Getting Started With GERGM

NEWS

Version 0.13.0 brings some major updates to the package which dramatically increase the robustness and speed of the estimation procedure, as well as extend the functionality of the GERGM to simplex constrained networks, and larger networks. Note that these updates may cause the estimation procedure to yield slightly different results from the previous version of the package. In general, we have found these changes to improve model fit, but please email matthewjdenny@gmail.com if you encounter any issues. If you would like to approximate the behavior of the `gergm()` function before package version 0.13.0, you should set `use_previous_thetas = FALSE` and `convex_hull_proportion = NULL`.

Introduction

This vignette is designed to introduce you to the **GERGM** R package. **GERGM** stands for **G**eneralized **E**xponential **R**andom **G**raph **M**odel. This class of models was developed to characterize the structure of networks with real-valued edges. GERGMs represent a generalization of ERGMs, which were developed to model the structure of networks with binary edge values, and many network statistics commonly included in ERGM specifications have identical formulations in the weighted case. The relevant papers detailing the model can be found at the links below:

- Bruce A. Desmarais, and Skyler J. Cranmer, (2012). “Statistical inference for valued-edge networks: the generalized exponential random graph model”. PloS One. [Available Here]
- James D. Wilson, Matthew J. Denny, Shankar Bhamidi, Skyler Cranmer, and Bruce Desmarais (2017). “Stochastic weighted graphs: Flexible model specification and simulation”. Social Networks, 49, 37–47. [Available Here]
- Paul E. Stillman, James D. Wilson, Matthew J. Denny, Bruce A. Desmarais, Shankar Bhamidi, Skyler Cranmer, and Zhong-lin Lu. (2017). “Statistical Modeling of the Default Mode Brain Network Reveals a Segregated Highway Structure”. Scientific Reports, 7(11694), 1–14. [Available Here]
- Matthew J. Denny (2016). “The Importance of Generative Models for Assessing Network Structure”. [Available Here]

The GERGM also relies on the initialization method of Hummel et al. as of version 0.13.0. The paper corresponding to this method is available here:

- Ruth M. Hummel, David R. Hunter, and Mark S. Handcock (2012). “Improving simulation-based algorithms for fitting ERGMs”. Journal of Computational and Graphical Statistics, 21(4), 920–939. [Available Here]

Getting Started With GERGM

The easiest way to do this is to install the package from CRAN via the standard `install.packages` command:

```
install.packages("GERGM")
```

This will take care of some weird compilation issues that can arise, and is the best option for most people. If you want the most current development version of the package, you will need to start by making sure you have Hadley Wickham's devtools package installed.

If you want to get the latest version from GitHub, start by checking out the **Requirements for using C++ code with R** section in the following tutorial: Using C++ and R code Together with Rcpp. You will likely need to install either `Xcode` or `Rtools` depending on whether you are using a Mac or Windows machine before you can install the GERGM package via GitHub, since it makes use of C++ code to speed up inference. That said, the development version often has additional functionality not found in the CRAN release.

```
install.packages("devtools")
```

Now we can install from Github using the following line:

```
devtools::install_github("matthewjdenny/GERGM")
```

Once the `GERGM` package is installed, you may access its functionality as you would any other package by calling:

```
library(GERGM)
```

If all went well, check out the `vignette("getting_started")` which will pull up this vignette!

Basic Usage

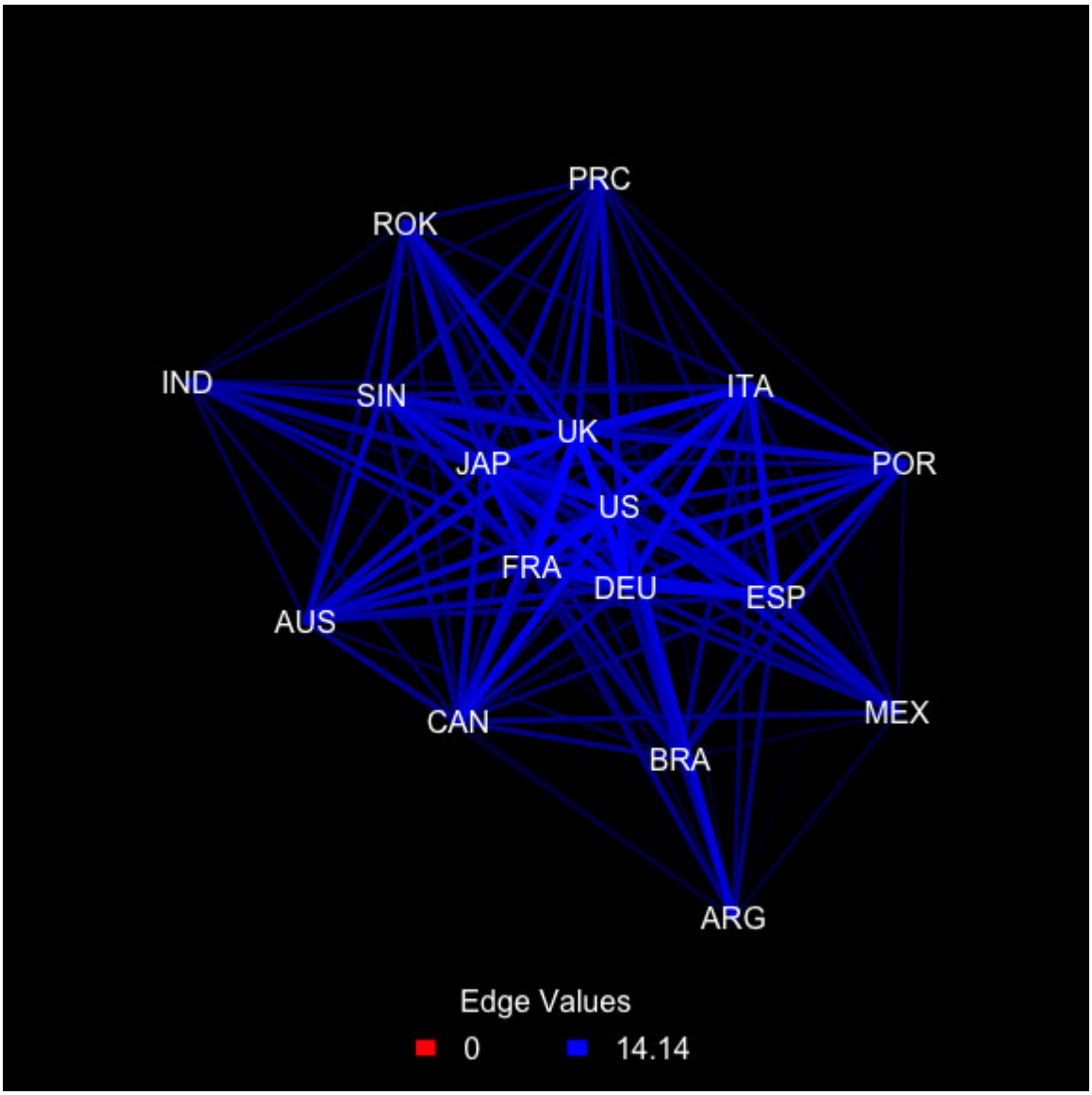
We begin by loading in some example network data. In our case, these data are (logged) aggregate public and private lending volumes between 17 large countries from 2005. The data are included in the GERGM package and were used in the Wilson et. al. study listed at the beginning

Getting Started With GERGM

countries in 2005. We will make use of this data in fitting our example GERGM model.

The GERGM package provides a `plot_network()` function, which we can use to visualize the network as follows:

```
library(GERGM)
set.seed(12345)
data("lending_2005")
data("covariate_data_2005")
data("net_exports_2005")
plot_network(lending_2005)
```



Alternatively, if we prefer a white background, and no legend, we can select options for this as well. Typing `?plot_network` into the console will pull up a manual for this function.

Getting Started With GERGM

accessed by typing `?gergm` into the console. We are going to focus on a simpler version of the application from the Wilson et. al. paper, that will highlight creating a formula object with node and network level covariates, as well as endogenous (network) effects. While this model will not provide a perfect fit to the data, it serves to illustrate a number of key concepts. If we look at the first couple of rows of the `covariate_data_2005` object, we can see that it include information about each country's log GDP and whether it was a member of the G8.

```
head(covariate_data_2005)
```

```
##              GDP  log_GDP  G8
## ARG 2.229108e+11 26.13004  No
## AUS 6.933386e+11 27.26478  No
## BRA 8.921068e+11 27.51685  No
## CAN 1.164144e+12 27.78301 Yes
## PRC 2.268594e+12 28.45018  No
## FRA 2.203679e+12 28.42115 Yes
```

To model this network, we are gong to include an `edges` term, which functions similarly to an intercept term in a regression model and parameterizes the density of the network. We are also going to include `sender` and `receiver` effects for a country's GDP. These parameters are designed to capture the effects of having a large economy on the amount of lending a borrowing a country does. We are also going to include a `nodemix` term to capture the propensity for members and non-members of the G8 to lend to each other, compared to the base case of non-G8 to non-G8 member lending. The last covariate effect we are going to include in the model is a `netcov`, or network covariate term, capturing the effect of the structure of the international trade network on the international lending network. Finally, we are going to include one endogenous statistic in the model, to capture the degree of reciprocal lending in the network. For this endogenous statistic, we are also going to include an exponential down-weight. this means that when the value of the network statistic is calculated, it will then be raised to the power of (in this case) 0.8. This will have the effect of reducing its value, but more importantly of smoothing out statistic values as the GERGM parameter controlling the propensity for mutual dyads in the network carries. Practically, this can make it easier to get starting values for the mutual dyads parameter that are in the right ball park, aiding in the estimation process. The formula object is defined below:

```
formula <- lending_2005 ~ edges + mutual(alpha = 0.8) + sender("log_GDP") +
  receiver("log_GDP") + nodemix("G8", base = "No") + netcov(net_exports_2005)
```

Getting Started With GERGM

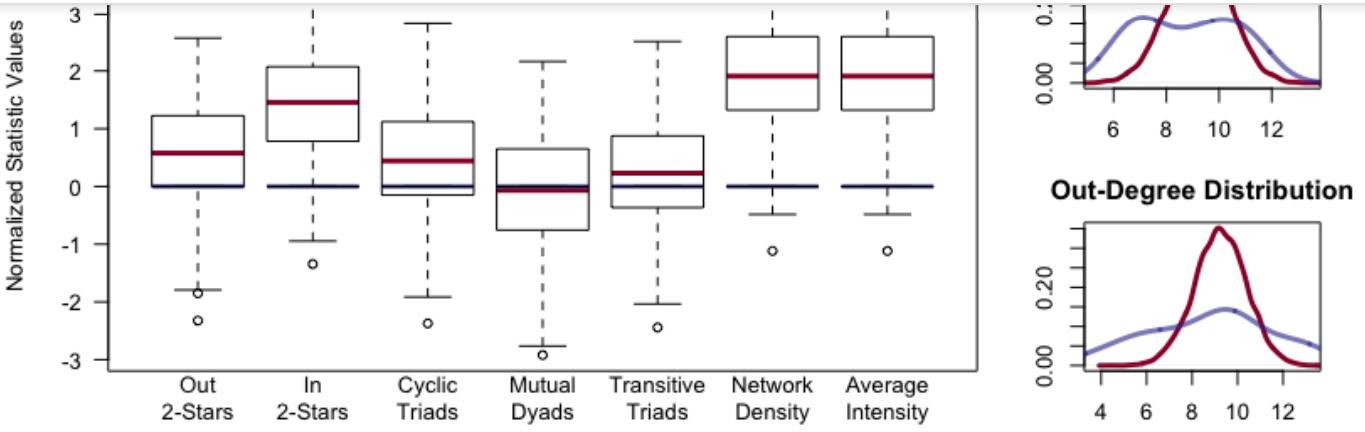
If you are familiar with ERGMs (for binary network data), you may have heard of an issue these models can run into called “degeneracy”, which can make certain models impossible to estimate. In this particular example, as with all GERGM specifications we have tried so far, the GERGM does not seem to suffer from this issue. However, GERGMs can still be difficult to estimate. This is primarily due to challenges in getting good starting values for our model parameters. The current implementation of the GERGM software does so using the method of Hummel et al. (2012), which does a pretty good job in most cases. However, in some cases, it can be enough off the mark that the initial parameter guesses from MPLE simulate networks that look a lot different from the observed network. This can cause the optimizer in R (which is used to update our estimates of the model parameters) to zoom off to infinity.

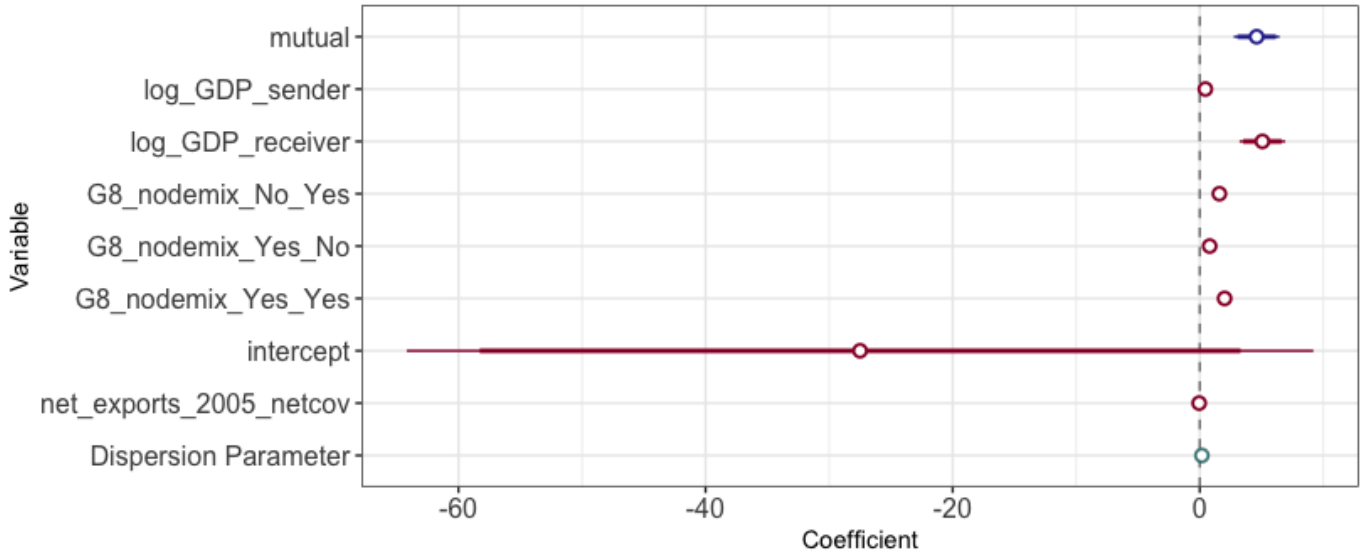
If this happens to you, do not (immediately) panic! This usually means you are dealing with a tricky network, or a tricky specification (typically one with lots of endogenous statistics included). The first thing to do is try to use alpha weighting. A good rule of thumb is to set `alpha = 0.8` for all of the endogenous statistics included in the model. Note that these currently include: `out2stars`, `in2stars`, `ctriads`, `mutual`, and `ttriads` (or just `twostars` and `ttriads` if your network is undirected). If this does not work, you can try cranking down the weights to around 0.5. If this still does not work, you will need to explore the `theta_grid_optimization_list` option in the `gergm` documentation, which should always work if given enough time (although this could be weeks, depending on how complex your model is). A fuller example is provided at the end of this vignette.

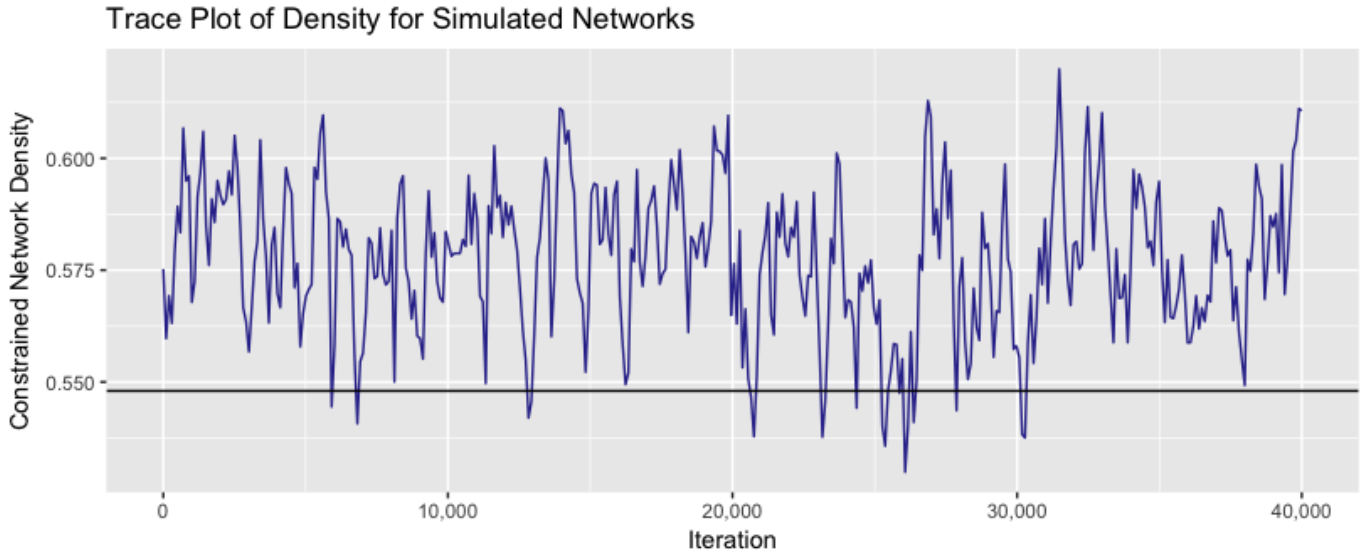
Having discussed the challenges that come with estimating a GERGM model, lets try an example!

```
test <- gergm(formula,
              covariate_data = covariate_data_2005,
                number_of_networks_to_simulate = 40000,
                thin = 1/100,
                proposal_variance = 0.05,
                MCMC_burnin = 10000,
                seed = 456,
                convergence_tolerance = 0.5)
```

Getting Started With GERGM







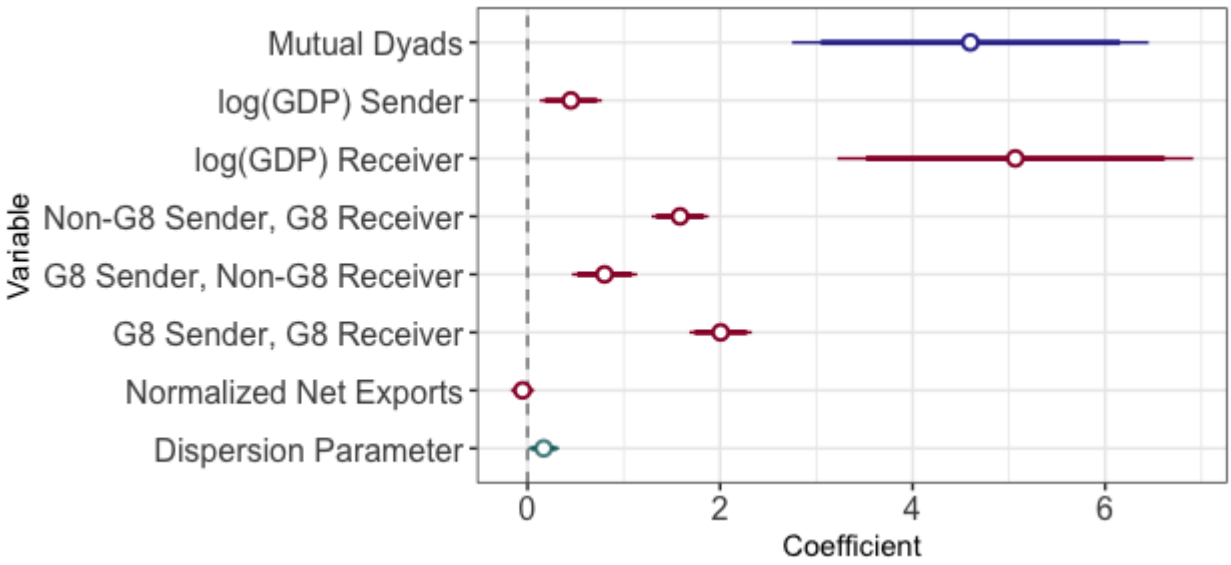
The output displayed in this vignette only includes diagnostic plots, and not all of the information that would be spit out by the `gergm()` function if you were to run this code on your computer. All of that output is meant to help you track the estimation process (which can take days or weeks for larger networks), and diagnose issues with the estimation. Note that if you wish to tweak some of

Getting Started With GERGM

```
# Generate Estimate Plot
Estimate_Plot(test)
# Generate GOF Plot
GOF(test)
# Generate Trace Plot
Trace_Plot(test)
```

In particular, we might want to make a nicer looking estimate plot. We can do this using the following block of code, where we leave out the intercept estimate, and provide a custom list of parameter names to produce a publication quality plot:

```
Estimate_Plot(test,
              coefficients_to_plot = "both",
              coefficient_names = c("Mutual Dyads",
                                    "log(GDP) Sender",
                                    "log(GDP) Receiver",
                                    "Non-G8 Sender, G8 Receiver",
                                    "G8 Sender, Non-G8 Receiver",
                                    "G8 Sender, G8 Receiver",
                                    "intercept",
                                    "Normalized Net Exports",
                                    "Dispersion Parameter"),
              leave_out_coefficients = "intercept")
```



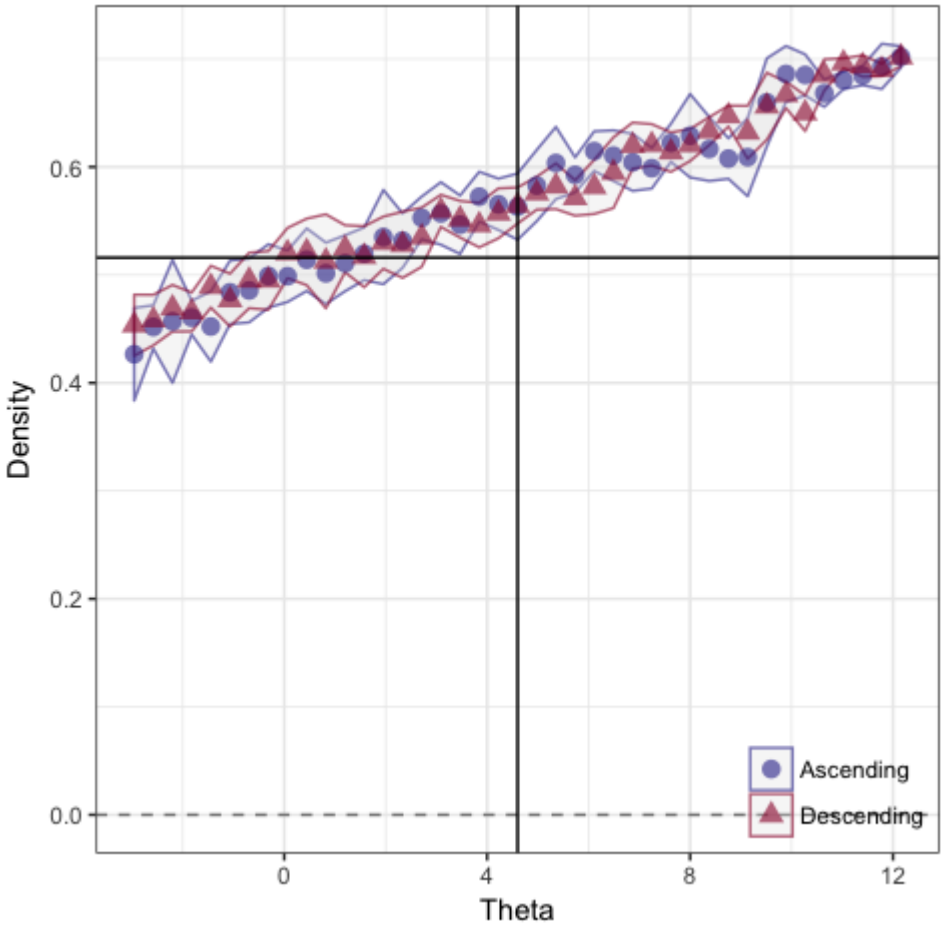
In order to verify the claim made earlier in this vignette that the current model is not degenerate, we can generate a hysteresis plot for this model using the `hysteresis()` function. This function simulates large numbers of networks at parameter values around the estimated parameter values

reference for details.

- Snijders, Tom AB, et al. “New specifications for exponential random graph models.” Sociological methodology 36.1 (2006): 99-153.

So long as we see a smooth upward sloping series of points, we have strong evidence that the specification is not degenerate.

```
# Generate Hysteresis plots for all structural parameter estimates
hysteresis_results <- hysteresis(test,
                                 networks_to_simulate = 1000,
                                 burnin = 300,
                                 range = 8,
                                 steps = 20,
                                 simulation_method = "Metropolis",
                                 proposal_variance = 0.05)
```



As we can see this specification does not display signs of degeneracy, even though we needed to use exponential down-weighting in order to fit the model.

Following on from the example above, we can also predict individual edge values, conditioning on the rest of the observed edges and estimated parameters. We can then calculate the mean edgewise mean squared error (MSE) for these predictions, and compare it against the MSE from a null model with no parameters included. First we generate the conditional edge predictions:

```
test2 <- conditional_edge_prediction(
  GERGM_Object = test,
  number_of_networks_to_simulate = 100,
  thin = 1,
  proposal_variance = 0.05,
  MCMC_burnin = 100,
  seed = 123)
```

Next we can calculate the MSE of these predictions and compare it to the null model predictions.

```
MSE_results <- conditional_edge_prediction_MSE(test2)
```

```
## Mean MSE for Predicted Edge Values: 2162.995
## Mean MSE for Max Ent Predicted Edge Values: 43107.43
## This represents a 94.98 percent reduction in the average edgewise MSE when using the GERGM model.
```

As we can see, this model does significantly better in terms of conditional edgewise predictive performance than the null model.

Correlation Matrix Estimation Functionality

The GERGM development team has recently added functionality to estimate GERGMs on correlation matrices, as well as simulation functionality. The GERGM can therefore be used both to assess the structural properties of correlation matrices, but also as an infinitely flexible prior for correlation matrices. To access the this functionality, set the optional argument `beta_correlation_model = TRUE`. Covariate effects are modeled using a Beta regression and then the correlation matrix is transformed onto an unconstrained space of matrices on [0,1], where standard GERGM modeling takes place. The resulting simulated networks can then be transformed back onto the correlation space. Correlation matrix estimation functionality is the same as with other undirected networks. More information on this model will be made available in mid 2017 with the publication of a paper that is now under review. Please contact the package maintainer with further questions.

```
# from a uniform distribution.
Posdef <- function (n, ev = runif(n, 0, 10)) {
  Z <- matrix(ncol=n, rnorm(n^2))
  decomp <- qr(Z)
  Q <- qr.Q(decomp)
  R <- qr.R(decomp)
  d <- diag(R)
  ph <- d / abs(d)
  O <- Q %*% diag(ph)
  Z <- t(O) %*% diag(ev) %*% O
  return(Z)
}

# Generate eigenvalues
x <- rnorm(10)
# generate a positive definite matrix
pdmat <- Posdef(n = 10)
# transform to correlations
correlations <- pdmat / max(abs(pdmat))
diag(correlations) <- 1
net <- (correlations + t(correlations)) / 2

# add in node names
colnames(net) <- rownames(net) <- letters[1:10]

# correlation GERGM specification
formula <- net ~ edges + ttriads

# model should run in under a minute
test <- gergm(formula,
              estimation_method = "Metropolis",
              number_of_networks_to_simulate = 100000,
              thin = 1/100,
              proposal_variance = 0.2,
              MCMC_burnin = 100000,
              seed = 456,
              convergence_tolerance = 0.5,
              beta_correlation_model = TRUE)
```

Common Problems and Errors

ERGMs and GERGM tend to be more challenging to fit to real world data than many other classes of models, due to the dependence between observations. We have sought to make the GERGM package as easy to use as possible, but it is likely that the user may still encounter some difficulties in fitting complex models, especially to large networks. Below, we outline several

Getting Started With GERGM

any estimation problems, as this will give you much more information to work with.

- **Time**: The GERGM naturally has at least order N^2 time complexity (where N is the number of nodes in the network), which means that runtime increases exponentially as the network gets bigger. It may also take a large number of iterations for the model to reach the target distribution (a long burnin), and a large number of parameter updates for the model to converge. It is not uncommon for the model to run for a week or two, even on networks as small as 50 nodes. Runtime can also be dramatically increased by using some of the features we have built to aid in model fitting. To some degree, this is just part of the process, and the user should plan to run their analyses on a computer that will not be unplugged or restarted for a few weeks. Additionally, the user may use parallelization to help speed up this process. By setting `parallel_statistic_calculation = TRUE` and the `cores` argument to a number greater than one, a speedup in estimation may be achieved, particularly for larger networks (more than 50 nodes).
- **Zero Acceptance Rate**: The Metropolis Hastings estimation algorithm works by simulating a large sample of networks from the GERGM generative process to approximate the normalizing constant. However, if the `proposal_variance` is too large or the model is very complex, then the algorithm may never be able to accept a new network proposal, which will cause the estimation procedure to fail. This will be evidenced by the following line in the model output, near where an error occurs `Metropolis Hastings Acceptance Rate (target = 0.25 ): 0`. The `gergm()` function will try to automatically deal with this issue by dividing the proposal variance by 10, and doubling the burnin and number of networks to simulate. To deal with this issue, try reducing the proposal variance by at least an order of magnitude. You can also set `hyperparameter_optimization = TRUE`, and the software will try to find an optimal proposal variance for you. However, you can speed things up by reducing the initial proposal variance yourself. If you are still not able to get a high enough proposal variance, try setting the `sample_edges_at_a_time` parameter to an odd number in the neighborhood of 1-200. The reason for setting it to an odd number like 159 is that this will cause the edges sampled together to cycle through estimation (providing better model fit). The basic idea behind this argument, is that if the model cannot accept all edges proposed together (which can be very difficult for large networks), it may have an easier time accepting smaller changes.
- **Poor Initialization**: While Wilson et al. (2017) show that the GERGM does not seem to exhibit classical likelihood degeneracy for most model specifications, if a particularly bad initialization for the parameters is selected by MPLE, then the Metropolis Hastings procedure may simulate networks such that the observed network is outside the convex hull of the simulated networks. By default, the `gergm()` function will seek to deal with this by using the convex hull initialization method developed by Hummel et al. (2012). The basic ideal behind this method is that if we have a poor initialization, we pretend like the target network is “closer” to our poor initialization, and let the optimizer try to move towards it. Then we keep iteratively moving our target network closer to the tru network until it is in the right neighborhood and regular optimization can take over. this method works very well most of the time, but sometimes we can still hit issues where the optimizer misbehaves. If this happens, the `optim` function in R (which we use to update the parameter estimates) will tend to zoom off to infinity in an effort to correct the error, and the estimation procedure will crash. This is desirable behavior because such parameter estimates (on the order of 1,000,000,000) will be meaningless. You can tell that the model is displaying this sort of behavior if the `final value` after the line reading `Optimizing theta estimates...` in the R output is equal to something like -100000000000. One way to deal

simulate networks where the observed network is inside the convex hull of simulated networks. If all else fails, the `theta_grid_optimization_list` option may be used, as in the example in the next section. This is a brute force approach which basically tries a number of parameter combinations around the MPLE estimates to find a combination that is good enough that the optimizer can do its work. The downside si that this requires a lot of computational power (and time), but this slowdown and be dramatically reduced by using multiple cores and parallelization. Adding more node level covariate effects to the model may also increase stability, but should not be done if those covariates are not theoretically justified.

While the above issues are not an exhaustive list, they are the most common ones we have encountered. Please check that your issue does not fall under one of the above categories (or that the solution described above does not work) before contacting the package maintainer.

Bonus: A More Complex Model

Here we have included code to run the full model which appears in the Wilson et al. paper. The original version of this code required a 30 core machine to run as currently specified, and took several days to weeks to run, depending on the computer setup. However, with the new convex hull initialization proceedure, this model will run in a few minutes on a standard laptop, while providing slightly better model fit.We include this more complex specification to highlight the flexibility the GERGM package gives users to deal with more difficult to model data, and the advances we have made in speeding up and improving estimation in the latest version of the package.

```
formula <- lending_2005 ~  mutual(0.8) + ttriads(0.8) + out2stars(0.8) +
  sender("log_GDP") + netcov(net_exports_2005) +
  receiver("log_GDP") + nodemix("G8", base = "No")


result <- gergm(formula,
                covariate_data = covariate_data_2005,
                number_of_networks_to_simulate = 100000,
                thin = 1/100,
                proposal_variance = 0.05,
                MCMC_burnin = 50000,
                seed = 456,
                convergence_tolerance = 0.8,
                target_accept_rate = 0.25)
```