An Introduction to Latent Order Logistic (LOLOG) Network Models

The Statnet Development Team

2018-09-24

Introduction to LOLOG

Latent Order Logistic (LOLOG) models are a general framework for generative statistical modeling of graph datasets motivated by the principle of network growth. This class of models is fully general and terms modeling different important network features can be mixed and matched to provide a rich generative description of complex networks.

Networks tend to evolve over time. This evolution can take the form of the addition or deletion of a tie from the network, or the addition or deletion of a vertex. To motivate our model we consider a growth process, where each edge variable is sequentially considered for edge creation, and edges are not deleted.

In particular the `lolog` package uses a vertex ordering process for the motivating data generating process of the graph. For some networks, it may be reasonable to posit that vertices entered the network in some order that may or may not be random, and upon entering the network, edge variables connecting them to vertices already in the network are considered in a completely random order. The probability that an ordering of edge variables (s) occurs is defined as $p(s)$. The probability of a tie, given the network grown up to a time-point is modeled as a logistic regression

$$\text{logit}\big(p(y_{s_t} = 1|\eta, y^{t-1}, s_{\leq t})\big) = \theta \cdot c(y_{s_t} = 1|y^{t-1}, s_{\leq t})$$

where $s_{\leq t}$ is the growth order of the network up to time t, y^{t-1} is the state of the graph at time $t-1$. $c(y_{s_t}|y^{t-1}, s_{\leq t})$ is a vector representing the change in some graph statistics from time $t-1$ to t if an edge is present, and θ is a vector of parameters.

The full likelihood of an observed graph y given an order s is then just the product of these logistic likelihoods

$$p(y|s, \theta) = \prod_{t=1}^{n_d} p(y_{s_t} = 1|\eta, y^{t-1}, s_{\leq t}),$$

and the marginal likelihood of an observed graph is the sum over all possible orderings

$$p(y|\theta) = \sum_{s} p(y|s, \theta)p(s).$$

Since the ordering of edge variable creation is rarely fully observed, the goal of this package is to perform graph inference on this marginal likelihood. This is done via Method of Moments, Generalized Method of Moments, or variational inference. The following document will show how this is operationalized within the `lolog` package.

Getting Started

This vignette is based on the vignette from the `ergm` package, and is designed to give a working introduction to LOLOG modeling.

This vignette will utilize the statnet suite of packages for data and network manipulation. To install these:

```
install.packages("statnet")
```

To install the latest version of the package from CRAN (not yet available):

```
install.packages("lolog")
```

To install the latest development version from github run the following:

```
# If devtools is not installed:
# install.packages("devtools")

devtools::install_github("statnet/lolog")
```

If this is your first R source package that you have installed, you'll also need a set of development tools. On Windows, download and install Rtools, and `devtools` takes care of the rest. On a Mac, install the Xcode command line tools. On Linux, install the R development package, usually called `r-devel` or `r-base-dev`. For details see Package Development Prerequisites.

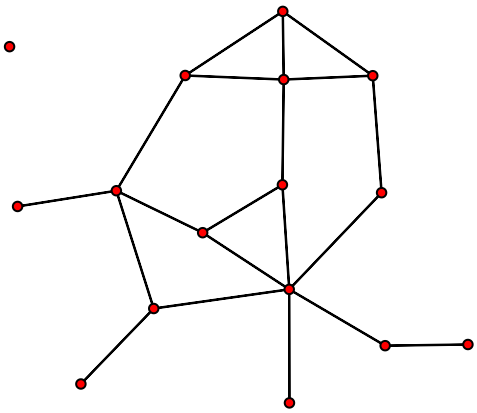
Statistical Network Modeling: The lolog Command and lolog Object

Make sure the `lolog` package is attached:

```
library(lolog)
set.seed(1)
```

The `ergm` package contains several network data sets that you can use for practice examples.

```
suppressPackageStartupMessages(library(ergm))
#data(package='ergm') # tells us the datasets in our packages
data(florentine) # loads flomarriage and flobusiness data
flomarriage # Let's look at the flomarriage data
#>  Network attributes:
#>   vertices = 16
#>   directed = FALSE
#>   hyper = FALSE
#>   loops = FALSE
#>   multiple = FALSE
#>   bipartite = FALSE
#>   total edges= 20
#>     missing edges= 0
#>     non-missing edges= 20
#>
#>  Vertex attribute names:
#>     priorates totalties vertex.names wealth
#>
#> No edge attributes
plot(flomarriage) # Let's view the flomarriage network
```



Networks tend to evolve over time. This evolution can take the form of the addition or deletion of a tie from the network, or the addition or deletion of a vertex. LOLOGs are motivated by a growth process, where each edge variable is sequentially considered for edge creation, and edges are not deleted.

Remember a LOLOG represents the probability of a tie, given the network grown up to a time-point as

$$\text{logit}\big(p(y_{s_t} = 1|\eta, y^{t-1}, s_{\leq t})\big) = \theta \cdot c(y_{s_t} = 1|y^{t-1}, s_{\leq t})$$

where $s_{\leq t}$ is the growth order of the network up to time t, y^{t-1} is the state of the graph at time $t-1$. $c(y_{s_t}|y^{t-1}, s_{\leq t})$ is a vector representing the change in graph statistics from time $t-1$ to t if an edge is present, and θ is a vector of parameters.

An Erdos-Renyi Model

We begin with the simplest possible model, the Bernoulli or Erdos-Renyi model, which contains only an edge term.

```
flomodel.01 <- lolog(flomarriage~edges) # fit model
#> Initializing Using Variational Fit
#>
#>  Model is dyad independent. Replications are redundant. Setting nReplicates <- 1L.
#> Model is dyad independent. Returning maximum likelihood estimate.
flomodel.01
#> MLE Coefficients:
#>     edges
#> -1.609438

summary(flomodel.01) # look in more depth
#>       observed_statistics     theta        se pvalue
#> edges                  20 -1.609438 0.2449451      0
```

How to interpret this model? The log-odds of any tie occurring is:

$$= -1.609 \times \text{change in the number of ties} = -1.609 \times 1$$

for all ties, since the addition of any tie to the network changes the number of ties by 1!

Corresponding probability is:

$$\exp(-1.609)/(1 + \exp(-1.609)) = 0.1667$$

which is what you would expect, since there are 20/120 ties.

A Triangle Model

Let's add a term often thought to be a measure of clustering: the number of completed triangles.

```
flomodel.02 <- lolog(flomarriage~edges()+triangles(), verbose=FALSE)
summary(flomodel.02)
#>           observed_statistics      theta        se pvalue
#> edges                      20 -1.6350235 0.2610002 0.0000
#> triangles                   3  0.2007203 0.7161925 0.7793
```

```
coef1 = flomodel.02$theta[1]
coef2 = flomodel.02$theta[2]
logodds = coef1 + c(0,1,2) * coef2
expit = function(x) 1/(1+exp(-x))
ps = expit(logodds)
coef1 = round(coef1, 3)
coef2 = round(coef2, 3)
logodds = round(logodds, 3)
ps = round(ps, 3)
```

Again, how to interpret coefficients?

Conditional log-odds of two actors forming a tie is:

$$-1.635 \times \text{change in the number of ties} + 0.201 \times \text{change in number of triangles}$$

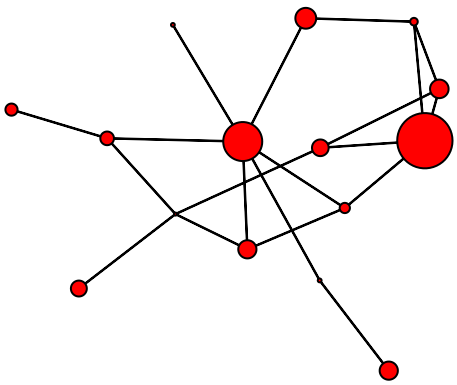
- if the tie will not add any triangles to the network, its log-odds is: -1.635.
- if it will add one triangle to the network, its log-odds is: $-1.635 + 0.201 = -1.434$
- if it will add two triangles to the network, its log-odds is: $-1.635 + 0.201 \times 2 = -1.234$
- the corresponding probabilities are 0.163, 0.192, and 0.226.

Let's take a closer look at the `lolog` object itself:

```
class(flomodel.02) # this has the class lolog
#> [1] "lologGmm" "lolog"    "list"

names(flomodel.02) # let's look straight at the lolog obj.
#>  [1] "method"          "formula"         "auxFromula"
#>  [4] "theta"           "stats"           "estats"
#>  [7] "auxStats"        "obsStats"        "targetStats"
#> [10] "obsModelStats"   "net"             "grad"
#> [13] "vcov"            "likelihoodModel"
```

```
flomodel.02$theta
#>      edges  triangles
#> -1.6350235  0.2007203
flomodel.02$formula
#> flomarriage ~ edges() + triangles()
wealth <- flomarriage %v% 'wealth' # the %v% extracts vertex
wealth # attributes from a network
#>  [1]  10  36  55  44  20  32   8  42 103  48  49   3  27  10 146  48
plot(flomarriage, vertex.cex=wealth/25) # network plot with vertex size
```



```
                                        # proportional to wealth
```

The Effect of Wealth

We can test whether edge probabilities are a function of wealth:

```
flomodel.03 <- lolog(flomarriage~edges+nodeCov('wealth'))
#> Initializing Using Variational Fit
#>
#>  Model is dyad independent. Replications are redundant. Setting nReplicates <- 1L.
#> Model is dyad independent. Returning maximum likelihood estimate.
summary(flomodel.03)
#>                observed_statistics       theta          se pvalue
#> edges                           20 -2.59492903 0.536051763 0.0000
#> nodecov.wealth                2168  0.01054591 0.004674236 0.0241
```

Yes, there is a significant positive wealth effect on the probability of a tie.

Does Large Wealth Inequality Matter?

We can test whether edge probabilities are a function of wealth:

```
wdiff<-outer(flomarriage %v% "wealth", flomarriage %v% "wealth",function(x,y){abs(x-y)>20})
table(wdiff)
#> wdiff
#> FALSE  TRUE
#>   110   146
flomodel.04 <- lolog(flomarriage~edges+nodeCov('wealth')+edgeCov(wdiff,"inequality"))
#> Initializing Using Variational Fit
#>
#>  Model is dyad independent. Replications are redundant. Setting nReplicates <- 1L.
#> Model is dyad independent. Returning maximum likelihood estimate.
summary(flomodel.04)
#>                    observed_statistics      theta          se pvalue
#> edges                               20 -3.5637462 0.798418212 0.0000
#> nodecov.wealth                    2168  0.0065339 0.004749717 0.1689
#> edgeCov.inequality                  18  1.7806414 0.791428126 0.0245
```

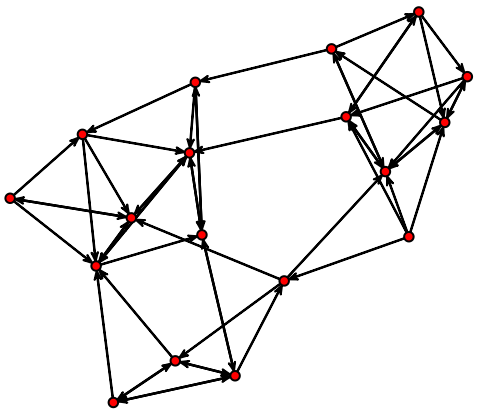
The inequality in wealth does seem to increase the probability of a tie.

Reciprocity

Let's try a model or two on:

Is there a statistically significant tendency for ties to be reciprocated ('mutuality')?

```
data(samplk)
ls() # directed data: Sampson's Monks
#>  [1] "coef1"            "coef2"            "emptyNetwork"
#>  [4] "expit"            "fituk"            "fitukErgm"
#>  [7] "fitukInd"         "flobusiness"      "flomarriage"
#> [10] "flomodel.01"      "flomodel.02"      "flomodel.03"
#> [13] "flomodel.04"      "g"                "logodds"
#> [16] "makeEmptyNetwork" "ps"               "samplk1"
#> [19] "samplk2"          "samplk3"          "src"
#> [22] "ukFaculty"        "wdiff"            "wealth"
samplk3
#>  Network attributes:
#>   vertices = 18
#>   directed = TRUE
#>   hyper = FALSE
#>   loops = FALSE
#>   multiple = FALSE
#>   bipartite = FALSE
#>   total edges= 56
#>     missing edges= 0
#>     non-missing edges= 56
#>
#>  Vertex attribute names:
#>     cloisterville group vertex.names
#>
#> No edge attributes
plot(samplk3)
```

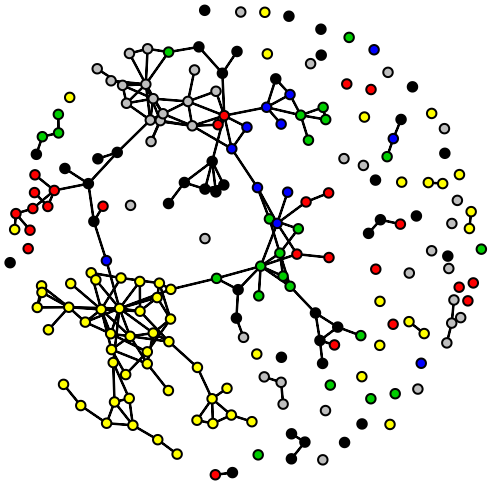


```
sampmodel.01 <- lolog(samplk3~edges+mutual, verbose=FALSE)
summary(sampmodel.01)
#>        observed_statistics     theta        se pvalue
#> edges                   56 -1.775532 0.1694364  0e+00
#> mutual                  15  2.485336 0.6849980  3e-04
```

Modeling Larger Networks

Let's try a larger network. First we will fit a dyad independent model.

```
data(faux.mesa.high)
mesa <- faux.mesa.high
mesa
#>  Network attributes:
#>   vertices = 205
#>   directed = FALSE
#>   hyper = FALSE
#>   loops = FALSE
#>   multiple = FALSE
#>   bipartite = FALSE
#>   total edges= 203
#>     missing edges= 0
#>     non-missing edges= 203
#>
#>  Vertex attribute names:
#>     Grade Race Sex
#>
#> No edge attributes
plot(mesa, vertex.col='Grade')
```



```
#legend('bottomleft',fill=7:12,legend=paste('Grade',7:12),cex=0.75)
mesa %v% "GradeCat" <- as.character(mesa %v% "Grade")
fauxmodel.01 <- lolog(mesa ~edges + nodeMatch('GradeCat') + nodeMatch('Race'))
#> Initializing Using Variational Fit
#>
#>  Model is dyad independent. Replications are redundant. Setting nReplicates <- 1L.
#> Model is dyad independent. Returning maximum likelihood estimate.
summary(fauxmodel.01)
#>                    observed_statistics      theta        se pvalue
#> edges                              203 -6.2254111 0.1737005 0.0000
#> nodematch.GradeCat                 163  2.8278899 0.1773356 0.0000
#> nodematch.Race                     103  0.4266748 0.1427599 0.0028
```

Now lets try adding in transitivity and 2-stars (a measure of degree spread)

```
# This may take a minute or two
fauxmodel.02 <- lolog(mesa ~edges + nodeMatch('GradeCat') + nodeMatch('Race') +
                        triangles + star(2), verbose=FALSE)
summary(fauxmodel.02)
#>                    observed_statistics      theta         se pvalue
#> edges                              203 -6.3216047 0.21722728 0.0000
#> nodematch.GradeCat                 163  2.5949595 0.19521630 0.0000
#> nodematch.Race                     103  0.4144164 0.17508446 0.0179
#> triangles                           62  3.0442395 0.70365678 0.0000
#> star.2                             659  0.0238069 0.05218006 0.6482
```

We see strong evidence of additional transitivity, but little evidence that the degrees have higher spread than expected given the other terms. With LOLOG models, we avoid some of the degeneracy problems present in ERGM models. Let's try the same model in `ergm`

```
fauxmodel.01.ergm <- ergm(mesa ~edges + nodematch('GradeCat') + nodematch('Race') +
                            triangles + kstar(2))
#> Starting maximum pseudolikelihood estimation (MPLE):
#> Evaluating the predictor and response matrix.
#> Maximizing the pseudolikelihood.
#> Finished MPLE.
#> Starting Monte Carlo maximum likelihood estimation (MCMLE):
#> Iteration 1 of at most 20:
#> Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL), : Number of edges in a simul
```

Of course it is highly recommended that you take great care in selecting terms for use with `ergm` for precisely the reason that many can lead to model degeneracy. A much better choice here for the `ergm` model would have been to use `gwesp` and `gwdegree` in place of `triangles` and 2-stars.

Specifying Vertex Inclusion Order

LOLOG models are motivated by a growth process where edge variables are "added" to the network one at a time. The package implements a sequential vetex ordering process. Specifically, vertices are "added" to the network in random order, and then edge variables connecting the added vertex to each other vertex already added are included in random order.

If you either observe or partially observe the inclusion order of vertices, this information can be included by specifying it in the model formula after a '|' character. In this example we will consider the network relations of a group of partners at a law firm. We observe the senority of the partner, the type of practice and the office where they work. (cSenority and cPractice are just categorical versions of the numerically coded practice and senority variables). It is plausible to posit that partners with more senority entered the law firm and hense the network prior to less senior individuals.

```
library(network)
data(lazega)
seniority <- as.numeric(lazega %v% "seniority") # Lower values are more senior
fit <- lolog(lazega ~  edges() + triangles() + nodeCov("cSeniority") +
               nodeCov("cPractice") + nodeMatch("gender") + nodeMatch("practice") +
               nodeMatch("office") | seniority, verbose=FALSE)
summary(fit)
#>                    observed_statistics       theta         se pvalue
#> edges                              115 -7.62480177 1.04368899 0.0000
#> triangles                          120  1.04200765 0.38241615 0.0064
#> nodecov.cSeniority                4687  0.02782641 0.01051917 0.0082
#> nodecov.cPractice                  359  0.71274690 0.18686134 0.0001
```

```
#> nodematch.gender                99  1.14111172 0.40977466 0.0054
#> nodematch.practice              72  0.94953124 0.27329167 0.0005
#> nodematch.office                85  1.60272724 0.28792514 0.0000
```

In some cases only a partial ordering is observed. For example we might observed the month someone joined a group, with multiple individuals joining per month. When ties like this exist in the ordering variable, network simulations respect the partial ordering, with ties broken at random for each simulated network.

Model Terms Available for LOLOG Estimation and Simulation

Model terms are the expressions (e.g., `triangle`) used to represent predictors on the right-hand size of equations used in:

- calls to `lolog` (to estimate an LOLOG model)
- calls to simulate (to simulate networks from an LOLOG model fit)
- calls to calculateStatistics (to obtain measurements of network statistics on a dataset)

Terms Provided with `lolog`

For a list of available terms that can be used to specify a LOLOG model, type:

```
help('lolog-terms')
```

This currently produces:

- `edges` (dyad-independent) (order-independent) (directed) (undirected)
- `star(k, direction=1L)` (order-independent) (directed) (undirected)
- `triangles()` (order-independent) (directed) (undirected)
- `clustering()` (order-independent) (undirected)
- `transitivity()` (order-independent) (undirected)
- `mutual()` (order-independent) (directed)
- `nodeCov(name)` (dyad-independent) (order-independent) (directed) (undirected)
- `nodeMatch(name)` (dyad-independent) (order-independent) (directed) (undirected)
- `nodeMix(name)` (dyad-independent) (order-independent) (directed) (undirected)
- `edgeCov(x)` (dyad-independent) (order-independent) (directed) (undirected)
- `degree(d, direction=0L, lessThanOrEqual=FALSE)` (order-independent) (directed) (undirected)
- `degreeCrossProd()` (order-independent) (undirected)
- `gwdsp(alpha)` (order-independent) (directed) (undirected)
- `gwesp(alpha)` (order-independent) (directed) (undirected)
- `gwdegree(alpha, direction=0L)` (order-independent) (directed) (undirected)
- `esp(d)` (order-independent) (directed) (undirected)
- `geoDist(long, lat, distCuts=Inf)` (dyad-independent) (order-independent) (undirected)
- `dist(names` (dyad-independent) (order-independent) (undirected)
- `preferentialAttachment(k=1)` (undirected)
- `sharedNbrs(k=1)` (undirected)
- `nodeLogMaxCov(name)` (order-independent) (undirected)
- `twoPath()` (order-independent) (directed) (undirected)
- `boundedDegree(lower,upper)` (order-independent) (undirected)

Coding New Terms

Full details on coding new terms is beyond the scope of this document. However, `lolog` provides facilities for extending the package to provide new terms. An example of this can be created via

```
lologPackageSkeleton()
```

and then at the command prompt, run

```
R CMD build LologExtension
R CMD INSTALL LologExtension_1.0.tar.gz
```

A new package will be installed implementing the minDegree (minimum degree) statistic.

Alternatively, C++ extensions can be added directly from R without the need of any package infrastructure via the inlining features of the Rcpp package. `lolog` provides an inline plugin for this feature. For details, see

```
help("inlineLologPlugin")
```

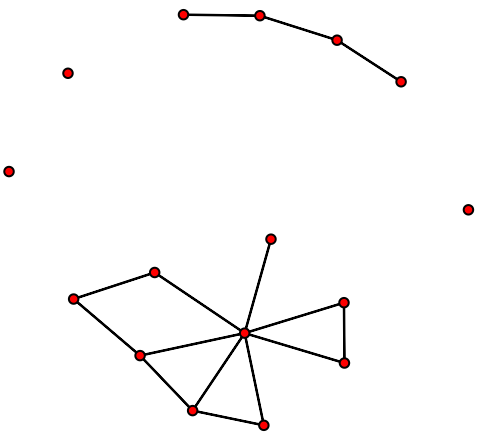
Network Simulation and Computation

Network Statistics can be calculated using the calculateStatistics function

```
calculateStatistics(mesa ~ edges + triangles + degree(0:15))
#>     edges triangles  degree.0  degree.1  degree.2  degree.3  degree.4
#>       203        62        57        51        30        28        18
#>  degree.5  degree.6  degree.7  degree.8  degree.9 degree.10 degree.11
#>        10         2         4         1         2         1         0
#> degree.12 degree.13 degree.14 degree.15
#>         0         1         0         0
```

Simulating from a fitted `lolog` object with

```
nets <- simulate(flomodel.03,nsim=10) #Generates a list of BinaryNet objects
plot(nets[[1]])
```



Voila. Of course, networks that you generate will look somewhat different. Note that the objects created are `BinaryNet` objects unless the convert parameter is set to TRUE, in which case they are `network` objects.

Working with `BinaryNet` Objects

`BinaryNet`s are network data structures native to the `lolog` package. They are special in a couple ways. First, they have a sparse representation of missingness, such that a directed network where halve of the nodes have been egocentrically sampled takes the same amount of space as a simple fully observed network. Secondly, they may be passed up and down easily from R to C++ and vis a versa. Finally, they are extensible

on the C++ level, so that their implementation may be replaced. For example, it might be useful to put in a file backed storage system for large problems.

`BinaryNet`s can be coerced to and from `network` objects.

```
data(sampson)

#coersion
net <- as.BinaryNet(samplike)
nw2 <- as.network(net)
print(nw2)
#>  Network attributes:
#>   vertices = 18
#>   directed = TRUE
#>   hyper = FALSE
#>   loops = FALSE
#>   multiple = FALSE
#>   bipartite = FALSE
#>   total edges= 88
#>     missing edges= 0
#>     non-missing edges= 88
#>
#>  Vertex attribute names:
#>     cloisterville group vertex.names
#>
#> No edge attributes

#dyad Extraction
net[1:2,1:5]
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE  TRUE FALSE  TRUE
#> [2,]  TRUE FALSE FALSE FALSE FALSE
net$outNeighbors(c(1,2,3))
#> [[1]]
#> [1]  2  3  5  8 12 14
#>
#> [[2]]
#> [1]  1  7 12 14 15
#>
#> [[3]]
#> [1]  1  2 13 17 18

#dyad assignment
net[1,1:5] <- rep(NA,5)
net[1:2,1:5]
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE    NA    NA    NA    NA
#> [2,]  TRUE FALSE FALSE FALSE FALSE
net[1:2,1:5,maskMissing=FALSE] #remove the mask over missing values and see
#>       [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] FALSE  TRUE  TRUE FALSE  TRUE
#> [2,]  TRUE FALSE FALSE FALSE FALSE
                                #nothing was really changed

#node variables
```

```
net$variableNames()
#> $discrete
#> [1] "cloisterville" "group"         "na"            "vertex.names"
#>
#> $continuous
#> character(0)
net[["group"]]
#>  [1] Turks    Turks    Outcasts Loyal    Loyal    Loyal    Turks
#>  [8] Loyal    Loyal    Loyal    Loyal    Turks    Outcasts Turks
#> [15] Turks    Turks    Outcasts Outcasts
#> Levels: Loyal Outcasts Turks
net[["rnorm"]] <- rnorm(18)
net[["rnorm"]]
#>  [1]  0.1258612  0.4620700 -0.9483070 -0.2279269  0.2882392 -0.3182463
#>  [7]  1.0249388 -0.8219556  1.0958955  1.3596402 -0.1191218  0.6464820
#> [13]  1.0195519  0.9726483  0.8808873 -0.6801720 -0.1446265  0.6971865

#See available methods
#print(DirectedNet)
#print(UndirectedNet)
```

All user facing functions in the `lolog` package accept `BinaryNet`s as arguments, and will convert `network`, `igraph` and `tidygraph` graph objects to `BinaryNet`s automatically.

Order Dependent Terms

LOLOG allows for network statistics that depend not just on the network, but also the (unobserved) order in which dyads were 'added' to the network. One model of this class is Barabasi-Albert preferential attachment model, which is closely approximated by a LOLOG model with an edges and `preferentialAttachment` term.

For each order dependent statistic, one or more order independent statistics must be specified as moment matching targets. In this case, we will use a two-star term:

```
flomodel.04 <- lolog(flomarriage ~ edges() + preferentialAttachment(),
                     flomarriage ~ star(2), verbose=FALSE)
summary(flomodel.04)
#>                        observed_statistics       theta        se pvalue
#> edges                                   20 -1.53333439 2.0523468 0.4550
#> preferentialAttachment                  NA  0.03107266 0.8450358 0.9707
```

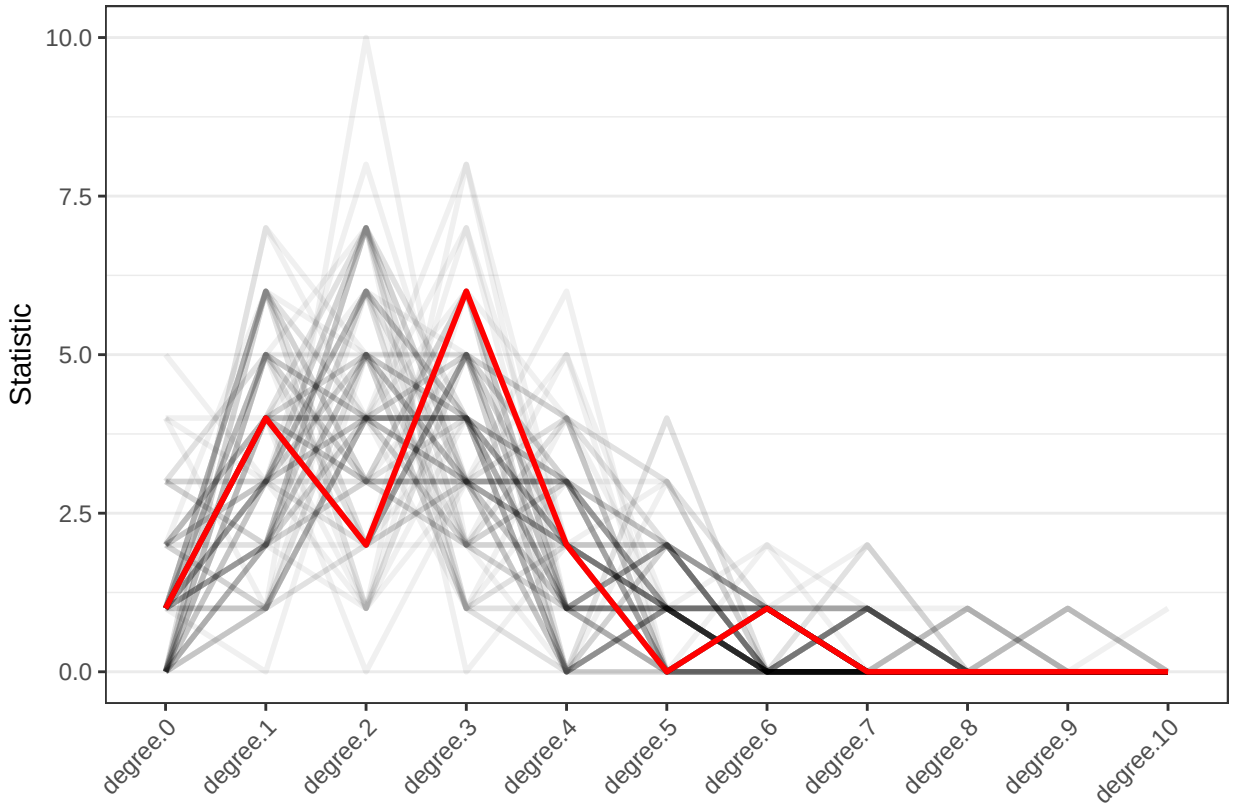
Examining the Quality of Model Fit - GOF

LOLOGs are generative models - that is, they represent the process that governs tie formation at a local level. These local processes in turn aggregate up to produce characteristic global network properties, even though these global properties are not explicit terms in the model. One test of whether a model "fits the data" is therefore how well it reproduces these global properties. We do this by choosing a network statistic that is not in the model, and comparing the value of this statistic observed in the original network to the distribution of values we get in simulated networks from our model.

We begin by comparing the degree structure of simulated networks compared to the observed (red)

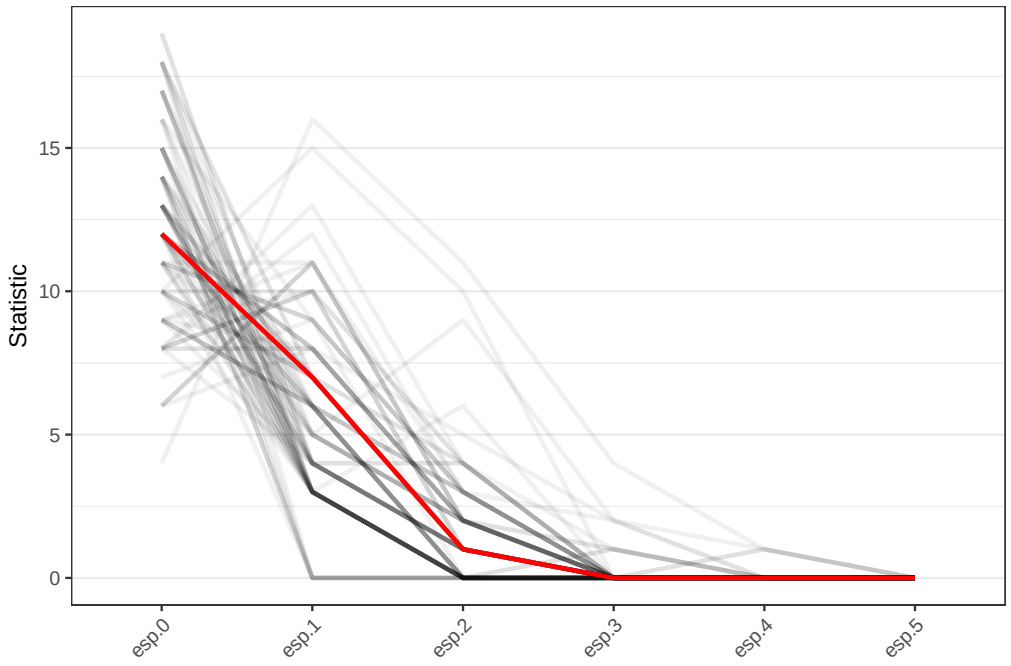
```
gdeg <- gofit(flomodel.03, flomarriage ~ degree(0:10))
gdeg
```

```
#>           obs min mean max    pvalue
#> degree.0    1   0 1.15   5 0.8901396
#> degree.1    4   0 3.39   7 0.7032129
#> degree.2    2   0 4.09  10 0.2609429
#> degree.3    6   0 3.63   8 0.1337452
#> degree.4    2   0 1.87   6 0.9242996
#> degree.5    0   0 1.07   4 0.2577892
#> degree.6    1   0 0.42   2 0.2785162
#> degree.7    0   0 0.26   2 0.6066035
#> degree.8    0   0 0.06   1 0.8015210
#> degree.9    0   0 0.05   1 0.8194396
#> degree.10   0   0 0.01   1 0.9203443
plot(gdeg)
```



Next we can look at the edgewise shared partner distribution in simulated networks compared with the observed distribution (red)

```
gesp <- gofit(flomodel.03, flomarriage ~ esp(0:5))
gesp
#>       obs min  mean max    pvalue
#> esp.0  12   4 12.33  19 0.9215674
#> esp.1   7   0  5.75  16 0.7030868
#> esp.2   1   0  1.56  11 0.7838569
#> esp.3   0   0  0.15   4 0.7878181
#> esp.4   0   0  0.04   1 0.8390561
#> esp.5   0   0  0.00   0        NA
plot(gesp)
```


Statistic
15
10
5
0
esp.0
esp.1
esp.2
esp.3
esp.4
esp.5