A Short JAGS Tutorial
with an Interpersonal Perception/Social Network Example

Written for the Professional Development Workshop
Why We All Should Be Bayesians: Opportunities of Bayesian Statistics for Management Research
Academy of Management 2011 Annual Meeting, San Antonio

Miguel Sousa Lobo
miguel.lobo@insead.edu

August 11, 2011

1 Preliminaries

This tutorial uses JAGS and MATLAB on Mac OS X. JAGS is free and open-source (GNU GPL), and there are also versions of it for Windows and for different Linux flavors. While it also includes some extensions, JAGS is mostly an implementation of the BUGS language, so that most models can be interchangeably run in JAGS, WinBUGS, and OpenBUGS. I use MATLAB because it has arguably the best graphical visualization functions, but you can instead analyze the samples of your model's posterior distribution generated by JAGS with whatever statistical package you're comfortable with (including free open source packages such as R – with rjags you can even use JAGS directly from within R).

JAGS is written and maintained by Martyn Plummer at the WHO's International Agency for Research on Cancer, Infection and Cancer Epidemiology group. To download JAGS, go to:

`http://www-ice.iarc.fr/~martyn/software/jags/`

Follow the link to Sourceforge. Under 'Manuals' you can get the user manual. Under 'JAGS' you can get the software (version 3.0.0 has just been released and is not compiled for the Mac yet; I'm using version 2.2.0). Download the version appropriate for your system and follow the instructions. For the Mac OS X version, download the package, double click to start the installer, and follow the installer's instructions.

The files for the example in this tutorial is available from:

`http://www.sousalobo.com/aom2011pdw/`

This includes a couple of MATLAB functions I wrote that are used in this tutorial (`readcoda.m` and `autocorr.m`). If you're using MATLAB, you should also download `saveR.m` (by Jeroen Janssens) from:

`http://www.mathworks.com/matlabcentral/fileexchange/28370`

2 The example

Our example is a complete social network (simulated, with summary statistics chosen to approximate those of the empirical datasets in [2]). A group of n people were asked to rate each other in a survey item (let's say the question was 'I trust this person.') The responses are in a seven-point Likert-like scale, which are coded 0 to 6. Let's use i to index the source (or perceiver, observer, rater, ego); and j to index the target (or alter).

2.1 What effects shape perception?

We'll follow the approach of modeling the response as the sum of an item effect, a source effect, a target effect, and a relationship effect (see [1, 2]). Different theoretical questions relate to different effects. For instance, the relationship effect is relevant for questions of how relationships between two people are formed and evolve (for instance, to what extent does reciprocity matter?) Questions about performance and outcomes for individuals, on the other hand, relate to the target effect (for instance, are people seen as more trustworthy also seen as more competent?)

Define the following effects, or latent variables:

- an item effect, c,
- a source effect (or bias), a_i,
- a target effect (or score), b_j, and
- a relationship effect, e_{ij}.

These effects add up to a compound latent response,

$$y_{ij} = c + a_i + b_j + e_{ij}.$$

2.2 From perception to data

By assumption, the compound response is real-valued, continuous and unbounded. However, the observed response is on a discrete and bounded scale. For modeling purposes, we'll assume that the respondent selects the discrete response which is closest to the sum of the effects:

$$x_{ij} = \max\left(0, \min\left(6, \lfloor y_{ij} + 0.5 \rfloor\right)\right).$$

That is: if $y_{ij} < 0.5$ the observation will be $x_{ij} = 0$; if $0.5 \leq y_{ij} < 1.5$ the observation will be $x_{ij} = 1$; *etc.*

This model should not be seen as a description of how people respond in the sense of representing the cognitive processes of the survey respondents. It simply proposes a plausible statistical structure for the survey responses, which must be validated by an assessment of the model's fit to the data.

2.3 What is the distribution of the effects?

Let's assume for now that each of these effects is drawn from a normal distribution, and that they are all independent of each other, with one exception: each person's source and target effects can be correlated. We then have

$$\begin{bmatrix} a_i \\ b_i \end{bmatrix} \sim \mathcal{N}(0, \Sigma_{ab}), \quad \text{and} \quad e_{ij} \sim \mathcal{N}(0, \sigma_e^2).$$

The goal of the simulation is to generate samples of the posterior distributions of Σ_{ab} and σ_e given the observations, that is $f(\Sigma_{ab}|X)$ and $f(\sigma_e|X)$. These distributions summarize what we know about the distribution of the effects based on the available empirical evidence.

2.4 The next level up: Priors for the distributions

The question of what we know about Σ_{ab} and σ_e^2 after seeing the observations cannot be answered without also asking what we knew about them *before* seeing the observations. While every approach to statistical inference necessarily makes prior assumptions, in a Bayesian model these assumptions are explicitly stated. A common approach is to select a prior distribution which is 'noninformative' or 'weakly informative'. Simplifying things somewhat: within a range of values that covers all possible values of the parameter, no value is specified to be especially more likely than any other. Another way of checking whether the prior

is informative is to try different distributions thought to be noninformative and check whether the final estimates change.

Another concern is that the prior should be such that the posterior distribution is proper (that is, integrable). For more on choosing prior distributions for variance parameters in hierarchical models see [3]. We'll use an inverse-gamma prior distribution for the variance of the relationship effect,

$$\tau_e \sim \text{gamma}\,(1, 1)\,,$$

where $\tau_e = 1/\sigma_e^2$, and an inverse-Wishart prior distribution for the covariance matrix of the source and target effects,

$$\Omega_{ab} \sim \text{Wishart}\,(I, 2)\,,$$

where $\Omega_{ab} = \Sigma_{ab}^{-1}$. These choices can be checked to not have a significant impact on the estimates, and are made for computational convenience (sampling with conjugate priors is more efficient). Note that the prior for σ_e^2 excludes the possibility of the variance of the relationship effects being zero (a very safe assumption...) For models where the variance could be zero, another choice of prior would be better, again see [3].

3 Setting up and running the model in JAGS

3.1 The script file (`myscript.cmd`)

You can type commands directly into the JAGS terminal, but you'll make your life easier if you write a script file with all the commands you need and then tell JAGS to run that script. This allows you to debug your data analysis procedure, to avoid mistakes, and to have documentation for what you did.

Here's the script file we'll use:

```
model in mymodel.bug
data in mydata.r
compile, nchains(3)
inits in myinits.r
initialize
update 2000
monitor set absigma, thin(5)
monitor set esigma, thin(5)
update 5000
coda *, stem(mycoda)
```

A brief explanation of each command:

- The first two are self-explanatory, they tell JAGS where to look for the file with the specification of the model, and where to look for the file with the data.
- The command `compile` tells JAGS to parse the model. The model will be checked for errors, and get translated into code that runs the simulation of the posterior distributions of the model parameters. The `nchains` option tells JAGS to run three independent Markov chains in parallel.
- The `inits` command instructs JAGS to use values you specify in a file to initialize some of the parameters in the simulation chain (in many models you can omit this, but in some cases JAGS may have trouble guessing what are reasonable values to start from).
- The `initialize` command gets the simulation ready to go, with the number of trials count at zero.
- The `update` command runs a specified number of simulation trials (steps of the Markov chain). This first set of samples is for adaptation and burn-in. The first time the update command is issued after initialization, the sampler runs in adaptive mode, adjusting the jump distributions to increase efficiency. For later update commands, adaptation is turned off in order to obtain unbiased samples.

- The `monitor` command tells JAGS to save the value of a variable in subsequent trials. The `thin` option tells JAGS to only save the value of the parameters every n steps. The `update` command that follows will result in 3000 samples of the parameters of interest being saved, 1000 from each chain.
- The `coda` command causes a file to be written with the saved samples ($*$ to save all monitored variables).

3.2 The data file (`mydata.r`)

This file is in R format, so if you're using R it'll be straightforward to create. Some statistical packages have the option to save in R format. In MATLAB, I use `saveR` to create my data files. The data file for our example begins like this:

```
"N" <- 40
"abmu" <- structure(c(0.0, 0.0), .Dim = c(2, 1))
"abR" <- structure(c(1.0, 0.0, 0.0, 1.0), .Dim = c(2, 2))
"abk" <- 2
"yt" <- structure(c(0.5, 1.5, 2.5, 3.5, 4.5, 5.5), .Dim = c(1, 6))
"X" <- structure(c(NA, 4, 1, 6, 3, 2, 5, 2, 1, 1, 5, 1, 6, 2, 3, 3, ...
```

This defines:

- `N`, a scalar, the number of nodes in the network (the number of people who took the survey).
- `X`, an $N \times N$ matrix, the actual data, the tie values (as defined by the survey responses), coded from 0 to 6.
- `abmu`, the mean for the Wishart prior on Ω_{ab} (a vector of zeros).
- `abR`, the shape matrix for the Wishart prior.
- `abk`, the degrees of freedom for the Wishart prior.
- `yt`, a vector with six entries with the thresholds for the discretization, from 0.5 to 5.5 in unit increments.

`NA` stands for 'not available', and are given the treatment appropriate for missing data (in MATLAB the equivalent is `NaN`, or 'not a number'). In this example only the diagonal of X is missing.

3.3 The model file (`mymodel.bug`)

With the model specification in §2, and because of the equation $y_{ij} = c + a_i + b_j + e_{ij}$, the observations are a deterministic function of other variables. This is an issue when setting up the sampler because, given a set of values for, say, c, a_i, and b_i, the value of e_{ij} is fixed (and, hence, cannot be randomly sampled from its distribution conditional on the other variables, so that generating samples of the posterior cannot be done one variable at a time).

This can, however, be addressed with a simple transformation in how we specify the model. Standard BUGS is not clever enough (as of yet) to do this for us. JAGS can do it for us if, instead of the usual addition operator, we use the distribution `dsum`. However, `dsum` still has some limitations and is not available in standard BUGS, so let's see how to do the transformation explicitly. Instead of

$$y_{ij} = c + a_i + b_j + e_{ij}, \quad \text{and} \quad e_{ij} \sim \mathcal{N}\left(0, \sigma_e^2\right),$$

we use the equivalent formulation

$$y_{ij} \sim \mathcal{N}\left(\mu_{ij}, \sigma_e^2\right), \quad \text{and} \quad \mu_{ij} = c + a_i + b_j.$$

With this formulation, the relationship terms for each tie in the network e_{ij} aren't explicitly computed during the simulation, rather, they are implied from $y_{ij} - (c + a_i + b_j)$. With this modification, the model in §2 is specified as follows in JAGS.

```
var X[N,N], Y[N,N], mu[N,N], ab[2,N], c, abomega[2,2], etau, absigma[2,2], esigma;
model {
    for (i in 1:N) {
        for (j in 1:N) {
            X[i,j] ~ dinterval(Y[i,j],yt)
            Y[i,j] ~ dnorm(mu[i,j],etau)
            mu[i,j] <- ab[1,i]+ab[2,j]+c
        }
    }
    for (i in 1:N) {
        ab[,i] ~ dmnorm(abmu,abomega)
    }
    c ~ dnorm(0.0,0.01)
    abomega ~ dwish(abR,abk)
    etau ~ dgamma(1.0,1.0)
    absigma <- inverse(abomega)
    esigma <- 1.0/sqrt(etau)
}
```

Here's a brief description of the elements of the model:

- *Declarations:* `var` is used to declare the variables that are going to be updated by the sampler.
- *Logical nodes:* `<-` defines a deterministic relationship; when a variable on the right-hand side has been updated, the one on the left is recomputed with the given formula. (`<-` is also used to specify link functions, *e.g.*, `logit(y)<-x` or, equivalently, `y<-ilogit(x)`.)
- *Stochastic nodes:* `~` defines a stochastic relationship; this relationship impacts the sampling distribution of the variables on both sides of the definition.

We used the following distributions in our model:

- `dnorm(mu,tau)`, a normal distribution (the variance is specified by the precision parameter $\tau = 1/\sigma^2$).
- `dmnorm(mu,Omega)`, a multivariate normal distribution (the covariance matrix is specified by the precision matrix $\Omega = \Sigma^{-1}$).
- `dinterval(t,c)`, this JAGS-only (not in standard BUGS), and implements the discretization (which plays the role of a link function in our model); it's not really a distribution, but sets up the observed values as stochastic nodes as required for the sampler to work.
- `dgamma(r,mu)`, a gamma distribution (note that the scale parameter `mu` is as for an inverse-gamma, *i.e.*, specified as a precision).
- `dwish(R,k)`, a Wishart distribution (note that the scale matrix parameter `R` is as for an inverse-Wishart, *i.e.*, specified as a precision matrix).

3.4 The inits file (`myinits.r`)

The `inits` command instructs JAGS to use values you specify in a file to initialize some or all of the parameters in the simulation chain. In many models you can omit this, but in some cases JAGS may have trouble guessing what are reasonable values to start from. (Trouble arises if JAGS picks a starting point which is has extremely low, or even zero, probability given the observations, in which case the Markov chain

may take a very long time to reach the region of interest in the distribution where most of the probability lies, or it may run into numerical overflow or underflow in the calculations.)

In our example, things go much more smoothly if we initialize the y_{ij} (the latent rating, or compound of item, source, target, and relationship effects) to be identical to the corresponding x_{ij} (the observed survey response). This is because the JAGS extension `dinterval`, while modeling-wise useful and computationally efficient, seldom does a good job of setting up consistent initial values. I created the `myinits.r` file by copying the matrix X to a matrix Y and saving it in R format.

3.5 Running the model

Open a terminal window. To do this on the Mac's OS X, open a Finder window, go to Applications, open Utilities, open Terminal.

Go to the directory where your JAGS files are. On the terminal window you opened, use `cd` to navigate to the directory where you saved your model files. For reference, here are some useful terminal commands:

- `pwd` (print working directory), tells you where you are.
- `ls` (list), the see contents of the current working directory.
- `cd` (change directory), should be followed by the directory where you want to go, typically the name of a subdirectory, or `..` (two dots) to go to the parent directory.

There are a lot of other commands for working with files, deleting, moving, etc., but you'll probably find it much easier to do that in a Finder window.

Run your script. Type `jags myscript.cmd`, followed by enter. This will execute all the commands in your script. If you enter `jags` without including a script name, you'll enter the interactive mode. You can then enter JAGS commands one at a time. To exit back to the terminal prompt, enter `exit` at the JAGS prompt. (Note that the actual JAGS executable files are saved in `/usr/local`, a directory which is not viewable in Finder, but can be accessed via the terminal.)

4 Analyzing and visualizing the results in MATLAB

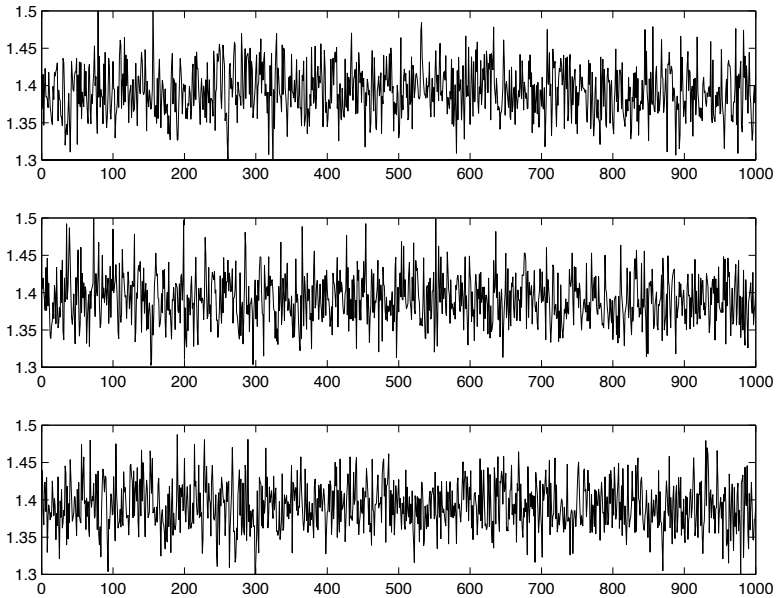
To load the simulation output into MATLAB, you can use the `readcoda` function. It returns the names of the monitored variables in `xname`, and the samples in `X` (the size of `X` is number of samples × number of chains × number of monitored variables). The `whos` function lists the variables present in the MATLAB workspace and their details. Entering the name of a variable in the MATLAB prompt displays its content.

```
>> clear all
>> [xname,X]=readcoda('mycoda');
>> whos
  Name          Size              Bytes  Class     Attributes

  X          1000x3x5             120000  double
  xname         5x1                  408  cell

>> xname

xname =

    'absigma[1,1]'
```

```
    'absigma[2,1]'
    'absigma[1,2]'
    'absigma[2,2]'
    'esigma'
```

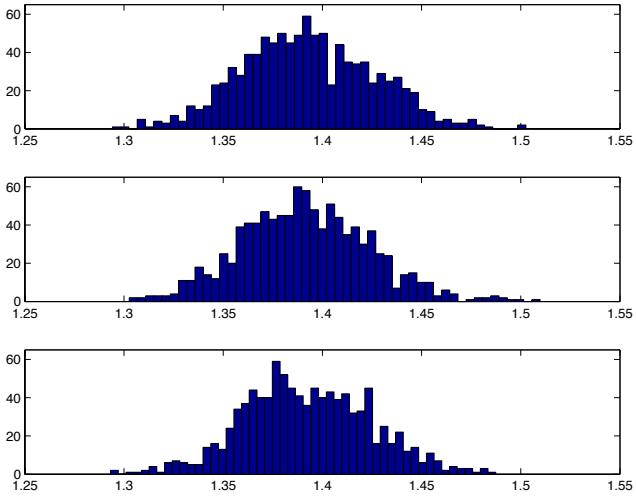
Let's plot the three chains of samples of σ_e:

```
>> for i=1:3, subplot(3,1,i), plot(X(:,i,5)), end
```



From visual inspection, the chains seem to have converged in the burn-in phase and to be well-mixed. Let's visualize the posterior distribution $f(\sigma_e|X)$ as estimated by each chain:

```
>> for i=1:3, subplot(3,1,i), hist(X(:,i,5)), end
```



The distributions are consistent across chains, as we'd expect if there are enough decorrelated samples in each chain. To confirm this, we can check the autocorrelation in the samples using the function `autocorr` (type `help autocorr` for details on its parameters):

```
>> autocorr(X(:,:,5),1,1:4)

ans =

    0.1686    0.1062    0.1039
    0.0600    0.0410    0.0166
   -0.0323   -0.0070    0.0091
   -0.0130   -0.0478   -0.0497
```

Each column in the results of `autocorr` is for one of the chains. The rows are the autocorrelation with a lag of 1, 2, 3, and 4 in the entries of `X` (which, on account of having thinned the chain by a factor of 5, correspond to lags of 5, 10, 15, and 20 samples). This confirms that there is little autocorrelation in the chains with a lag of 5 samples, and none with a lag of 10 or more.

Other checks of the reliability of the results involve comparing the sample distributions across the chains. Several more or less sophisticated tests are commonly used, here we'll simply check the correlation across chains (correlation would indicate that the burn-in phase wasn't long enough for the chains to reach the limiting distribution):

```
>> corrcoef(X(:,:,5))

ans =

    1.0000   -0.0134   -0.0717
   -0.0134    1.0000   -0.0148
   -0.0717   -0.0148    1.0000
```

We can compute an estimate of σ_e separately from each chain, which confirms that we're getting consistent results:

```
>> mean(X(:,:,5))

ans =

    1.3926    1.3915    1.3922
```

We can now pool the sample from all the chains:

```
>> m=size(X); XX=reshape(X,[m(1)*m(2),m(3)]);
```

Let's define a vector with the samples of the posterior distribution of each of the parameters of interest:

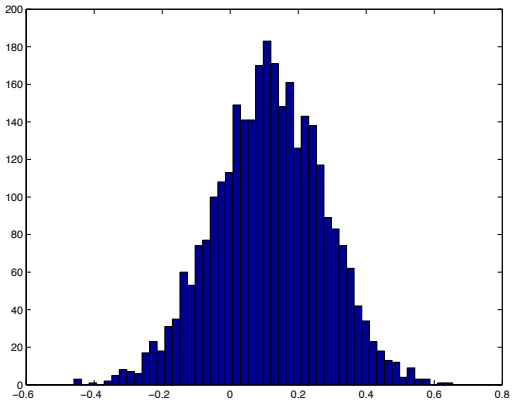
```
>> sigmaa=sqrt(XX(:,1));
>> sigmab=sqrt(XX(:,4));
>> rhoab=XX(:,2)./(sigmaa.*sigmab);
>> sigmae=XX(:,5);
```

We can now get estimates and 95% confidence intervals (sometimes called *credibility intervals*, to make their Bayesian nature explicit) for each parameter:

```
>> mean([sigmaa sigmab rhoab sigmae])

ans =

    0.9394    1.1877    0.1150    1.3921

>> prctile([sigmaa sigmab rhoab sigmae],[2.5 97.5])

ans =

    0.7450    0.9444   -0.2064    1.3297
    1.1854    1.5144    0.4250    1.4567
```

If we want to investigate whether those who are more trusting of others are also more likely to be more trusted by others, the posterior distribution of interest is $f(\rho_{ab}|X)$ (note that, for this to be a sound empirical investigation, we'd have to address the question of whether lower or higher source effects reflect a meaningful characteristic of survey respondents, rather than an inconsequential difference in how people used the survey scale). We can visualize this distribution:

```
>> hist(rhoab,50)
```



And compute the one-sided posterior probability $P(\rho_{ab} \leq 0|X)$ (the inequality operator returns a vector of zeros and ones for each sample, the mean of that vector is the fraction of ones):

```
>> mean(rhoab<=0)

ans =

    0.2343
```

5 Extensions

To illustrate the flexibility of Bayesian models and of BUGS model specifications, let's briefly consider some extensions of this model. If reciprocity is likely to be present, as is the case in most network data, we should

model correlation between the relationship effect from i to j and the relationship effect from j to i, which we have so far assumed to be independent of each other. There are two good reasons to model this correlation. First, we may be interested in estimating the degree of reciprocity. Second, even if reciprocity in itself is not of interest, controlling for it in the model will lead to better answers to other questions. In particular, if e_{ij} and e_{ji} are strongly correlated, the effective sample size can be reduced, so that if we don't control for reciprocity the posterior distribution for some parameters may be narrower than it should be.

The relationship effects in our model should then be drawn from a jointly normal distribution,

$$\begin{bmatrix} e_{ij} \\ e_{ji} \end{bmatrix} \sim \mathcal{N}(0, \Sigma_e).$$

Since e_{ij} and e_{ji} are drawn from the same distribution (this must be the case because they can be interchangeably defined), they must have the same variance. This means that the covariance matrix, in addition to being symmetric and positive definite (as all covariance matrices must be), must be skew-symmetric, that is

$$\Sigma_e = \begin{bmatrix} \sigma_e^2 & \rho\,\sigma_e^2 \\ \rho\,\sigma_e^2 & \sigma_e^2 \end{bmatrix}.$$

While there are ways of enforcing this in other simulation environments, in BUGS/JAGS we have the constraint in how we define the y_{ij} which led the e_{ij} to not be explicitly defined. One way around this to implement reciprocity in BUGS is as follows. With $y_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_e)$ for all (i,j), for each (i,j) such that $j > i$ define

$$\mu_{ij} = c + a_i + b_j + r_{ij}, \quad \mu_{ji} = c + a_i + b_j + r_{ij}, \quad \text{and} \quad r_{ij} \sim \mathcal{N}(0, \sigma_r)$$

(r_{ij} can be termed the 'reciprocity effect'). We can obtain estimates of the covariance of the relationship effects from

$$\Sigma_e = \begin{bmatrix} \sigma_e^2 + \sigma_r^2 & \sigma_r^2 \\ \sigma_r^2 & \sigma_e^2 + \sigma_r^2 \end{bmatrix}.$$

This approach assumes $\rho \geq 0$, which is an appropriate prior assumption for many social network data. If non-negative reciprocity is not an appropriate prior assumption, another variation can be used,

$$\begin{aligned} \mu_{ij} &= c + a_i + b_j + r_{ij} \\ \mu_{ji} &= c + a_i + b_j + (2p-1)\,r_{ij}, \end{aligned}$$

where p is a Bernoulli r.v. (with prior probability $P(p=1) = 0.5$). In BUGS this is:

```
for (i in 1:N) {
    for (j in i+1:N) {
        Y[i,j] ~ dnorm(mu[i,j],etau)
        Y[j,i] ~ dnorm(mu[j,i],etau)
        mu[i,j] <- ab[1,i]+ab[2,j]+c+r[i,j]
        mu[j,i] <- ab[1,j]+ab[2,i]+c+(2*p-1)*r[i,j]
        r[i,j] ~ dnorm(0,rtau)
    }
}
p ~ dbern(0.5)
```

The posterior probability $\mathrm{P}(p=1|X)$ (the fraction of simulation trials where the indicator p is one) tells us the strength of empirical evidence for positive reciprocity. We can also plot the histogram of the simulated posterior distribution of the correlation coefficient (without prior restriction on its sign) by computing for each sample $\rho = (2p-1)\,\sigma_r^2 / \left(\sigma_r^2 + \sigma_e^2\right)$.

Other important extensions and variations in this class of models include:

- Other link functions may be more appropriate for other kinds of data, for instance if the distribution of the values associated with the ties is long tailed.

- In social network data, higher-level network structure (beyond node-level and dyad-level effects) is usually present, most commonly block structure. This can be the consequence, for instance, of homophily associated with unmodeled node attributes or, consistent with balance theory, the outcome of a dynamic process of triadic closure. The type of models described here can be extended to account for block structure.

- In the case of multiple items or networks, the dependence structure between the relationship effects in the different networks is typically of especial interest. With an approach along the lines outlined here, we can investigate, for instance, wether ties in one network predict ties in another network after controlling for node characteristics, correlation in node characteristics across networks, reciprocity, and block structure.

Finally, a brief mention of alternative ways of investigating theoretical hypotheses in a Bayesian framework, still using as an example the question of whether there is significant reciprocity in ties:

- The reciprocity model can be extended to allow for probability mass at $\rho = 0$ by defining

$$\begin{aligned} \mu_{ij} &= c + a_i + b_j + q\, r_{ij} \\ \mu_{ji} &= c + a_i + b_j + q\,(2p-1)\, r_{ij}, \end{aligned}$$

  with q Bernoulli. The strength of the empirical evidence for positive reciprocity in the ties is the posterior probability $\mathrm{P}(q = 1 \text{ and } p = 1|X)$ (*i.e.*, the fraction of simulation trials where both indicators are one).

- Other approaches involve running separate models with and without reciprocity, and comparing the fit of each model to the data (in a way that accounts for the different number of parameters in each model).

References

[1] Kenny, D.A. (1994). *Interpersonal Perception: A Social Relations Analysis*, The Guilford Press.

[2] Casciaro, T. and Sousa Lobo, M. (2008). *When competence is irrelevant: The role of interpersonal affect in task-related ties* Administrative Science Quarterly, 53, 655–684.

[3] Gelman, A. (2006). *Prior distributions for variance parameters in hierarchical models*, Bayesian Analysis, 1, 515–533.