Modeling valued networks with *statnet*

Pavel N. Krivitsky, Carter T. Butts,
The Statnet Development Team

May 22, 2013

Contents

1 Getting the software

If you have not already done so, please make sure that you have a reasonably new version of R, preferably the latest (3.0.0) (R Core Team, 2013). Then, download and install the latest versions of the Statnet (Handcock et˜al., 2008; Goodreau et˜al., 2008) packages, in particular `ergm` version 3.1 (Hunter et˜al., 2008; Handcock et˜al., 2013), `ergm.count` version 3.1 (Krivitsky, 2013), `latentnet` version 2.4-4 (Krivitsky and Handcock, 2013), and their dependencies. You can accomplish this by typing:

```
> install.packages("ergm.count")
> install.packages("latentnet")
```

```
> library(ergm.count)
> library(latentnet)
```

We will also need a temporary workaround for a bug:

```
> as.mcmc.default <- coda:::as.mcmc.default
> as.mcmc.list.default <- coda:::as.mcmc.list.default
```

2 `network` and edge attributes

`network` (Butts, 2008; Butts et˜al., March 15, 2013) objects have three types of attributes:

network attributes attributes which pertain to the whole network and include such information as network size, directedness, and multiplicity;

vertex attributes attributes which pertain to the individual vertices in the network and include such information as vertex label, as well as group assignment or some other property of the individual being represented;

edge attributes attributes which pertain to edges in the network and include such information as edge value.

An edge attribute is only defined for edges that exist in the network. Thus, in a matter of speaking, to set an edge value, one first has to *create* an edge and then *set* its attribute.

As with network and vertex attributes, edge attributes that have been set can be listed with `list.edge.attributes`. Every network has at least one edge attribute: `"na"`, which, if set to `TRUE`, marks an edge as missing.

2.1 Constructing valued networks

2.1.1 Sampson's Monks, pooled

The first dataset that we'll be using is the (in)famous Sampson's monks. Dataset `samplk` in package `ergm` contains three (binary) networks: `samplk1`, `samplk2`, and `samplk3`, containing the Monks' top-tree friendship nominations at each of the three survey time points. We are going to construct a valued network that pools these nominations.

Method 1: From a sociomatrix Suppose that a valued sociomatrix is available.

```
> data(samplk)
> ls()
```

```
[1] "as.mcmc.default"      "as.mcmc.list.default" "samplk1"
[4] "samplk2"              "samplk3"
```

```
> as.matrix(samplk1)[1:5,1:5]
```

```
            Ramuald Bonaventure Ambrose Berthold Peter
Ramuald           0           0       0        0     1
Bonaventure       0           0       0        0     1
Ambrose           0           1       0        0     0
Berthold          0           0       1        0     1
Peter             1           1       0        1     0
```

```
> # A sociomatrix totaling the nominations.
> samplk.tot.m<-as.matrix(samplk1)+as.matrix(samplk2)+as.matrix(samplk3)
> samplk.tot.m[1:5,1:5]
```

```
            Ramuald Bonaventure Ambrose Berthold Peter
Ramuald           0           2       1        0     3
Bonaventure       0           0       1        0     3
Ambrose           0           3       0        0     0
Berthold          0           1       3        0     3
Peter             1           3       0        3     0
```

```
> # Create a network where the number of nominations becomes an attribute of an edge.
> samplk.tot <- as.network(samplk.tot.m, directed=TRUE, matrix.type="a",
                           ignore.eval=FALSE, names.eval="nominations" # Important!
                           )
> # Add vertex attributes.
> samplk.tot %v% "group" <- samplk1 %v% "group" # Groups identified by Sampson
> samplk.tot %v% "group"
```

```
 [1] "Waverers" "Loyal"    "Loyal"    "Loyal"    "Loyal"    "Loyal"
 [7] "Waverers" "Turks"    "Turks"    "Turks"    "Turks"    "Turks"
[13] "Turks"    "Turks"    "Waverers" "Outcasts" "Outcasts" "Outcasts"
```

```
> samplk.tot %v% "vertex.names" <- samplk1 %v% "vertex.names" # Names
> # We can view the attribute as a sociomatrix.
> as.matrix(samplk.tot,attrname="nominations")[1:5,1:5]
```

	Ramuald	Bonaventure	Ambrose	Berthold	Peter
Ramuald	0	2	1	0	3
Bonaventure	0	0	1	0	3
Ambrose	0	3	0	0	0
Berthold	0	1	3	0	3
Peter	1	3	0	3	0

```
> # Also, note that samplk.tot now has an edge if i nominated j *at least once*.
> as.matrix(samplk.tot)[1:5,1:5]
```

	Ramuald	Bonaventure	Ambrose	Berthold	Peter
Ramuald	0	1	1	0	1
Bonaventure	0	0	1	0	1
Ambrose	0	1	0	0	0
Berthold	0	1	1	0	1
Peter	1	1	0	1	0

Method 2: From an edgelist Now, suppose that instead we have an edgelist with values:

```
> samplk.tot.el <- as.matrix(samplk.tot, attrname="nominations",
                             matrix.type="edgelist")
> samplk.tot.el[1:5,]
```

	[,1]	[,2]	[,3]
[1,]	5	1	1
[2,]	7	1	1
[3,]	1	2	2
[4,]	3	2	3
[5,]	4	2	1

```
> # and an initial empty network.
> samplk.tot2 <- samplk1 # Copy samplk1
> delete.edges(samplk.tot2, seq_along(samplk.tot2$mel)) # Empty it out
> samplk.tot2

 Network attributes:
  vertices = 18
  directed = TRUE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
```

```
  total edges= 0
    missing edges= 0
    non-missing edges= 0

 Vertex attribute names:
    cloisterville group vertex.names
```

```
> samplk.tot2[samplk.tot.el[,1:2], names.eval="nominations", add.edges=TRUE] <-
    samplk.tot.el[,3]
> as.matrix(samplk.tot2,attrname="nominations")[1:5,1:5]
```

```
            Ramuald Bonaventure Ambrose Berthold Peter
Ramuald           0           2       1        0     3
Bonaventure       0           0       1        0     3
Ambrose           0           3       0        0     0
Berthold          0           1       3        0     3
Peter             1           3       0        3     0
```

In general, the construction `net[i,j, names.eval="attrname", add.edges=TRUE] <- value` can be used to modify individual edge values for attribute `"attrname"`. This way, we can also add more than one edge attribute to a network.

2.1.2 Zachary's Karate club

The other dataset we'll be using is almost as (in)famous Zachary's Karate Club dataset is a collapsed multiplex network that counts the number of social contexts in which each pair of individuals associated with the Karate Club in question interacted. A total of 8 contexts were considered, but as the contexts themselves were determined by the network process, this limit itself can be argued to be endogenous.

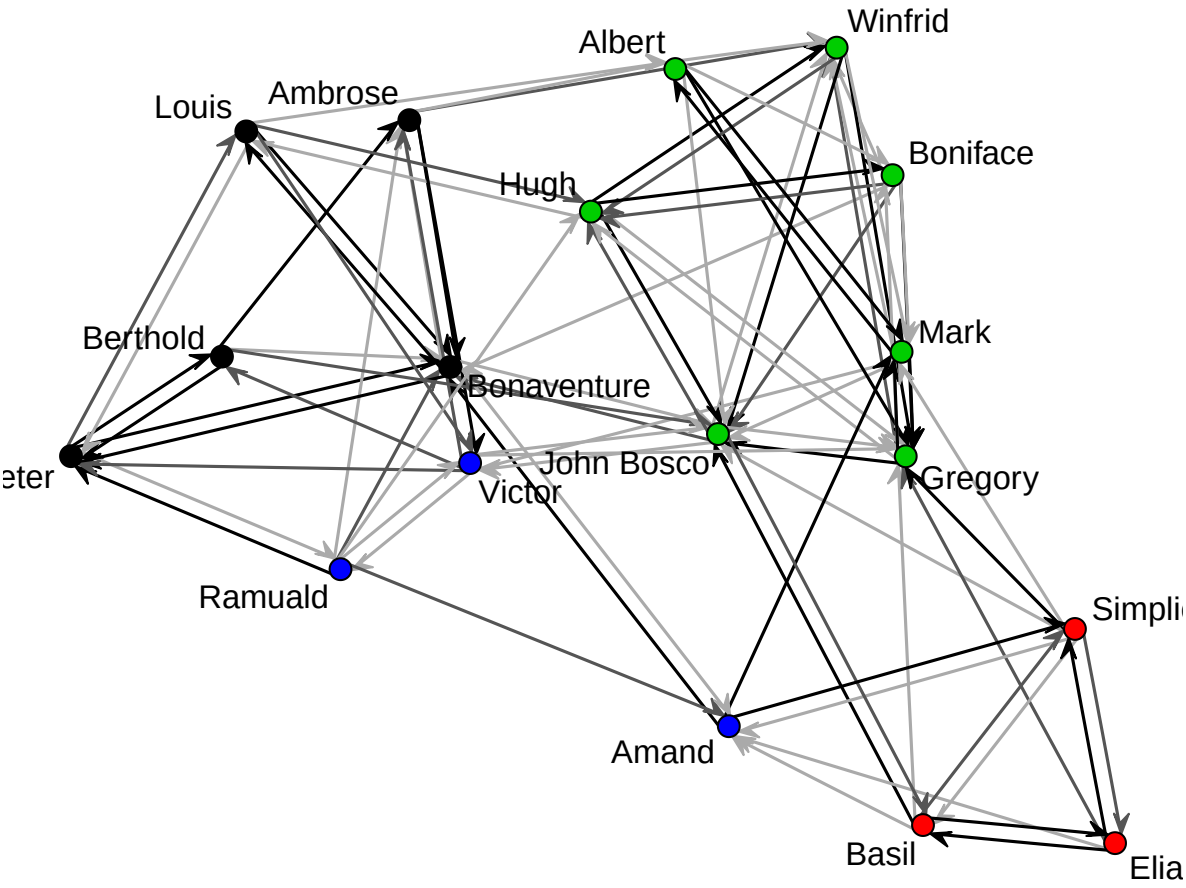
Over the course of the study, the club split into two factions one led by the instructor, "Mr. Hi" and the other led by the Club President, "John A.". Zachary also recorded the faction alignment of every regular attendee in the club. This dataset is included in the `ergm.count` package, as `zach`.

2.2 Visualizing a valued network

The `network`'s `plot` method for `networks` can be used to plot a sociogram of a network. When plotting a valued network, we might want to do color the ties depending on their value. Function `gray` can be used to generate a gradient of colors, with `gray(0)` generating black and `gray(1)` generating white. This can then be passed to the `edge.col` argument of `plot.network`.

Sampson's Monks For the monks, let's do it as a matrix.

```
> par(mar=rep(0,4))
> samplk.ecol <-
    matrix(gray(1 - (as.matrix(samplk.tot, attrname="nominations")/3)),
           nrow=network.size(samplk.tot))
> plot(samplk.tot, edge.col=samplk.ecol, usecurve=TRUE, edge.curve=0.0001,
       displaylabels=TRUE, vertex.col=as.factor(samplk.tot%v%"group"))
```

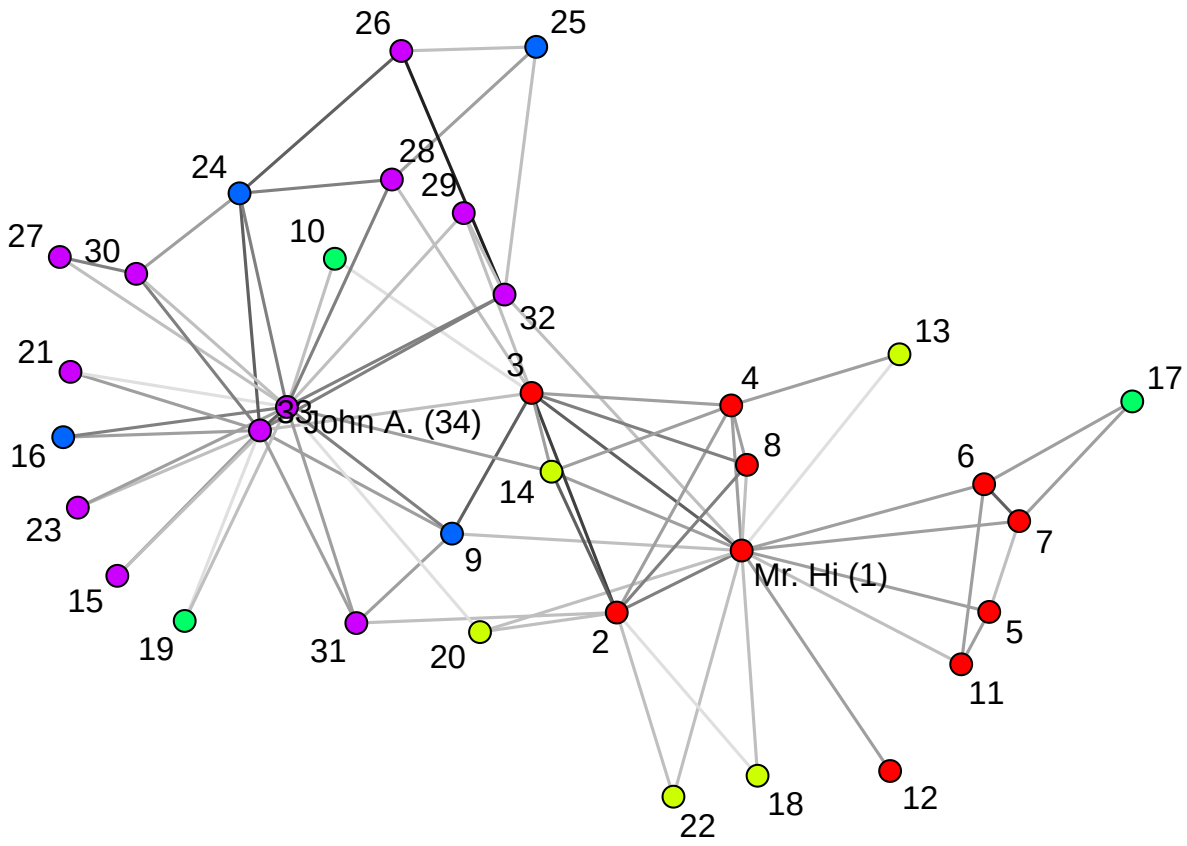


Edge color can also be passed as a vector of colors corresponding to edges. It's more efficient, but the ordering in the vector must correspond to `network` object's internal ordering, so it should be used with care.

Zachary's Karate Club In the following plot, we plot those strongly aligned with Mr. Hi as red, those with John A. with purple, those neutral as green, and those weakly aligned with

colors in between.

```
> data(zach)
> zach.ecol <- gray(1 - (zach %e% "contexts")/8)
> zach.vcol <- rainbow(5)[zach %v% "faction.id"+3]
> par(mar=rep(0,4))
> plot(zach, edge.col=zach.ecol, vertex.col=zach.vcol, displaylabels=TRUE)
```



3 Modeling dyad-dependent interaction counts with valued ERGMs using `ergm.count`

Many of the functions in package **ergm**, including **ergm**, **simulate**, and **summary**, have been extended to handle networks with valued relatins. They switch into this valued mode when passed the **response** argument, specifying the name of the edge attribute to use as the response variable. For example, a new valued term **sum** evaluates the sum of the values of all of the relations: $\sum_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}$. So,

```
> summary(samplk.tot~sum)
```

produces an error (because no such term has been implemented for binary mode), while

```
> summary(samplk.tot~sum, response="nominations")
```

```
sum
168
```

gives the summary statistics. We will introduce more statistics shortly. First, we need to introduce the notion of valued ERGMs.

For a more in-depth discussion of the following, see (Krivitsky, 2012).

3.1 Valued ERGMs

Generally, a valued ERGM (for discrete variables) looks like this:

$$\Pr_{h,\boldsymbol{g}}(\boldsymbol{Y}=\boldsymbol{y};\boldsymbol{\theta}) = \frac{h(\boldsymbol{y})\exp(\boldsymbol{\theta}^\top \boldsymbol{g}(\boldsymbol{y}))}{c_{h,\boldsymbol{g}}(\boldsymbol{\theta})},\ \boldsymbol{y}\in\mathcal{Y},$$

where the normalizing constant

$$c_{h,\boldsymbol{g}}(\boldsymbol{\theta}) = \sum_{\boldsymbol{y}\in\mathcal{Y}} h(\boldsymbol{y})\exp\left(\boldsymbol{\theta}^\top \boldsymbol{g}(\boldsymbol{y})\right).$$

3.1.1 New concept: a reference distribution

With binary ERGMs, we only concern ourselves with presence and absence of ties among actors — who is connected with whom? If we want to model values as well, we need to think about who is connected with whom *and* how strong or intense these connections are. In particular, we need to think about how the values for connections we measure are distributed. The reference distribution (a *reference measure*, for the mathematically inclined) specifies the model for the data *before* we add any ERGM terms, and is the first step in modeling these values. The reference distribution is specified using a one-sided formula as a **reference** argument to an **ergm** or **simulate** call. Running

```
> help("ergm-references")
```

will list the choices implemented in the various packages, and are given as a one-sided formula.

Conceptually, it has two ingredients: the sample space and the baseline distribution ($h(\boldsymbol{y})$). An ERGM that "borrows" these from a distribution X for which we have a name is called an *X-reference ERGM*.

The sample space For binary ERGMs, the sample space $\mathcal{Y}$ — the set of possible networks that can occur — is some subset of $2^{\mathbb{Y}}$, the set of all possible ways in which relationships among the actors may occur.

For the sample space of valued ERGMs, we need to define $\mathbb{S}$, the set of possible values each relationship may take. For example, for count data, that's $\mathbb{S} = \{0, 1, \ldots, s\}$ if the maximum count is s and $\{0, 1, \ldots\}$ if there is no *a priori* upper bound. Having specified that, $\mathcal{Y}$ is defined as some subset of $\mathbb{S}^{\mathbb{Y}}$: the set of possible ways to assign to each relationship a value.

As with binary ERGMs, other constraints like degree distribution may be imposed on $\mathcal{Y}$.

$h(\boldsymbol{y})$: The baseline distribution What difference does it make?

Suppose that we have a sample space with $\mathbb{S} = \{0, 1, 2, 3\}$ (e.g., number of monk-monk nominations) and let's have one ERGM term: the sum of values of all relations: $\sum_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}$:

$$\Pr_{h,\boldsymbol{g}}(\boldsymbol{Y} = \boldsymbol{y}; \boldsymbol{\theta}) \propto h(\boldsymbol{y}) \exp\left(\boldsymbol{\theta} \sum_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}\right).$$

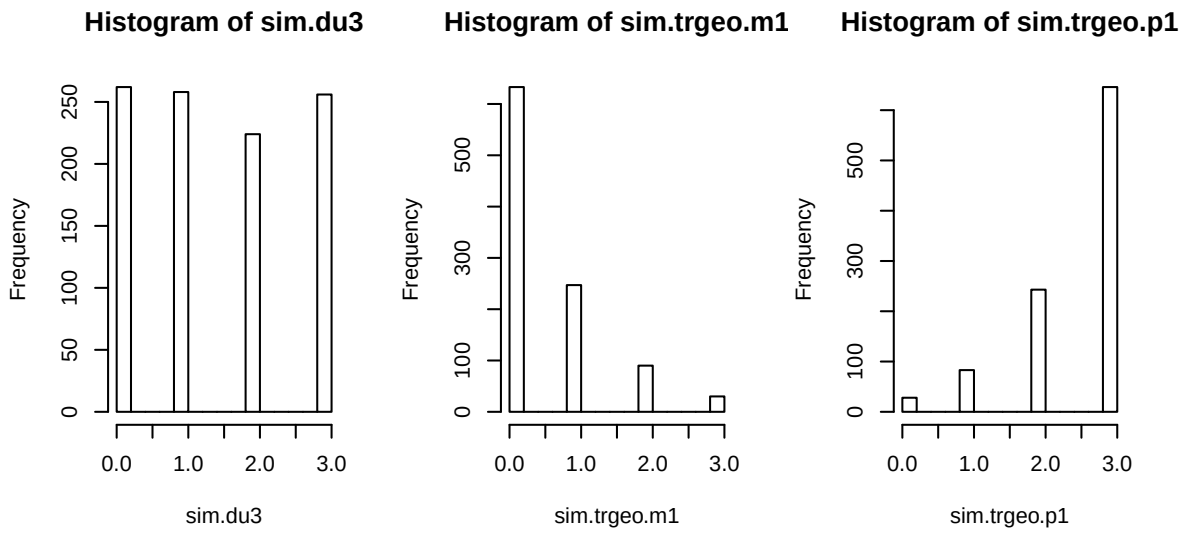
There are two values for $h(\boldsymbol{y})$ that might be familiar:

$h(\boldsymbol{y}) = 1$ **(or any constant)** $\implies \boldsymbol{Y}_{i,j} \overset{\text{i.i.d.}}{\sim}$ Uniform or truncated geometric

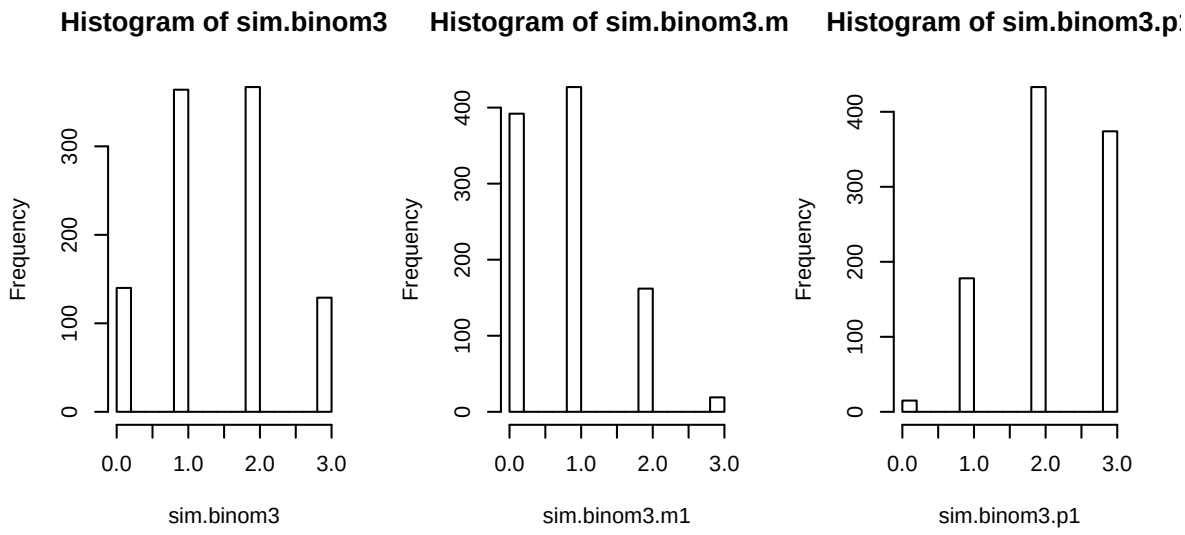
$h(\boldsymbol{y}) = \binom{m}{\boldsymbol{y}_{i,j}} = \frac{m!}{\boldsymbol{y}_{i,j}!(m-\boldsymbol{y}_{i,j})!} \implies \boldsymbol{Y}_{i,j} \overset{\text{i.i.d.}}{\sim} \text{Binomial}(m, \text{logit}^{-1}(\boldsymbol{\theta}))$

What do they look like? Let's simulate!

```
> y <- network.initialize(2,directed=FALSE) # A network with one dyad!
> ## Discrete Uniform reference
> # 0 coefficient: discrete uniform
> sim.du3<-simulate(y~sum, coef=0, reference=~DiscUnif(0,3),
                    response="w",statsonly=TRUE,nsim=1000)
> # Negative coefficient: truncated geometric skewed to the right
> sim.trgeo.m1<-simulate(y~sum, coef=-1, reference=~DiscUnif(0,3),
                         response="w",statsonly=TRUE,nsim=1000)
> # Positive coefficient: truncated geometric skewed to the left
> sim.trgeo.p1<-simulate(y~sum, coef=+1, reference=~DiscUnif(0,3),
                        response="w",statsonly=TRUE,nsim=1000)
> # Plot them:
> par(mfrow=c(1,3))
> hist(sim.du3,breaks=diff(range(sim.du3))*4)
> hist(sim.trgeo.m1,breaks=diff(range(sim.trgeo.m1))*4)
> hist(sim.trgeo.p1,breaks=diff(range(sim.trgeo.p1))*4)
```




```
> ## Binomial reference
> # 0 coefficient: Binomial(3,1/2)
> sim.binom3<-simulate(y~sum, coef=0, reference=~Binomial(3),
                       response="w",statsonly=TRUE,nsim=1000)
> # -1 coefficient: Binomial(3, exp(-1)/(1+exp(-1)))
> sim.binom3.m1<-simulate(y~sum, coef=-1, reference=~Binomial(3),
                          response="w",statsonly=TRUE,nsim=1000)
> # +1 coefficient: Binomial(3, exp(1)/(1+exp(1)))
> sim.binom3.p1<-simulate(y~sum, coef=+1, reference=~Binomial(3),
                          response="w",statsonly=TRUE,nsim=1000)
> # Plot them:
> par(mfrow=c(1,3))
> hist(sim.binom3,breaks=diff(range(sim.binom3))*4)
> hist(sim.binom3.m1,breaks=diff(range(sim.binom3.m1))*4)
> hist(sim.binom3.p1,breaks=diff(range(sim.binom3.p1))*4)
```



Now, suppose that we don't have an *a priori* upper bound on the counts — $\mathbb{S} = \{0, 1, \dots\}$ — then there are two familiar reference distributions:

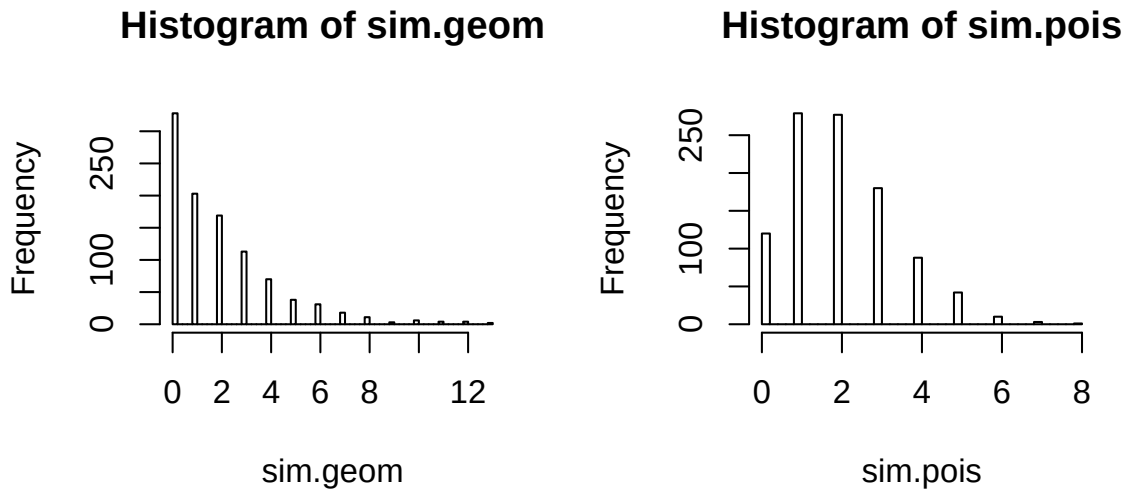
$h(\boldsymbol{y}) = 1$ **(or any constant)** $\implies \boldsymbol{Y}_{i,j} \overset{\text{i.i.d.}}{\sim} \text{Geometric}(p = 1 - \exp(\boldsymbol{\theta}))$

$h(\boldsymbol{y}) = 1/\prod_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}! \implies \boldsymbol{Y}_{i,j} \overset{\text{i.i.d.}}{\sim} \text{Poisson}(\mu = \exp(\boldsymbol{\theta}))$

```
> sim.geom<-simulate(y~sum, coef=log(2/3), reference=~Geometric,
                     response="w",statsonly=TRUE,nsim=1000)
> mean(sim.geom)

[1] 1.955

> sim.pois<-simulate(y~sum, coef=log(2), reference=~Poisson,
                     response="w",statsonly=TRUE,nsim=1000)
> mean(sim.pois)

[1] 2.024
```

Similar means. But, what do they look like?

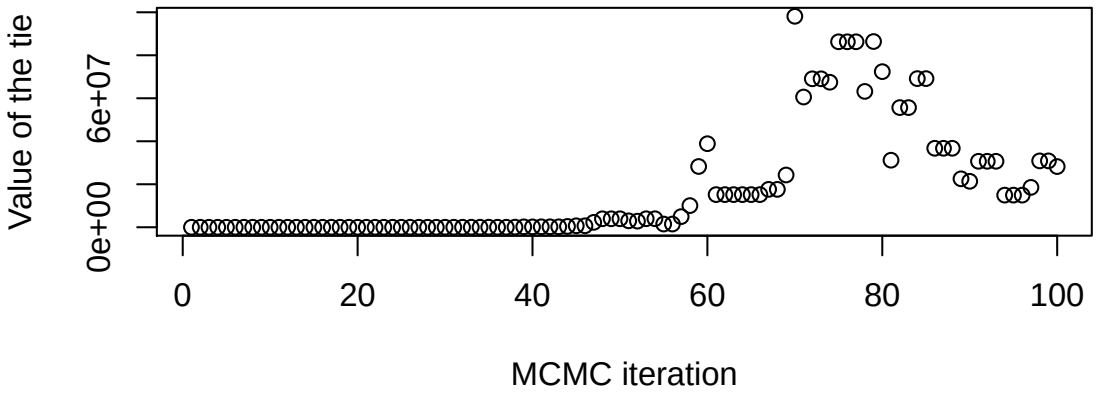
```
> par(mfrow=c(1,2))
> hist(sim.geom,breaks=diff(range(sim.geom))*4)
> hist(sim.pois,breaks=diff(range(sim.pois))*4)
```



Where did `log(2)` and `log(2/3)` come from? Later.

Warning: Parameter space What happens if we simulate from a geometric-reference ERGM with all coefficients set to 0?

```
> par(mfrow=c(1,1))
> sim.geo0<-simulate(y~sum, coef=0, reference=~Geometric,
                     response="w",statsonly=TRUE,nsim=100,
                     control=control.simulate(MCMC.burnin=0,MCMC.interval=1))
> mean(sim.geo0)

[1] 17925807

> plot(c(sim.geo0),xlab="MCMC iteration",ylab="Value of the tie")
```



Why does it do that? Because

$$\Pr_{h,\boldsymbol{g}}(\boldsymbol{Y}=\boldsymbol{y};\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}\sum_{(i,j)\in\mathbb{Y}}\boldsymbol{y}_{i,j})}{c_{h,\boldsymbol{g}}(\boldsymbol{\theta})}$$

for $\boldsymbol{\theta} \geq 0$, is not a valid distribution, because $c_{h,\boldsymbol{g}}(\boldsymbol{\theta}) = \infty$. Using `reference=~Geometric` can be dangerous for this reason. This issue only arises with ERGMs that have an infinite sample space.

3.2 Initial values

Before we can start fitting these models, we need a place to start. Unfortunately, there is no general MPLE for these models that we have (yet), so there are two options:

$\vec{0}$ This is the default. It can work reasonably well for some models, but it can also bog down for larger networks.

A closed-form submodel If the model fit has a simpler submodel that we can estimate exactly, we can use that as a starting point. That is, we find a coefficient for the submodel, and then start the main fit with all other coefficients set to 0.

Just as almost ever application of binary ERGMs uses the `edges` term as the baseline, almost every model we are likely to use will have the baseline intensity (`sum`) term. Given a network `y` with attribute `"a"` to be used as the response, the following should give adequate starting coefficients for `sum` for:

`Bernoulli:` Binary ERGM. Use MPLE.

```
Binomial(trials):
     p <- sum(y %e% "a")/trials/network.dyadcount(y)
     init.sum <- log(p/(1-p))
```

```
Poisson:
     m <- sum(y %e% "a")/network.dyadcount(y)
     init.sum <- log(m)

Geometric:
     m <- sum(y %e% "a")/network.dyadcount(y)
     init.sum <- log(1-1/(m+1))
```

This will probably be automated in the next version.

3.2.1 Example: Sampson's Monks with Binomial(3) reference

Suppose our model of interest contains the term `sum`, and we are using the `Binomial(3)` reference distribution. Then, plugging in `samplk.tot` for `y` and `"nominations"` for `"a"`,

```
> p <- sum(samplk.tot %e% "nominations")/3/network.dyadcount(samplk.tot)
> p
[1] 0.1830065
> samplk.sum.init <- log(p/(1-p)) # i.e., logit(p)
> samplk.sum.init
[1] -1.496109
```

3.2.2 Example: Zachary's Karate Club with Poisson reference

Suppose our model of interest contains the term `sum`, and we are using the `Poisson` reference distribution. Then, plugging in `zach` for `y` and `"contexts"` for `"a"`,

```
> m <- sum(zach %e% "contexts")/network.dyadcount(zach)
> m
[1] 0.4117647
> zach.sum.init <- log(m)
> zach.sum.init
[1] -0.8873032
```

We will be using these in the following examples.

3.3 ERGM terms

3.3.1 GLM-style terms

Many of the familiar ERGM effects can be modeled using the very same terms, but applied a little differently.

Any dyad-independent binary ERGM statistic can be expressed as $\boldsymbol{g}_{\mathrm{k}} = \sum_{(i,j)\in\mathbb{Y}} \boldsymbol{x}_{k,i,j}\boldsymbol{y}_{i,j}$ for some covariate matrix $\boldsymbol{x}_k$. If $\boldsymbol{y}_{i,j}$ is allowed to have values other than 0 and 1, then simply using such a term in a Poisson-reference ERGM creates the familiar log-linear effect. Similarly, in a Binomial-reference ERGM, such terms produce an effect on log-odds of a success.

The good news is that almost every dyad-independent `ergm` term has been reimplemented to allow this. It is invoked by specifying "`form="sum"`" argument for one of the terms inherited from binary ERGMs, though this not required, as it's the default. Also, note that for valued ERGMs, the "intercept" term is `sum`, not `edges`.

```
> help("ergm-terms")
```

has the complete list.

Example: Sampson's Monks For example, we can fit the equivalent of logistic regression on the probability of nomination, with every ordered pair of monks observed 3 times. We will look at differential homophily on group. That is, $\boldsymbol{Y}_{i,j} \overset{\text{ind.}}{\sim} \text{Binomial}(3, \boldsymbol{\pi}_{i,j})$ where

$$\begin{aligned}\text{logit}(\boldsymbol{\pi}_{i,j}) = \boldsymbol{\beta}_1 &+ \boldsymbol{\beta}_2 \mathbb{I}\left(i \text{ and } j \text{ are both in the Loyal Opposition}\right) \\ &+ \boldsymbol{\beta}_3 \mathbb{I}\left(i \text{ and } j \text{ are both Outcasts}\right) + \boldsymbol{\beta}_4 \mathbb{I}\left(i \text{ and } j \text{ are both Young Turks}\right) \\ &+ \boldsymbol{\beta}_5 \mathbb{I}\left(i \text{ and } j \text{ are both Waverers}\right)\end{aligned}$$

To actually fit it, we need to specify the initial parameters. How many do we need?

```
> npar <- length(summary(samplk.tot~sum + nodematch("group",diff=TRUE,form="sum"),
                         response = "nominations"))
> npar
[1] 5
```

```
> samplk.tot.nm <-
    ergm(samplk.tot~sum + nodematch("group",diff=TRUE,form="sum"),
         response="nominations", reference=~Binomial(3),
         control=control.ergm(init=c(samplk.sum.init, rep(0, npar-1))))
> mcmc.diagnostics(samplk.tot.nm)
```

```
> summary(samplk.tot.nm)

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

Formula:   samplk.tot ~ sum + nodematch("group", diff = TRUE, form = "sum")

Iterations:  20
```

```
Monte Carlo MLE Results:
                             Estimate Std. Error MCMC % p-value
sum                           -2.3253     0.1346      0  <1e-04 ***
nodematch.sum.group.Loyal      2.2553     0.2940      0  <1e-04 ***
nodematch.sum.group.Outcasts   3.5661     0.5886      0  <1e-04 ***
nodematch.sum.group.Turks      2.1999     0.2202      0  <1e-04 ***
nodematch.sum.group.Waverers   1.0732     0.5813      0  0.0658 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance:    0.0  on 306  degrees of freedom
 Residual Deviance: -555.5  on 301  degrees of freedom

Note that the null model likelihood and deviance are defined to be 0.

AIC: -545.5    BIC: -526.8    (Smaller is better.)
```

Based on this, we can say that the odds of a monk nominating another monk not in the same group during a given time step are $\exp(\boldsymbol{\beta}_1) = \exp(-2.3253) = 0.0978$, that the odds of a Loyal Opposition monk nominating another Loyal Opposition monk are $\exp(\boldsymbol{\beta}_2) = \exp(2.2553) = 9.5379$ times higher, etc..

Example: Zachary's Karate Club We will use a Poisson log-linear model for the number of contexts in which each pair of individuals interacted, as a function of whether this individual is a faction leader (Mr. Hi or John A.) That is, $\boldsymbol{Y}_{i,j} \overset{\text{ind.}}{\sim} \text{Poisson}(\boldsymbol{\mu}_{i,j})$ where

$$\log(\boldsymbol{\mu}_{i,j}) = \boldsymbol{\beta}_1 + \boldsymbol{\beta}_2(\mathbb{I}\,(i \text{ is a faction leader}) + \mathbb{I}\,(j \text{ is a faction leader}))$$

We will do this by constructing a dummy variable, a vertex attribute `"leader"`:

```
> unique(zach %v% "role")
[1] "Instructor" "Member"     "President"

> # Vertex attr. "leader" is TRUE for Hi and John, FALSE for others.
> zach %v% "leader" <- zach %v% "role" != "Member"


> zach.lead <-
    ergm(zach~sum + nodefactor("leader"),
         response="contexts", reference=~Poisson,
         control=control.ergm(init=c(zach.sum.init, 0)))
> mcmc.diagnostics(zach.lead)


> summary(zach.lead)
```

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

Formula:   zach ~ sum + nodefactor("leader")

Iterations:  20

Monte Carlo MLE Results:
                           Estimate Std. Error MCMC % p-value
sum                         -1.2204     0.0799      0  <1e-04 ***
nodefactor.sum.leader.TRUE   1.4396     0.1176      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance:    0.0  on 561  degrees of freedom
 Residual Deviance: -349.9  on 559  degrees of freedom

Note that the null model likelihood and deviance are defined to be 0.

AIC: -345.9    BIC: -337.2    (Smaller is better.)
```

Based on this, we can say that the expected number of contexts of interaction between two non-leaders is $\exp(\boldsymbol{\beta}_1) = \exp(-1.2204) = 0.2951$, that the expected number of contexts of interaction between a leader and a non-leader is $\exp(\boldsymbol{\beta}_2) = \exp(1.4396) = 4.2192$ times higher, and that the expected number of contexts of interaction between the two leaders is $\exp(2\boldsymbol{\beta}_2) = \exp(2 \cdot 1.4396) = 17.8016$ times higher than that between two non-leaders. (Because the leaders were hostile to each other, this may not be a very good prediction.)

3.3.2 Sparsity and zero-modification

It is often the case that in networks of counts, the network is sparse, yet if two actors do interact, their interaction count is relatively high. This amounts to zero-inflation.

We can model this using the binary-ERGM-based terms with the term `nonzero` ($\boldsymbol{g}_{\mathrm{k}} = \sum_{(i,j)\in\mathbb{Y}} \mathbb{I}\left(\boldsymbol{y}_{i,j} \neq 0\right)$) and GLM-style terms with argument `form="nonzero"`: $\boldsymbol{g}_{\mathrm{k}} = \sum_{(i,j)\in\mathbb{Y}} \boldsymbol{x}_{k,i,j} \mathbb{I}\left(\boldsymbol{y}_{i,j} \neq 0\right)$. For example,

```
> samplk.tot.nm.nz <-
    ergm(samplk.tot~sum + nonzero + nodematch("group",diff=TRUE,form="sum"),
         response="nominations", reference=~Binomial(3),
         control=control.ergm(init=c(samplk.sum.init,0, 0,0,0,0)))
> mcmc.diagnostics(samplk.tot.nm.nz)


> summary(samplk.tot.nm.nz)
```

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

Formula:   samplk.tot ~ sum + nonzero + nodematch("group", diff = TRUE,
    form = "sum")

Iterations:  20

Monte Carlo MLE Results:
                             Estimate Std. Error MCMC % p-value
sum                           -0.3334     0.1968      0  0.0912 .
nonzero                       -2.9730     0.3220      1  <1e-04 ***
nodematch.sum.group.Loyal      1.2191     0.2220      0  <1e-04 ***
nodematch.sum.group.Outcasts   1.9709     0.4837      0  <1e-04 ***
nodematch.sum.group.Turks      1.1895     0.1779      0  <1e-04 ***
nodematch.sum.group.Waverers   0.5998     0.4195      1  0.1538
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance:    0.0  on 306  degrees of freedom
 Residual Deviance: -646.3  on 300  degrees of freedom

Note that the null model likelihood and deviance are defined to be 0.

AIC: -634.3    BIC: -612    (Smaller is better.)
```

fits a zero-modified Binomial model, with a coefficient on the number of non-zero relations -2.973 is negative and highly significant, indicating that there is an excess of zeros in the data relative to the binomial distribution, and given the rest of the model.

3.3.3 Dispersion

Similarly, even if we may use Poisson as a starting distribution, the counts might be overdispersed or underdispersed relative to it. For now, `ergm` offers two ways to do so:

Conway–Maxwell–Poisson

- Implemented by adding a `CMP` term to a Poisson- or geometric-reference ERGM.
- Effectively replaces the "$1/y!$" part of a Poisson density with "$1/(y!)^{\boldsymbol{\theta}_{\text{CMP}}}$".

\+ Produces a continuum between a geometric distribution and a Bernoulli distribution.

\+ Can represent both over- and under-dispersion.

\− Has the parameter space problem; also, has some theoretical issues.

Fractional moments

- Implemented by adding a `sum(pow=1/2)` term to a Poisson-reference ERGM.
- Adds a statistic of the form $\sum_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}^{1/2}$ to the model.

\+ More stable.

\+ For Poisson-like data, $\sqrt{\boldsymbol{y}_{i,j}}$ is a variance-stabilizing transformation, so it could be interpreted as modeling (along with `sum` the first two moments of $\sqrt{\boldsymbol{y}_{i,j}}$.

– Not well-understood.

– In extreme cases, creates a bimodal shape in the counts.

3.3.4 Mutuality

`ergm` binary `mutuality` statistic has the form $\boldsymbol{g}_{\leftrightarrow} = \sum_{(i,j)\in\mathbb{Y}} \boldsymbol{y}_{i,j}\boldsymbol{y}_{j,i}$. It turns out that directly plugging counts into that statistic is a bad idea. `mutuality(form)` is a valued ERGM term, permitting the following generalizations:

`"geometric"`: $\sum_{(i,j)\in\mathbb{Y}} \sqrt{\boldsymbol{y}_{i,j}\boldsymbol{y}_{j,i}}$ — can be viewed as uncentered covariance of variance-stabilized counts

`"min"`: $\sum_{(i,j)\in\mathbb{Y}} \min \boldsymbol{y}_{i,j}, \boldsymbol{y}_{j,i}$ — easiest to interpret

`"nabsdiff"`: $\sum_{(i,j)\in\mathbb{Y}} -|\boldsymbol{y}_{i,j} - \boldsymbol{y}_{j,i}|$

Figure 1 visualizes their effects.

3.3.5 Individual heterogeneity

Different actors may have different overall propensities to interact. This has been modeled using random effects (as in the p_2 model and using degeneracy-prone terms like k-star counts.

`ergm` implements a number of statistics to model it, but the one that seems to work best so far seems to be

$$\boldsymbol{g}_{\text{actor cov.}}(\boldsymbol{y}) = \sum_{i\in N} \frac{1}{n-2} \sum_{j,k\in\mathbb{Y}_i\wedge j<k} (\sqrt{\boldsymbol{y}_{i,j}} - \overline{\sqrt{\boldsymbol{y}}})(\sqrt{\boldsymbol{y}_{i,k}} - \overline{\sqrt{\boldsymbol{y}}}),$$

essentially a measure of covariance between the $\sqrt{\boldsymbol{y}_{i,j}}$s incident on the same actor. The term `nodesqrtcovar` implements it.

3.3.6 Triadic closure

Finally, to generalize the notion of triadic closure, `ergm` implements very flexible `transitiveweights(twopath, combine, affect)` and similar `cyclicalweights` statistics. The transitive weight statistic has the following general form:

$$\boldsymbol{g}_{\boldsymbol{v}}(\boldsymbol{y}) = \sum_{(i,j)\in\mathbb{Y}} v_{\text{affect}}\left(\boldsymbol{y}_{i,j}, v_{\text{combine}}\left(v_{\text{2-path}}(\boldsymbol{y}_{i,k}, \boldsymbol{y}_{k,j})_{k\in N\setminus\{i,j\}}\right)\right),$$

and can be "customized" by varying the three functions

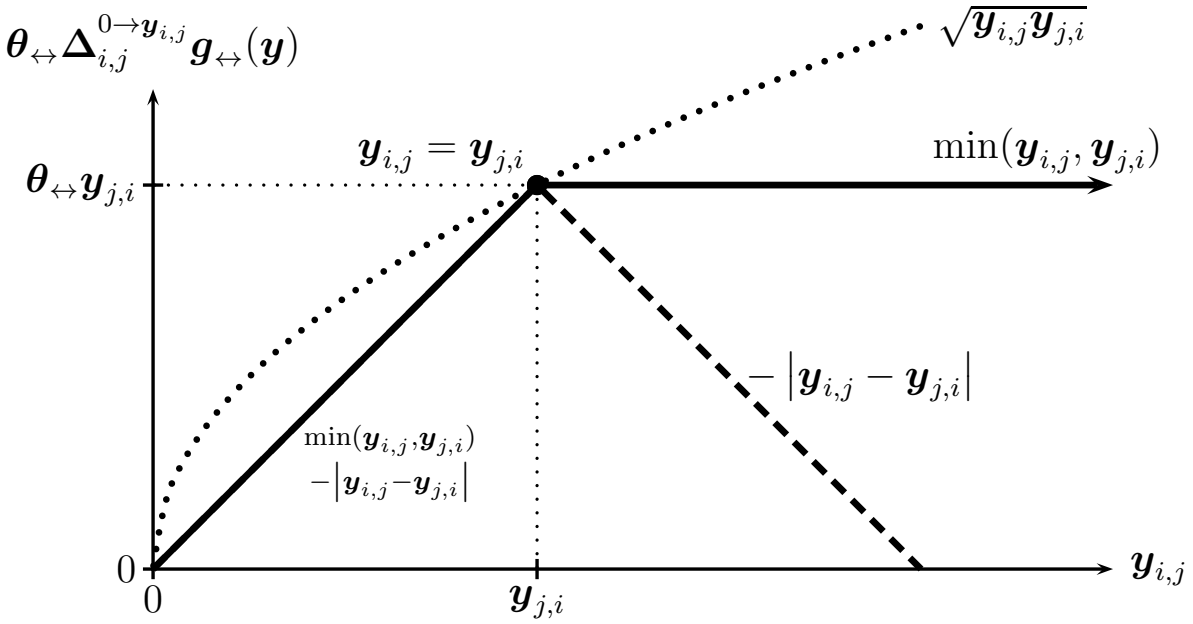


Figure 1: Effect of several mutuality forms on the probability of $\boldsymbol{Y}_{i,j}$ having a certain value given a particular $\boldsymbol{y}_{j,i}$.

$v_{\textbf{2-path}}$ Given $\boldsymbol{y}_{i,k}$ and $\boldsymbol{y}_{k,j}$, what is the strength of the two-path they form?

- `"min"` the minimum of their values — conservative
- `"geomean"` their geometric mean — more able to detect effects, but more likely to cause "degeneracy"

$v_{\textbf{combine}}$ Given the strengths of the two-paths $\boldsymbol{y}_{i\to k\to j}$ for all $k \neq i, j$, what is the combined strength of these two-paths between i and j?

- `"max"` the strength of the strongest path — conservative; analogous to `transitiveties`
- `"sum"` the sum of path strength — more able to detect effects, but more likely to cause "degeneracy"; analogous to `triangles`

$v_{\textbf{affect}}$ Given the combined strength of the two-paths between i and j, how should they affect $\boldsymbol{Y}_{i,j}$?

- `"min"` conservative
- `"geomean"` more able to detect effects, but more likely to cause "degeneracy"

These effects are analogous to mutuality.

3.4 Examples

3.4.1 Sampson's Monks

Suppose that we want to fit a model with a zero-modified Binomial baseline, mutuality, transitive (hierarchical) triads, and cyclical (antihierarchical) triads, to this dataset. Such a

model has 5 parameters, so

```
> theta0 <- c(samplk.sum.init, 0, 0, 0 ,0)
> samplk.tot.ergm <-
    ergm(samplk.tot ~ sum + nonzero + mutual("min") +
         transitiveweights("min","max","min") +
         cyclicalweights("min","max","min"),
         reference=~Binomial(3), response="nominations",
         control=control.ergm(init=theta0))
> mcmc.diagnostics(samplk.tot.ergm)

> summary(samplk.tot.ergm)

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

Formula:   samplk.tot ~ sum + nonzero + mutual("min") + transitiveweights("min",
    "max", "min") + cyclicalweights("min", "max", "min")

Iterations:  20

Monte Carlo MLE Results:
                              Estimate Std. Error MCMC % p-value
sum                           -0.04967    0.18854      2  0.7924
nonzero                       -3.45575    0.31520      4  <1e-04 ***
mutual.min                     1.38499    0.24720      3  <1e-04 ***
transitiveweights.min.max.min  0.21051    0.16318      0  0.1980
cyclicalweights.min.max.min   -0.24760    0.14486      1  0.0884 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance:    0  on 306  degrees of freedom
 Residual Deviance: -603  on 301  degrees of freedom

Note that the null model likelihood and deviance are defined to be 0.

AIC: -593    BIC: -574.4    (Smaller is better.)
```

What does it tell us? The negative coefficient on `nonzero` suggests zero-inflation, there is strong evidence of mutuality, and the positive coefficient on transitive weights and negative on the cyclical weights suggests hierarchy, but they are not significant.

3.4.2 Zachary's Karate Club

Now, let's try using Poisson to model the Zachary Karate Club data: a zero-modified Poisson, with potentially different levels of activity for the faction leaders, heterogeneity in actor

activity level overall, and an effect of difference in faction membership, a model that looks like this:

```
> summary(zach~sum+nonzero+nodefactor("leader")+absdiffcat("faction.id")+
          nodesqrtcovar(TRUE), response="contexts")
```

```
                          sum                       nonzero
                    231.00000                     78.00000
nodefactor.sum.leader.TRUE   absdiff.sum.faction.id.1
                     90.00000                     74.00000
  absdiff.sum.faction.id.2   absdiff.sum.faction.id.3
                     12.00000                     11.00000
  absdiff.sum.faction.id.4     nodesqrtcovar.centered
                     10.00000                     16.17248
```

The parameter vector will have length 8, so

```
> theta0 <- c(zach.sum.init, rep(0,7))
```

A few other notes:

- We can make sampling from zero-inflated Poisson more efficient by telling `ergm` that that's what we expect by setting `MCMC.prop.weights="0inflated"` control parameter. The current version has a bug that requires the degree of inflation to be set by the user, though this will be automatic in the next release, so `MCMC.prop.args=list(p0=0.5)` is also needed.
- Informally, a Poisson random variable contains more "information" than a Bernoulli random variable, which means that large changes in likelihood are not necessarily symptomatic of a problem. Thus, it often helps to set `MCMLE.trustregion`, which is normally 20 to something higher.

Now, for the fit and the diagnostics:

```
> zach.pois <-
    ergm(zach~sum+nonzero+nodefactor("leader")+absdiffcat("faction.id")+
         nodesqrtcovar(TRUE),
         response="contexts", reference=~Poisson,
         control=control.ergm(MCMLE.trustregion=100,
           init=theta0,
           MCMC.prop.weights="0inflated",
           MCMC.prop.args=list(p0=0.5) # Should not be necessary in the next version.
           ), verbose=TRUE)
> mcmc.diagnostics(zach.pois)


> summary(zach.pois)
```

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

Formula:   zach ~ sum + nonzero + nodefactor("leader") + absdiffcat("faction.id") +
    nodesqrtcovar(TRUE)

Iterations:  20

Monte Carlo MLE Results:
                           Estimate Std. Error MCMC % p-value
sum                         0.99512    0.09250      7 < 1e-04 ***
nonzero                    -3.85145    0.27512     11 < 1e-04 ***
nodefactor.sum.leader.TRUE  0.20062    0.06633      1 0.00261 **
absdiff.sum.faction.id.1   -0.10837    0.07810      0 0.16582
absdiff.sum.faction.id.2   -0.66743    0.19232      1 0.00056 ***
absdiff.sum.faction.id.3   -0.88254    0.21637      1 < 1e-04 ***
absdiff.sum.faction.id.4   -1.16067    0.23913      1 < 1e-04 ***
nodesqrtcovar.centered      1.81829    0.22266      1 < 1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance:    0.0  on 561  degrees of freedom
 Residual Deviance: -805.4  on 553  degrees of freedom

Note that the null model likelihood and deviance are defined to be 0.

AIC: -789.4    BIC: -754.7    (Smaller is better.)
```

What does it tell us? The negative coefficient on `nonzero` suggests zero-inflation, the faction leaders clearly have more activity than others, and the more ideologically separated two individuals are, the less they interact. Over and above that, there is some additional heterogeneity in how active individuals are: if i has a lot of interaction with j, it is likely that i has more with j'. Could this mean a form of preferential attachment?

We can try seeing whether there is some friend of a friend effect above and beyond that. This can be done by fitting a model with transitivity and seeing whether the coefficient is significant, or we can perform a simulation test. In the following

- `simulate` unpacks the `zach.pois` ERGM fit, extracting the formula, the coefficient, and the rest of the information.
- `nsim` says how many networks to generate.
- `statsonly=TRUE` says that we only want to see the simulated statistics, not the networks.
- `monitor=~transitiveweights("geomean","sum","geomean")` says that in addition to the statistics used in the fit, we want `simulate` to keep track of the transitive weights statistic.

We do not need to worry about degeneracy in this case, because we are not actually using that statistic in the model, only "monitoring" it.

```
> # Simulate from model fit:
> zach.sim <-
    simulate(zach.pois, monitor=~transitiveweights("geomean","sum","geomean"),
             nsim = 1000, statsonly=TRUE,
             control=control.simulate.ergm(
               MCMC.prop.weights="0inflated",
               MCMC.prop.args=list(p0=0.5) # Should not be necessary in the next version.
               ))
```

```
> # What have we simulated?
> colnames(zach.sim)
```

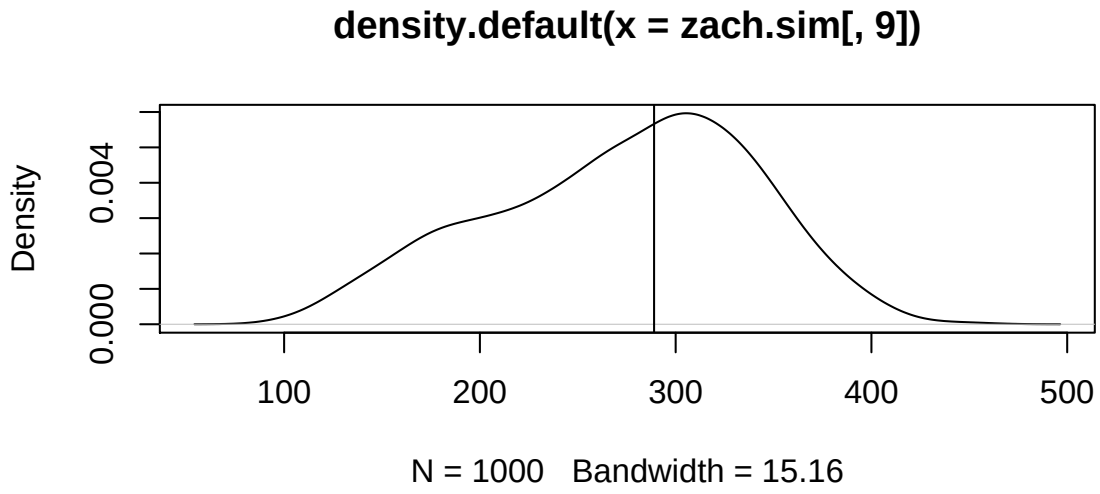
```
[1] "sum"
[2] "nonzero"
[3] "nodefactor.sum.leader.TRUE"
[4] "absdiff.sum.faction.id.1"
[5] "absdiff.sum.faction.id.2"
[6] "absdiff.sum.faction.id.3"
[7] "absdiff.sum.faction.id.4"
[8] "nodesqrtcovar.centered"
[9] "transitiveweights.geomean.sum.geomean"
```

```
> # How high is the transitiveweights statistic in the observed network?
> zach.obs <- summary(zach ~ transitiveweights("geomean","sum","geomean"),
                      response="contexts")
> zach.obs
```

```
transitiveweights.geomean.sum.geomean
                            288.9793
```

Let's plot the density of the simulated values of transitive weights statistic:

```
> par(mar=c(5, 4, 4, 2) + 0.1)
> # 9th col. = transitiveweights
> plot(density(zach.sim[,9]))
> abline(v=zach.obs)
```




```
> # Where does the observed value lie in the simulated?
> # This is a p-value for the Monte-Carlo test:
> min(mean(zach.sim[,9]>zach.obs), mean(zach.sim[,9]<zach.obs))*2
```

```
[1] 0.916
```

Looks like individual heterogeneity and faction alignment account for appearance of triadic effects. (Notably, the factions themselves may be endogenous, if social influence is a factor. Untangling selection from influence is hard enough when dynamic network data are available. We cannot do it here.)

3.5 Other notes

- Missing (`NA`) edges are handled automatically for valued ERGMs as well.
- `ergm` has an argument `eval.loglik`, which is `TRUE` by default. For valued ERGMs, it's quite a bit less efficient than for binary, at least for now. So, unless you need the AICs or BICs to compare models, and especially if your networks are not small, pass `eval.loglik=FALSE`.
- Writing user terms is possible, but the API is a little different from that of binary change statistics API.

3.6 Medium-term Roadmap

- A public release for rank ERGM implementation.
- Automatic starting values for valued ERGMs.
- More efficient sampling for valued ERGMs.

- Goodness of fit diagnostics.
- Other reference distributions, including ranks and continuous data. (Continuous uniform is already implemented, but not well-understood.)

4 Latent space models with non-binary response with `latentnet`

`latentnet`, as the name suggests, is designed to fit latent space models, but it can fit other dyad-independent network models as well. Let

$\boldsymbol{Y}$ be the random network being modeled;

$\boldsymbol{y}$ be the observed network;

$\boldsymbol{x}$ is a $p \times n \times n$ array of dyadic covariates, with

$\boldsymbol{x}_{\cdot,i,j}$ being a p-vector of covariates for dyad (i, j);

$\boldsymbol{\beta}$ be the p-vector of covariate coefficients;

$\boldsymbol{Z}$ be the $n \times d$ array of latent positions, with

$\boldsymbol{Z}_i$ being the d-vector position of actor i;

$\boldsymbol{\delta}$ be the n-vector of sender effects, with

$\boldsymbol{\delta}_i$ being the sender effect of $\boldsymbol{\delta}$; and

$\boldsymbol{\gamma}$ being a n-vector of receiver effects, with

$\boldsymbol{\gamma}_i$ being the receiver effect of $\boldsymbol{\gamma}$.

For brevity, let $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{Z}, \boldsymbol{\delta}, \boldsymbol{\gamma})$ and let $\boldsymbol{\theta}_{i,j} = (\boldsymbol{\beta}, \boldsymbol{Z}_i, \boldsymbol{Z}_j, \boldsymbol{\delta}_i, \boldsymbol{\gamma}_j)$. Generally, a latent space model that can be fit by `latentnet` has the following form:

$$\Pr(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) = \prod_{(i,j)\in\mathbb{Y}} \Pr(\boldsymbol{Y}_{i,j} = \boldsymbol{y}_{i,j}|\boldsymbol{\theta}_{i,j}, \boldsymbol{x}_{\cdot,i,j}), \tag{1}$$

$$\Pr(\boldsymbol{Y}_{i,j} = \boldsymbol{y}_{i,j}|\boldsymbol{\theta}_{i,j}, \boldsymbol{x}_{\cdot,i,j}) = f(\boldsymbol{y}_{i,j}|\boldsymbol{\mu}_{i,j}), \tag{2}$$

$$\boldsymbol{\mu}_{i,j} = g^{-1}(\boldsymbol{\eta}_{i,j}), \tag{3}$$

$$\boldsymbol{\eta}_{i,j} = \boldsymbol{x}_{\cdot,i,j}^{\top}\boldsymbol{\beta} + \mathrm{d}(\boldsymbol{Z}_i, \boldsymbol{Z}_j) + \boldsymbol{\delta}_i + \boldsymbol{\gamma}_j, \tag{4}$$

for some latent position effect $\mathrm{d}(\cdot, \cdot)$. Effects supported by `latentnet` are Euclidean, having $\mathrm{d}(\boldsymbol{Z}_i, \boldsymbol{Z}_j) = -|\boldsymbol{Z}_i - \boldsymbol{Z}_j|$ and bilinear, having $\mathrm{d}(\boldsymbol{Z}_i, \boldsymbol{Z}_j) = \boldsymbol{Z}_i^{\top}\boldsymbol{Z}_j$. Latent space positions $\boldsymbol{Z}$ can further be modeled as a Gaussian mixture, and $\boldsymbol{\delta}$ and $\boldsymbol{\gamma}$ are likewise modeled as random effects.

In words, the values of individual relations are assumed to be independent (1), and each potential relation has a value modeled by a GLM, having some distribution with density f,

parametrized by its expected value (2), which is, in turn, a function, via a GLM link function g, of the linear predictor $\boldsymbol{\eta}_{i,j}$ (3), which is a linear function combining all the parameters (4).

A latent space model for a binary network has

$$f(\boldsymbol{y}_{i,j}|\boldsymbol{\mu}_{i,j}) = (\boldsymbol{\mu}_{i,j})^{\boldsymbol{y}_{i,j}}(1-\boldsymbol{\mu}_{i,j})^{1-\boldsymbol{y}_{i,j}}$$
$$g(\boldsymbol{\mu}_{i,j}) = \text{logit}(\boldsymbol{\mu}_{i,j}),$$

a Bernoulli model with a logit link: a logistic regression model.

This is easily extended to any GLM. In particular,

Poisson log-linear model:

$$f(\boldsymbol{y}_{i,j}|\boldsymbol{\mu}_{i,j}) = \exp\left(-\boldsymbol{\mu}_{i,j}\right)\boldsymbol{\mu}_{i,j}^{\boldsymbol{y}_{i,j}}/\boldsymbol{y}_{i,j}!$$
$$g(\boldsymbol{\mu}_{i,j}) = \log(\boldsymbol{\mu}_{i,j})$$

Binomial logit with m trials:

$$f(\boldsymbol{y}_{i,j}|\boldsymbol{\mu}_{i,j}) = \binom{m}{\boldsymbol{y}_{i,j}}(\boldsymbol{\mu}_{i,j}/m)^{\boldsymbol{y}_{i,j}}(1-\boldsymbol{\mu}_{i,j}/m)^{m-\boldsymbol{y}_{i,j}}$$
$$g(\boldsymbol{\mu}_{i,j}) = \text{logit}(\boldsymbol{\mu}_{i,j})$$

Normal linear with variance σ^2:

$$f(\boldsymbol{y}_{i,j}|\boldsymbol{\mu}_{i,j}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\boldsymbol{y}_{i,j}-\boldsymbol{\mu}_{i,j})^2}{2\sigma^2}\right)$$
$$g(\boldsymbol{\mu}_{i,j}) = \boldsymbol{\mu}_{i,j}$$

These are currently supported by `latentnet`. Unfortunately, σ^2 cannot be estimated by `latentnet` at this time.

4.1 A very quick overview of `latentnet`

The workhorse function `ergmm` (for Exponential Random Graph *Mixed* Model) has a syntax very similar to `ergm`: the model is specified as `network ~ term1 + term2 + ...` and some of the *dyad-independent* `ergm` terms still work. (We are working on replicating the full set of `ergm` terms.) For a full list of terms implemented, see `? terms.ergmm`. Note that `ergmm`, like `glm` and unlike `ergm`, sets the first covariate $\boldsymbol{x}_{1,i,j} \equiv 1$ for an intercept term, the (binary) ERGM equivalent of an edge count term.

For example,

```
> samplk.nm.l <- ergmm(samplk.tot~nodematch("group",diff=TRUE),tofit="mle", verbose=TRUE)
```

sets $\boldsymbol{x}_{1,i,j} \equiv 1$, $\boldsymbol{x}_{2,i,j} \equiv \mathbb{I}\,(i \in \text{Loyal} \wedge j \in \text{Loyal})$, $\boldsymbol{x}_{3,i,j} \equiv \mathbb{I}\,(i \in \text{Outcasts} \wedge j \in \text{Outcasts})$, etc., to produce a model of the form

$$\text{logit}(\Pr(\boldsymbol{Y}_{i,j}=1)) = \boldsymbol{\beta}_1 + \boldsymbol{\beta}_2\mathbb{I}\,(i \text{ and } j \text{ are both Loyal Opposition}) + \boldsymbol{\beta}_3\mathbb{I}\,(i \text{ and } j \text{ are both Outcasts}) + \cdots,$$

equivalent to a `ergm` specification of

```
> samplk.nm.e <- ergm(samplk.tot~edges+nodematch("group",diff=TRUE))
```

In fact, the produce the same coefficients and standard errors:

```
> summary(samplk.nm.l, point.est="mle", se=TRUE)
==========================
Summary of model fit
==========================

Formula:   samplk.tot ~ nodematch("group", diff = TRUE)
Attribute: edges
Model:     Bernoulli
MCMC sample of size 4000, draws are 10 iterations apart, after burnin of 10000 iterations.
Covariate coefficients MLE:
                         Estimate Std. Error z value  Pr(>|z|)
edges                    -1.66208    0.17932 -9.2689 < 2.2e-16 ***
nodematch.group.Loyal     2.06755    0.49040  4.2161 2.486e-05 ***
nodematch.group.Outcasts 17.05930  900.30202  0.0189   0.98488
nodematch.group.Turks     2.57837    0.38577  6.6836 2.331e-11 ***
nodematch.group.Waverers  1.66208    0.83596  1.9882   0.04678 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> summary(samplk.nm.e)
==========================
Summary of model fit
==========================

Formula:   samplk.tot ~ edges + nodematch("group", diff = TRUE)

Iterations:  20

Monte Carlo MLE Results:
                         Estimate Std. Error MCMC % p-value
edges                     -1.6621     0.1793     NA  <1e-04 ***
nodematch.group.Loyal      2.0675     0.4904     NA  <1e-04 ***
nodematch.group.Outcasts  18.3450  1038.5246     NA  0.9859
nodematch.group.Turks      2.5784     0.3858     NA  <1e-04 ***
nodematch.group.Waverers   1.6621     0.8360     NA  0.0477 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 424.2  on 306  degrees of freedom
 Residual Deviance: 289.1  on 301  degrees of freedom

AIC: 299.1    BIC: 317.7    (Smaller is better.)
```

`latentnet` fits latent space models using terms `euclidean` and `bilinear`, which take two main arguments: `d` for dimension and `G` for number of clusters to use. So, for a classic example of the Sampson's monks fit, i.e.,

$$\text{logit}(\Pr(\boldsymbol{Y}_{i,j} = 1)) = \boldsymbol{\beta}_1 + |\boldsymbol{Z}_i - \boldsymbol{Z}_j|,$$

with $\boldsymbol{Z}_i$ modeled as 3 spherical Gaussian clusters with 2 dimensions,

```
> samplk.d2G3<-ergmm(samplk.tot~euclidean(d=2,G=3), verbose=TRUE)
```

Random actor-specific effects can also be used, with terms `rsender`, `receiver`, and `rsociality`, respectively: a receiver effects model of the form

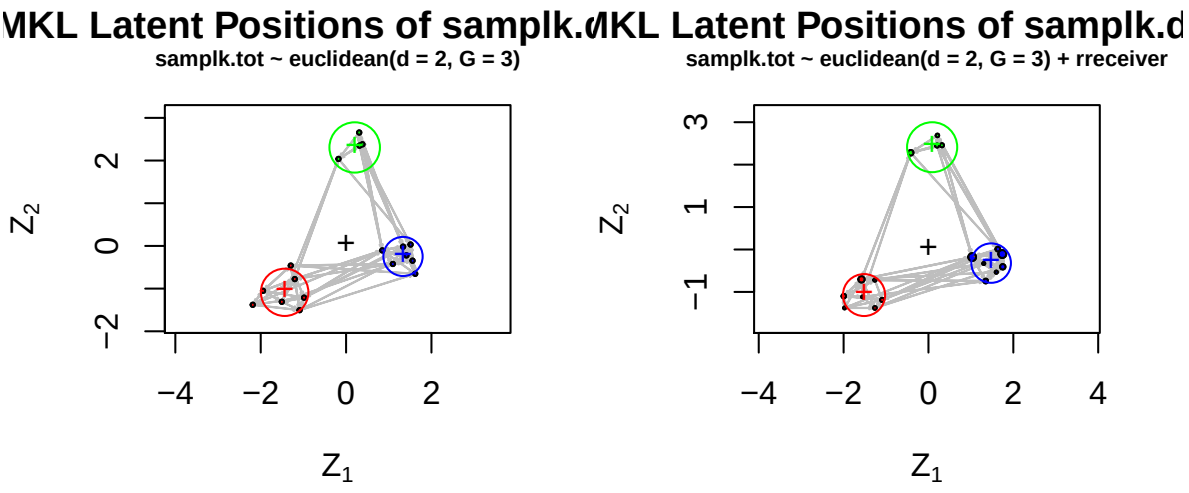
$$\text{logit}(\Pr(\boldsymbol{Y}_{i,j} = 1)) = \boldsymbol{\beta}_1 + |\boldsymbol{Z}_i - \boldsymbol{Z}_j| + \boldsymbol{\gamma}_j,$$

can be fit with

```
> samplk.d2G3r<-ergmm(samplk.tot~euclidean(d=2,G=3)+rreceiver, verbose=TRUE)
> mcmc.diagnostics(samplk.d2G3r)
```

The results can be visualized using

```
> par(mfrow=c(1,2))
> # Extract a clustering
> Z.K.ref <- summary(samplk.d2G3,point.est="pmean")$pmean$Z.K
> # Plot one model, saving positions, using Z.K.ref to set reference clustering.
> Z.ref <- plot(samplk.d2G3, pie=TRUE, Z.K.ref=Z.K.ref)
> # Plot the other model, using Z.ref and Z.K.ref to ensure similar
> # orientation and coloring.
> plot(samplk.d2G3r, rand.eff="receiver", pie=TRUE, Z.ref=Z.ref, Z.K.ref=Z.K.ref)
```



All `ergmm` terms also take additional arguments, specifying prior distributions. Their defaults are generally sensible, so we will not discuss them here.

Table 1: GLM families implemented by `latentnet`

Family	Link	`family=`	`fam.par=list(...)`
Bernoulli	logit	`"Bernoulli"`	
binomial	logit	`"binomial"`	`trials`$= m$
Poisson	log	`"Poisson"`	
Normal	linear	`"normal"`	`var`$= \sigma^2$

4.2 Specifying non-binary models in `ergmm`

Fitting non-binary models using `ergmm` requires three additional arguments:

`response` An edge attribute in the network whose values are to be used as the response.

`family` A string specifying the family and the link to be used.

`fam.par` A named list of parameters required by some families.

The families listed above are specified using parameter values listed in Table 1. Also, see

```
> ? families.ergmm
```

for more information.

We have our network of nomination counts, with a maximum of three "successes", so we might model it using a Binomial distribution, i.e.,

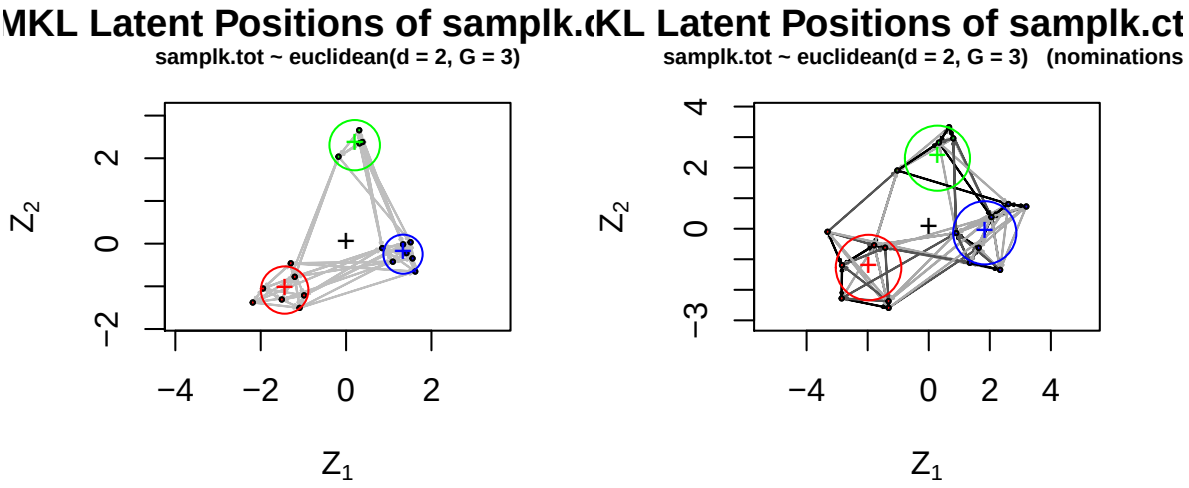
$$\Pr(\boldsymbol{Y}_{i,j} = \boldsymbol{y}_{i,j} | \boldsymbol{\eta}_{i,j}) = \binom{3}{\boldsymbol{y}_{i,j}} (\text{logit}^{-1}(\boldsymbol{\eta}_{i,j}))^{\boldsymbol{y}_{i,j}} (\text{logit}^{-1}(\boldsymbol{\eta}_{i,j}))^{3-\boldsymbol{y}_{i,j}},$$

with $\boldsymbol{\eta}_{i,j}$ being the same as in the binary case.

```
> # Bernoulli logit fit (recall)
> # samplk.d2G3 <- ergmm(samplk.tot~euclidean(d=2,G=3))
> # Binomial(trials=3) logit fit
> samplk.ct.d2G3 <-
    ergmm(samplk.tot~euclidean(d=2,G=3),response="nominations",
          family="binomial",fam.par=list(trials=3), verbose=TRUE)
```

We can plot the fit

```
> # Plot them side-by-side, using edge.col argument:
> par(mfrow=c(1,2))
> plot(samplk.d2G3,pie=TRUE, Z.ref=Z.ref, Z.K.ref=Z.K.ref)
> plot(samplk.ct.d2G3,pie=TRUE, Z.ref=Z.ref, Z.K.ref=Z.K.ref, edge.col=samplk.ecol)
```



Now, consider a latent cluster random effects model for the number of contexts of interactions, which takes into account that faction leaders are likely to have greater propensity to interact than non-leaders: $\boldsymbol{Y}_{i,j} \overset{\text{ind.}}{\sim} \text{Poisson}(\boldsymbol{\mu}_{i,j})$ with

$$\log \boldsymbol{\mu}_{i,j} = \boldsymbol{\eta}_{i,j} = \boldsymbol{\beta}_1 + \boldsymbol{\beta}_2(\mathbb{I}\,(i \text{ is a faction leader}) + \mathbb{I}\,(j \text{ is a faction leader})) + |\boldsymbol{Z}_i - \boldsymbol{Z}_j| + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j.$$

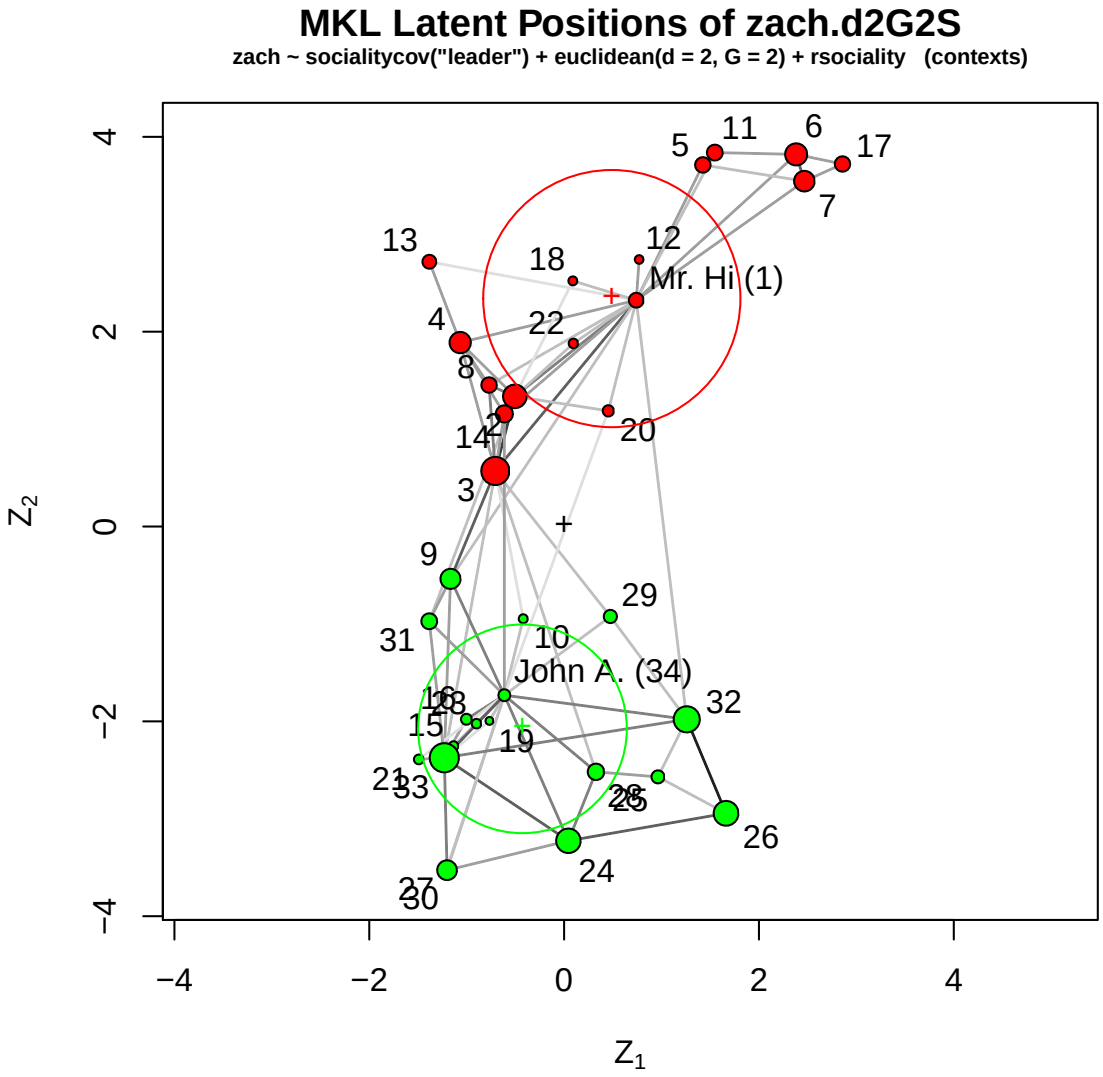
Some `ergmm` are missing that (for now), but the `"leader"` dummy variable can be used with `ergmm`'s `sendercov`, `receivercov`, and/or `socialitycov`.

The term `socialitycov(a)` sets $\boldsymbol{x}_{k,i,j} \equiv a_i + a_j$. Its `ergm` name is `nodecov`. Now, let's fit the model:

```
> zach.d2G2S <- ergmm(zach~socialitycov("leader")+euclidean(d=2,G=2)+rsociality,
                      response="contexts",family="Poisson",
                      verbose=TRUE)
> mcmc.diagnostics(zach.d2G2S)


> summary(zach.d2G2S)

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

Formula:   zach ~ socialitycov("leader") + euclidean(d = 2, G = 2) + rsociality
Attribute: contexts
Model:     Poisson
MCMC sample of size 4000, draws are 10 iterations apart, after burnin of 10000 iterations.
Covariate coefficients posterior means:
                        Estimate    2.5%  97.5% Quantile of 0
edges                    0.85346 0.18999 1.5176       0.00325 **
```

```
socialitycov.leader.TRUE  1.96461 0.70017 3.3625       0.00200 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Sociality effect variance: 0.7768003.
Overall BIC:        1149.352
Likelihood BIC:     801.9636
Latent space/clustering BIC:     267.818
Sociality effect BIC:     79.57072

Covariate coefficients MKL:
                         Estimate
edges                   0.3029993
sociality.leader.TRUE 2.0621726
```

```
> par(mar=c(5, 4, 4, 2) + 0.1)
> plot(zach.d2G2S, rand.eff="sociality", edge.col=zach.ecol, labels=TRUE)
```



Another useful application of `socialitycov` and others is when an undirected network of counts is a product of collapsing an affiliation network of actors to events to produce a network of actors only, counting the number of shared events for each pair of actors. If there is no particular pattern to the interaction, then the expected number of shared events between i and j would be proportional to the total number of events of i and to the number of events of j. If a_i is the total number of events associated with actor i, then using a vertex attribute equaling to $\log(a_i)$ in a `socialitycov` can be a good baseline model:

$$\log \boldsymbol{\mu}_{i,j} = \boldsymbol{\beta}_1 + \boldsymbol{\beta}_2(\log(a_i) + \log(a_j)) \Leftrightarrow \boldsymbol{\mu}_{i,j} = \exp(\boldsymbol{\beta}_1)\, a_i^{\boldsymbol{\beta}_2} a_j^{\boldsymbol{\beta}_2},$$

which produces proportionality when $\boldsymbol{\beta}_2 \approx 1$. For an application of this, see Krivitsky et˜al. (2009).

4.3 Medium-term Roadmap

- Transition `ergmm` to use dyad-independent `ergm` terms.
- Enable `latentnet` to fit heterogeneous trial counts.
- Enable `latentnet` to fit normal distribution variance.

References

Carter˜T. Butts. `network`: A package for managing relational data in R. *Journal of Statistical Software*, 24(2):1–36, May 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v24/i02`.

Carter˜T. Butts, Mark˜S. Handcock, and David˜R. Hunter. *`network`: Classes for Relational Data*. Irvine, CA, March 15, 2013. URL `http://statnet.org/`. R package version 1.7.2.

Steven˜M. Goodreau, Mark˜S. Handcock, David˜R. Hunter, Carter˜T. Butts, and Martina Morris. A `statnet` tutorial. *Journal of Statistical Software*, 24(9):1–26, May 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v24/i09`.

Mark˜S. Handcock, David˜R. Hunter, Carter˜T. Butts, Steven˜M. Goodreau, and Martina Morris. `statnet`: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11, May 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v24/i01`.

Mark˜S. Handcock, David˜R. Hunter, Carter˜T. Butts, Steven˜M. Goodreau, Pavel˜N. Krivitsky, and Martina Morris. *`ergm`: Fit, Simulate and Diagnose Exponential-Family Models for Networks*. The Statnet Project (`http://www.statnet.org`), 2013. URL `CRAN.R-project.org/package=ergm`. R package version 3.1-0.

David˜R. Hunter, Mark˜S. Handcock, Carter˜T. Butts, Steven˜M. Goodreau, and Martina Morris. `ergm`: A package to fit, simulate and diagnose exponential-family models for networks. *Journal of Statistical Software*, 24(3):1–29, May 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v24/i03`.

Pavel˜N. Krivitsky. Exponential-family random graph models for valued networks. *Electronic Journal of Statistics*, 6:1100–1128, 2012. doi: 10.1214/12-EJS696.

Pavel˜N. Krivitsky. *`ergm.count`: Fit, Simulate and Diagnose Exponential-Family Models for Networks with Count Edges*. The Statnet Project (`http://www.statnet.org`), 2013. URL `CRAN.R-project.org/package=ergm.count`. R package version 3.1-0.

Pavel˜N. Krivitsky and Mark˜S. Handcock. *`latentnet`: Latent position and cluster models for statistical networks*. The Statnet Project (`http://www.statnet.org`), 2013. URL `CRAN.R-project.org/package=latentnet`. R package version 2.4-4.

Pavel˜N. Krivitsky, Mark˜S. Handcock, Adrian˜E. Raftery, and Peter˜D. Hoff. Representing degree distributions, clustering, and homophily in social networks with latent cluster

random effects models. *Social Networks*, 31(3):204–213, July 2009. ISSN 0378-8733. doi: 10.1016/j.socnet.2009.04.001.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`.