata(mesa.ego, by="Sex")
```

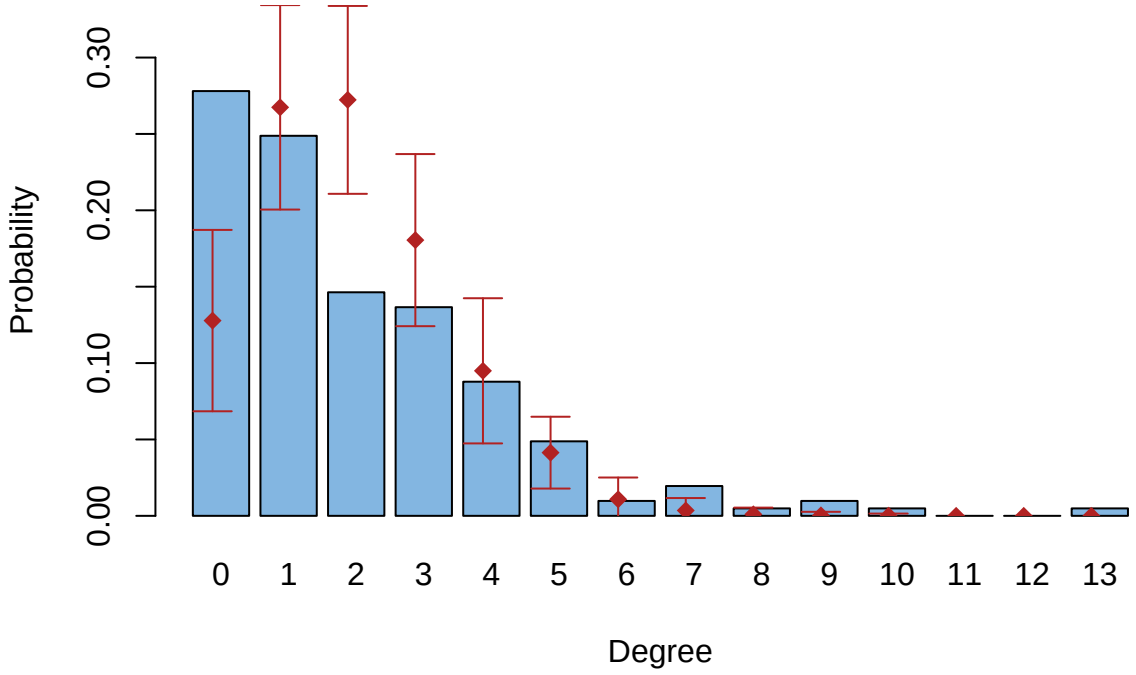
```
# to get the proportion at each degree level

degreedist.egodata(mesa.ego, by="Sex", prob=T)
```



The `degreedist` function also has an argument that lets you overplot the expected degree distribution for a Bernoulli random graph with the same expected density.

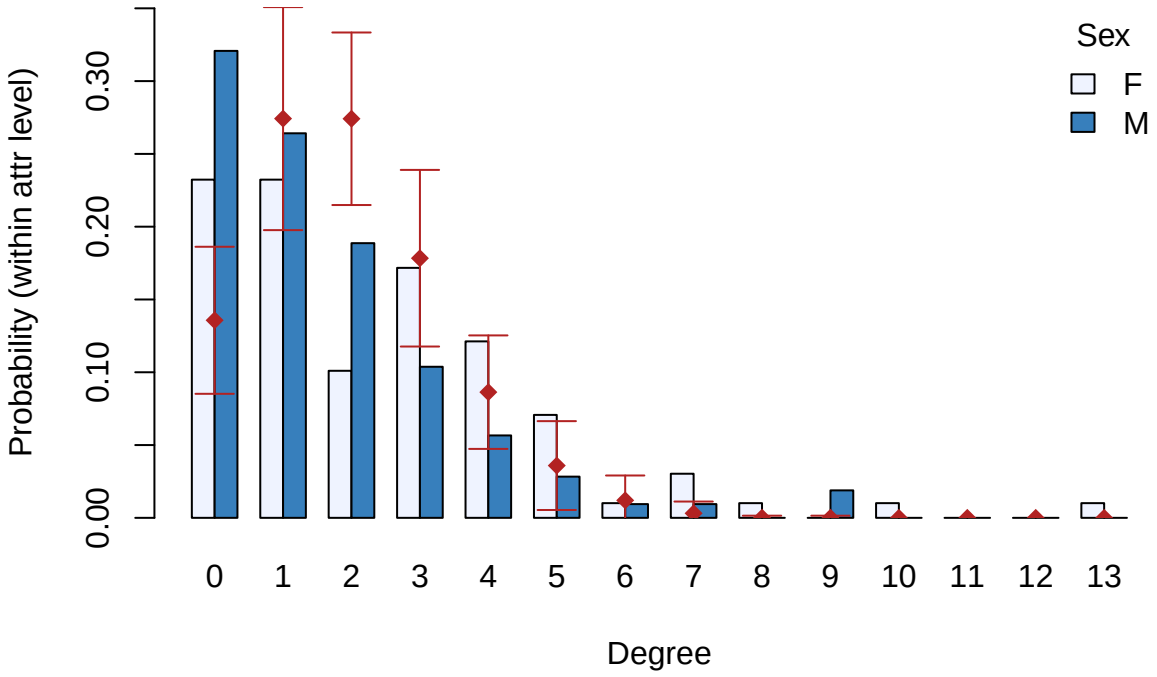
```
degreedist.egodata(mesa.ego, brg=T)
```

```
Constructing pseudopopulation network.

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.0004653
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.0005796
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag
```




```
degreedist.egodata(mesa.ego, by="Sex", prob=T, brg=T)
```

```
Constructing pseudopopulation network.
Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.000209
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by < 0.0001
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag
```



The `brg` overplot is based on 50 simulations of a Bernoulli random graph with the same number of nodes and expected density, implemented by using an ergm.ego simulation from an edges only model with $\theta =$ logit(probability of a tie) from the observed data. The overplot shows the mean and 2 standard deviations obtained for each degree value from the 50 simulations. Note that the `brg` automatically scales to the proportions when `prob=T`.

6c. Model Specification

The key characteristics we would like to capture in a model for this network are:

- Variation in mean degree by nodal attributes (race, sex and grade)
- Patters of mixing by race, sex and grade
- The degree distribution, (in particular the disproportionate isolate fraction)

We can use `ergm.ego` to fit a sequence of nested models to both estimate the parameters associated with these statistics, and test their significance. We can diagnose the both the estimation process (to verify convergence and good mixing in the MCMC sampler) and the fit of the model to the data. In both cases, we will use functionality that will be familiar to `ergm` users: MCMC diagnostics, and GOF.

One thing that is different from a standard `ergm` call is that we need to specify the scaling, both for the *pseudo-population (N′)* that will be used to set the target statistics during estimation, and for the *population (N)* size that the final rescaled coefficients will represent. Recall,

Population size ($|N|$) If we wanted to rescale the final estimates to reproduce the expected values in the true population network, we would need to know N, the size of the popuation. In general, we do not know this, so the most useful scaling is a per capita scaling, which can easily be transformed into any value of $|N|$ later (for simulation, or other purposes). For per capita scaling, $|N| = 1$. In `ergm.ego`, it is controlled by the `popsize` top-level argument.

Pseudo-population ($|N'|$) Recall from above that the target statistics can be scaled to an arbitrary population size for estimation. What size should we use? Several principles guide this choice:

- *Bias* – In general, estimation bias is reduced the closer N' is to N (usually larger).
- *Computing time* – The larger the pseudo-population, the longer the estimation takes.
- *Sample weights* – In general, it is good practice for the smallest sample weight to produce at least 1 observations in the pseudo-population network, though more is better.

This leads to different guidelines for data with and without weights.

Simulation studies in Krivitsky & Morris (2015) suggest that a good rule of thumb is to have a minimum pseudo-population size of 1,000 for unweighted data. For weighted data the pseudo-populations size should be at least 1 * sampleSize/smallestWeight (or 3 * sampleSize/smallestWeight to be safe), or 1000 (whichever is larger).

In `ergm.ego`, $|N'|$ is controlled by a combination of four factors:

- the sample size $|S|$ (i.e., number of egos),
- the top-level argument `popsize` ($|N|$ or 1) (default: 1),
- the `control.ergm.ego` control parameter `ppopsize` (default: `"auto"`),
- the `control.ergm.ego` control parameter `ppopsize.mul` (default: 1).

If `ppopsize` is left at its defualt (`"auto"`),

- If `popsize` is left at 1, $|S|\times$`ppopsize.mul`.
- If `popsize` is specified, use $|N|\times$`ppopsize.mul`.

You can also force one of these two regimes by setting `ppopsize` to `"samp"` or `"pop"`, respectively, or set it to a number to force a particular $|N'|$ ignoring `ppopsize.mul`.

For more information, see

```
?control.ergm.ego
```

We will give an example below.

In both cases, the scaling will ultimately only affect the estimate of the *edges* term, and we will demonstrate this below.

Let's start with simple edges-only model to see what's the same and what is different from a call to `ergm`:

```
fit.edges <- ergm.ego(mesa.ego ~ edges)
```

```
Constructing pseudopopulation network.

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.0002044
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.0007946
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

```
summary(fit.edges)
```

```
==========================
Summary of model fit
==========================

Formula:   mesa.ego ~ edges

Iterations:  2 out of 20

Monte Carlo MLE Results:
            Estimate Std. Error MCMC % p-value
netsize.adj -5.32301    0.00000      0  <1e-04 ***
edges        0.69941    0.07717      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


 The following terms are fixed by offset and are not estimated:
  netsize.adj
```

This is a simple model, for a homogenous tie probability – a Bernoulli random graph with the mean degree observed in our sampled data. Let's look under the hood at the components that are output to the `fit.edges` object:

```
names(fit.edges)
```

```
 [1] "coef"             "sample"           "sample.obs"
 [4] "iterations"       "MCMCtheta"        "loglikelihood"
 [7] "gradient"         "hessian"          "covar"
[10] "failure"          "network"          "newnetworks"
[13] "newnetwork"       "coef.init"        "est.cov"
[16] "coef.hist"        "stats.hist"       "steplen.hist"
[19] "control"          "etamap"           "formula"
[22] "target.stats"     "target.esteq"     "constrained"
[25] "constraints"      "reference"        "estimate"
[28] "offset"           "drop"             "estimable"
[31] "null.lik"         "v"                "m"
[34] "ergm.formula"     "ergm.offset.coef" "egodata"
[37] "ppopsize"         "popsize"          "ergm.covar"
[40] "DtDe"
```

```
fit.edges$ppopsize
```

```
[1] 205
```

```
fit.edges$popsize
```

```
[1] 1
```

Many of the elements of the object are the same as you would get from an `ergm` fit, but the last few elements are unique to `ergm.ego`. Here you can see the `ppopsize` – the pseudo-population size used to construct the target statistics, and `popsize` – the final scaled population size after network size adjustment is applied. The values that were used in the fit were the default values, since we did not specify otherwise. So, `ppopsize`= 205 (the sample size, or number of egos), and `popsize`= 1, so the scaling returns the per capita estimates from the model parameters.

The summary shows the `netsize.adj`$= -5.32301, which is $-\log(205)$.

The summary function also reports that:

```
 The following terms are fixed by offset and are not estimated:
  netsize.adj
```

So what would happen if we fit the model instead with target statistics from a pseudo-population of size 1000? To do this, we explicitly change the value of the `ppopsize` parameter through the control argument:

Now the netsize.adj value is $-6.9077553 = -\log(1000)$.

Note that the value of the estimated edges coefficient is the same in both models, 0.698. This is the behavior we expect – the model is returning the same per capita value in both cases; it is just using a different scaling for the target statistics used in the fit. For this simple model, there may not be much difference in the properties of the estimates for these two different pseudo-population sizes.

We will examine the impact of modifying the `popsize` parameter in a later section below.

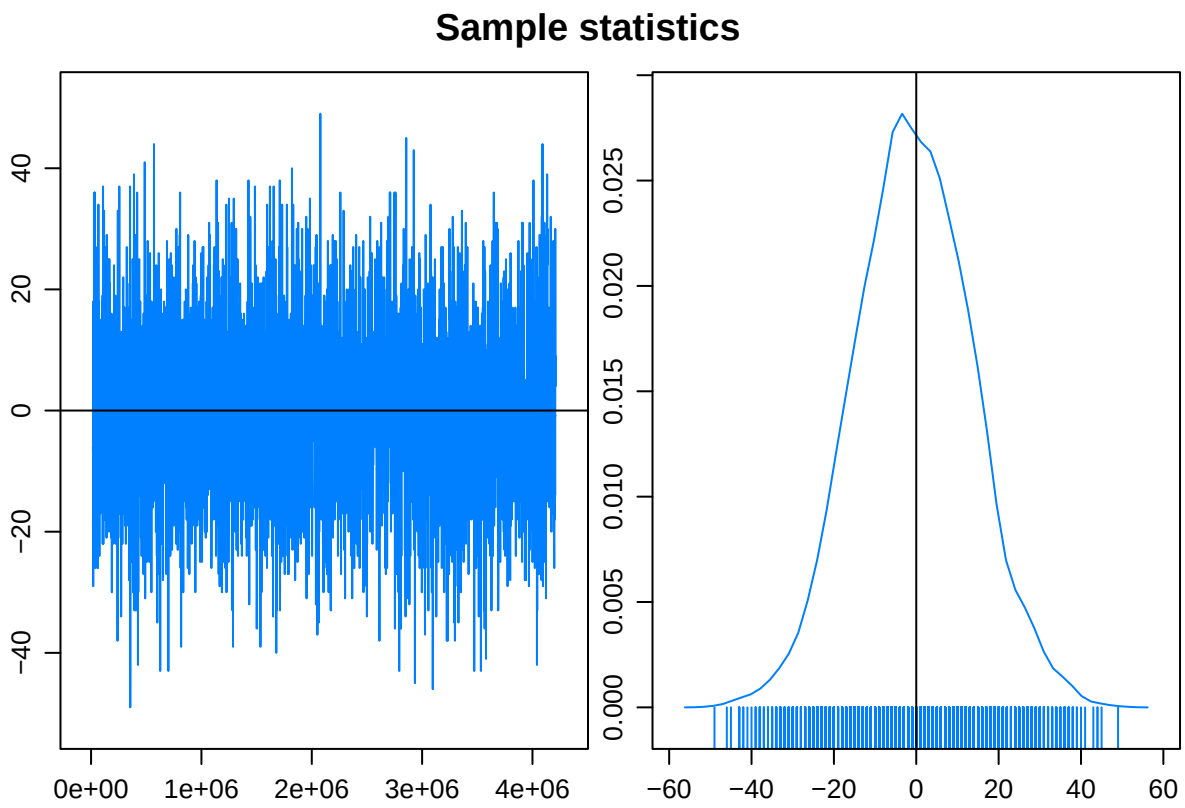
As the output shows, the model fit was fit using MCMC. This, too is different from the edges-only model using `ergm`. For `ergm`, models with only dyad-dependent terms are fit using Newton-Raphson algorthims (the same algorithm used for logistic regression), not MCMC. For `ergm.ego`, estimation is alway based on MCMC, regardless of the terms in the model.

Now let's see what the MCMC diagnostics for this model look like

```
mcmc.diagnostics(fit.edges)
```

```
Sample statistics summary:

Iterations = 16384:4209664
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 4096

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
       -0.5596        14.0366         0.2193         0.2462

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
  -27   -10    -1     9    28


Sample statistics cross-correlations:
      edges
```

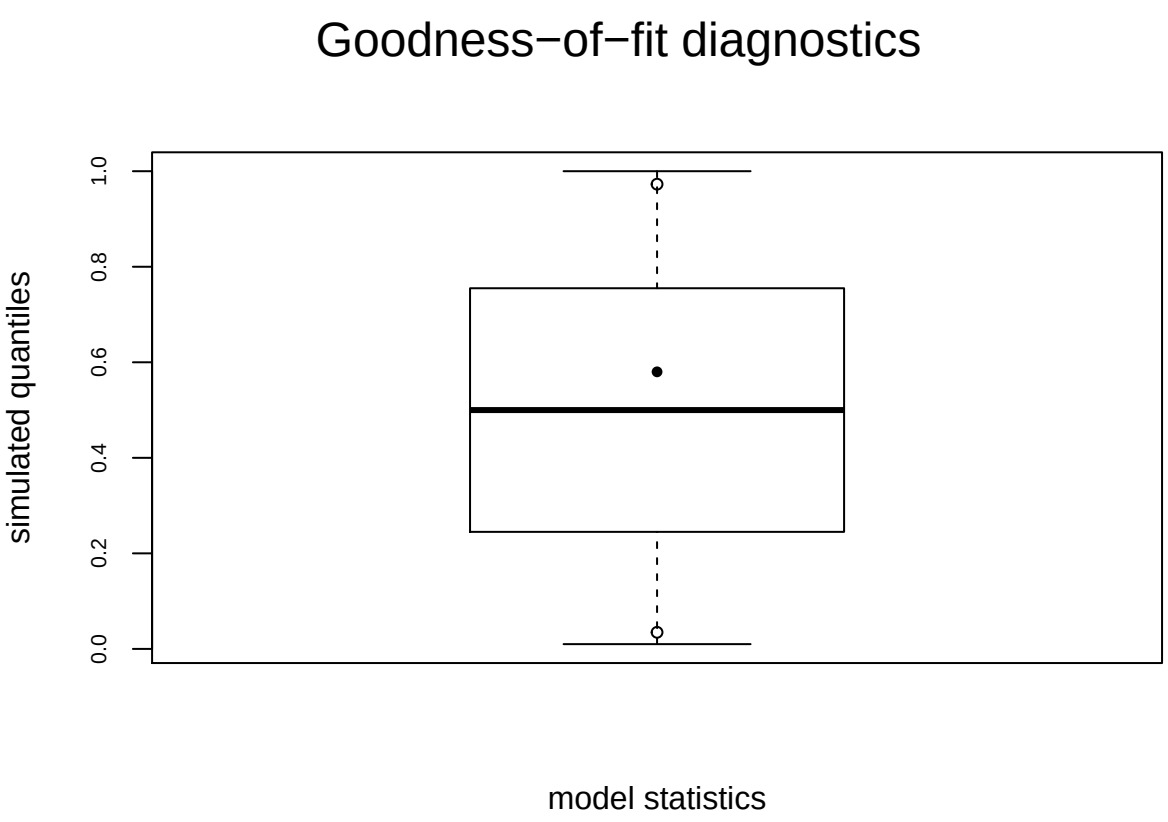
```
edges     1

Sample statistics auto-correlation:
Chain 1
               edges
Lag 0     1.000000000
Lag 1024  0.115057551
Lag 2048 -0.001844062
Lag 3072  0.002638155
Lag 4096 -0.029599062
Lag 5120 -0.004611425

Sample statistics burn-in diagnostic (Geweke):
Chain 1

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  edges
-0.1746

Individual P-values (lower = worse):
    edges
0.8613922
Joint P-value (lower = worse):  0.8520016 .
```




```
MCMC diagnostics shown here are from the last round of simulation, prior to computation of final parame
```

Again, this is a simple model, so the diagnostics suggest good mixing, and even though the distribution of the sample statistic deviations from the targets (on the right panel) can not technically be used to verify that the simulations from the model match the target stats. But the values here suggest they do. We can check that with a call to GOF.

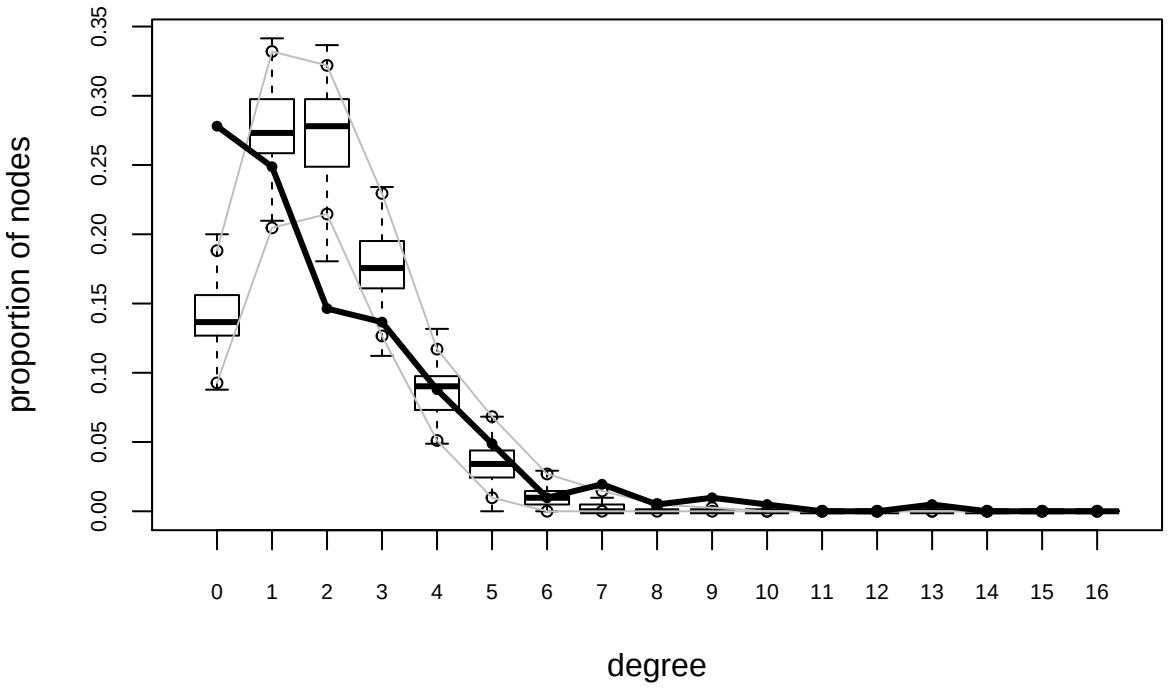
```
plot(gof(fit.edges, GOF="model"))
```



So, it looks good there too.

Finally, we should evaluate the model fit. We can also use GOF to do this, by comparing observed statistics that are not in the model, like the full degree distribution, with simulations from the fitted model. This is the same procedure that we use for `ergm`, but now with a more limited set of observed higher-order statistics to use for assessment.

```
plot(gof(fit.edges, GOF="degree"))
```



And here, finally, we see some bad behavior, but this too is expected from such a simple model. The GOF plot shows there are almost twice as many isolates in the observed data than would be predicted from a simple edges-only model.
Of course we knew this from having looked at the degree distribution plots with the Bernoulli random graph overlay.

Ok, so that's a full cycle of description, estimation, and model assessment.

From here, let's try fitting a `degree(0)` term to see how that changes the degree distribution assessment.

```
set.seed(1)
```

```
fit.deg0 <- ergm.ego(mesa.ego ~ edges + degree(0), control=control.ergm.ego(ppopsize=1000))
```

```
Constructing pseudopopulation network.

Note: Constructed network has size 1025, different from requested 1000. Estimation should not be meanin

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 19.43
Iteration 2 of at most 20:
The log-likelihood improved by 4.87
Iteration 3 of at most 20:
The log-likelihood improved by 3.848
```

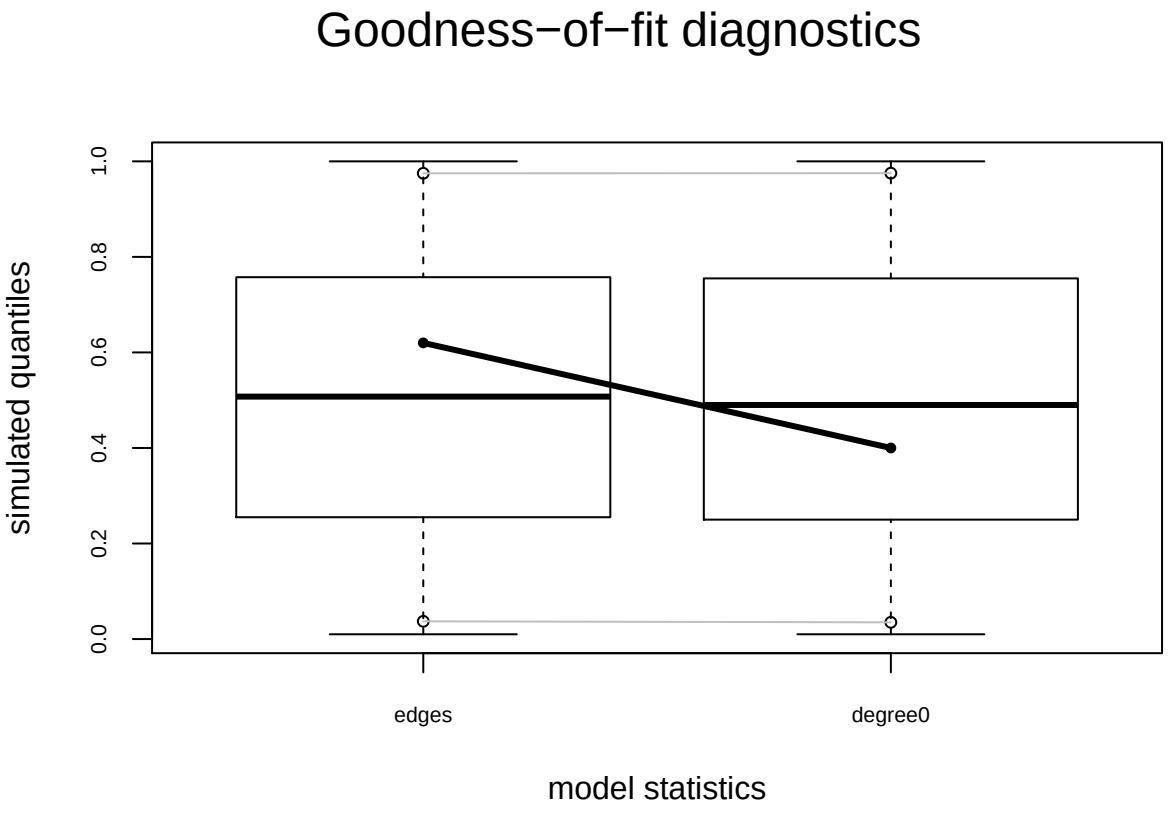
```
Iteration 4 of at most 20:
The log-likelihood improved by 3.182
Iteration 5 of at most 20:
The log-likelihood improved by 2.621
Iteration 6 of at most 20:
The log-likelihood improved by 2.711
Iteration 7 of at most 20:
The log-likelihood improved by 2.645
Step length converged once. Increasing MCMC sample size.
Iteration 8 of at most 20:
The log-likelihood improved by 0.1677
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

```
summary(fit.deg0)
```

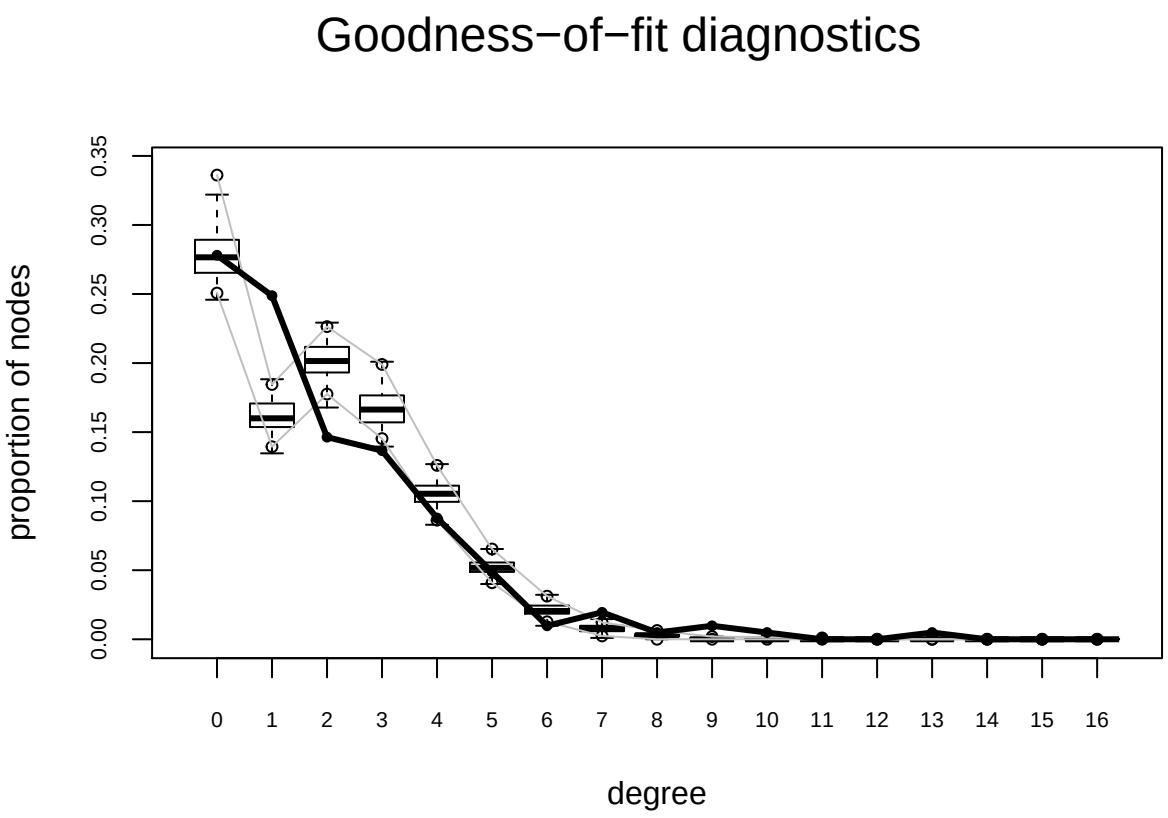
```
==========================
Summary of model fit
==========================

Formula:   mesa.ego ~ edges + degree(0)

Iterations:  8 out of 20

Monte Carlo MLE Results:
            Estimate Std. Error MCMC % p-value
netsize.adj  -6.9324     0.0000      0  <1e-04 ***
edges         1.1738     0.1127      0  <1e-04 ***
degree0       1.4952     0.2666      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


 The following terms are fixed by offset and are not estimated:
  netsize.adj
```

```
mcmc.diagnostics(fit.deg0)
```

```
plot(gof(fit.deg0, GOF="model"))
```




```
plot(gof(fit.deg0, GOF="degree"))
```



So, we've now fit the isolates exactly, and the fit is better, but this now suggests there are more nodes with just one tie than would be expected, given the mean degree, and the number of isolates.

And just to round things off, let's fit a relatively large model

```
fit.full <- ergm.ego(mesa.ego ~ edges + degree(0:1)
                     + nodefactor("Sex")
                     + nodefactor("Race", base=2)
                     + nodefactor("Grade")
                     + nodematch("Sex") + nodematch("Race") + nodematch("Grade"))
```

```
Constructing pseudopopulation network.

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 3.326
Iteration 2 of at most 20:
The log-likelihood improved by 1.923
Iteration 3 of at most 20:
The log-likelihood improved by 0.9883
Step length converged once. Increasing MCMC sample size.
Iteration 4 of at most 20:
The log-likelihood improved by 0.5091
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

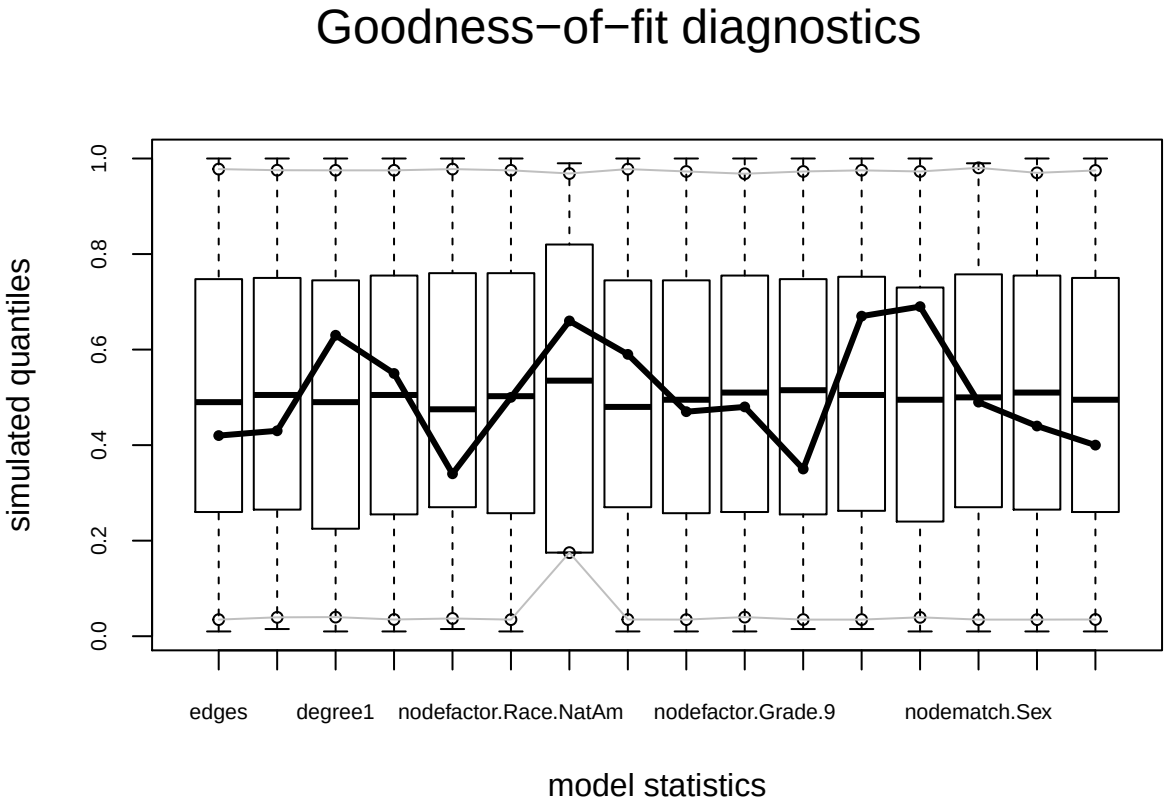
```
summary(fit.full)
```

```
==========================
Summary of model fit
==========================

Formula:   mesa.ego ~ edges + degree(0:1) + nodefactor("Sex") + nodefactor("Race",
    base = 2) + nodefactor("Grade") + nodematch("Sex") + nodematch("Race") +
    nodematch("Grade")

Iterations:  4 out of 20

Monte Carlo MLE Results:
                     Estimate Std. Error MCMC %  p-value
netsize.adj          -5.32301    0.00000      0  < 1e-04 ***
edges                -1.38134    0.18848      0  < 1e-04 ***
degree0               2.15596    0.34569      0  < 1e-04 ***
degree1               1.04281    0.27445      0 0.000145 ***
nodefactor.Sex.M     -0.16966    0.05881      0 0.003920 **
nodefactor.Race.Black  1.21731    0.20096      0  < 1e-04 ***
nodefactor.Race.NatAm  0.29951    0.05700      0  < 1e-04 ***
nodefactor.Race.Other -0.86136    0.61006      0 0.157986
nodefactor.Race.White  0.58534    0.11948      0  < 1e-04 ***
```

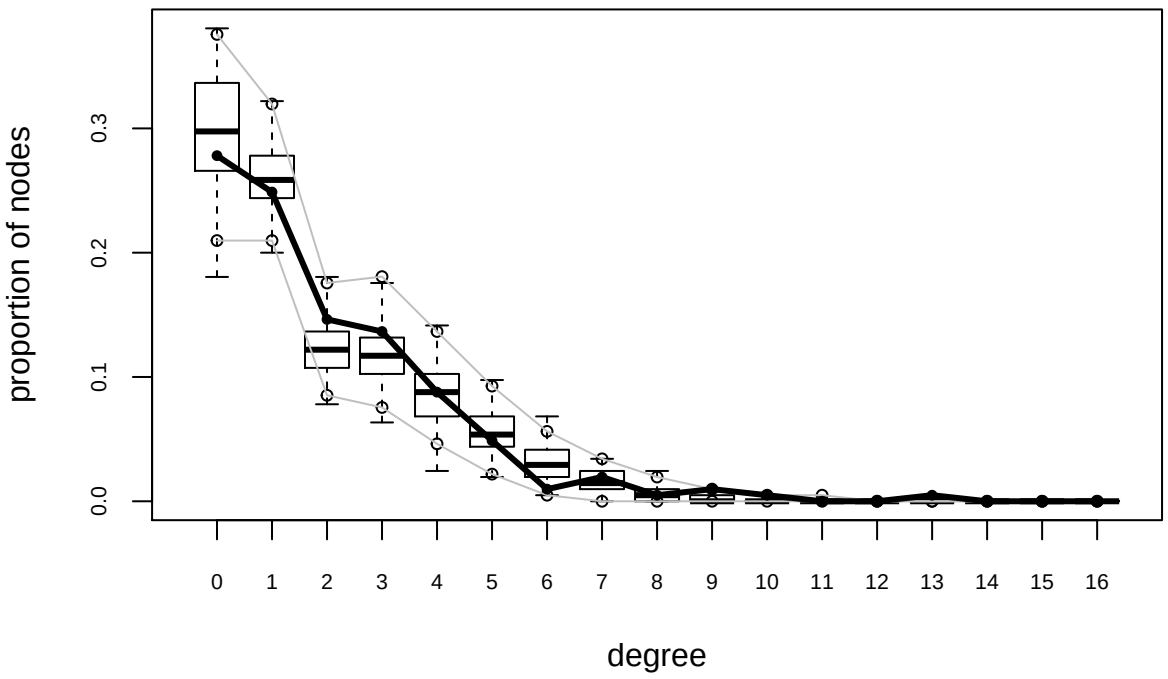
```
nodefactor.Grade.8     0.14330    0.04999       0 0.004156 **
nodefactor.Grade.9     0.14108    0.05261       0 0.007332 **
nodefactor.Grade.10    0.32322    0.08241       0  < 1e-04 ***
nodefactor.Grade.11    0.40861    0.05678       0  < 1e-04 ***
nodefactor.Grade.12    0.78660    0.07036       1  < 1e-04 ***
nodematch.Sex          0.63525    0.11576       0  < 1e-04 ***
nodematch.Race         0.83624    0.12919       0  < 1e-04 ***
nodematch.Grade        3.06967    0.13975       0  < 1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


 The following terms are fixed by offset and are not estimated:
  netsize.adj
```

```
mcmc.diagnostics(fit.full)
```

```
plot(gof(fit.full, GOF="model"))
```




```
plot(gof(fit.full, GOF="degree"))
```



In general the model diagnostics look good. *If* this were a genuine sample of 205 students from a larger school, we could infer the following:

- there are still many more isolates, and more degree 1 nodes than expected by chance
- there are significant differences in mean degree by race, with the dominant group (Hispanics, the reference category) nominating fewer friends that the other groups
- there may a difference in mean degree for 11th and 12th graders when compared to all the lower grades.
- there are strong and significant homophily effects, for all three attributes

We will leave this model here and go on to explore how the idea of sampling uncertainty is being used to produce the standard errors for our coefficients.

Note that we have implicitly used simulate here – it's the basis of the GOF results – but it is also possible to simulate complete networks from this fit:

```
sim.full <- simulate(fit.full)
summary(mesa.ego ~ edges + degree(0:1)
                       + nodefactor("Sex")
                       + nodefactor("Race", base=2)
                       + nodefactor("Grade")
                       + nodematch("Sex") + nodematch("Race") + nodematch("Grade"))
```

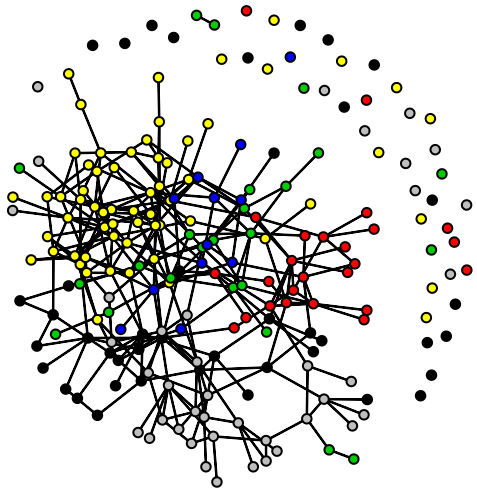
```
                edges               degree0               degree1
                  203                    57                    51
     nodefactor.Sex.M nodefactor.Race.Black nodefactor.Race.NatAm
                  171                    26                   156
nodefactor.Race.Other nodefactor.Race.White    nodefactor.Grade.8
```

```
                     1                     45                     75
   nodefactor.Grade.9   nodefactor.Grade.10   nodefactor.Grade.11
                    65                     36                     49
  nodefactor.Grade.12         nodematch.Sex         nodematch.Race
                    28                    132                    103
       nodematch.Grade
                   163
```

```
summary(sim.full ~ edges + degree(0:1)
                   + nodefactor("Sex")
                   + nodefactor("Race", base=2)
                   + nodefactor("Grade")
                   + nodematch("Sex") + nodematch("Race") + nodematch("Grade"))
```

```
                 edges               degree0               degree1
                   277                    43                    44
     nodefactor.Sex.M nodefactor.Race.Black nodefactor.Race.NatAm
                   235                    31                   248
nodefactor.Race.Other nodefactor.Race.White    nodefactor.Grade.8
                     1                    57                    92
   nodefactor.Grade.9   nodefactor.Grade.10   nodefactor.Grade.11
                    82                    62                    59
  nodefactor.Grade.12         nodematch.Sex         nodematch.Race
                    39                   174                   155
       nodematch.Grade
                   217
```

```
plot(sim.full, vertex.col="Grade")
```



We can use network size invariance to simulate networks of a different size, albeit one has to be careful:

```
sim.full2 <- simulate(fit.full, popsize=network.size(mesa)*2)
summary(mesa~edges + degree(0:1)
                   + nodefactor("Sex")
                   + nodefactor("Race", base=2)
                   + nodefactor("Grade")
                   + nodematch("Sex") + nodematch("Race") + nodematch("Grade"))*2
```

```
                edges               degree0               degree1
                  406                   114                   102
     nodefactor.Sex.M nodefactor.Race.Black nodefactor.Race.NatAm
                  342                    52                   312
nodefactor.Race.Other nodefactor.Race.White    nodefactor.Grade.8
                    2                    90                   150
   nodefactor.Grade.9   nodefactor.Grade.10   nodefactor.Grade.11
                  130                    72                    98
  nodefactor.Grade.12         nodematch.Sex         nodematch.Race
                   56                   264                   206
      nodematch.Grade
                  326
```

```
summary(sim.full2~edges + degree(0:1)
                    + nodefactor("Sex")
                    + nodefactor("Race", base=2)
                    + nodefactor("Grade")
                    + nodematch("Sex") + nodematch("Race") + nodematch("Grade"))
```

```
                edges               degree0               degree1
                  373                   121                   105
     nodefactor.Sex.M nodefactor.Race.Black nodefactor.Race.NatAm
                  296                    52                   282
nodefactor.Race.Other nodefactor.Race.White    nodefactor.Grade.8
                    4                    67                   118
   nodefactor.Grade.9   nodefactor.Grade.10   nodefactor.Grade.11
                  151                    61                    89
  nodefactor.Grade.12         nodematch.Sex         nodematch.Race
                   48                   253                   196
      nodematch.Grade
                  301
```

We only demostrate the functionality here, but it is in fact a powerful way to diagnose other structural properties of the fitted model. We refer you to the section on egocentric estimation in the ERGM workshop tutorial. That section does not deal with statistical inference for the coefficients, it focuses instead on using simulation to identify systematic lack of fit and revise a model.

6d. Example of parameter recovery and sampling based on Faux Magnolia High:

When we estimate parameters based on sampled data, the sampling uncertainty in our estimates comes from the differences in the observations we draw from sample to sample, and the magnitude of uncertainty is a function of our sample size. This is why we typically see something like $\sqrt{n}$ in the denominator of the standard error of a sample mean or sample proportion. The same principle holds in the context of egocentric network sampling: the standard errors will depend on the number of egos sampled.
This is true despite the fact that we are rescaling first to pseudo-population size, then back down to per capita values. Neither of these influences the estimates of the standard errors – those are influenced only by the size of the egocentric sample.

So let's use the `sample` function from `ergm.ego` to demonstrate this effect. For this section we will use the larger built-in network, `faux.magnolia.high`.

```
data(faux.magnolia.high)
faux.magnolia.high -> fmh
(N <- network.size(fmh))
```

```
[1] 1461
```

Let's start by fitting an ERGM to the complete network, and looking at the coefficients:

```
fit.ergm <- ergm(fmh~edges+degree(0:3)+nodefactor("Race")+nodematch("Race")
                    +nodefactor("Sex")+nodematch("Sex")+absdiff("Grade"))
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 2.436
Iteration 2 of at most 20:
The log-likelihood improved by 2.833
Iteration 3 of at most 20:
The log-likelihood improved by 1.38
Iteration 4 of at most 20:
The log-likelihood improved by 0.3842
Step length converged once. Increasing MCMC sample size.
Iteration 5 of at most 20:
The log-likelihood improved by 0.3488
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag
```

Now, suppose we only observe an egocentric view of the data – as an egocentric census. With an egocentric census, it's as though we give a survey to all of the students. Each student nominates her friends, but does not report the name of the friend, she only reports their sex, race and grade. How does the fit from ergm.ego to this egocentric census compare to the complete-network ergm estimates?

```
fmh.ego <- as.egodata(fmh)
head(fmh.ego)
```

```
$egos
  vertex.names Grade  Race Sex
1            1     9 Black   F
2            2    10 Black   M
3            3    12 Black   F
4            4    11  Hisp   F
5            5     9 Black   F
6            6    11 White   M

$alters
  vertex.names Grade  Race Sex
1            1     9 Black   F
2            2     8 White   M
3            2    10 White   M
4            2    10 White   F
```

```
5            3    12 Black   F
6            3    12 Black   F

$egoWt
[1] 1 1 1 1 1 1

$egoIDcol
[1] "vertex.names"
```

```
egofit <- ergm.ego(fmh.ego~edges+degree(0:3)+nodefactor("Race")+nodematch("Race")
                  +nodefactor("Sex")+nodematch("Sex")+absdiff("Grade"), popsize=N,
                  ppopsize=N)
```

```
Constructing pseudopopulation network.

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 1.866
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 1.087
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

```
modse <- function(fit) sqrt(diag(vcov(fit, sources="model"))) # A convendience function.

# Parameters recovered:
param.compare <- cbind(coef(fit.ergm), coef(egofit)[-1], (coef(fit.ergm)-coef(egofit)[-1])/modse(egofit
colnames(param.compare) <- c("Full nw est", "Ego census est", "diff Z")
round(param.compare, 3)
```

```
                     Full nw est Ego census est diff Z
edges                     -4.938         -4.981  0.151
degree0                    0.955          0.941  0.032
degree1                    0.269          0.261  0.023
degree2                    0.037          0.031  0.025
degree3                   -0.240         -0.238 -0.009
nodefactor.Race.Black     -0.575         -0.550 -0.202
nodefactor.Race.Hisp      -0.243         -0.207 -0.240
nodefactor.Race.NatAm      0.219          0.246 -0.138
nodefactor.Race.Other     -0.056         -0.178  0.458
nodefactor.Race.White     -0.915         -0.901 -0.131
nodematch.Race             1.686          1.683  0.043
nodefactor.Sex.M          -0.088         -0.084 -0.164
nodematch.Sex              0.855          0.850  0.088
absdiff.Grade             -2.115         -2.112 -0.036
```

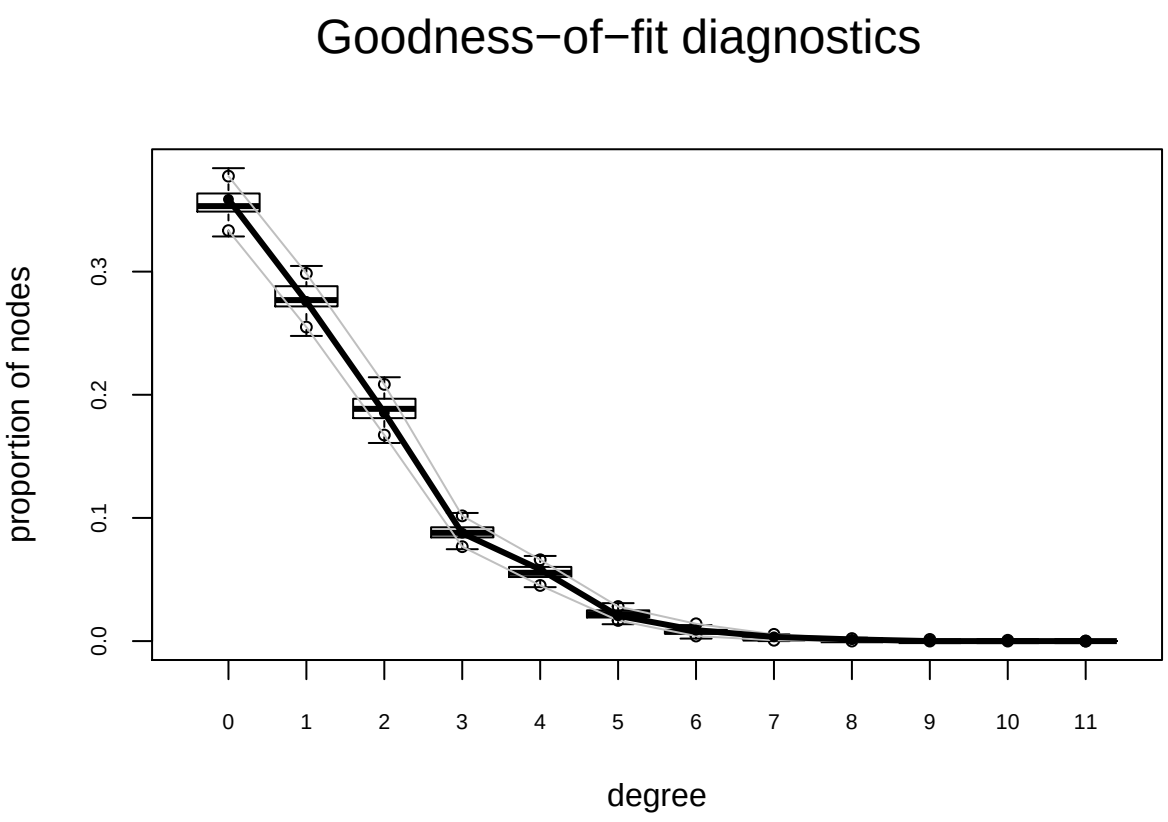
Again, we can diagnose the fitted egocentric model for proper convergence.

```
# MCMC diagnostics.
mcmc.diagnostics(egofit)
```

```
# Check whether the model converged to the right statistics:
plot(gof(egofit, GOF="model"))
```

Let's check whether the fitted model can be used to reconstruct the degree distribution.

```
plot(gof(egofit, GOF="degree"))
```



What if we only had a large sample, instead of an egocentric census? Here, we sample N students with replacement.

```
set.seed(1)
```

```
fmh.egosampN <- sample(fmh.ego, N, replace=TRUE)
egofitN <- ergm.ego(fmh.egosampN ~ edges+degree(0:3)+nodefactor("Race")
                                  +nodematch("Race")+nodefactor("Sex")+nodematch("Sex")
                                  +absdiff("Grade"),popsize=N)
```

```
Constructing pseudopopulation network.
```

```
Unable to match target stats. Using MCMLE estimation.
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 1.54
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 1.293
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag
```

```
# compare the coef
param.compare <- cbind(coef(fit.ergm), coef(egofit)[-1], coef(egofitN)[-1])
colnames(param.compare) <- c("Full nw est", "Ego census est", "Sample N est")
round(param.compare, 3)
```

```
                     Full nw est Ego census est Sample N est
edges                     -4.938         -4.981       -4.714
degree0                    0.955          0.941        1.282
degree1                    0.269          0.261        0.554
degree2                    0.037          0.031        0.263
degree3                   -0.240         -0.238        0.004
nodefactor.Race.Black     -0.575         -0.550       -0.659
nodefactor.Race.Hisp      -0.243         -0.207       -0.338
nodefactor.Race.NatAm      0.219          0.246       -0.099
nodefactor.Race.Other     -0.056         -0.178       -0.653
nodefactor.Race.White     -0.915         -0.901       -0.943
nodematch.Race             1.686          1.683        1.641
nodefactor.Sex.M          -0.088         -0.084       -0.122
nodematch.Sex              0.855          0.850        0.906
absdiff.Grade             -2.115         -2.112       -2.164
```

```
# compare the s.e.'s
se.compare <- cbind(modse(fit.ergm), modse(egofit)[-1], modse(egofitN)[-1])
colnames(se.compare) <- c("Full nw SE", "Ego census SE", "Sample N SE")
round(se.compare, 3)
```

```
                     Full nw SE Ego census SE Sample N SE
edges                     0.315         0.287       0.246
degree0                   0.452         0.431       0.459
degree1                   0.364         0.346       0.367
degree2                   0.272         0.259       0.280
degree3                   0.196         0.195       0.210
nodefactor.Race.Black     0.141         0.122       0.087
nodefactor.Race.Hisp      0.162         0.151       0.133
nodefactor.Race.NatAm     0.192         0.195       0.193
nodefactor.Race.Other     0.469         0.268       0.312
nodefactor.Race.White     0.131         0.112       0.095
nodematch.Race            0.103         0.080       0.077
nodefactor.Sex.M          0.032         0.028       0.033
nodematch.Sex             0.073         0.056       0.059
absdiff.Grade             0.068         0.070       0.072
```

What if we have a smaller sample? If we have a sample of $N/4 = 365$ students, How will our standard errors be affected?

```
fmh.egosampN4 <- sample(fmh.ego, round(N/4), replace=TRUE)

egofitN4 <- ergm.ego(fmh.egosampN4~edges+degree(0:3)+nodefactor("Race")
                     +nodematch("Race")+nodefactor("Sex")+nodematch("Sex")
                     +absdiff("Grade"), popsize=N)
```

```
Constructing pseudopopulation network.

Note: Constructed network has size 1460, different from requested 1461. Estimation should not be meanin

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.6912
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.2918
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

```
# compare the coef
param.compare <- cbind(coef(fit.ergm), coef(egofit)[-1], coef(egofitN)[-1], coef(egofitN4)[-1])
colnames(param.compare) <- c("Full nw est", "Ego census est", "Sample N est", "Sample N/4 est")
round(param.compare, 3)
```

```
                     Full nw est Ego census est Sample N est
edges                     -4.938         -4.981       -4.714
degree0                    0.955          0.941        1.282
degree1                    0.269          0.261        0.554
degree2                    0.037          0.031        0.263
degree3                   -0.240         -0.238        0.004
nodefactor.Race.Black     -0.575         -0.550       -0.659
nodefactor.Race.Hisp      -0.243         -0.207       -0.338
nodefactor.Race.NatAm      0.219          0.246       -0.099
nodefactor.Race.Other     -0.056         -0.178       -0.653
nodefactor.Race.White     -0.915         -0.901       -0.943
nodematch.Race             1.686          1.683        1.641
nodefactor.Sex.M          -0.088         -0.084       -0.122
nodematch.Sex              0.855          0.850        0.906
absdiff.Grade             -2.115         -2.112       -2.164
                     Sample N/4 est
edges                        -5.623
degree0                       0.220
degree1                      -0.233
degree2                      -0.233
degree3                      -0.440
nodefactor.Race.Black        -0.480
nodefactor.Race.Hisp         -0.615
nodefactor.Race.NatAm         0.253
nodefactor.Race.Other         0.544
```

```
nodefactor.Race.White         -0.861
nodematch.Race                 1.819
nodefactor.Sex.M              -0.079
nodematch.Sex                  0.895
absdiff.Grade                 -1.935
```

```
# compare the s.e.'s
se.compare <- cbind(modse(fit.ergm), modse(egofit)[-1], modse(egofitN)[-1], modse(egofitN4)[-1])
colnames(se.compare) <- c("Full nw SE", "Ego census SE", "Sample N SE", "Sample N/4 SE")
round(se.compare, 3)
```

```
                     Full nw SE Ego census SE Sample N SE Sample N/4 SE
edges                    0.315         0.287       0.246         0.690
degree0                  0.452         0.431       0.459         1.165
degree1                  0.364         0.346       0.367         0.927
degree2                  0.272         0.259       0.280         0.691
degree3                  0.196         0.195       0.210         0.463
nodefactor.Race.Black    0.141         0.122       0.087         0.260
nodefactor.Race.Hisp     0.162         0.151       0.133         0.303
nodefactor.Race.NatAm    0.192         0.195       0.193         0.338
nodefactor.Race.Other    0.469         0.268       0.312         0.490
nodefactor.Race.White    0.131         0.112       0.095         0.224
nodematch.Race           0.103         0.080       0.077         0.181
nodefactor.Sex.M         0.032         0.028       0.033         0.063
nodematch.Sex            0.073         0.056       0.059         0.120
absdiff.Grade            0.068         0.070       0.072         0.157
```

As with ordinary statistics, standard error is inverse-proportional to the square root of the sample size.

Suppose that we oversampled the minority groups by a factor of 4. We survey a non-white student 4 times more likely than in a simple random sample.

```
w <- 1+3*((fmh %v% "Race")!="White")
fmh.egosampN4w <- sample(fmh.ego, round(N/4), replace=TRUE, prob=w)

head(fmh.egosampN4w)
```

```
$egos
     vertex.names Grade  Race Sex
78             78    12  Hisp   F
1326         1326    10 White   F
890           890     7 NatAm   M
473           473     8 Black   F
1324         1324    12 Black   F
270           270    11 White   M

$alters
  vertex.names Grade  Race Sex
1          1.1     9 Black   F
2            1     9 Black   F
3         1000    11 White   M
4         1000    11 White   F
5         1000    11 White   M
```

```
6         1013     9 NatAm   F

$egoWt
[1] 0.25 1.00 0.25 0.25 0.25 1.00

$egoIDcol
[1] "vertex.names"
```

```
egofitN4w<-ergm.ego(fmh.egosampN4w~edges+degree(0:3)+nodefactor("Race")+nodematch("Race")+nodefactor("S
```

```
Constructing pseudopopulation network.

Note: Constructed network has size 1522, different from requested 1461. Estimation should not be meanin

Unable to match target stats. Using MCMLE estimation.

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 2.337
Iteration 2 of at most 20:
The log-likelihood improved by 1.499
Iteration 3 of at most 20:
The log-likelihood improved by 1.143
Iteration 4 of at most 20:
The log-likelihood improved by 0.7771
Step length converged once. Increasing MCMC sample size.
Iteration 5 of at most 20:
The log-likelihood improved by 0.4886
Step length converged twice. Stopping.

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

Which standard errors are improved by this sampling scheme?

```
# compare the coef

param.compare <- cbind(coef(fit.ergm), coef(egofitN4)[-1], coef(egofitN4w)[-1])
colnames(param.compare) <- c("Full nw est", "Sample N/4 est", "Oversampled")
round(param.compare, 3)
```

```
                     Full nw est Sample N/4 est Oversampled
edges                     -4.938         -5.623      -4.261
degree0                    0.955          0.220       3.510
degree1                    0.269         -0.233       2.449
degree2                    0.037         -0.233       1.579
degree3                   -0.240         -0.440       1.397
nodefactor.Race.Black     -0.575         -0.480      -0.338
nodefactor.Race.Hisp      -0.243         -0.615      -0.170
nodefactor.Race.NatAm      0.219          0.253       0.276
nodefactor.Race.Other     -0.056          0.544       0.981
nodefactor.Race.White     -0.915         -0.861      -0.728
nodematch.Race             1.686          1.819       1.523
```

```
nodefactor.Sex.M           -0.088         -0.079      -0.045
nodematch.Sex               0.855          0.895       0.659
absdiff.Grade              -2.115         -1.935      -2.149
```

```
# compare the s.e.'s

se.compare <- cbind(modse(fit.ergm),  modse(egofitN4)[-1], modse(egofitN4w)[-1])
colnames(se.compare) <- c("Full nw SE", "Sample N/4 SE", "Oversampled SE")
round(se.compare, 3)
```

```
                     Full nw SE Sample N/4 SE Oversampled SE
edges                     0.315         0.690          0.574
degree0                   0.452         1.165          0.951
degree1                   0.364         0.927          0.806
degree2                   0.272         0.691          0.642
degree3                   0.196         0.463          0.494
nodefactor.Race.Black     0.141         0.260          0.171
nodefactor.Race.Hisp      0.162         0.303          0.227
nodefactor.Race.NatAm     0.192         0.338          0.245
nodefactor.Race.Other     0.469         0.490          0.335
nodefactor.Race.White     0.131         0.224          0.164
nodematch.Race            0.103         0.181          0.142
nodefactor.Sex.M          0.032         0.063          0.071
nodematch.Sex             0.073         0.120          0.137
absdiff.Grade             0.068         0.157          0.171
```

7. Ongoing Developments

The package is under active development. The following features are planned for near future:

- Finite population correction.
- Support for more complex sampling designs, perhaps integrated with **R** package `survey`.
- Support for directed relaions.
- Support for main effects of covariates measured only on the egos.
- Support for alter–alter nominations, perhaps integrated with `egonetR` and other egocentric data management platforms.
- Support for automatic fitting of `tergm`s.