Statistical Inference in Network Science

UDD networks 2024

George G. Vega Yon, Ph.D.

2024-01-14

Table of contents

1 Home

Introduction

If you are reading this, it is because you know that networks are everywhere. Network science is a rapidly growing field that has been applied to many different disciplines, from biology to sociology, from computer science to physics. In this course, we will go over advanced network science topics; particularly, statistical inference in networks. The course contents are:

- Overview of statistical inference.

- Introduction to network science inference.

- Motif detection.

- Global statistics (e.g., modularity).

- Random graphs (static).

- Random graphs (dynamic).

- Coevolution of networks and behavior.

- Advanced topics (sampling and conditional models).

About the author

Dr. George G. Vega Yon is an Assistant Professor of Research at the Division of Epidemiology at the University of Utah and a Lead Scientist at Booz Allen Hamilton. He studies Complex Systems using Statistical Computing. George has over ten years of experience developing scientific software focusing on high-performance computing, data visualization, and social network analysis. His training is in Public Policy (M.A. UAI, 2011), Economics (M.Sc. Caltech, 2015), and Biostatistics (Ph.D. USC, 2020).

Dr. Vega Yon obtained his Ph.D. in Biostatistics under the supervision of Prof. Paul Marjoram and Prof. Kayla de la Haye, with his dissertation titled "Essays on Bioinformatics and Social Network Analysis: Statistical and Computational Methods for Complex Systems."

Disclaimer

This is an ongoing project. The course is being developed and will be updated as we go. If you have any comments or suggestions, please let me know. The generation of the course materials was assisted by AI tools, namely, GitHub copilot.

Code of Conduct

Please note that the networks-udd2024 project is released with a [Contributor Code of Conduct](). By contributing to this project, you agree to abide by its terms.

Part I

Day 1

2 Overview

Before jumping into network science details, we need to cover some fundamentals. I assume that most of the contents here are well known to you–we will be brief–but I want to ensure we are all on the same page.

2.1 Programming with R

The R programming language (R Core Team 2023) is the defacto language for social network analysis[1]. Furthermore, R homes the most comprehensive collection of packages implementing the methods we will cover here. Let's start by the fundamentals

2.2 Getting help

Unlike other languages, R's documentation is highly reliable. The Comprehensive R Archive Network [CRAN] is the official repository of R packages. All packages posted on CRAN must pass a series of tests to ensure the quality of the code, including the documentation.

To get help on a function, we can use the `help()` function. For example, if we wanted to get help on the `mean()` function, we would do:

```
help("mean")
```

2.3 Naming conventions

R has a set of naming conventions that we should follow to avoid confusion. The most important ones are:

- Use lowercase letters (optional)
- Use underscores to separate words (optional)
- Do not start with a number
- Do not use special characters

---

[1] Although, not for network science in general.

- Do not use reserved words

> **Question**
>
> Of the following list, which are valid names and which are valid but to be avoided?
>
> ```
> _my.var
> my.var
> my_var
> myVar
> myVar1
> 1myVar
> my var
> my-var
> ```

2.4 Assignment

In R, we have two (four) ways of assigning values to objects: the `<-` and `=` binary operators[2]. Although both are equivalent, the former is the preferred way of assigning values to objects since the latter can be confused with function arguments.

```
x <- 1
x = 1
```

> **Question**
>
> What is the difference between the following two assignments? Use the help function to find out.
>
> ```
> x <- 1
> x <<- 1
> ```

> **Question**
>
> What are other ways in which you can assign values to objects?

---

[2]In mathematics and computer science, a binary operator is a function that takes two arguments. In R, binary operators are implemented as `variable 1 [operator] variable 2`. For example, `1 + 2` is a binary operation.

2.5 Using functions and piping

In R, we use functions to perform operations on objects. Functions are implemented as `function_name ( argument_1 , argument_2 , ... )`. For example, the `mean()` function takes a vector of numbers and returns the mean of the values:

```r
x <- c(1, 2, 3) # The c() function creates a vector
mean(x)
## [1] 2
```

Furthermore, we can use the pipe operator (`|>`) to improve readability. The pipe operator takes the output of the left-hand side expression and passes it as the first argument of the right-hand side expression. Our previous example could be rewritten as:

```r
c(1, 2, 3) |> mean()
## [1] 2
```

2.6 Data structures

Atomic types are the minimal building blocks of R. They are logical, integer, double, character, complex, raw:

```r
x_logical   <- TRUE
x_integer   <- 1L
x_double    <- 1.0
x_character <- "a"
x_complex   <- 1i
x_raw       <- charToRaw("a")
```

Unlike other languages, we do not need to declare the data type before creating the object; R will infer it from the value.

> **Pro-tip**
>
> Adding the `L` suffix to the value is good practice when dealing with integers. Some R packages like `data.table` (Barrett, Dowle, and Srinivasan 2023) have internal checks that will throw an error if you are not explicit about the data type.

The next type is the vector. A vector is a collection of elements of the same type. The most common way to create a vector is with the `c()` function:

```r
x_integer <- c(1, 2, 3)
x_double  <- c(1.0, 2.0, 3.0)
x_logical <- c(TRUE, FALSE, TRUE)
## etc.
```

R will coerce the data types to the most general type. For example, if we mix integers and doubles, R will coerce the integers into doubles. The coercion order is logical < integer < double < character

> **Question**
>
> Why is the coercion order logical < integer < double < character?

The next data structure is the list. A list is a collection of elements of any type. We can create a list with the `list()` function:

```r
x_list       <- list(1, 2.0, TRUE, "a")
x_list_named <- list(a = 1, b = 2.0, c = TRUE, d = "a")
```

To access elements in a list, we have two options: by position or by name, the latter only if the elements are named:

```r
x_list[[1]]
## [1] 1
x_list_named[["a"]]
## [1] 1
x_list_named$a
## [1] 1
```

After lists, we have matrices. A matrix is a collection of elements of the same type arranged in a two-dimensional grid. We can create a matrix with the `matrix()` function:

```r
x_matrix <- matrix(1:9, nrow = 3, ncol = 3)
x_matrix
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

## We can access elements in a matrix by row column, or position:
x_matrix[1, 2]
## [1] 4
x_matrix[cbind(1, 2)]
## [1] 4
```

```r
x_matrix[4]
## [1] 4
```

> **Matrix is a vector**
>
> Matrices in R are vectors with dimensions. In base R, matrices are stored in column-major order. This means that the elements are stored column by column. This is important to know when we are accessing elements in a matrix

The two last data structures are arrays and data frames. An array is a collection of elements of the same type arranged in a multi-dimensional grid. We can create an array with the `array()` function:

```r
x_array <- array(1:27, dim = c(3, 3, 3))

## We can access elements in an array by row, column, and dimension, or
## position:
x_array[1, 2, 3]
## [1] 22
x_array[cbind(1, 2, 3)]
## [1] 22
x_array[22]
## [1] 22
```

Data frames are the most common data structure in R. In principle, these objects are lists of vectors of the same length, each vector representing a column. Columns (lists) in data frames can be of different types, but elements in each column must be of the same type. We can create a data frame with the `data.frame()` function:

```r
x_data_frame <- data.frame(
  a = 1:3,
  b = c("a", "b", "c"),
  c = c(TRUE, FALSE, TRUE)
)

## We can access elements in a data frame by row, column, or position:
x_data_frame[1, 2]
## [1] "a"
x_data_frame[cbind(1, 2)]
## [1] "a"
x_data_frame$b[1]    # Like a list
## [1] "a"
x_data_frame[[2]][1] # Like a list too
```

```
## [1] "a"
```

2.7 Functions

Functions are the most important building blocks of R. A function is a set of instructions that takes one or more inputs and returns one or more outputs. We can create a function with the `function()` function:

```
## This function has two arguments (y is optional)
f <- function(x, y = 1) {
  x + 1
}

f(1)
## [1] 2
```

Starting with R 4, we can use the lambda syntax to create functions:

```
f <- \(x, y) x + 1

f(1)
## [1] 2
```

2.8 Control flow

Control flow statements allow us to control the execution of the code. The most common control flow statements are `if`, `for`, `while`, and `repeat`. We can create a control flow statement with the `if()`, `for()`, `while()`, and `repeat()` functions:

```
## if
if (TRUE) {
  "a"
} else {
  "b"
}
## [1] "a"

## for
for (i in 1:3) {
```

```
    cat("This is the number ", i, "\n")
}
## This is the number  1
## This is the number  2
## This is the number  3

## while
i <- 1
while (i <= 3) {
    cat("This is the number ", i, "\n")
    i <- i + 1
}
## This is the number  1
## This is the number  2
## This is the number  3

## repeat
i <- 1
repeat {
    cat("This is the number ", i, "\n")
    i <- i + 1
    if (i > 3) {
        break
    }
}
## This is the number  1
## This is the number  2
## This is the number  3
```

2.9 R packages

R is so powerful because of its extensions. R extensions (different from other programming languages) are called packages. Packages are collections of functions, data, and documentation that provide additional functionality to R. Although anyone can create and distribute R packages to other users, the Comprehensive R Archive Network [CRAN] is the official repository of R packages. All packages posted on CRAN are thoroughly tested, so generally, their quality is high.

To install R packages, we use the `install.packages()` function; to load them, we use the `library()` function. For example, the following code chunk installs the `ergm` package and loads it:

```
install.packages("ergm")
library(ergm)
```

2.10 Statistics

Generally, statistics are used for two purposes: to describe and to infer. We observe data samples in descriptive statistics, recording and reporting the mean, median, and standard deviation, among other statistics. Statistical inference, on the other hand, is used to infer the properties of a population from a sample, particularly, about population parameters.

Population   Sample

Data

$$(x_1, x_2, \cdots, x_n)$$

Parameters   Statistics

μ $\longleftarrow$ $\bar{x}$

σ^2 $\longleftarrow$ s^2

σ $\longleftarrow$ s

p $\longleftarrow$ $\hat{p}$
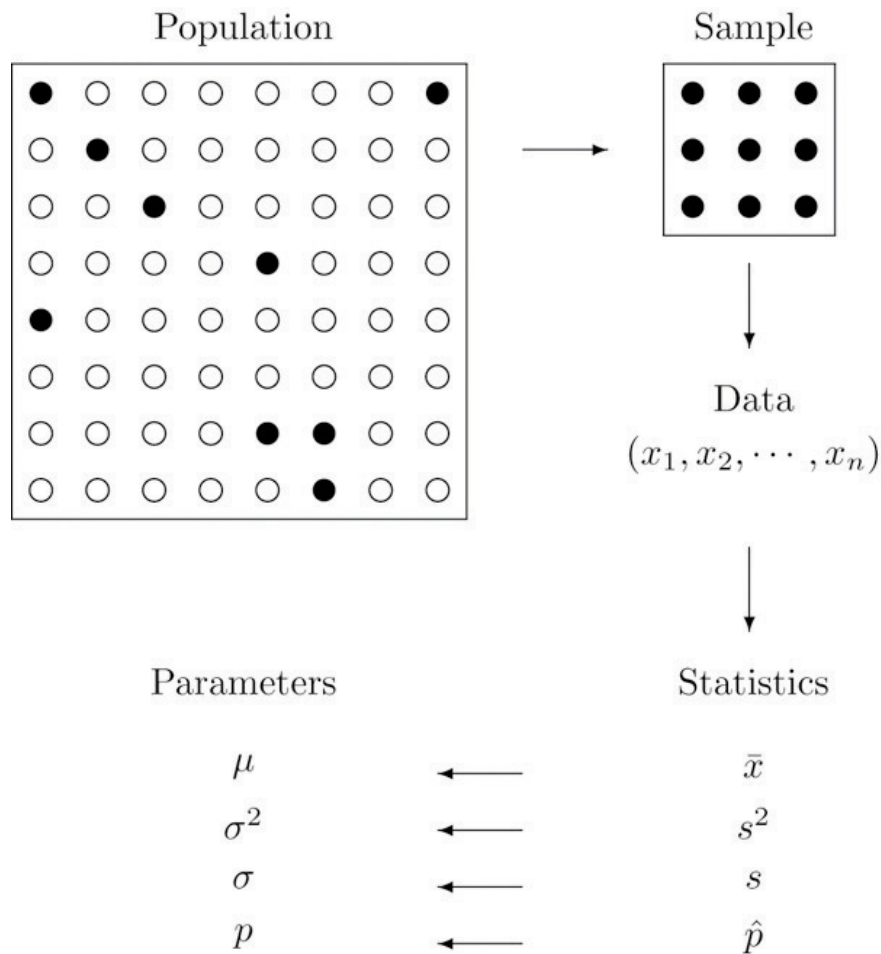
Figure 2.1: Grad view of statistics – Reproduced from "1.1: Basic Definitions and Concepts" (2014)

From the perspective of network science, descriptive statistics are used to describe the properties of a network, such as the number of nodes and edges, the degree distribution, the clustering coefficient, etc. Statistical inference in network science has to do with addressing

questions about the underlying properties of networked systems; some questions include the following:

- Are two networks different?
- Is the number of observed triangles in a network higher than expected by chance?
- Are individuals in a network more likely to be connected to individuals with similar characteristics?
- etc.

Part of statistical inference is hypothesis testing.

2.11 Hypothesis testing

According to Wikipedia

> A statistical hypothesis test is a method of statistical inference used to decide whether the data at hand sufficiently support a particular hypothesis. More generally, hypothesis testing allows us to make probabilistic statements about population parameters. More informally, hypothesis testing is the processes of making decisions under uncertainty. Typically, hypothesis testing procedures involve a user selected tradeoff between false positives and false negatives. – Wiki

In a nutshell, hypothesis testing is performed by following these steps:

1. State the null and alternative hypotheses. In general, the null hypothesis is a statement about the population parameter that challenges our research question; for example, given the question of whether two networks are different, the null hypothesis would be that the two networks are the same.

2. Compute the corresponding test statistic. It is a data function that reduces the information to a single number.

3. Compare the observed test statistic with the distribution of the test statistic under the null hypothesis. The sometimes infamous p-value: "[...] the probability that the chosen test statistic would have been at least as large as its observed value *if* every **model assumption were correct**, including the test hypothesis.'' (Greenland et al. 2016) [3]

4. Report the observed effect and p-value, *i.e.*, $\Pr(t \in H_0)$

---

[3]The discussion about interpreting p-values and hypothesis testing is vast and relevant. Although we will not review this here, I recommend looking into the work of Andrew Gelman Gelman (2018).

We usually say that we either *reject the null hypothesis* or *fail to reject it* (we never accept the null hypothesis,) but, in my view, it is always better to talk about it in terms of "suggests evidence for" or "suggests evidence against."

We will illustrate statistical concepts more concretely in the next section.

2.12 Statistical programming

Statistical programming (or computing) is the science of leveraging modern computing power to solve statistical problems. The R programming language is the defacto language for statistical programming, and so it has an extensive collection of packages implementing statistical methods and functions.

2.13 Probability distributions

R has a standard way of naming probability functions. The naming structure is `[type of function][distribution]`, where `[type of function]` can be `d` for density, `p` for cumulative distribution function, `q` for quantile function, and `r` for random generation. For example, the normal distribution has the following functions:

```
dnorm(0, mean = 0, sd = 1)
## [1] 0.3989423
pnorm(0, mean = 0, sd = 1)
## [1] 0.5
qnorm(0.5, mean = 0, sd = 1)
## [1] 0
```

Now, if we wanted to know what is the probability of observing a value smaller than -2 comming from a standard normal distribution, we would do:

```
pnorm(-2, mean = 0, sd = 1)
## [1] 0.02275013
```

Currently, R has a wide range of probability distributions implemented.

> **Question**
>
> How many probability distributions are implemented in R's stats package?

2.14 Random number generation

Random numbers, and more precisely, pseudo-random numbers, are a vital component of statistical programming. Pure randomness is hard to come by, and so we rely on pseudo-random number generators (PRNGs) to generate random numbers. These generators are deterministic algorithms that produce sequences of numbers we can then use to generate random samples from probability distributions. Because of the latter, PRNGs need a starting point called the seed. As a statistical computing program, R has a variety of PRNGs. As suggested in the previous subsection, we can generate random numbers from a probability distribution with the `r` function. In what follows, we will draw random numbers from a few distributions and plot histograms of the results:

```r
set.seed(1)

## Saving the current graphical parameters
op <- par(mfrow = c(2,2))
rnorm(1000) |> hist(main = "Normal distribution")
runif(1000) |> hist(main = "Uniform distribution")
rpois(1000, lambda = 1) |> hist(main = "Poisson distribution")
rbinom(1000, size = 10, prob = 0.1) |> hist(main = "Binomial distribution")
```
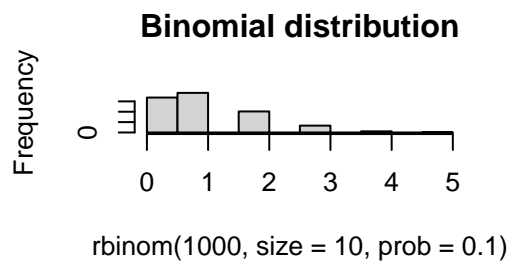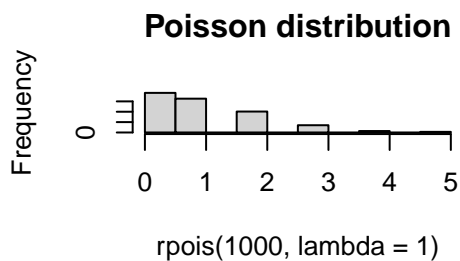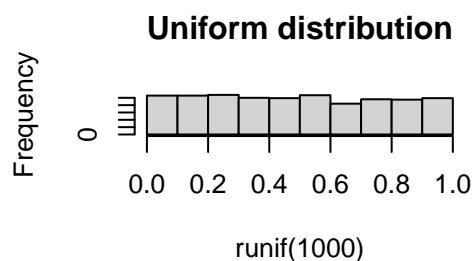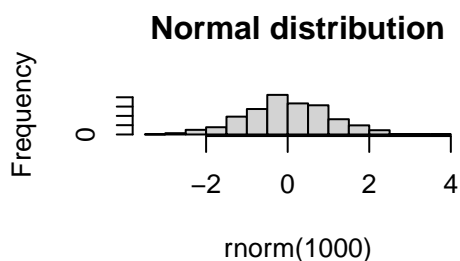


```r
par(op)
```

2.15 Simulations and sampling

Simulations are front and center in statistical programming. We can use them to test the properties of statistical methods, generate data, and perform statistical inference. The following example uses the sample function in R to compute the bootstrap standard error of the mean (see Casella and Berger 2021):

```r
set.seed(1)
x <- rnorm(1000)

## Bootstrap standard error of the mean
n <- length(x)
B <- 1000

## We will store the results in a vector
res <- numeric(B)

for (i in 1:B) {
  # Sample with replacement
  res[i] <- sample(x, size = n, replace = TRUE) |>
    mean()
}

## Plot the results
hist(res, main = "Bootstrap standard error of the mean")
```
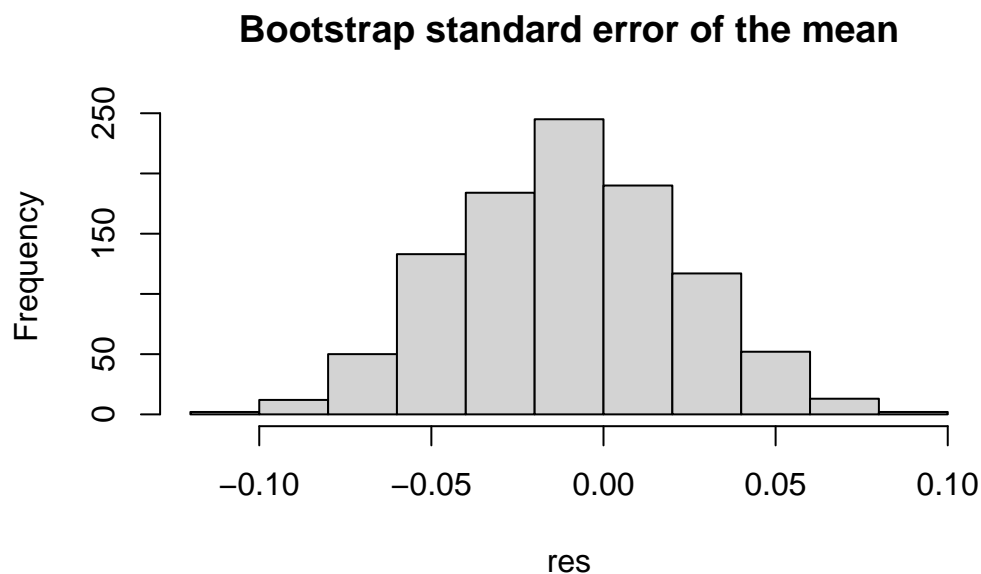

Since the previous example is rather extensive, let us review it in detail.

- `set.seed(1)` sets the seed of the PRNG to 1. It ensures we get the same results every time we run the code.

- `rnorm()` generates a sample of 1,000 standard-normal values.

- `n <- length(x)` stores the length of the vector in the `n` variable.

- `B <- 1000` stores the number of bootstrap samples in the `B` variable.

- `res <- numeric(B)` creates a vector of length `B` to store the results.

- `for (i in 1:B)` is a for loop that iterates from 1 to `B`.

- `res[i] <- sample(x, size = n, replace = TRUE) |> mean()` samples `n` values from `x` with replacement and computes the mean of the sample.

- The pipe operator (`|>`) passes the output of the left-hand side expression as the first argument of the right-hand side expression.

- `hist(res, main = "Bootstrap standard error of the mean")` plots the histogram of the results.

3 Random graphs

3.1 Introduction

In this section, we will focus on reviewing the most common random graph models, how these are used, and what things are important to consider when using them. Later on in the course, we will focus on Exponential-Family Random Graph Models [ERGMs], which are a generalization of the models we will discuss here.

3.2 Erdős–Rényi model

The Erdős–Rényi model is the simplest random graph model. It is defined by two parameters: n and p. The parameter n is the number of nodes in the graph, and p is the probability that any two nodes are connected by an edge. The model is named after Paul Erdős and Alfréd Rényi, who first introduced it in 1959.

Formally, we can describe the ER model as follows: (V, E) where $V = \{1, ..., n\}$ and E is a set of edges, where each edge is included with probability p.

> 💡 Tip
>
> **Computing note**: In the case of large networks, sampling ER graphs can be done effectively in a two-step process. First, we sample the number of edges in the graph from a binomial distribution. Then, we sample the edges uniformly at random from the set of all possible edges. This is much more efficient than sampling each edge independently since the number of possible edges is much smaller than the number of possible graphs.

Most of the time, the ER is used as a reference distribution for studying real-world networks. Nevertheless, using the ER model as a null model for a real-world network is not always a good idea, as it may inflate the type two error rate.

3.3 Code example

```
## Model parameters
n <- 40
p <- 0.1

## Generating the graph, version 1
set.seed(3312)
g <- matrix(as.integer(runif(n * n) < p), nrow = n, ncol = n)
diag(g) <- 0

## Visualizing the network
library(igraph)
library(netplot)
nplot(graph_from_adjacency_matrix(g))
```
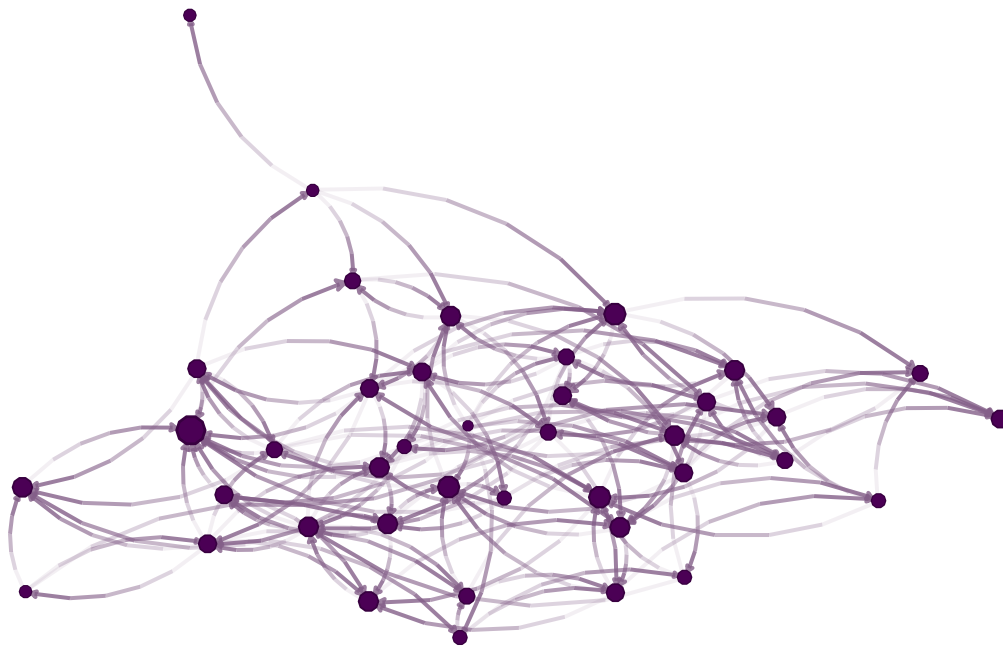

Challenge 1: How would you generate the graph using the two-step process described above?

Challenge 2: Using a Generalized-Linear-Model [GLM], estimate p and its variance from the above network.

3.4 Watts-Strogatz model

The second model in our list is the small-world model, introduced by Duncan Watts and Steven Strogatz in 1998. The model is defined by three parameters: n, k, and p. The parameter n is the number of nodes in the graph, k is the number of neighbors each node is connected to, and p is the probability that an edge is rewired. As its name suggests, the networks sampled from this model hold the small-world property, which means that the average distance between any two nodes is small.

Networks from the WS model are generated as follows:

1. Start with a ring of n nodes, where each node is connected to its k nearest neighbors.

2. For each edge (u, v), rewire it with probability p by replacing it with a random edge (u, w), where w is chosen uniformly at random from the set of all nodes.

Challenge: How would you generate a WS graph using the two-step process described above?

3.5 Code example

```
## Creating a ring
n <- 10
V <- 1:n
k <- 3
p <- .2

E <- NULL
for (i in 1:k) {
  E <- rbind(E, cbind(V, c(V[-c(1:i)], V[1:i])))
}

## Generating the ring layout
lo <- layout_in_circle(graph_from_edgelist(E))

## Plotting with netplot
nplot(
  graph_from_edgelist(E),
  layout = lo
  )
```
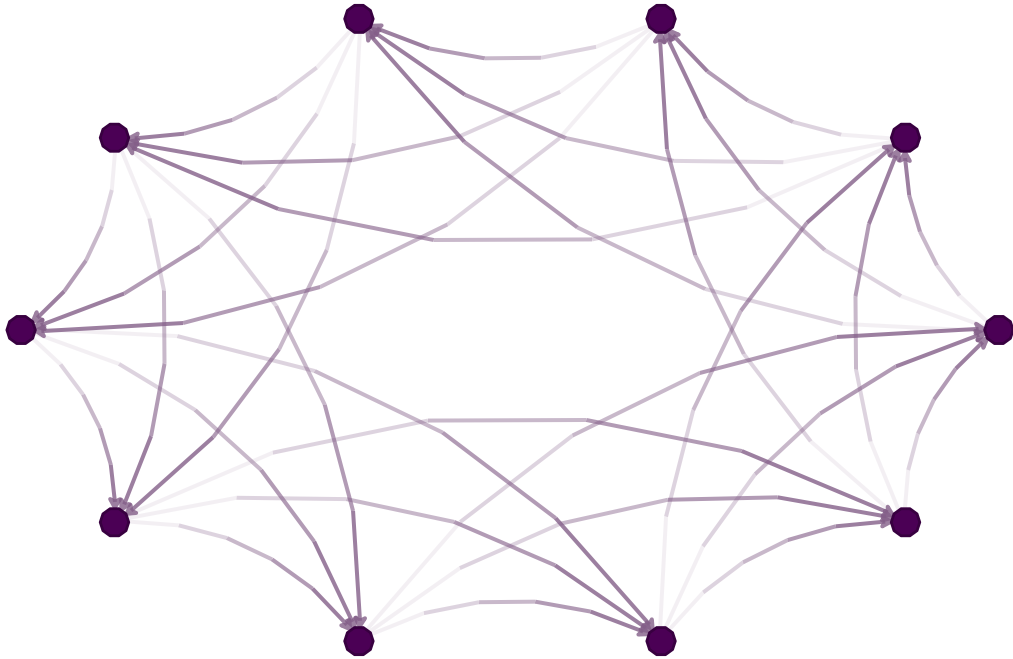
```
## Rewiring
ids <- which(runif(nrow(E)) < p)
E[ids, 2] <- sample(V, length(ids), replace = TRUE)
nplot(
  graph_from_edgelist(E),
  layout = lo
  )
```

3.6 Scale-free networks

Scale-free networks are networks where the degree distribution follows a power-law distribution. The power-law distribution is a heavy-tailed distribution, which means that it has a long tail of high-degree nodes. The power-law distribution is defined as follows:

$$p(k) = Ck^{-\gamma}$$

where C is a normalization constant and γ is the power-law exponent. The power-law exponent is usually between 2 and 3, but it can be any value larger than 2. The power-law distribution is a special case of the more general class of distributions called the Pareto distribution.

Scale-free networks are generated using the Barabási–Albert model, which was introduced by Albert-László Barabási and Réka Albert in 1999. The model is defined by two parameters: n and m. The parameter n is the number of nodes in the graph, and m is the number of edges added at each time step. The model is generated as follows:

1. Start with a graph of m nodes, where each node is connected to all other nodes.

2. At each time step, add a new node to the graph and connect it to m existing nodes. The probability that a new node is connected to an existing node i is proportional to the degree of i.

3.7 Code example

```
## Model parameters
n <- 500
m <- 2

## Generating the graph
set.seed(3312)
g <- matrix(0, nrow = n, ncol = n)
g[1:m, 1:m] <- 1
diag(g) <- 0

## Adding nodes
for (i in (m + 1):n) {

  # Selecting the nodes to connect to
  ids <- sample(
    x     = 1:(i-1), # Up to i-1
    size  = m,       # m nodes
```

```
    replace = FALSE,    # No replacement
    # Probability proportional to the degree
    prob     = colSums(g[, 1:(i-1), drop = FALSE])
    )

  # Adding the edges
  g[i, ids] <- 1
  g[ids, i] <- 1

}

## Visualizing the degree distribution
library(ggplot2)
data.frame(degree = colSums(g)) |>
  ggplot(aes(degree)) +
  geom_histogram() +
  scale_x_log10() +
  labs(
    x = "Degree\n(log10 scale)",
    y = "Count"
  )
```
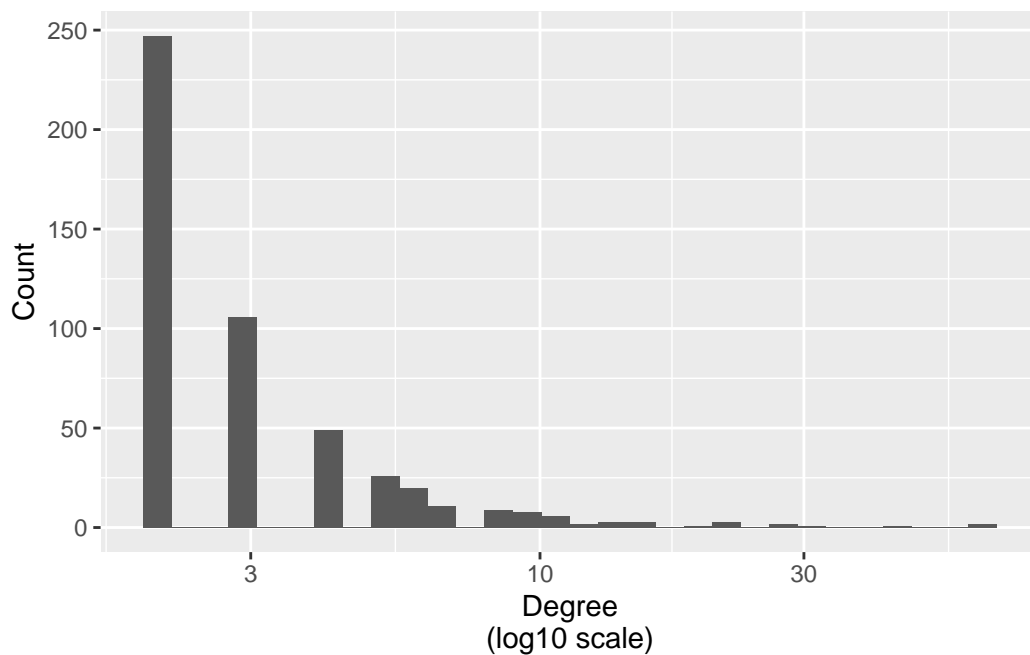
4 Behavior and coevolution

4.1 Introduction

This section focuses on inference involving network and a secondary outcome. While there are many ways of studying the coevolution or dependence between network and behavior, this section focuses on two classes of analysis: When the network is fixed and when both network and behavior influence each other.

Whether we treat the network as given or endogenous sets the complexity of conducting statistical inference. Data analysis becomes much more straightforward if our research focuses on individual-level outcomes embedded in a network and not on the network itself. Here, we will deal with three particular cases: (1) when network effects are lagged, (2) egocentric networks, and (3) when network effects are contemporaneous.

4.2 Lagged exposure

If we assume that network influence in the form of exposure is lagged, we have one of the most straightforward cases for network inference (Haye et al. 2019; Valente and Vega Yon 2020; Valente, Wipfli, and Vega Yon 2019). Here, instead of dealing with convoluted statistical models, the problem reduces to estimating a simple linear regression model. Generally, lagged exposure effects look like this:

$$y_{it} = \rho \left(\sum_{j \neq i} X_{ij} \right)^{-1} \left(\sum_{j \neq i} y_{jt-1} X_{ij} \right) + \theta^{\mathbf{t}} w_i + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

where y_{it} is the outcome of individual i at time t, X_{ij} is the ij-th entry of the adjacency matrix, θ is a vector of coefficients, w_i is a vector of features/covariates of individual i, and ε_i is a normally distributed error. Here, the key component is ρ: the coefficient associated with the network exposure effect.

The exposure statistic, $\left(\sum_{j \neq i} X_{ij} \right)^{-1} \left(\sum_{j \neq i} y_{jt-1} X_{ij} \right)$, is the weighted average of i's neighbors' outcomes at time $t - 1$.

4.3 Code example: Lagged exposure

The following code example shows how to estimate a lagged exposure effect using the `glm` function in R. The model we will simulate and estimate features a Bernoulli graph with 1,000 nodes and a density of 0.01.

$$y_{it} = \theta_1 + \rho \mathrm{Exposure}_{it} + \theta_2 w_i + \varepsilon$$

where $\mathrm{Exposure}_{it}$ is the exposure statistic defined above, and w_i is a vector of covariates.

```
## Simulating data
n <- 1000
time <- 2
theta <- c(-1, 3)

## Sampling a bernoilli network
set.seed(3132)
p <- 0.01
X <- matrix(rbinom(n^2, 1, p), nrow = n)
diag(X) <- 0

## Covariate
W <- matrix(rnorm(n), nrow = n)

## Simulating the outcome
rho <- 0.5
Y0 <- cbind(rnorm(n))

## The lagged exposure
expo <- (X %*% Y0)/rowSums(X)
Y1 <- theta[1] + rho * expo + W * theta[2] + rnorm(n)
```

Now we fit the model using GLM, in this case, linear Regression

```
fit <- glm(Y1 ~ expo + W, family = "gaussian")
summary(fit)
```

```
Call:
glm(formula = Y1 ~ expo + W, family = "gaussian")

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept) -1.07187    0.03284 -32.638  < 2e-16 ***
expo          0.61170    0.10199   5.998  2.8e-09 ***
W             3.00316    0.03233  92.891  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 1.071489)

    Null deviance: 10319.3  on 999  degrees of freedom
Residual deviance:  1068.3  on 997  degrees of freedom
AIC: 2911.9

Number of Fisher Scoring iterations: 2
```

4.4 Egocentric networks

Generally, when we use egocentric networks and egos' outcomes, we are thinking in a model where one observation is the pair (y_i, X_i), this is, the outcome of individual i and the egocentric network of individual i. When such is the case, since (a) networks are independent across egos and (b) the networks are fixed, like the previous case, a simple linear regression model is enough to conduct the analyses. A typical model looks like this:

$$y = \theta_x^{\mathbf{t}} s(X) + \theta^{\mathbf{t}} w + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

Where y is a vector of outcomes, X is a matrix of egocentric networks, w is a vector of covariates, θ is a vector of coefficients, and ε is a vector of errors. The key component here is $s(X)$, which is a vector of sufficient statistics of the egocentric networks. For example, if we are interested in the number of ties, $s(X)$ is a vector of the number of ties of each ego.

4.5 Code example: Egocentric networks

For this example, we will simulate a stream of 1,000 Bernoulli graphs looking into the probability of school dropout. Each network will have between 4 and 10 nodes and have a density of 0.4. The data-generating process is as follows:

$$\mathrm{Pr}_{\theta}(Y_i = 1) = \mathrm{logit}^{-1}(\theta_x s(X_i))$$

Where $s(X) \equiv (\mathrm{density}, \mathrm{n\ mutual\ ties})$, and $\theta_x = (0.5, -1)$. This model only features sufficient statistics. We start by simulating the networks

```
set.seed(331)
n <- 1000
sizes <- sample(4:10, n, replace = TRUE)

## Simulating the networks
X <- lapply(sizes, function(x) matrix(rbinom(x^2, 1, 0.4), nrow = x))
X <- lapply(X, \(x) {diag(x) <- 0; x})

## Inspecting the first 5
head(X, 5)
```

```
[[1]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    0
[2,]    0    0    0    0    0
[3,]    0    1    0    0    0
[4,]    0    0    0    0    0
[5,]    1    0    0    1    0

[[2]]
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    1    0    1    0

[[3]]
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    1    0    1    0    0
[2,]    0    0    0    0    0    0
[3,]    0    1    0    0    0    1
[4,]    0    0    0    0    1    0
[5,]    0    0    0    0    0    0
[6,]    0    0    0    0    0    0

[[4]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    1    0
[2,]    0    0    0    0    1
[3,]    0    1    0    0    0
[4,]    0    1    1    0    1
[5,]    1    0    1    0    0
```

```
[[5]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    1    0    0    0    0
[3,]    1    0    0    0    0
[4,]    0    0    0    0    0
[5,]    1    0    0    0    0
```

Using the `ergm` R package (Handcock et al. 2023; Hunter et al. 2008), we can extract the associated sufficient statistics of the egocentric networks:

```r
library(ergm)
stats <- lapply(X, \(x) summary_formula(x ~ density + mutual))

## Turning the list into a matrix
stats <- do.call(rbind, stats)

## Inspecting the first 5
head(stats, 5)
```

```
      density mutual
[1,] 0.3000000      0
[2,] 0.1666667      0
[3,] 0.1666667      0
[4,] 0.4500000      0
[5,] 0.1500000      0
```

We now simulate the outcomes

```r
y <- rbinom(n, 1, plogis(stats %*% c(0.5, -1)))
glm(y ~ stats, family = binomial(link = "logit")) |>
  summary()
```

```
Call:
glm(formula = y ~ stats, family = binomial(link = "logit"))

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    0.07319    0.41590   0.176    0.860
statsdensity   0.42568    1.26942   0.335    0.737
statsmutual   -1.14804    0.12166  -9.436   <2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 768.96  on 999  degrees of freedom
Residual deviance: 518.78  on 997  degrees of freedom
AIC: 524.78

Number of Fisher Scoring iterations: 7
```

4.6 Network effects are endogenous

Here we have two different approaches: Spatial Autocorrelation [SAR], and Autologistic actor attribute model [ALAAM] (Robins, Pattison, and Elliott 2001). The first one is a generalization of the linear regression model that accounts for spatial dependence. The second is a close relative of ERGMs that treats covariates as endogenous and network as exogenous. Overall, ALAAMs are more flexible than SARs, but SARs are easier to estimate.

SAR Formally, SAR models (see LeSage 2008) can be used to estimate network exposure effects. The general form is:

$$y = \rho W y + \theta^{\mathbf{t}} X + \epsilon, \quad \epsilon \sim \text{MVN}(0, \Sigma)$$

where $y \equiv \{y_i\}$ is a vector of outcomes the outcome, ρ is an autocorrelation coefficient, $W \in \{w_{ij}\}$ is a row-stochastic square matrix of size n, θ is a vector of model parameters, X is the corresponding matrix with exogenous variables, and ϵ is a vector of errors that distributes multivariate normal with mean 0 and covariance makeΣ.[^notation] The SAR model is a generalization of the linear regression model that accounts for spatial dependence. The SAR model can be estimated using the `spatialreg` package in R (Roger Bivand 2022).

> 💡 Tip
>
> What is the appropriate network to use in the SAR model? According to LeSage and Pace (2014), it is not very important. Since $(I_n - \rho \mathbf{W})^{-1} = \rho \mathbf{W} + \rho^2 \mathbf{W}^2 + ...$

Although the SAR model was developed for spatial data, it is easy to apply it to network data. Furthermore, each entry of the vector Wy has the same definition as network exposure, namely

$$W y \equiv \left\{ \sum_j y_j w_{ij} \right\}_i$$

Since W is row-stochastic, Wy is a weighted average of the outcome of the neighbors of i, *i.e.*, a vector of network exposures.

ALAAM The simplest way we can think of this class of models is as if a given covariate switched places with the network in an ERGM, so the network is now fixed and the covariate is the random variable. While ALAAMs can also estimate network exposure effects, we can use them to build more complex models beyond exposure. The general form is:

$$\Pr\left(Y = y | W, X\right) = \exp\left(\theta^{\mathbf{t}} s(y, W, X)\right) \times \eta(\theta)^{-1}$$

$$\eta(\theta) = \sum_y \exp\left(\theta^{\mathbf{t}} s(y, W, X)\right)$$

Where $Y \equiv \{y_i \in (0, 1)\}$ is a vector of binary individual outcomes, W denotes the social network, X is a matrix of exogenous variables, θ is a vector of model parameters, $s(y, W, X)$ is a vector of sufficient statistics, and $\eta(\theta)$ is a normalizing constant.

4.7 Code example: SAR

Simulation of SAR models can be done using the following observation: Although the outcome shows on both sides of the equation, we can isolate it in one side and solve for it; formally:

$$y = \rho X y + \theta^{\mathbf{t}} W + \varepsilon \implies y = (I - \rho X)^{-1} \theta^{\mathbf{t}} W + (I - \rho X)^{-1} \varepsilon$$

The following code chunk simulates a SAR model with a Bernoulli graph with 1,000 nodes and a density of 0.01. The data-generating process is as follows:

```
set.seed(4114)
n <- 1000

## Simulating the network
p <- 0.01
X <- matrix(rbinom(n^2, 1, p), nrow = n)

## Covariate
W <- matrix(rnorm(n), nrow = n)

## Simulating the outcome
rho <- 0.5
library(MASS) # For the mvrnorm function
```

```
## Identity minus rho * X
X_rowstoch <- X / rowSums(X)
I <- diag(n) - rho * X_rowstoch

## The outcome
Y <- solve(I) %*% (2 * W) + solve(I) %*% mvrnorm(1, rep(0, n), diag(n))
```

Using the `spatialreg` R package, we can fit the model using the `lagsarlm` function:

```
library(spdep) # for the function mat2listw
library(spatialreg)
fit <- lagsarlm(
  Y ~ W,
  data  = as.data.frame(X),
  listw = mat2listw(X_rowstoch)
  )

## Using texreg to get a pretty print
texreg::screenreg(fit, single.row = TRUE)
```

```
========================================
                   Model 1
----------------------------------------
(Intercept)              -0.01 (0.03)
W                         1.97 (0.03) ***
rho                       0.54 (0.04) ***
----------------------------------------
Num. obs.              1000
Parameters                4
Log Likelihood        -1373.02
AIC (Linear model)     2920.37
AIC (Spatial model)    2754.05
LR test: statistic      168.32
LR test: p-value          0.00
========================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

The interpretation of this model is almost the same as a linear regression, with the difference that we have the autocorrelation effect (`rho`). As expected, the model got an estimate close enough to the population parameter: $\rho = 0.5$.

4.8 Code example: ALAAM

To date, there is no R package implementing the ALAAM framework. Nevertheless, you can fit ALAAMs using the PNet software developed by the Melnet group at the University of Melbourne (click here).

Because of the similarities, ALAAMs can be implemented using ERGMs. Because of the novelty of it, the coding example will be left as a potential class project. We will post a fully-featured example after the workshop.

4.9 Coevolution

Finally, we discuss coevolution when both network and behavior are embedded in a feedback loop. Coevolution should be the default assumption when dealing with social networks. Nevertheless, models capable of capturing coevolution are hard to estimate. Here, we will discuss two of such models: Stochastic Actor-Oriented Models (or Siena Models) (first introduced in T. a. B. Snijders (1996); see also Tom A. B. Snijders (2017)) and Exponential-family Random Exponential-family Random Network Models [ERNMs,] a generalization of ERGMs (Wang, Fellows, and Handcock 2023; Fellows 2012).

Siena Stochastic Actor-Oriented Models [SOAMs] or Siena Models are dynamic models of network and behavior that describe the transition of a network system within two or more time points. The general form of a Siena model is:

ERNMs This model is closely related to ERGMs, with the difference that they incorporate a vertex-level output. Conceptually, it is moving from a having a random network, to a model where a given vertex feature and network are random:

$$P_{y,\theta}(Y = y | X = x) \to P_{y,\theta}(Y = y, X = x)$$

4.10 Code example: Siena

This example was adapted from the `RSiena` R package (see `?sienaGOF-auxiliary` page).We start by loading the package and taking a look at the data we will use:

```r
library(RSiena)

## Visualizing the adjacency matrix & behavior
op <- par(mfrow=c(2, 2))
image(s501, main = "Net: s501")
image(s502, main = "Net: s502")
hist(s50a[,1], main = "Beh1")
```

```
hist(s50a[,2], main = "Beh2")
```

Net: s501

Net: s502

Beh1

Beh2
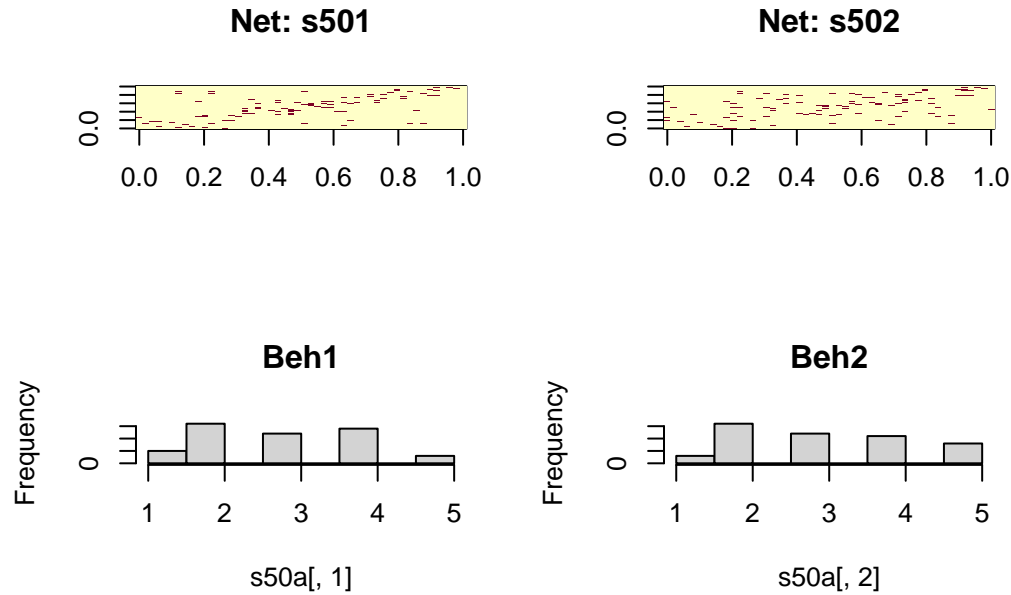

```
par(op)
```

The next step is the data preparation process. `RSiena` does not receive raw data as is. We need to explicitly declare the networks and outcome variable. Siena models can also model network changes

```
## Initializing the dependent variable (network)
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet1
```

```
Type          oneMode
Observations  2
Nodeset       Actors (50 elements)
```

```
mybeh  <- sienaDependent(s50a[,1:2], type="behavior")
mybeh
```

```
Type          behavior
Observations  2
Nodeset       Actors (50 elements)
```

36

```
## Node-level covariates (artificial)
mycov  <- c(rep(1:3,16),1,2)

## Edge-level covariates (also artificial)
mydycov <- matrix(rep(1:5, 500), 50, 50)


## Creating the data object
mydata <- sienaDataCreate(mynet1, mybeh)

## Adding the effects (first get them!)
myeff <- getEffects(mydata)

## Notice that Siena adds some default effects
myeff
##   name    effectName                    include fix    test  initialValue parm
## 1 mynet1 basic rate parameter mynet1 TRUE     FALSE FALSE    4.69604   0
## 2 mynet1 outdegree (density)          TRUE     FALSE FALSE   -1.48852   0
## 3 mynet1 reciprocity                  TRUE     FALSE FALSE    0.00000   0
## 4 mybeh   rate mybeh period 1          TRUE     FALSE FALSE    0.70571   0
## 5 mybeh   mybeh linear shape           TRUE     FALSE FALSE    0.32247   0
## 6 mybeh   mybeh quadratic shape        TRUE     FALSE FALSE    0.00000   0

## Adding a few extra effects (automatically prints them out)
myeff <- includeEffects(myeff, transTies, cycle3)
##   effectName       include fix    test  initialValue parm
## 1 3-cycles          TRUE    FALSE FALSE        0    0
## 2 transitive ties TRUE     FALSE FALSE        0    0
```

To add more effects, first, call the function `effectsDocumentation(myeff)`. It will show you explicitly how to add a particular effect. For instance, if we wanted to add network exposure (`avExposure`,) under the documentation of `effectsDocumentation(myeff)` we need to pass the following arguments:

```
## And now, exposure effect
myeff <- includeEffects(
  myeff,
  avExposure,
  # These last three are specified by effectsDocum...
  name         = "mybeh",
  interaction1 = "mynet1",
  type         = "rate"
  )
```

```
    effectName                                  include fix    test  initialValue parm
1 average exposure effect on rate mybeh TRUE     FALSE FALSE            0    0
```

The next step involves creating the model with (`sienaAlgorithmCreate`,) where we specify
all the parameters for fitting the model (e.g., MCMC steps.) Here, we modified the values
of `n3` and `nsub` to half of the default values to reduce the time it would take to fit the model;
yet this degrades the quality of the fit.

```
## Shorter phases 2 and 3, just for example:
myalgorithm <- sienaAlgorithmCreate(
  nsub = 2, n3 = 500, seed = 122, projname = NULL
  )
## If you use this algorithm object, siena07 will create/use an output file Siena.ta

## Fitting and printing the model
ans <- siena07(
  myalgorithm,
  data = mydata, effects = myeff,
  returnDeps = TRUE, batch = TRUE
  )
##
## Start phase 0
## theta:  4.696 -1.489  0.000  0.000  0.000  0.706  0.000  0.322  0.000
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 1%
## Phase 1 Iteration 15 Progress: 1%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 2%
## Phase 1 Iteration 30 Progress: 2%
## Phase 1 Iteration 35 Progress: 2%
## Phase 1 Iteration 40 Progress: 3%
## Phase 1 Iteration 45 Progress: 3%
## Phase 1 Iteration 50 Progress: 3%
## theta:  5.380 -1.734  0.481  0.147  0.166  1.168 -0.340  0.250  0.140
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 33%
## Phase 2 Subphase 1 Iteration 2 Progress: 33%
```

```
## theta  6.044 -1.877  0.840  0.300  0.473  1.096 -0.332  0.112  0.191
## ac  0.375 -0.183  3.835  4.574 23.412  0.922  0.908  0.626  3.302
## Phase 2 Subphase 1 Iteration 3 Progress: 33%
## Phase 2 Subphase 1 Iteration 4 Progress: 33%
## theta  6.1075 -2.0372  1.2512  0.0341  0.5207  1.2593 -0.1534  0.3018  0.1928
## ac  0.9859  0.2280 -0.0703 -2.2973 -2.7455  1.1518  1.0749  0.8732  0.5399
## Phase 2 Subphase 1 Iteration 5 Progress: 33%
## Phase 2 Subphase 1 Iteration 6 Progress: 33%
## theta  6.8650 -2.3185  1.7577  0.2694  0.7636  1.1997 -0.0433  0.3000  0.1762
## ac  0.4789  0.4883  0.0199 -1.9449 -2.2609  1.1444  1.0397  0.9282  0.6470
## Phase 2 Subphase 1 Iteration 7 Progress: 33%
## Phase 2 Subphase 1 Iteration 8 Progress: 33%
## theta  6.801140 -2.485273  1.966471  0.301197  0.817324  1.228010  0.000627  0.37
## ac  0.4538  0.4933 -0.0158 -1.9326 -2.2588  1.1102  1.0395  0.9239  0.6905
## Phase 2 Subphase 1 Iteration 9 Progress: 33%
## Phase 2 Subphase 1 Iteration 10 Progress: 33%
## theta  6.1258 -2.5124  1.8704  0.1206  0.6054  1.4538 -0.0145  0.5091 -0.0671
## ac  0.472  0.103 -0.330 -1.625 -1.991  1.152  1.090  0.767  0.514
## Phase 2 Subphase 1 Iteration 1 Progress: 33%
## Phase 2 Subphase 1 Iteration 2 Progress: 33%
## Phase 2 Subphase 1 Iteration 3 Progress: 33%
## Phase 2 Subphase 1 Iteration 4 Progress: 33%
## Phase 2 Subphase 1 Iteration 5 Progress: 33%
## Phase 2 Subphase 1 Iteration 6 Progress: 34%
## Phase 2 Subphase 1 Iteration 7 Progress: 34%
## Phase 2 Subphase 1 Iteration 8 Progress: 34%
## Phase 2 Subphase 1 Iteration 9 Progress: 34%
## Phase 2 Subphase 1 Iteration 10 Progress: 34%
## theta  6.4976 -2.5900  1.9265  0.2750  0.8543  1.0420  0.0446  0.3234 -0.0686
## ac -0.212 -0.697 -0.818 -0.831 -0.765 -0.442 -0.268 -0.240 -0.267
## theta:  6.4976 -2.5900  1.9265  0.2750  0.8543  1.0420  0.0446  0.3234 -0.0686
##
## Start phase 2.2
## Phase 2 Subphase 2 Iteration 1 Progress: 48%
## Phase 2 Subphase 2 Iteration 2 Progress: 48%
## Phase 2 Subphase 2 Iteration 3 Progress: 48%
## Phase 2 Subphase 2 Iteration 4 Progress: 48%
## Phase 2 Subphase 2 Iteration 5 Progress: 48%
## Phase 2 Subphase 2 Iteration 6 Progress: 48%
## Phase 2 Subphase 2 Iteration 7 Progress: 49%
## Phase 2 Subphase 2 Iteration 8 Progress: 49%
## Phase 2 Subphase 2 Iteration 9 Progress: 49%
## Phase 2 Subphase 2 Iteration 10 Progress: 49%
```

```
## theta  6.7988 -2.5810  1.9335  0.3941  0.7843  1.1111  0.0358  0.3252 -0.0415
## ac -0.0757 -0.3672 -0.4025 -0.3676 -0.4157 -0.0166  0.0222 -0.0874  0.0651
## theta:  6.7988 -2.5810  1.9335  0.3941  0.7843  1.1111  0.0358  0.3252 -0.0415
##
## Start phase 3
## Phase 3 Iteration 500 Progress 100%

ans
## Estimates, standard errors and convergence t-ratios
##
##                                                     Estimate   Standard   Convergence
##                                                                 Error       t-ratio
## Network Dynamics
##   1. rate basic rate parameter mynet1                6.7988  ( 1.2496  )   -0.0460
##   2. eval outdegree (density)                       -2.5810  ( 0.1505  )    0.0113
##   3. eval reciprocity                                1.9335  ( 0.2627  )    0.0205
##   4. eval 3-cycles                                   0.3941  ( 0.2742  )   -0.0658
##   5. eval transitive ties                            0.7843  ( 0.2379  )   -0.0014
##
## Behavior Dynamics
##   6. rate rate mybeh period 1                        1.1111  ( 1.3096  )   -0.0199
##   7. rate average exposure effect on rate mybeh  0.0358  ( 0.4373  )   -0.0190
##   8. eval mybeh linear shape                         0.3252  ( 0.2329  )   -0.0048
##   9. eval mybeh quadratic shape                     -0.0415  ( 0.1139  )    0.0992
##
## Overall maximum convergence ratio:     0.2163
##
##
## Total of 940 iteration steps.
```

As a rule of thumb, absolute t-values below 0.1 show good convergence, below 0.2 we say "reasonably well," and above is no convergence. Let's highlight two of the effects we have in our model

1. Transitive ties (number five) are positive 0.78 with a t-value of smaller than 0.01. Thus, we say that the network has a tendency towards transitivity (balance) that is significant.

2. Exposure effect (number seven) is also positive, but small, 0.03, but still significant (t-value of -0.01)

:Like with ERGMs, we also do goodness-of-fit:

```
sienaGOF(ans, OutdegreeDistribution, varName="mynet1") |>
  plot()
```

Goodness of Fit of OutdegreeDistribution

p: 0.584
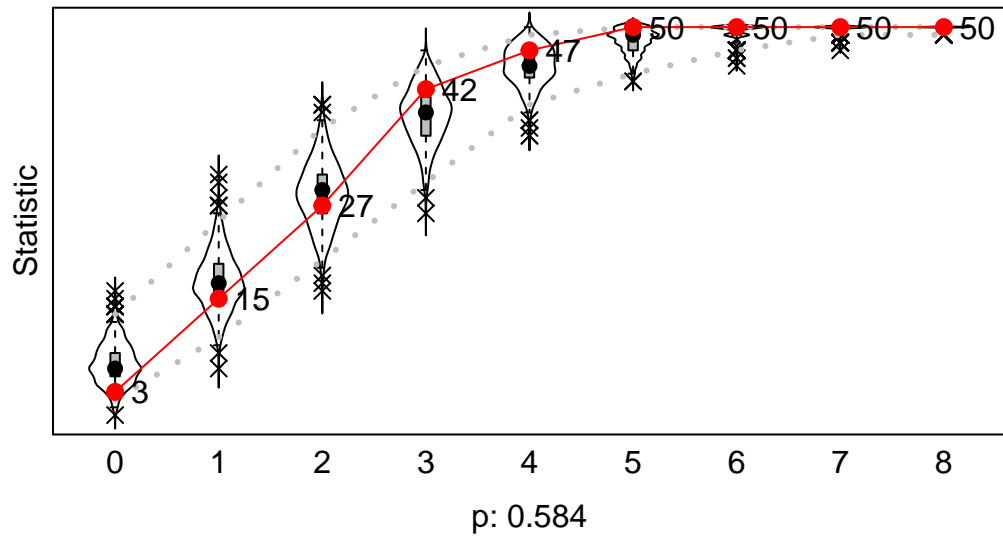
```r
sienaGOF(ans, BehaviorDistribution, varName = "mybeh") |>
  plot()
```

Note: some statistics are not plotted because their variance is 0.
This holds for the statistic: 5.

Goodness of Fit of BehaviorDistribution

p: 0.616
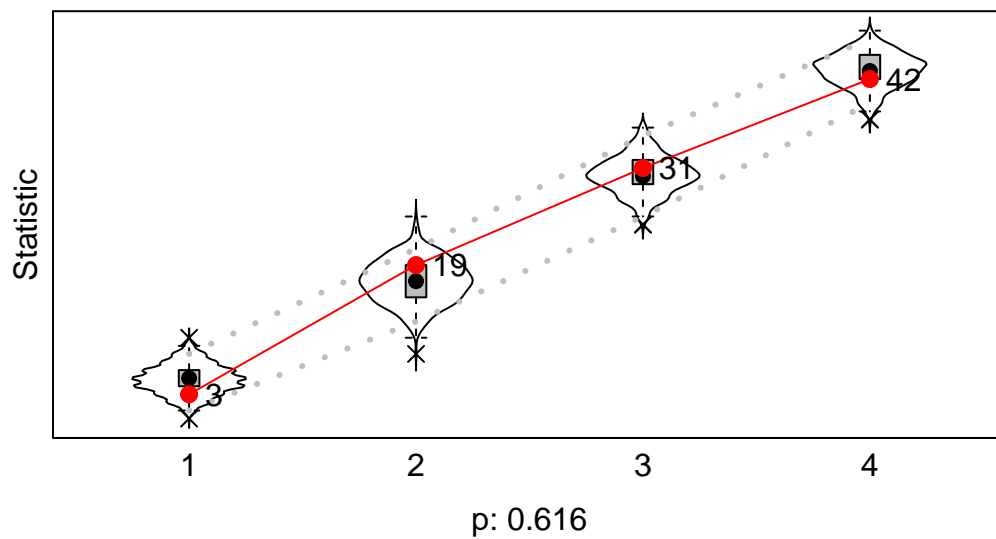
4.11 Code example: ERNM

To date, there is no CRAN release for the ERNM model. The only implementation I am aware of is one of the leading authors, which is available on GitHub: https://github.com/fellstat/ernm. Unfortunately, the current version of the package seems to be broken.

Just like the ALAAM case, as ERNMs are closely related to ERGMS, building an example using the ERGM package could be a great opportunity for a class project!

5 Lab 1

5.1 Types of problems

Overall, we can classify the problems according to two dimensions:

- **Behavior** Is the individual level behavior random or deterministic?
- **Network** Is the network random or deterministic?

Of the following cases, which ones can be treated with "regular statistical methods"?

a. Egocentric study 1: Analyze how the network structure affects health outcomes.

b. Egocentric study 2: Investigate what network structures are more prevalent in a population sample.

c. Egocentric study 3: Elucidate whether the prevalence of a given network structure is higher in one population than in another.

d. Country-level networks: Analyze whether neighboring countries tend to adopt international treaties at the same time.

e. Phylogenomics: Study a given phenotype in a population of organisms related by a phylogenetic tree.

5.2 Programming

Simulating convolutions: Using what you have learned about statistical functions in R, simulate the convolution of two normal distributions, one with $(\mu, \sigma^2) = (-3, 1)$ and the other with $(\mu, \sigma^2) = (2, 2)$. Plot the histogram of the results. Draw 1,000 samples.

```
set.seed(1)
x <- rnorm(1000, mean = -3, sd = 1)
y <- rnorm(1000, mean = 2, sd = 2)
z <- x + y

hist(z)
```

Bimodal distribution: Using the previous two normal distributions, simulate a bimodal distribution where the probability of sampling from the first distribution is 0.3 and the probability of sampling from the second distribution is 0.7. Plot the histogram of the results. (**Hint:** use a combination of `runif()` and `ifelse()`).

```r
z <- ifelse(runif(1000) < 0.3, x, y)
density(z) |> plot()
```

6 Introduction to ERGMs

Have you ever wondered if the friend of your friend is your friend? Or if the people you consider to be your friends feel the same about you? Or if age is related to whether you seek advice from others? All these (and many others certainly more creative) questions can be answered using Exponential-Family Random Graph Models.

6.1 Introduction

Exponential-Family Random Graph Models, known as ERGMs, ERG Models, or p^* models (Holland and Leinhardt (1981), Frank and Strauss (1986), Wasserman and Pattison (1996), Tom A. B. Snijders et al. (2006), Robins et al. (2007), and many others) are a large family of statistical distributions for random graphs. In ERGMs, the focus is on the processes that give origin to the network rather than the individual ties.[1]

The most common representation of ERGMs is the following:

$$P_{y,\theta}(Y = y) = \exp\left(\theta^{\mathbf{t}} s(y)\right) \kappa(\theta)^{-1}$$

where Y is a random graph, $y \in \mathcal{Y}$ is a particular realization of Y, θ is a vector of parameters, $s(y)$ is a vector of sufficient statistics, and $\kappa(\theta)$ is a normalizing constant. The normalizing constant is defined as:

$$\kappa(\theta) = \sum_{y \in \mathcal{Y}} \exp\left(\theta^{\mathbf{t}} s(y)\right)$$

From the statistical point of view, the normalizing constant makes this model attractive; only cases where $\mathcal{Y}$ is small enough to be enumerated are feasible (G. G. Vega Yon, Slaughter, and Haye 2021). Because of that reason, estimating ERGMs is a challenging task.

In more simple terms, ERG models resemble Logistic regressions. The term $\theta^{\mathbf{t}} s(y)$ is simply the sum of the product between the parameters and the statistics (like you would see in a Logistic regression,) and its key component is the composition of the vector of sufficient statistics, $s(y)$. The latter is what defines the model.

---

[1]While ERG Models can be used to predict individual ties (which is another way of describing them), the focus is on the processes that give origin to the network.

The vector of sufficient statistics dictates the data-generating process of the network. Like the mean and variance characterize the normal distribution, sufficient statistics (and corresponding parameters) characterize the ERG distribution. Figure 6.1 shows some examples of sufficient statistics with their corresponding parametrizations.
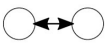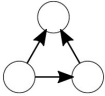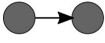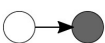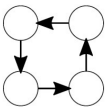
Representation	Description
	Mutual Ties (Reciprocity) $\sum_{i \neq j} y_{ij} y_{ji}$
	Transitive Triad (Balance) $\sum_{i \neq j \neq k} y_{ij} y_{jk} y_{ik}$
	Homophily $\sum_{i \neq j} y_{ij} \mathbf{1} \left(x_i = x_j \right)$
	Attribute-receiver effect $\sum_{i \neq j} y_{ij} x_j$
	Four Cycle $\sum_{i \neq j \neq k \neq l} y_{ij} y_{jk} y_{kl} y_{li}$

Figure 6.1: Example ERGM terms from G. G. Vega Yon, Slaughter, and Haye (2021)

The `ergm` package has many terms we can use to fit these models. You can explore the available terms in the manual `ergm.terms` in the `ergm` package. Take a moment to explore the manual and see the different terms available.

6.2 Dyadic independence (p1)

The simplest ERG model we can fit is a Bernoulli (Erdos-Renyi) model. Here, the only statistic is the edgecount. Now, if we can write the function computing sufficient statistics as a sum over all edges, *i.e.*, $s(y) = \sum_{ij} s'(y_{ij})$, the model reduces to a Logistic regression.

```r
library(ergm)

## Simulating a Bernoulli network
Y <- matrix(rbinom(100^2, 1, 0.1), 100, 100)
diag(Y) <- NA

cbind(
  logit = glm(as.vector(Y) ~ 1, family = binomial) |> coef(),
  ergm  = ergm(Y ~ edges) |> coef(),
  avg   = mean(Y, na.rm = TRUE) |> qlogis()
)
```

```
##                    logit      ergm       avg
## (Intercept) -2.218733 -2.218733 -2.218733
```

When we see this, we say that dyads are independent of each other; this is also called *p1* models (Holland and Leinhardt 1981). As a second example, imagine that we are interested in assessing whether gender homophily (the tendency between individuals of the same gender to connect) is present in a network. ERG models are the right tool for this task. Moreover, if we assume that gender homophily is the only mechanism that governs the network, the problem reduces to a Logistic regression:

$$P_{y,\theta}(y_{ij} = 1) = \text{Logit}^{-1}\left(\theta_{edges} + \theta_{homophily}\mathbb{1}\left(X_i = X_j\right)\right)$$

where $\mathbb{1}\left(\cdot\right)$ is the indicator function, and X_i equals one if the ith individual is female, and zero otherwise. To see this, let's simulate some data. We will use the `simulate_formula` function in the `ergm` package. All we need is a network, a model, and some arbitrary coefficients. We start with an indicator vector for gender (1: female, male otherwise):

```
## Simulating data
set.seed(7731)
X <- rbinom(100, 1, 0.5)
head(X)
```

```
[1] 0 1 0 0 0 1 0
```

Next, we use the functions from the `ergm` and `network` packages to simulate a network with homophily:

```
## Simulating the network with homophily
library(ergm)

## A network with 100 vertices
Y <- network(100)

## Adding the vertex attribute with the %v% operator
Y %v% "X" <- X

## Simulating the network
Y <- ergm::simulate_formula(
  # The model
  Y ~ edges + nodematch("X"),
  # The coefficients
  coef = c(-3, 2)
  )
```

In this case, the network tends to be sparse (negative edges coefficient) and present homophilic ties (positive nodematch coefficient). Using the `sna` package (also from the `statnet` suite), we can plot the network:

```
library(sna)
gplot(Y, vertex.col = c("green", "red")[Y %v% "X" + 1])
```
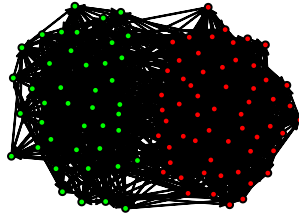

Using the ERGM package, we can fit the model using code very similar to the one used to simulate the network:

```
fit_homophily <- ergm(Y ~ edges + nodematch("X"))
```

We will check the results later and compare them against the following: the Logit model. The following code chunk implements the logistic regression model for this particular model. This example only proves that ERGMs are a generalization of Logistic regression.

> **Only to learn!**
>
> This example was created only for learning purposes. In practice, you should **always** use the `ergm` package or `PNet` software to fit ERGMs.

```
n <- 100
sstats <- summary_formula(Y ~ edges + nodematch("X"))
Y_mat <- as.matrix(Y)
diag(Y_mat) <- NA

## To speedup computations, we pre-compute this value
X_vec <- outer(X, X, "==") |> as.numeric()

## Function to compute the negative loglikelihood
obj <- \(theta) {

  # Compute the probability according to the value of Y
  p <- ifelse(
    Y_mat == 1,

    # If Y = 1
```

```r
    plogis(theta[1] + X_vec * theta[2]),

    # If Y = 0
    1 - plogis(theta[1] + X_vec * theta[2])
  )

  # The -loglikelihood
  -sum(log(p[!is.na(p)]))


}
```

And, using the `optim` function, we can fit the model:

```r
## Fitting the model
fit_homophily_logit <- optim(c(0,0), obj, hessian = TRUE)
```

Now that we have the values let's compare them:

```r
## The coefficients
cbind(
  theta_ergm  = coef(fit_homophily),
  theta_logit = fit_homophily_logit$par,
  sd_ergm     = vcov(fit_homophily) |> diag() |> sqrt(),
  sd_logit    = sqrt(diag(solve(fit_homophily_logit$hessian)))
)
```

	theta_ergm	theta_logit	sd_ergm	sd_logit
edges	-2.973331	-2.973829	0.06659648	0.06661194
nodematch.X	1.926039	1.926380	0.07395580	0.07397025

As you can see, both models yielded the same estimates because they are the same! Before continuing, let's review a couple of important results in ERGMs.

6.3 The most important results

If we were able to say what two of the most important results in ERGMs are, I would say the following: (a) conditioning on the rest of the graph, the probability of a tie distributes Logistic (Bernoulli), and (b) the ratio between two loglikelihoods can be approximated through simulation.

6.4 The logistic distribution

Let's start by stating the result: Conditioning on all graphs that are not y_{ij}, the probability of a tie Y_{ij} is distributed Logistic; formally:

$$P_{y,\theta}(Y_{ij} = 1|y_{-ij}) = \frac{1}{1 + \exp\left(\theta^{\mathbf{t}}\delta_{ij}(y)\right)},$$

where $\delta_{ij}(y) \equiv s_{ij}^{+}(y) - s_{ij}^{-}(y)$ is the change statistic, and $s_{ij}^{+}(y)$ and $s_{ij}^{-}(y)$ are the statistics of the graph with and without the tie Y_{ij}, respectively.

The importance of this result is two-fold: (a) we can use this equation to interpret fitted models in the context of a single graph (like using odds,) and (b) we can use this equation to simulate from the model, **without touching the normalizing constant**.

6.5 The ratio of loglikelihoods

The second significant result is that the ratio of loglikelihoods can be approximated through simulation. It is based on the following observation by Geyer and Thompson (1992):

$$\frac{\kappa(\theta)}{\kappa(\theta_0)} = \mathbb{E}_{y,\theta_0}\left((\theta - \theta_0)s(y)^{\mathbf{t}}\right),$$

Using the latter, we can approximate the following loglikelihood ratio:

$$l(\theta) - l(\theta_0) = (\theta - \theta_0)^{\mathbf{t}}s(y) - \log\left[\frac{\kappa(\theta)}{\kappa(\theta_0)}\right]$$

$$\approx (\theta - \theta_0)^{\mathbf{t}}s(y) - \log\left[M^{-1}\sum_{y^{(m)}}(\theta - \theta_0)^{\mathbf{t}}s(y^{(m)})\right]$$

Where θ_0 is an arbitrary vector of parameters, and $y^{(m)}$ are sampled from the distribution P_{y,θ_0}. In the words of Geyer and Thompson (1992), "[...] it is possible to approximate θ by using simulations from one distribution P_{y,θ_0} no matter which θ_0 in the parameter space is."

6.6 Start to finish example

There is no silver bullet to fit ERGMs. However, the following steps are a good starting point:

1. **Inspect the data**: Ensure the network you work with is processed correctly. Some common mistakes include isolates being excluded, vertex attributes not being adequately aligned (mismatch), etc.

2. **Start with endogenous effects first**: Before jumping into vertex/edge-covariates, try fitting models that only include structural terms such as edgecount, triangles (or their equivalent,) stars, reciprocity, etc.

3. **After structure is controlled**: You can add vertex/edge-covariates. The most common ones are homophily, nodal-activity, etc.

4. **Evaluate your results**: Once you have a model you are happy with, the last couple of steps are (a) assess convergence (which is usually done automagically by the `ergm` package,) and (b) assess goodness-of-fit, which in this context means how good was our model to capture (not-controlled for) properties of the network.

Although we could go ahead and use an existing dataset to play with, instead, we will simulate a directed graph with the following properties:

- 100 nodes.
- Homophily on Music taste.
- Gender heterophily.
- Reciprocity

The following code chunk illustrates how to do this in R. Notice that, like the previous example, we need to create a network with the vertex attributes needed to simulate homophily.

```
## Simulating the covariates (vertex attribute)
set.seed(1235)

## Simulating the data
Y <- network(100, directed = TRUE)
Y %v% "fav_music" <- sample(c("rock", "jazz", "pop"), 100, replace = TRUE)
Y %v% "female"    <- rbinom(100, 1, 0.5)
```

Now that we have a starting point for the simulation, we can use the `simulate_formula` function to get our network:

```
Y <- ergm::simulate_formula(
  Y ~
    edges +
    nodematch("fav_music") +
```

```
    nodematch("female") +
    mutual,
  coef = c(-4, 1, -1, 2)
  )

## And visualize it!
gplot(Y, vertex.col = c("green", "red")[Y %v% "female" + 1])
```
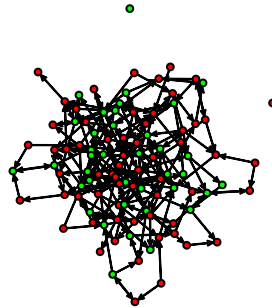

This figure is precisely why we need ERGMs (and why many Sunbelt talks don't include a network visualization!). We know the graph has structure (it's not random), but visually, it is hard to see.

6.7 Inspect the data

For the sake of time, we will not take the time to investigate our network properly. However, you should always do so. Make sure you do descriptive statistics (density, centrality, modularity, etc.), check missing values, isolates (disconnected nodes), and inspect your data visually through "notepad" and visualizations before jumping into your ERG model.

6.8 Start with endogenous effects first

The step is to check whether we can fit an ERGM or not. We can do so with the Bernoulli graph:

```
model_1 <- ergm(Y ~ edges)
summary(model_1)
```

```
Call:
ergm(formula = Y ~ edges)
```

```
Maximum Likelihood Results:

      Estimate Std. Error MCMC % z value Pr(>|z|)
edges -3.78885    0.06833       0  -55.45   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Null Deviance: 13724  on 9900  degrees of freedom
 Residual Deviance:  2102  on 9899  degrees of freedom

AIC: 2104  BIC: 2112  (Smaller is better. MC Std. Err. = 0)
```

It is rare to see a model in which the edgecount is not significant. The next term we will add is reciprocity (`mutual` in the `ergm` package)

```
model_2 <- ergm(Y ~ edges + mutual)
summary(model_2)
## Call:
## ergm(formula = Y ~ edges + mutual)
##
## Monte Carlo Maximum Likelihood Results:
##
##         Estimate Std. Error MCMC % z value Pr(>|z|)
## edges   -3.93277    0.07833      0 -50.205   <1e-04 ***
## mutual   2.15152    0.31514      0   6.827   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 13724  on 9900  degrees of freedom
##  Residual Deviance:  2064  on 9898  degrees of freedom
##
## AIC: 2068  BIC: 2083  (Smaller is better. MC Std. Err. = 0.8083)
```

As expected, reciprocity is significant (we made it like this!.) Notwithstanding, there is a difference between this model and the previous one. This model was not fitted using MLE. Instead, since the reciprocity term involves more than one tie, the model cannot be reduced to a Logistic regression, so it needs to be estimated using one of the other available estimation methods in the `ergm` package.

The model starts gaining complexity as we add higher-order terms involving more ties. An infamous example is the number of triangles. Although highly important for social sciences, including triangle terms in your ERGMs results in a degenerate model (when the MCMC chain jumps between empty and fully connected graphs). One exception is if you deal with small networks. To address this, Tom A. B. Snijders et al. (2006) and Hunter (2007) introduced various new terms that significantly reduce the risk of degeneracy. Here, we will

illustrate the use of the term "geometrically weighted dyadic shared partner" (`gwdsp`,) which Prof. David Hunter proposed. The `gwdsp` term is akin to triadic closure but reduces the chances of degeneracy.

```
## Fitting two more models (output message suppressed)
model_3 <- ergm(Y ~ edges + mutual + gwdsp(.5, fixed = TRUE))
## model_4 <- ergm(Y ~ edges + triangles) # bad idea
```

Right after fitting a model, we want to inspect the results. An excellent tool for this is the R package `texreg` (Leifeld 2013):

```
library(texreg)
screenreg(list(model_1, model_2, model_3))
```

```
============================================================
                        Model 1        Model 2        Model 3
------------------------------------------------------------
edges                   -3.79 ***      -3.93 ***      -3.84 ***
                        (0.07)         (0.08)         (0.21)
mutual                                  2.15 ***       2.14 ***
                                       (0.32)         (0.29)
gwdsp.OTP.fixed.0.5                                   -0.02
                                                      (0.05)
------------------------------------------------------------
AIC                      2104.43        2068.22        2071.94
BIC                      2111.63        2082.62        2093.54
Log Likelihood          -1051.22       -1032.11       -1032.97
============================================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

So far, `model_2` is winning. We will continue with this model.

6.9 Let's add a little bit of structure

Now that we only have a model featuring endogenous terms, we can add vertex/edge-covariates. Starting with `"fav_music"`, there are a couple of different ways to use this node feature:

a. Directly through homophily (assortative mixing): Using the `nodematch` term, we can control for the propensity of individuals to connect based on shared music taste.

b. Homophily (v2): We could activate the option `diff = TRUE` using the same term. By doing this, the homophily term is operationalized differently, adding as many terms as options in the vertex attribute.

c. Mixing: We can use the term `nodemix` for individuals' tendency to mix between musical tastes.

> **Question**
>
> In this context, what would be the different hypotheses behind each decision?

```r
model_4 <- ergm(Y ~ edges + mutual + nodematch("fav_music"))
model_5 <- ergm(Y ~ edges + mutual + nodematch("fav_music", diff = TRUE))
model_6 <- ergm(Y ~ edges + mutual + nodemix("fav_music"))
```

Now, let's inspect what we have so far:

```r
screenreg(list(`2` = model_2, `4` = model_4, `5` = model_5, `6` = model_6))
```

```
==========================================================================
                            2            4            5            6
--------------------------------------------------------------------------
edges                                 -3.93 ***    -4.29 ***    -4.29 ***    -3.56 ***
                                      (0.08)       (0.11)       (0.11)       (0.24)
mutual                                 2.15 ***     1.99 ***     2.00 ***     2.02 ***
                                      (0.32)       (0.30)       (0.30)       (0.30)
nodematch.fav_music                                 0.85 ***
                                                   (0.14)
nodematch.fav_music.jazz                                         0.74 **
                                                                (0.25)
nodematch.fav_music.pop                                          0.82 ***
                                                                (0.18)
nodematch.fav_music.rock                                         0.87 ***
                                                                (0.17)
mix.fav_music.pop.jazz                                                       -0.54
                                                                            (0.34)
mix.fav_music.rock.jazz                                                      -0.75 *
                                                                            (0.38)
mix.fav_music.jazz.pop                                                       -0.60
                                                                            (0.35)
mix.fav_music.pop.pop                                                         0.10
                                                                            (0.27)
mix.fav_music.rock.pop                                                       -0.50
```

```
                                                                   (0.31)
mix.fav_music.jazz.rock                                           -1.40 **
                                                                   (0.44)
mix.fav_music.pop.rock                                            -0.86 *
                                                                   (0.33)
mix.fav_music.rock.rock                                            0.15
                                                                   (0.27)
-----------------------------------------------------------------------------
AIC                           2068.22        2030.85        2033.57       2036.37
BIC                           2082.62        2052.45        2069.57       2108.38
Log Likelihood               -1032.11       -1012.42       -1011.79      -1008.19
=============================================================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

Although model 5 has a higher loglikelihood, using AIC or BIC suggests model 4 is a better candidate. For the sake of time, we will jump ahead and add `nodematch("female")` as the last term of our model. The next step is to assess (a) convergence and (b) goodness-of-fit.

```r
model_final <- ergm(Y ~ edges + mutual + nodematch("fav_music") + nodematch("female"))

## Printing the pretty table
screenreg(list(`2` = model_2, `4` = model_4, `Final` = model_final))
```

```
============================================================
                        2              4              Final
------------------------------------------------------------
edges                 -3.93 ***      -4.29 ***      -3.95 ***
                      (0.08)         (0.11)         (0.12)
mutual                 2.15 ***       1.99 ***       1.86 ***
                      (0.32)         (0.30)         (0.33)
nodematch.fav_music                   0.85 ***       0.81 ***
                                     (0.14)         (0.14)
nodematch.female                                    -0.74 ***
                                                    (0.15)
------------------------------------------------------------
AIC                   2068.22        2030.85        2002.18
BIC                   2082.62        2052.45        2030.98
Log Likelihood       -1032.11       -1012.42        -997.09
============================================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

6.10 Conclusion

We have shown here just a glimpse of what ERG models are. A large, active, collaborative community of social network scientists is working on new extensions and advances. If you want to learn more about ERGMs, I recommend the following resources:

- The Statnet website (link).

- The book "Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications" by Lusher, Koskinen, and Robins (2013).

- Melnet's PNEt website (link).

Part II

Day 2

7 Convergence on ERGMs

Let's continue what we left off in the previous session: Evaluating our ERGMs. Like most models, ERGMs are a mix between art and science. Here is again a list of steps/considerations to have when fitting ERGMs:

1. **Inspect the data**.

2. **Start with endogenous effects first**.

3. **After structure is controlled**.

4. **Evaluate your results**: Once you have a model you are happy with, the last couple of steps are (a) assess convergence (which is usually done automagically by the `ergm` package,) and (b) assess goodness-of-fit, which in this context means how good was our model to capture (not-controlled for) properties of the network.

We are in the last step. Let's recall what was our final model with the simulated data (code folded intentionally):

```r
## Loading the packages
library(ergm)
library(sna)

## Simulating the covariates (vertex attribute)
set.seed(1235)

## Simulating the data
Y <- network(100, directed = TRUE)
Y %v% "fav_music" <- sample(c("rock", "jazz", "pop"), 100, replace = TRUE)
Y %v% "female"    <- rbinom(100, 1, 0.5)

## Simulating the ERGM
Y <- ergm::simulate_formula(
  Y ~
    edges +
    nodematch("fav_music") +
    nodematch("female") +
    mutual,
  coef = c(-4, 1, -1, 2)
```

```r
  )

gplot(Y, vertex.col = c("green", "red")[Y %v% "female" + 1])
```
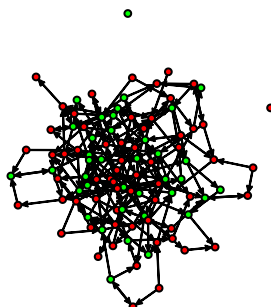


```r
model_final <- ergm(Y ~ edges + mutual + nodematch("fav_music") + nodematch("female")

summary(model_final)
```

```
Call:
ergm(formula = Y ~ edges + mutual + nodematch("fav_music") +
    nodematch("female"))

Monte Carlo Maximum Likelihood Results:

                     Estimate Std. Error MCMC %  z value Pr(>|z|)
edges                 -3.9573     0.1112       0 -35.588   <1e-04 ***
mutual                 1.8950     0.2853       0   6.643   <1e-04 ***
nodematch.fav_music    0.8295     0.1291       0   6.426   <1e-04 ***
nodematch.female      -0.7436     0.1379       0  -5.394   <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 13724  on 9900  degrees of freedom
 Residual Deviance:  1994  on 9896  degrees of freedom

AIC: 2002   BIC: 2031   (Smaller is better. MC Std. Err. = 0.8189)
```

7.1 Convergence

As our model was fitted using MCMC, we must ensure the chains converged. We can use the `mcmc.diagnostics` function from the `ergm` package to check model convergence. This

function looks at the last set of simulations of the MCMC model and generates various diagnostics for the user.

Under the hood, the fitting algorithm generates a stream of networks based on those parameters for each new proposed set of model parameters. The last stream of networks is thus simulated using the final state of the model. The `mcmc.diagnostics` function takes that stream of networks and plots the sequence of the sufficient statistics included in the model. A converged model should show a stationary statistics sequence, moving around a fixed value without (a) becoming stuck at any point and (b) chaining the tendency. This model shows both:

```
mcmc.diagnostics(model_final, which  = c("plots"))
```
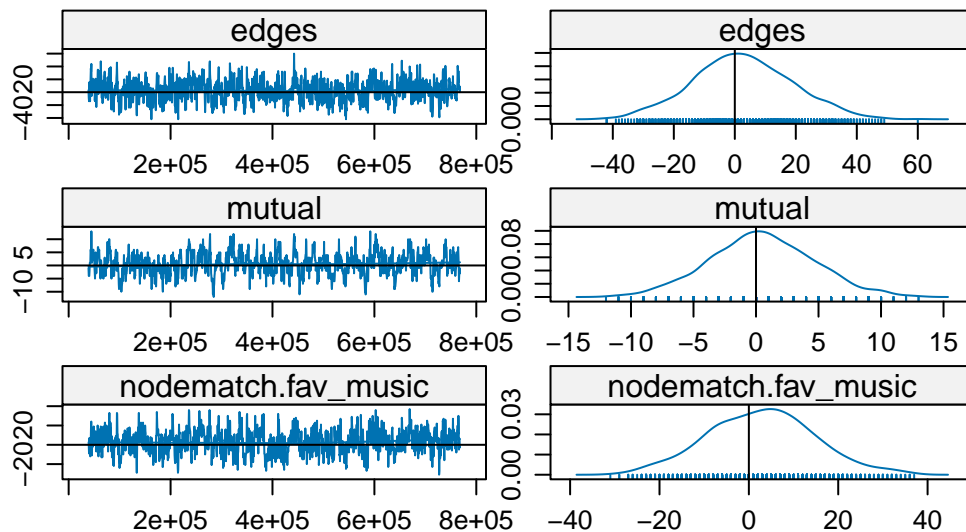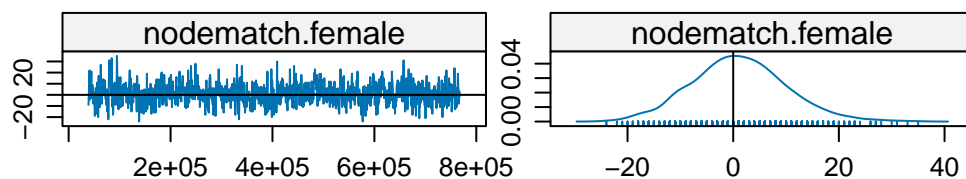

Note: MCMC diagnostics shown here are from the last round of
  simulation, prior to computation of final parameter estimates.
  Because the final estimates are refinements of those used for this
  simulation run, these diagnostics may understate model performance.
  To directly assess the performance of the final model on in-model
  statistics, please use the GOF command: gof(ergmFitObject,
  GOF=~model).

Now that we know our model was good enough to represent the observed statistics (sample them, actually,) let's see how good it is at capturing other features of the network that were not included in the model.

7.2 Goodness-of-fit

This would be the last step in the sequence of steps to fit an ERGM. As we mentioned before, the idea of Goodness-of-fit [GOF] in ERG models is to see how well our model captures other properties of the graph that were not included in the model. By default, the `gof` function from the `ergm` package computes GOF for:

a. The model statistics.

b. The outdegree distribution.

c. The indegree distribution.

d. The distribution of edge-wise shared partners.

e. The distribution of the geodesic distances (shortest path).

The process of evaluating GOF is relatively straightforward. Using samples from the posterior distribution, we check whether the observed statistics from above are covered (fall within the CI) of our model. If they do, we say that the model has a good fit. Otherwise, if we observe significant anomalies, we return to the bench and try to improve our model.

As with all simulated data, our `gof()` call shows that our selected model was an excellent choice for the observed graph:

```
gof_final <- gof(model_final)
print(gof_final)
##
## Goodness-of-fit for in-degree
##
##           obs min  mean max MC p-value
## idegree0   11   3 10.93  20       1.00
## idegree1   22  16 24.80  34       0.58
## idegree2   23  17 26.54  36       0.52
## idegree3   30   9 19.41  31       0.02
## idegree4   10   4 10.94  20       0.94
## idegree5    3   1  4.97  11       0.46
## idegree6    1   0  1.68   5       0.96
## idegree7    0   0  0.58   3       1.00
## idegree8    0   0  0.14   1       1.00
## idegree10   0   0  0.01   1       1.00
##
```

```
## Goodness-of-fit for out-degree
##
##             obs min   mean max MC p-value
## odegree0    10    3 10.71  18         1.00
## odegree1    30   13 24.67  35         0.28
## odegree2    23   18 27.50  40         0.38
## odegree3    17   12 19.01  28         0.74
## odegree4    10    3 10.72  18         1.00
## odegree5     8    1  4.84  10         0.12
## odegree6     2    0  1.79   6         1.00
## odegree7     0    0  0.57   3         1.00
## odegree8     0    0  0.15   2         1.00
## odegree9     0    0  0.03   1         1.00
## odegree10    0    0  0.01   1         1.00
##
## Goodness-of-fit for edgewise shared partner
##
##             obs min    mean max MC p-value
## esp.OTP0 212 178 209.21 241         0.78
## esp.OTP1   7    3  10.47  22         0.44
## esp.OTP2   0    0   0.40   3         1.00
##
## Goodness-of-fit for minimum geodesic distance
##
##        obs min    mean   max MC p-value
## 1      219 190  220.08   255         0.94
## 2      449 326  459.25   619         0.98
## 3      790 499  842.85  1183         0.90
## 4     1151 708 1252.11  1960         0.80
## 5     1245 775 1396.75  2266         0.66
## 6     1043 656 1181.38  1680         0.44
## 7      745 499  802.34  1211         0.72
## 8      434 194  466.85   791         0.86
## 9      198  57  244.78   569         0.72
## 10      80  11  121.93   408         0.76
## 11      10   1   60.50   285         0.42
## 12       1   0   30.83   251         0.46
## 13       0   0   15.96   186         0.74
## 14       0   0    8.42   133         1.00
## 15       0   0    5.02    96         1.00
## 16       0   0    3.03    65         1.00
## 17       0   0    1.83    60         1.00
## 18       0   0    1.05    47         1.00
```

```
## 19       0   0    0.54    23        1.00
## 20       0   0    0.18     7        1.00
## 21       0   0    0.07     3        1.00
## 22       0   0    0.01     1        1.00
## Inf 3535 776 2784.24 4680          0.38
##
## Goodness-of-fit for model statistics
##
##                          obs min    mean max MC p-value
## edges                    219 190 220.08 255       0.94
## mutual                    16   9  16.36  25       1.00
## nodematch.fav_music 123   97 123.55 152       1.00
## nodematch.female          72  49  71.53  90       0.90
```
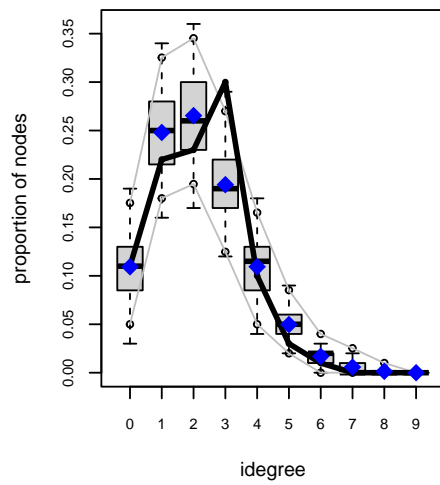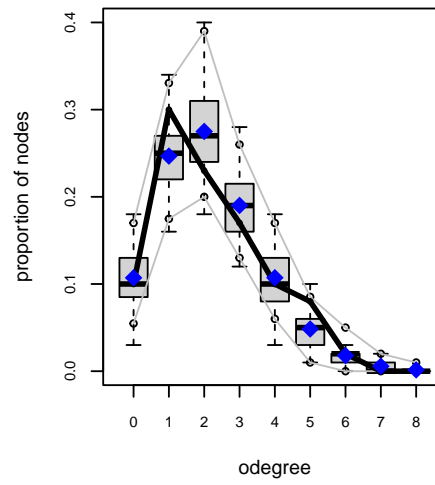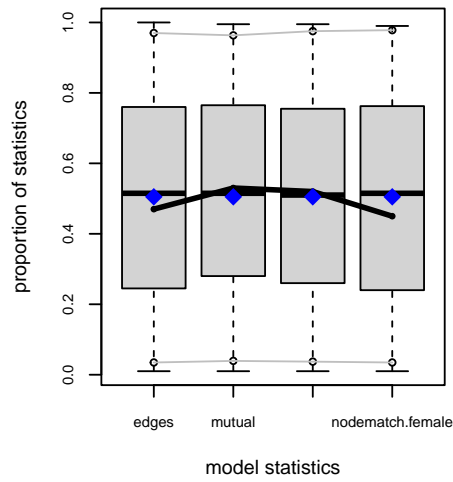
It is easier to see the results using the `plot` function:

```
## Plotting the result (5 figures)
op <- par(mfrow = c(3,2))
plot(gof_final)
par(op)
```

Goodness−of−fit diagnostics
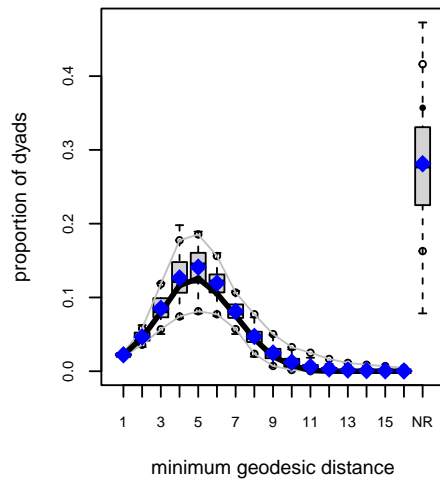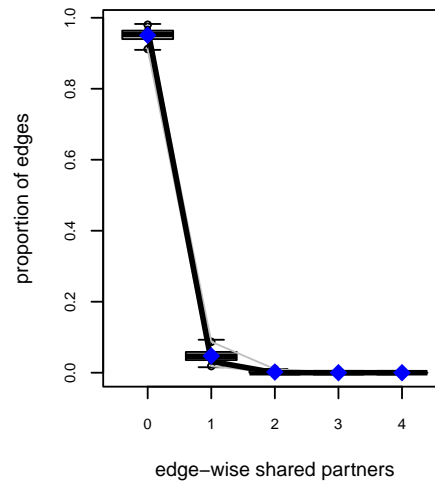
8 Odd balls

Other common scenarios involve more convoluted/complex questions. For instance, in the case of dyadic behavior Bell et al. (2019).

In Tanaka and Vega Yon (2024), we study the prevalence of perception-based network motifs. While the ERGM framework would be a natural choice, as a first approach, we used non-parametric tests for hypothesis testing.

9 Lab II

The `ergm` package comes with a handful of vignettes (extended R examples) that you can use to learn more about the package. The vignette with the same name as the package shows an example fitting a model to the network `faux.mesa.high`. Address the following questions:

a. What type of ERG model is this?

b. Looking at the plot in the vignette, what other term(s) could you consider worth adding to the model? Why?

c. The vignette did not include a goodness-of-fit analysis. Can you add one? What do you find?

d. **Challenge**: without using `ergm` to fit the model, estimate the following $p1$ model: `~ edges + nodefactor("Race")`. Compare your results with what you get using `ergm`.

10 Advanced ERGMs: Constraints

10.1 Introduction

For this section, we will dive in into ERGM constranints. Using constraints, you will be able to modify the sampling space of the model to things such as:

- Pool (multilevel) models.
- Accounting for data generating process.
- Make your model behave (with caution.)
- Even fit Discrete-Exponential Family Models (DEFM.)

We will start with formally understanding what constraining the space means and then continue with some examples.

10.2 Constraining ERGMs

Exponential Random Graph Models [ERGMs] can represent a variety of network classes. We often look at "regular" social networks like school students, colleagues in the workplace, or families. Nonetheless, some social networks we study have features that restrict how connections can occur. Typical examples are bi-partite graphs and multilevel networks. There are two classes of vertices in bi-partite networks, and ties can only occur between classes. On the other hand, multilevel networks may feature multiple classes with inter-class ties that are somewhat restricted. Structural constraints exist in both cases, meaning some configurations may not be plausible.

Mathematically, what we are trying to do is, instead of assuming that all network configurations are possible:

$$\left\{ y \in \mathcal{Y} : y_{ij} = 0, \forall i = j \right\}$$

we want to go a bit further avoiding loops, namely:

$$\left\{ y \in \mathcal{Y} : y_{ij} = 0, \forall i = j; \mathbf{y} \in C \right\},$$

where C is a constraint, for example, only networks with no triangles. The `ergm` R package has built-in capabilities to deal with some of these cases. Nonetheless, we can specify models with arbitrary structural constraints built into the model. The key is in using offset terms.

10.3 Example 1: Pool (block-diagonal/multilevel) ERGM

Nowadays, multi-network studies featuring a large sample of networks are easy to see. When we deal with multiple networks, there are usually two approaches to analyzing them: (a) estimate separate ERGMs and do a meta-analysis like in [ANN CITATION], or (b) fit a pooled ERGM. Here, we will follow the latter.

Pooled ERGMs resemble typical statistical methods. Assuming that the networks are independent, we can estimate a model like the following:

$$P_{y,\theta}(Y_1 = y_1, Y_2 = y_2, ..., Y_1 = y_L) = \prod_{l}^{L} \exp\left(\theta^{\mathbf{t}} s(y_l)\right) \kappa_l(\theta)^{-1}$$

In this case, we assume that all networks have the same data-generating process, meaning they all come from the same $\mathcal{Y}$. If that's not the case and, like in any regression analysis, we could write a hierarchical model where model parameters come from the same distribution or fit a random effects ERGM as described in Slaughter and Koehly (2016).

We have a handful of different ways of fitting pooled ERGMs. If you are dealing with small networks (about six nodes if directed and eight if undirected,) you can use the `ergmito` R package (G. G. Vega Yon, Slaughter, and Haye 2021; G. Vega Yon 2020). The `ergm` package can still assist you with simple models if you have larger networks. But if you want to leverage the power of having multiple networks, the recently published `ergm.multi` R package should be your choice (Krivitsky, Coletti, and Hens 2023a; Krivitsky 2023). Since we are learning to use constraints, we will use the `ergm` package. Let's start with an example from the `ergmito` R package:

```
library(sna)
library(ergm)
library(ergmito)

## Loading five artificial networks (5 networks)
data(fivenets)

## Taking a look at the first two
fivenets[1:2]
## [[1]]
##  Network attributes:
##    vertices = 4
```

```
##    directed = TRUE
##    hyper = FALSE
##    loops = FALSE
##    multiple = FALSE
##    bipartite = FALSE
##    total edges= 2
##      missing edges= 0
##      non-missing edges= 2
##
##  Vertex attribute names:
##      female name
##
## No edge attributes
##
## [[2]]
##  Network attributes:
##    vertices = 4
##    directed = TRUE
##    hyper = FALSE
##    loops = FALSE
##    multiple = FALSE
##    bipartite = FALSE
##    total edges= 7
##      missing edges= 0
##      non-missing edges= 7
##
##  Vertex attribute names:
##      female name
##
## No edge attributes
```
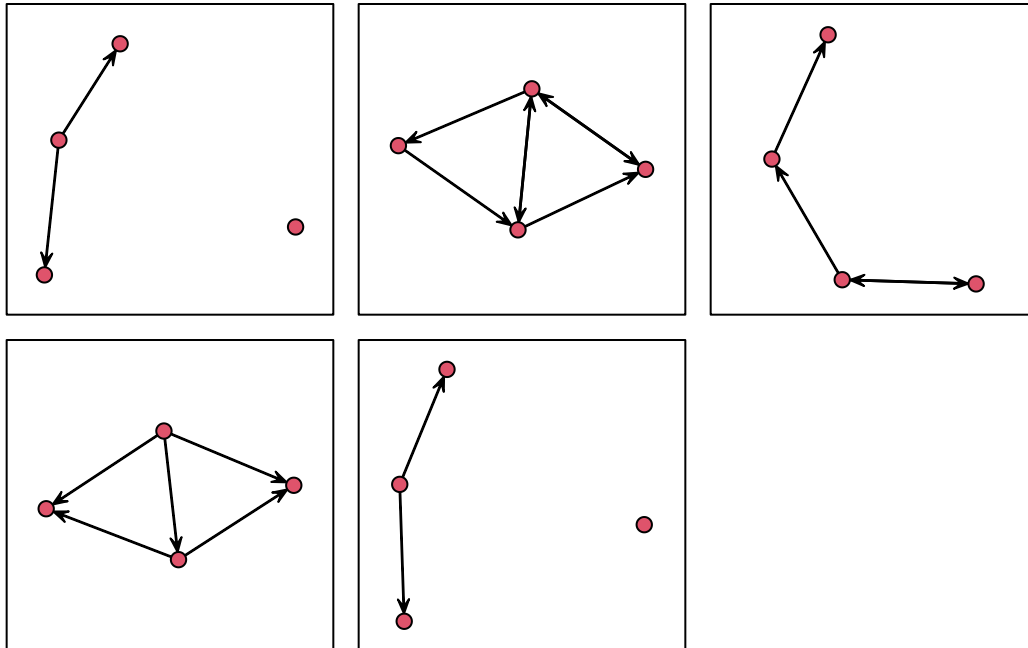
We can also visualize these five networks:

```
## Quickly visualizing fine networks
op <- par(mfrow = c(2, 3), mar = c(1,1,1,1)/2)
res <- lapply(fivenets, \(x) {gplot(x); box()})
par(op)
```

These five networks were generated using an edges+gender homophily model. Since all the graphs in the list are small (five vertices each,) we can fit a pooled ERGM using the `ergmito` function from the package of the same name:

```r
summary(ergmito(fivenets ~ edges + nodematch("female")))
```

```
ERGMito estimates (MLE)
This model includes 5 networks.

formula:
  fivenets ~ edges + nodematch("female")

                 Estimate Std. Error z value Pr(>|z|)
edges            -1.70475    0.54356 -3.1363 0.001711 **
nodematch.female  1.58697    0.64305  2.4679 0.013592 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
AIC: 73.34109    BIC: 77.52978    (Smaller is better.)
```

Who cares about small networks?

You may be wondering why someone would care about small networks. Well, the truth is that small networks are the fabric of societies (think about family nucleous,) and are more complicated than you think. A directed graph with five vertices has about

> 1,000,000 possible configurations.

> **HPC in small networks**
>
> The key of ERGMitos is that, with a small enough support size, we can skip using MCMC and simulation-based approximations and directly calculate likelihoods for pooled models. This is extremely powerful as it opens the door to diverse statistical methods.

The question is: How can we use constraints to fit the same model using the `ergm` package? We have to follow these steps:

1. We must combine all the networks into a single network object.

2. Specify the constraint via the `constraint` argument in the `ergm` function. In this case, use the `blockdiag` constraint (check out the manual `?ergm.constraints` to learn more about what's available with the tool).

3. Fit the model!

The `ergmito` package has a convenient function that will take any list of network objects (or matrices) and turn it into a single network. The passed networks don't need to have the same size. After combining the networks, the function creates a new vertex attribute that holds the id of the original group.

```
## Combining the networks
bd <- blockdiagonalize(fivenets)
bd
```

```
 Network attributes:
  vertices = 20
  directed = TRUE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 20
    missing edges= 0
    non-missing edges= 20

 Vertex attribute names:
    block female name_original vertex.names

No edge attributes
```

Just out of curiosity, what would we get if we fitted a regular ERGM to this data? Here is the answer:

```
ergm(bd ~ edges + nodematch("female"))
```

```
Call:
ergm(formula = bd ~ edges + nodematch("female"))

Maximum Likelihood Coefficients:
         edges  nodematch.female
        -3.882            1.542
```

i.e., biased estimates. Although the homophily term is very similar to the original estimate, the edges parameter is twice the size. The estimate is inflated because the support of this model includes networks where there are ties between the groups, yet these networks cannot be connected one another.

To solve this issue, as we said, we use the constraints:

```
ergm(bd ~ edges + nodematch("female"), constraints = ~ blockdiag("block"))
```

```
Call:
ergm(formula = bd ~ edges + nodematch("female"), constraints = ~blockdiag("block"))

Maximum Likelihood Coefficients:
         edges  nodematch.female
        -1.705            1.587
```

Success! This model is now correctly specified. Although it seems like we have a single network, by imposing the blockdiagonal constraint, we are fitting a pooled model. The `ergm` package comes with more alternatives to restricting the sample space of your model, but there may be a case where you need more. For those cases, the offset comes in handy.

10.4 Example 2: Interlocking egos and disconnected alters[1]

Imagine that we have two sets of vertices. The first, group `E`, are egos in an egocentric study. The second group, called `A`, is composed of people mentioned by egos in `E` but were not surveyed. Assume that individuals in `A` can only connect to individuals in `E`; moreover,

---

[1]Thanks to Laura Koehly, who devised this complicated model.

individuals in E have no restrictions on connecting. In other words, only two types of ties exist: E-E and A-E. The question is now, how can we enforce such a constraint in an ERGM?

Using offsets, and in particular, setting coefficients to -Inf provides an easy way to restrict the support set of ERGMs. For example, if we wanted to constrain the support to include networks with no triangles, we would add the term offset(triangle) and use the option offset.coef = -Inf to indicate that realizations including triangles are not possible. Using R:

```
ergm(net ~ edges + offset(triangle), offset.coef = -Inf)
```

In this model, a Bernoulli graph, we reduce the sample space to networks with no triangles. In our example, such a statistic should only take non-zero values whenever ties within the A class happen. We can use the nodematch() term to do that. Formally

$$\text{NodeMatch}(x) = \sum_{i,j} y_{ij} \mathbf{1}(x_i = x_j)$$

This statistic will sum over all ties in which source (i) and target (j)'s X attribute is equal. One way to make this happen is by creating an auxiliary variable that equals, e.g., 0 for all vertices in A, and a unique value different from zero otherwise. For example, if we had 2 As and three Es, the data would look like $\{0, 0, 1, 2, 3\}$. The following code block creates an empty graph with 50 nodes, 10 of which are in group E (ego).

```
library(ergm, quietly =  TRUE)
library(sna, quietly =  TRUE)

n <- 50
n_egos <- 10
net <- as.network(matrix(0, ncol = n, nrow = n), directed = TRUE)

## Let's assing the groups
net %v% "is.ego" <- c(rep(TRUE, n_egos), rep(FALSE, n - n_egos))
net %v% "is.ego"
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[49] FALSE FALSE
```

```
gplot(net, vertex.col = net %v% "is.ego")
```
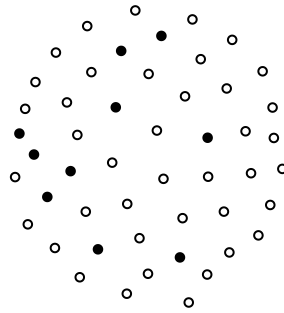
To create the auxiliary variable, we will use the following function:

```r
## Function that creates an aux variable for the ergm model
make_aux_var <- function(my_net, is_ego_dummy) {

  n_vertex <- length(my_net %v% is_ego_dummy)
  n_ego_   <- sum(my_net %v% is_ego_dummy)

  # Creating an auxiliary variable to identify the non-informant non-informant ties
  my_net %v% "aux_var" <- ifelse(
    !my_net %v% is_ego_dummy, 0, 1:(n_vertex - n_ego_)
    )

  my_net
}
```

Calling the function in our data results in the following:

```r
net <- make_aux_var(net, "is.ego")

## Taking a look over the first 15 rows of data
cbind(
  Is_Ego = net %v% "is.ego",
  Aux    = net %v% "aux_var"
) |> head(n = 15)
```

```
     Is_Ego Aux
[1,]      1   1
[2,]      1   2
[3,]      1   3
[4,]      1   4
[5,]      1   5
[6,]      1   6
[7,]      1   7
```

```
 [8,]       1   8
 [9,]       1   9
[10,]       1  10
[11,]       0   0
[12,]       0   0
[13,]       0   0
[14,]       0   0
[15,]       0   0
```

We can now use this data to simulate a network in which ties between `A`-class vertices are not possible:

```
set.seed(2828)
net_sim <- simulate(net ~ edges + nodematch("aux_var"), coef = c(-3.0, -Inf))
gplot(net_sim, vertex.col = net_sim %v% "is.ego")
```
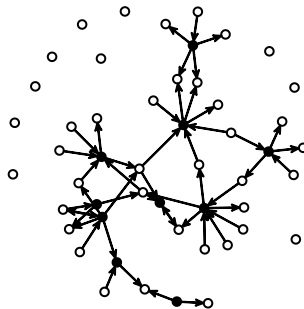

As you can see, this network has only ties of the type `E-E` and `A-E`. We can double-check by (i) looking at the counts and (ii) visualizing each induced-subgraph separately:

```
summary(net_sim ~ edges + nodematch("aux_var"))
```

```
          edges nodematch.aux_var
             49                 0
```

```
net_of_alters <- get.inducedSubgraph(
  net_sim, which((net_sim %v% "aux_var") == 0)
  )

net_of_egos <- get.inducedSubgraph(
  net_sim, which((net_sim %v% "aux_var") != 0)
  )

## Counts
```
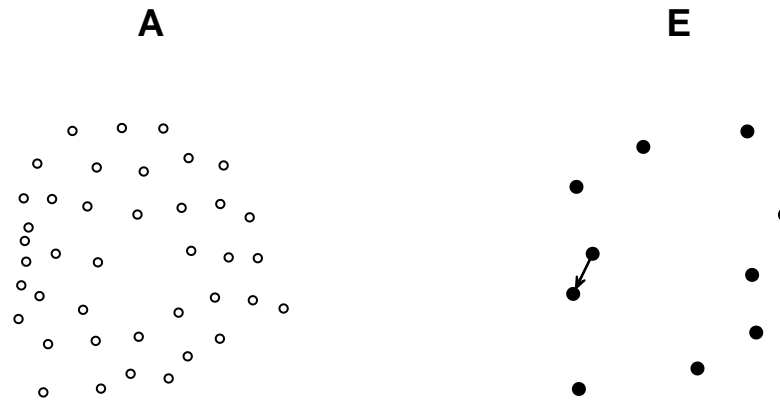
```r
summary(net_of_alters ~ edges + nodematch("aux_var"))
```

```
        edges nodematch.aux_var
            0                 0
```

```r
summary(net_of_egos ~ edges + nodematch("aux_var"))
```

```
        edges nodematch.aux_var
            1                 0
```

```r
## Figures
op <- par(mfcol = c(1, 2))
gplot(net_of_alters, vertex.col = net_of_alters %v% "is.ego", main = "A")
gplot(net_of_egos, vertex.col = net_of_egos %v% "is.ego", main = "E")
```



```r
par(op)
```

Now, to fit an ERGM with this constraint, we simply need to make use of the offset terms. Here is an example:

```r
ans <- ergm(
  net_sim ~ edges + offset(nodematch("aux_var")), # The model (notice the offset)
  offset.coef = -Inf                              # The offset coefficient
  )
## Starting maximum pseudolikelihood estimation (MPLE):
## Obtaining the responsible dyads.
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
```

```
## Finished MPLE.
## Evaluating log-likelihood at the estimate.
summary(ans)
## Call:
## ergm(formula = net_sim ~ edges + offset(nodematch("aux_var")),
##     offset.coef = -Inf)
##
## Maximum Likelihood Results:
##
##                                 Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                             -2.843      0.147      0  -19.34   <1e-04 ***
## offset(nodematch.aux_var)           -Inf      0.000      0    -Inf   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 1233.8  on 890  degrees of freedom
##  Residual Deviance:  379.4  on 888  degrees of freedom
##
## AIC: 381.4  BIC: 386.2  (Smaller is better. MC Std. Err. = 0)
##
##  The following terms are fixed by offset and are not estimated:
##    offset(nodematch.aux_var)
```

This ERGM model–which by the way only featured dyadic-independent terms, and thus can be reduced to a logistic regression–restricts the support by excluding all networks in which ties within the class A exists. To finalize, let's look at a few simulations based on this model:

```
set.seed(1323)
op <- par(mfcol = c(2,2), mar = rep(1, 4))
for (i in 1:4) {
  gplot(simulate(ans), vertex.col = net %v% "is.ego", vertex.cex = 2)
  box()
}
```
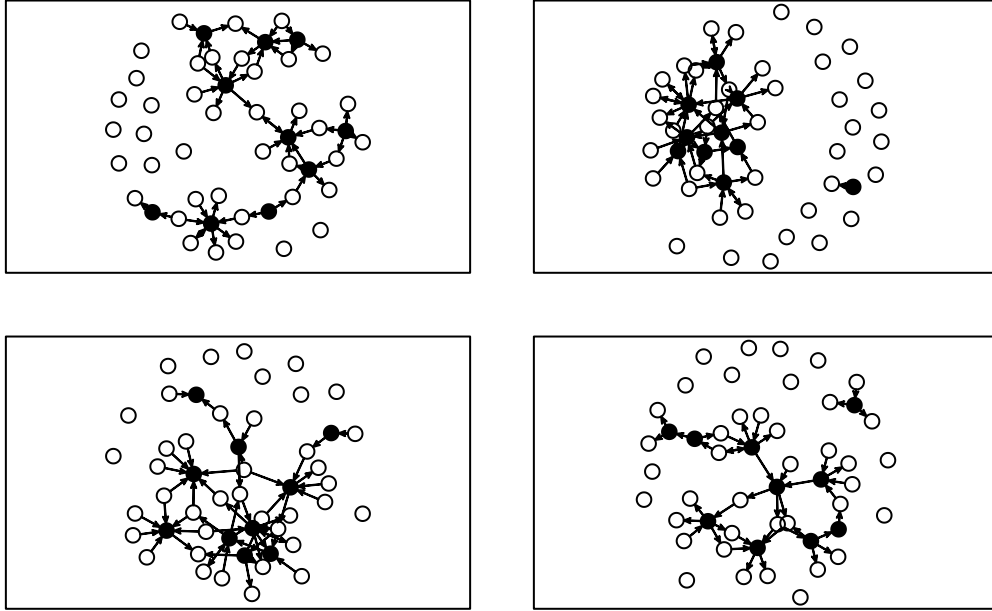
```
par(op)
```

All networks with no ties between `A` nodes.

10.5 Example 3: Bi-partite networks

In the case of bipartite networks (sometimes called affiliation networks,) we can use `ergm`'s terms for bipartite graphs to corroborate what we discussed here. For example, the two-star term. Let's start simulating a bipartite network using the `edges` and `two-star` parameters. Since the `k-star` term is usually complex to fit (tends to generate degenerate models,) we will take advantage of the `Log()` transformation function in the `ergm` package to smooth the term.[2]

The bipartite network that we will be simulating will have 100 actors and 50 entities. Actors, which we will map to the first level of the `ergm` terms, this is, `b1star b1nodematch`, etc. will send ties to the entities, the second level of the bipartite ERGM. To create a bipartite network, we will create an empty matrix of size `nactors x nentitites`; thus, actors are represented by rows and entities by columns.

---

[2]After writing this example, it became apparent the use of the `Log()` transformation function may not be ideal. Since many terms used in ERGMs can be zero, e.g., triangles, the term `Log(~ ostar(2))` is undefined when `ostar(2) = 0`. In practice, the ERGM package sets a lower limit for the log of 0, so, instead of having `Log(0) ~ -Inf`, they set it to be a really large negative number. This causes all sorts of issues to the estimates; in our example, an overestimation of the population parameter and a positive log-likelihood. Therefore, I wouldn't recommend using this transformation too often.

```r
## Parameters for the simulation
nactors   <- 100
nentities <- floor(nactors/2)
n         <- nactors + nentities

## Creating an empty bipartite network (baseline)
net_b <- network(
  matrix(0, nrow = nactors, ncol = nentities), bipartite = TRUE
)

## Simulating the bipartite ERGM,
net_b <- simulate(net_b ~ edges + Log(~b1star(2)), coef = c(-3, 1.5), seed = 55)
```
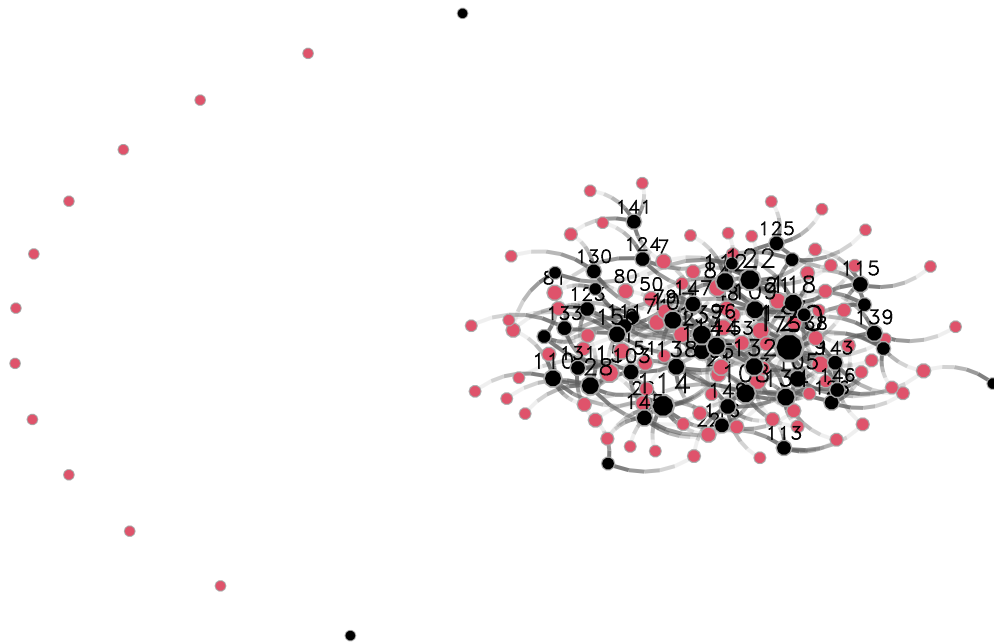
Let's see what we got here:

```r
summary(net_b ~ edges + Log(~b1star(2)))
```

```
      edges Log~b1star2
245.000000    5.746203
```

```r
netplot::nplot(net_b, vertex.col = (1:n <= nactors) + 1)
```


Notice that the first `nactors` vertices in the network are the actors, and the remaining are the entities. Now, although the `ergm` package features bipartite network terms, we can

still fit a bipartite ERGM without explicitly declaring the graph as such. In such case, the `b1star(2)` term of a bipartite network is equivalent to an `ostar(2)` in a directed graph. Likewise, `b2star(2)` in a bipartite graph matches the `istar(2)` term in a directed graph. This information will be relevant when fitting the ERGM. Let's transform the bipartite network into a directed graph. The following code block does so:

```
## Identifying the edges
net_not_b <- which(as.matrix(net_b) != 0, arr.ind = TRUE)

## We need to offset the endpoint of the ties by nactors
## so that the ids go from 1 through (nactors + nentitites)
net_not_b[,2] <- net_not_b[,2] + nactors

## The resulting graph is a directed network
net_not_b <- network(net_not_b, directed = TRUE)
```

Now we are almost done. As before, we need to use node-level covariates to put the constraints in our model. For this ERGM to reflect an ERGM on a bipartite network, we need two constraints:

1. Only ties from actors to entities are allowed, and
2. entities can only receive ties.

The corresponding offset terms for this model are: `nodematch("is.actor") ~ -Inf`, and `nodeocov("isnot.actor") ~ -Inf`. Mathematically:

$$\text{NodeMatch}(\text{x} = \text{"is.actor"}) = \sum_{i<j} y_{ij} \mathbb{1}\left(x_i = x_j\right)$$
$$\text{NodeOCov}(\text{x} = \text{"isnot.actor"}) = \sum_i x_i \times \sum_{j<i} y_{ij}$$

In other words, we are setting that ties between nodes of the same class are forbidden, and outgoing ties are forbidden for entities. Let's create the vertex attributes needed to use the aforementioned terms:

```
net_not_b %v% "is.actor" <- as.integer(1:n <= nactors)
net_not_b %v% "isnot.actor" <- as.integer(1:n > nactors)
```

Finally, to make sure we have done all well, let's look how both networks–bipartite and unimodal–look side by side:

```
## First, let's get the layout
fig <- netplot::nplot(net_b, vertex.col = (1:n <= nactors) + 1)
gridExtra::grid.arrange(
```

```
    fig,
    netplot::nplot(
      net_not_b, vertex.col = (1:n <= nactors) + 1,
      layout = fig$.layout
      ),
    ncol = 2, nrow = 1
  )
```
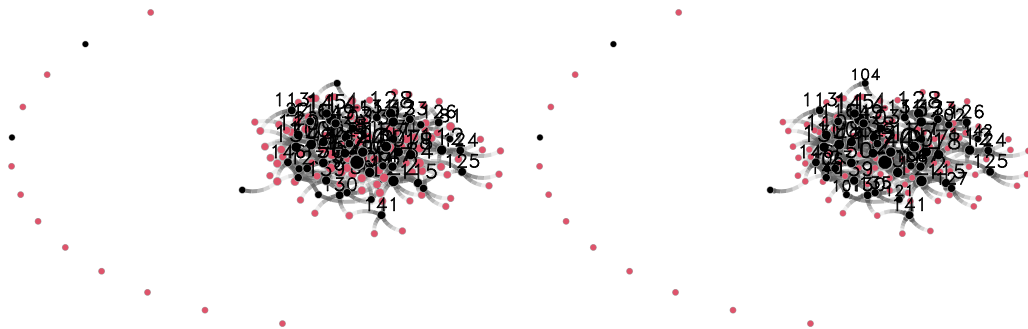


```
## Looking at the counts
summary(net_b ~ edges + b1star(2) + b2star(2))
```

```
 edges b1star2 b2star2
   245     313     645
```

```
summary(net_not_b ~ edges + ostar(2) + istar(2))
```

```
 edges ostar2 istar2
   245    313    645
```

With the two networks matching, we can now fit the ERGMs with and without offset terms and compare the results between the two models:

```
## ERGM with a bipartite graph
res_b      <- ergm(
  # Main formula
  net_b ~ edges + Log(~b1star(2)),

  # Control parameters
  control = control.ergm(seed = 1)
  )
```

```
## ERGM with a digraph with constraints
res_not_b <- ergm(
  # Main formula
  net_not_b ~ edges + Log(~ostar(2)) +

  # Offset terms
  offset(nodematch("is.actor")) + offset(nodeocov("isnot.actor")),
  offset.coef = c(-Inf, -Inf),

  # Control parameters
  control = control.ergm(seed = 1)
  )
```

Here are the estimates (using the **texreg** R package for a prettier output):

```
texreg::screenreg(list(Bipartite = res_b, Directed = res_not_b))
```

```
=================================================================
                                 Bipartite      Directed
-----------------------------------------------------------------
edges                            -3.14 ***            -3.11 ***
                                 (0.15)               (0.14)
Log~b1star2                      21.89
                                 (17.13)
Log~ostar2                                            19.66
                                                      (16.75)
offset(nodematch.is.actor)                            -Inf

offset(nodeocov.isnot.actor)                          -Inf

-----------------------------------------------------------------
AIC                              1958.00   -2134192392498170112.00
BIC                              1971.03   -2134192392498170112.00
Log Likelihood                   -977.00    1067096196249085056.00
=================================================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

As expected, both models yield the "same" estimate. The minor differences observed between the models are how the **ergm** package performs the sampling. In particular, in the bipartite case, **ergm** has special routines for making the sampling more efficient, having a higher acceptance rate than that of the model in which the bipartite graph was not explicitly declared. We can tell this by inspecting rejection rates:

```
data.frame(
  Bipartite = coda::rejectionRate(res_b$sample[[1]]) * 100,
  Directed  = coda::rejectionRate(res_not_b$sample[[1]][, -c(3,4)]) * 100
) |> knitr::kable(digits = 2, caption = "Rejection rate (percent)")
```

Table 10.1: Rejection rate (percent)

	Bipartite	Directed
edges	2.48	3.67
Log~b1star2	1.24	2.04

The ERGM fitted with the offset terms has a much higher rejection rate than that of the ERGM fitted with the bipartite ERGM.

Finally, the fact that we can fit ERGMs using offset does not mean that we need to use it ALL the time. Unless there is a very good reason to go around `ergm`'s capabilities, I wouldn't recommend fitting bipartite ERGMs as we just did, as the authors of the package have included (MANY) features to make our job easier.

11 New topics in network modeling

11.1 Overview

- With **more data** and **computing resources**, the things that we can ask and do with networks are becoming increasingly (even more) exciting and complex.

- In this section, I will introduce **some** of the latest advancements and forthcoming topics in network modeling.

11.2 Part I: New models and extensions

11.3 Mutli-ERGMs

- In Krivitsky, Coletti, and Hens (2023a), the authors present a start-to-finish pooled ERGM example featuring heterogeneous data sources.

- They increase power and allow exploring heterogeneous effects across types/classes of networks.
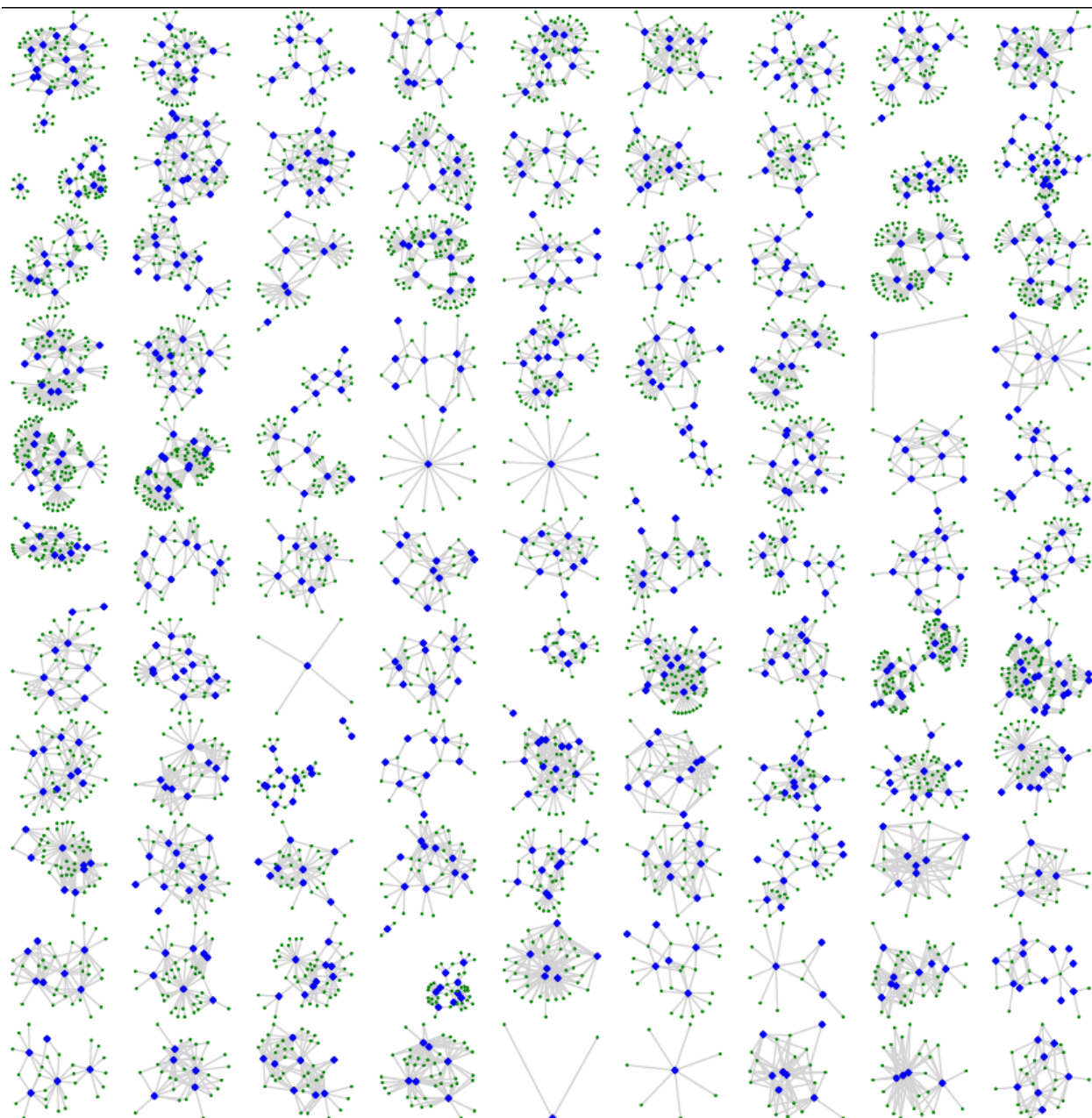
11.4 Statistical power of SOAM

- **SOAM** Stadtfeld et al. (2020) proposes ways to perform power analysis for Siena models. At the center of their six-step approach is simulation.
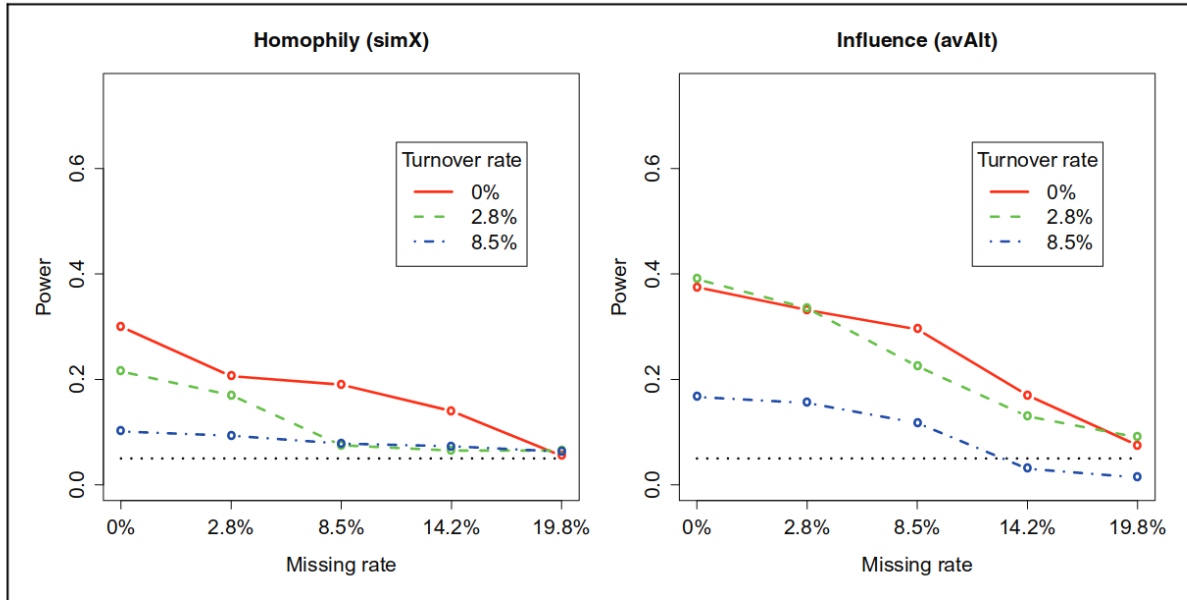
Figure 4. Power of models with smaller homophily (simX = 0.4) and influence (avAlt = 0.3) parameters. Missing rates are indicated in the *x*-axis, turnover rates are given by the three lines. The black dotted line indicates the chosen significance level (5 percent).

11.5 Bayesian ALAAM

Ever wondered how to model influence exclusively?

- The Auto-Logistic Actor Attribute Model [ALAAM] is a model that allows us to do just that.

- Koskinen and Daraganova (2022) extends the ALAAM model to a Bayesian framework.

- It provides greater flexibility to accommodate more complicated models and add extensions such as hierarchical models.

11.6 Relational Event Models

- REMs are great for modeling sequences of ties (instead of panel or cross-sectional.)

- Butts et al. (2023) provides a general overview of Relational Event Models [REMs,] new methods, and future steps.
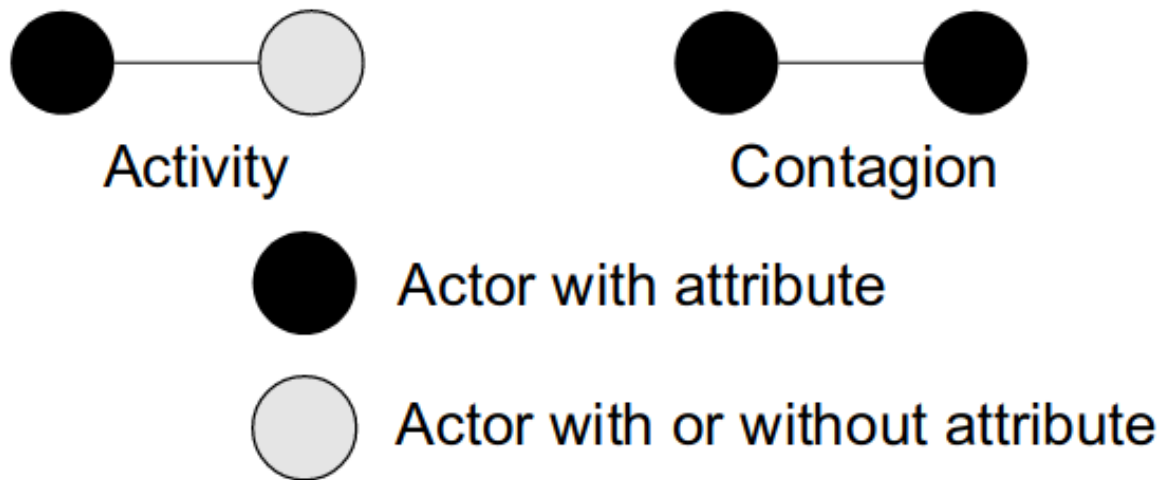
Figure 11.1: Figure 1 reproduced from A. D. Stivala et al. (2020)
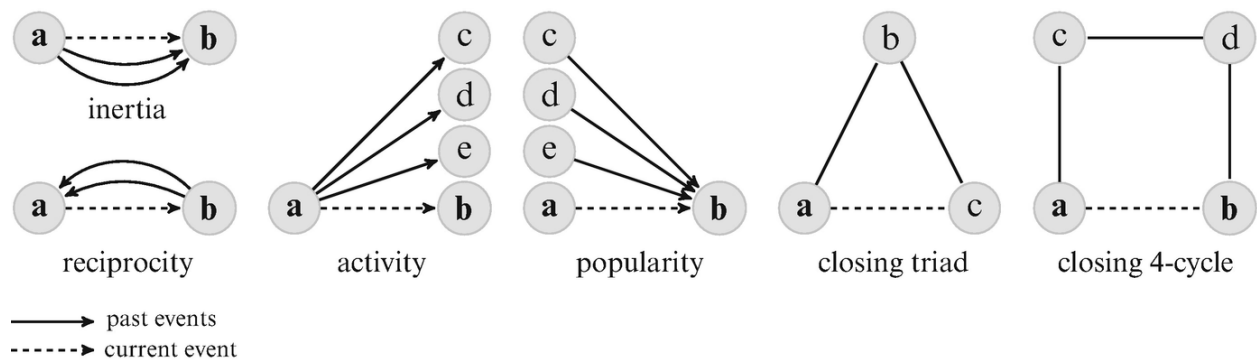

Figure 11.2: Figure 3 reproduced from Brandenberger (2020)

11.7 Big ERGMs

- **ERGMs** In A. Stivala, Robins, and Lomi (2020), a new method is proposed to estimate large ERGMs (featuring millions of nodes).

11.8 Exponential Random *Network* Models

- Wang, Fellows, and Handcock recently published a re-introduction of the ERNM framework (Wang, Fellows, and Handcock 2023).

- ERNMs generalize ERGMs to incorporate behavior and are the cross-sectional causing of SIENA models.

$$\text{ER}\mathbf{G}\text{M} : P_{y,\theta}(Y = y | X = x)$$
$$\text{ER}\mathbf{N}\text{M} : P_{y,\theta}(Y = y, X = x)$$

11.9 Part II: Shameless self-promotion
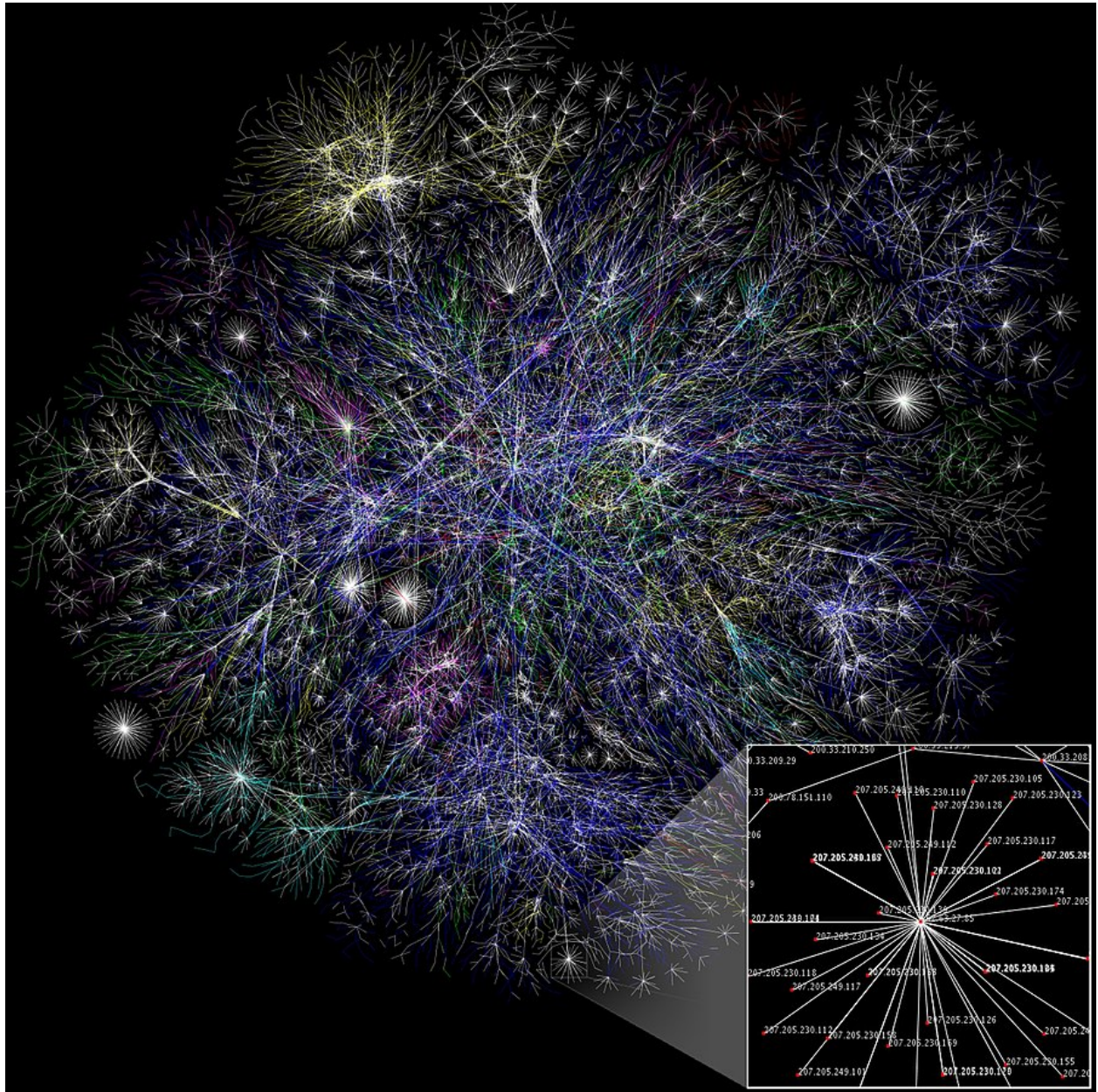
11.10 ERGMitos: Small ERGMs

- **ERGMitos**[1] (G. G. Vega Yon, Slaughter, and Haye 2021) leverage small network sizes to use exact statistics.

11.11 Discrete Exponential-family Models

- ERGMs are a particular case of Random Markov fields.

- We can use the ERGM framework for modeling vectors of binary outcomes, e.g., the consumption of $\{tobacco, MJ, alcohol\}$
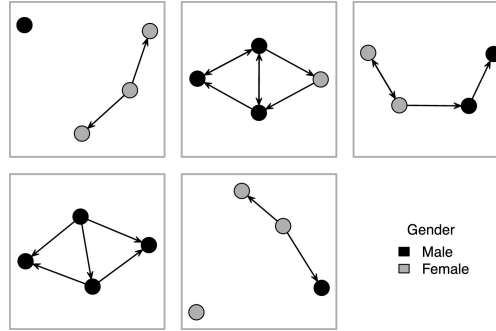
---

[1]From the Spanish suffix meaning small.

Figure 11.3: Partial map of the Internet based on the January 15, 2005 data found on opte.org. – Wiki

Figure 11.4: Five small networks from the `ergmito` R package

alcohol	
(Intercept)	$-3.24\ (0.27)^{***}$
x Hispanic	$0.91\ (0.21)^{***}$
x Year 3	$1.09\ (0.29)^{***}$
x Year 4	$0.99\ (0.27)^{***}$
tobacco	
(Intercept)	$-5.11\ (0.39)^{***}$
x Female	$-0.61\ (0.25)^{*}$
x Expo. Smoke	$1.72\ (0.40)^{***}$
x Sib. Smokes	$1.03\ (0.29)^{***}$
x Year 4	$1.63\ (0.31)^{***}$
mj	
x Grades	$-0.61\ (0.07)^{***}$
x Female	$-0.67\ (0.23)^{**}$
x Year 4	$1.78\ (0.28)^{***}$
Pairwise Motifs	
(a) $(alcohol^{+}, \underline{tobacco^{-}}) \to (alcohol^{+}, \underline{tobacco^{+}})$	$1.89\ (0.33)^{***}$
(c) $(\underline{alcohol^{-}}, tobacco^{+}) \to (\underline{alcohol^{+}}, tobacco^{+})$	$2.00\ (0.50)^{***}$
(d) $(tobacco^{+}, \underline{mj^{-}}) \to (tobacco^{+}, \underline{mj^{+}})$	$0.93\ (0.36)^{**}$
(e) $(\underline{alcohol^{-}}, \underline{tobacco^{-}}) \to (\underline{alcohol^{+}}, \underline{tobacco^{+}})$	$2.30\ (0.36)^{***}$
Motifs involving all three outcomes	
(f) $(alcohol^{-}, tobacco^{-}, \underline{mj^{-}}) \to (alcohol^{-}, tobacco^{-}, \underline{mj^{+}})$	$-2.39\ (0.35)^{***}$
AIC	1647.04
BIC	1733.01
N events (transitions)	1161

$^{***}p < 0.001;\ ^{**}p < 0.01;\ ^{*}p < 0.05$

11.12 Power analysis in ERGMs

- Using conditional ERGMs (closely related to constrained), we can do power analysis for network samples (G. G. Vega Yon 2023).
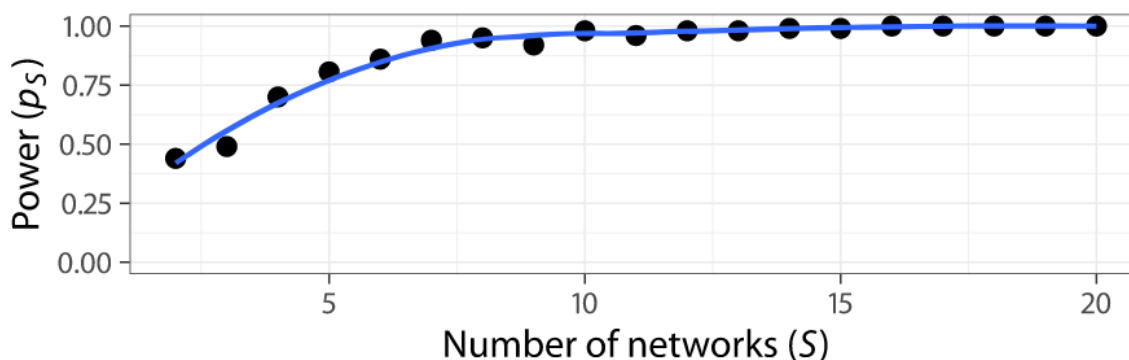
Figure 1. Empirical power p_S for detecting a gender homophily effect with an assumed $\theta_{\text{homophily}} = 1.1$ as a function of number of networks S, where each network has size 8 nodes and 20 edges in the scenario of Vega Yon (2023, sec. 1).

Figure 11.5: Reproduced from Krivitsky, Coletti, and Hens (2023b)

11.13 Two-step estimation ERGMs

- Conditioning the ERGM on an observed statistic "drops" the associated coefficient.

- Hypothesis: As n increases, conditional ERGM estimates are consistent with the full model:

11.14 Thanks!

11.15 Bonus track: Why network scientists don't use ERGMs?

Attempts to overcome these problems by extending the blockmodel have focused particularly on the use of (more complicated) p^* or exponential random graph models, but **while these are conceptually appealing**, they quickly **lose the analytic tractability** of the original blockmodel **as their complexity increases**.

	Full	Conditional	Mispecified
edges	-2.94^{***}	-2.94^{***}	-2.84^{***}
	(0.04)	(0.04)	(0.04)
mutual	1.89^{***}	1.89^{***}	4.83^{***}
	(0.11)	(0.12)	(0.08)
triangle	0.50^{***}		
	(0.01)		
AIC	8709.44	6979.75	17708.82
BIC	8733.15	6995.56	17724.63
Log Likelihood	-4351.72	-3487.87	-8852.41
Num. networks	1000	1000	1000
Convergence	0	0	0
Time (seconds)	2.50	0.12	1.41

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

Figure 11.6: Simulation study trying to demonstrate the concept (Work in progress)

– Karrer and Newman (2011)

References

"1.1: Basic Definitions and Concepts." 2014. *Statistics LibreTexts.* https://stats.libretexts.org/Bookshelves

Barrett, Tyson, Matt Dowle, and Arun Srinivasan. 2023. *Data.table: Extension of 'Data.frame'.* https://r-datatable.com.

Bell, Brooke M., Donna Spruijt-Metz, George G. Vega Yon, Abu S. Mondol, Ridwan Alam, Meiyi Ma, Ifat Emi, John Lach, John A. Stankovic, and Kayla De La Haye. 2019. "Sensing Eating Mimicry Among Family Members." *Translational Behavioral Medicine.* https://doi.org/10.1093/tbm/ibz051.

Brandenberger, Laurence. 2020. "Interdependencies in Conflict Dynamics: Analyzing Endogenous Patterns in Conflict Event Data Using Relational Event Models." In *Computational Conflict Research*, edited by Emanuel Deutschmann, Jan Lorenz, Luis G. Nardin, Davide Natalini, and Adalbert F. X. Wilhelm, 67–80. Computational Social Sciences. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-29333-8_4.

Butts, Carter T., Alessandro Lomi, Tom A. B. Snijders, and Christoph Stadtfeld. 2023. "Relational Event Models in Network Science." *Network Science* 11 (2): 175–83. https://doi.org/10.1017/nws.2023.9.

Casella, George, and Roger L. Berger. 2021. *Statistical Inference.* Cengage Learning.

Fellows, Ian E. 2012. "Exponential Family Random Network Models." *ProQuest Dissertations and Theses.* PhD thesis. https://login.ezproxy.lib.utah.edu/login?url=https://www.proquest.com/dissertations-theses/exponential-family-random-network-models/docview/1221548720/se-2.

Frank, O, and David Strauss. 1986. "Markov graphs." *Journal of the American Statistical Association* 81 (395): 832–42. https://doi.org/10.2307/2289017.

Gelman, Andrew. 2018. "The Failure of Null Hypothesis Significance Testing When Studying Incremental Changes, and What to Do About It." *Personality and Social Psychology Bulletin* 44 (1): 16–23. https://doi.org/10.1177/0146167217729162.

Geyer, Charles J., and Elizabeth A. Thompson. 1992. "Constrained Monte Carlo Maximum Likelihood for Dependent Data." *Journal of the Royal Statistical Society. Series B (Methodological)* 54 (3): 657–99. https://www.jstor.org/stable/2345852.

Greenland, Sander, Stephen J. Senn, Kenneth J. Rothman, John B. Carlin, Charles Poole, Steven N. Goodman, and Douglas G. Altman. 2016. "Statistical Tests, P Values, Confidence Intervals, and Power: A Guide to Misinterpretations." *European Journal of Epidemiology* 31 (4): 337–50. https://doi.org/10.1007/s10654-016-0149-3.

Handcock, Mark S., David R. Hunter, Carter T. Butts, Steven M. Goodreau, Pavel N. Krivitsky, and Martina Morris. 2023. *Ergm: Fit, Simulate and Diagnose Exponential-Family Models for Networks.* The Statnet Project (https://statnet.org). https://CRAN.R-project.org/package=ergm.

Haye, Kayla de la, Heesung Shin, George G. Vega Yon, and Thomas W. Valente. 2019. "Smoking Diffusion Through Networks of Diverse, Urban American Adolescents over the High School Period." *Journal of Health and Social Behavior.* https://doi.org/10.1177/0022146519870521.

Holland, Paul W., and Samuel Leinhardt. 1981. "An exponential family of probability distributions for directed graphs." *Journal of the American Statistical Association* 76 (373): 33–50. https://doi.org/10.2307/2287037.

Hunter, David R. 2007. "Curved Exponential Family Models for Social Networks." *Social Networks* 29 (2): 216–30. https://doi.org/10.1016/j.socnet.2006.08.005.

Hunter, David R., Mark S. Handcock, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2008. "ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks." *Journal of Statistical Software* 24 (3): 1–29. https://doi.org/10.18637/jss.v024.i03.

Karrer, Brian, and M. E. J. Newman. 2011. "Stochastic Blockmodels and Community Structure in Networks." *Physical Review E* 83 (1): 016107. https://doi.org/10.1103/PhysRevE.83.016107.

Koskinen, Johan, and Galina Daraganova. 2022. "Bayesian Analysis of Social Influence." *Journal of the Royal Statistical Society Series A: Statistics in Society* 185 (4): 1855–81. https://doi.org/10.1111/rssa.12844.

Krivitsky, Pavel N. 2023. *Ergm.multi: Fit, Simulate and Diagnose Exponential-Family Models for Multiple or Multilayer Networks.* The Statnet Project (https://statnet.org). https://CRAN.R-project.org/package=ergm.multi.

Krivitsky, Pavel N., Pietro Coletti, and Niel Hens. 2023a. "A Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks." *Journal of the American Statistical Association* 0 (0): 1–21. https://doi.org/10.1080/01621459.2023.2242627.

———. 2023b. "Rejoinder to Discussion of 'A Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks'." *Journal of the American Statistical Association* 118 (544): 2235–38. https://doi.org/10.1080/01621459.2023.2280383.

Leifeld, Philip. 2013. "**Texreg** : Conversion of Statistical Model Output in *R* to L A T E X and HTML Tables." *Journal of Statistical Software* 55 (8). https://doi.org/10.18637/jss.v055.i08.

LeSage, James P. 2008. "An Introduction to Spatial Econometrics." *Revue d'économie Industrielle* 123 (123): 19–44. https://doi.org/10.4000/rei.3887.

LeSage, James P., and R. Kelley Pace. 2014. "The Biggest Myth in Spatial Econometrics." *Econometrics* 2 (4): 217–49. https://doi.org/10.2139/ssrn.1725503.

Lusher, Dean, Johan Koskinen, and Garry Robins. 2013. *Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications.* Cambridge University Press.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Robins, Garry, Philippa Pattison, and Peter Elliott. 2001. "Network Models for Social Influence Processes." *Psychometrika* 66 (2): 161–89. https://doi.org/10.1007/BF02294834.

Robins, Garry, Pip Pattison, Yuval Kalish, and Dean Lusher. 2007. "An introduction to

exponential random graph (p*) models for social networks." *Social Networks* 29 (2): 173–91. https://doi.org/10.1016/j.socnet.2006.08.002.

Roger Bivand. 2022. "R Packages for Analyzing Spatial Data: A Comparative Case Study with Areal Data." *Geographical Analysis* 54 (3): 488–518. https://doi.org/10.1111/gean .12319.

Slaughter, Andrew J., and Laura M. Koehly. 2016. "Multilevel Models for Social Networks: Hierarchical Bayesian Approaches to Exponential Random Graph Modeling." *Social Networks* 44: 334–45. https://doi.org/10.1016/j.socnet.2015.11.002.

Snijders, Tom a B. 1996. "Stochastic Actor-Oriented Models for Network Change." *The Journal of Mathematical Sociology* 21 (1-2): 149–72. https://doi.org/10.1080/0022250X .1996.9990178.

Snijders, Tom A B, Philippa E Pattison, Garry L Robins, and Mark S Handcock. 2006. "New specifications for exponential random graph models." *Sociological Methodology* 36 (1): 99–153. https://doi.org/10.1111/j.1467-9531.2006.00176.x.

Snijders, Tom A. B. 2017. "Stochastic Actor-Oriented Models for Network Dynamics." *Annual Review of Statistics and Its Application* 4 (1): 343–63. https://doi.org/10.1146/an nurev-statistics-060116-054035.

Stadtfeld, Christoph, Tom A. B. Snijders, Christian Steglich, and Marijtje van Duijn. 2020. "Statistical Power in Longitudinal Network Studies." *Sociological Methods & Research* 49 (4): 1103–32. https://doi.org/10.1177/0049124118769113.

Stivala, Alex D., H. Colin Gallagher, David A. Rolls, Peng Wang, and Garry L. Robins. 2020. "Using Sampled Network Data With The Autologistic Actor Attribute Model." arXiv. https://doi.org/10.48550/arXiv.2002.00849.

Stivala, Alex, Garry Robins, and Alessandro Lomi. 2020. "Exponential Random Graph Model Parameter Estimation for Very Large Directed Networks." *PLoS ONE* 15 (1): 1–23. https://doi.org/10.1371/journal.pone.0227804.

Tanaka, Kyosuke, and George G. Vega Yon. 2024. "Imaginary Network Motifs: Structural Patterns of False Positives and Negatives in Social Networks." *Social Networks* 78 (July): 65–80. https://doi.org/10.1016/j.socnet.2023.11.005.

Valente, Thomas W., and George G. Vega Yon. 2020. "Diffusion/Contagion Processes on Social Networks." *Health Education & Behavior* 47 (2): 235–48. https://doi.org/10.117 7/1090198120901497.

Valente, Thomas W., Heather Wipfli, and George G. Vega Yon. 2019. "Network Influences on Policy Implementation: Evidence from a Global Health Treaty." *Social Science and Medicine.* https://doi.org/10.1016/j.socscimed.2019.01.008.

Vega Yon, George. 2020. *ergmito: Exponential Random Graph Models for Small Networks.* CRAN. https://cran.r-project.org/package=ergmito.

Vega Yon, George G. 2023. "Power and Multicollinearity in Small Networks: A Discussion of 'Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks' by Krivitsky, Coletti & Hens." *Journal Of The American Statistical Association.*

Vega Yon, George G., Andrew Slaughter, and Kayla de la Haye. 2021. "Exponential Random Graph Models for Little Networks." *Social Networks* 64 (August 2020): 225–38. https://doi.org/10.1016/j.socnet.2020.07.005.

Wang, Zeyi, Ian E. Fellows, and Mark S. Handcock. 2023. "Understanding Networks

with Exponential-Family Random Network Models." *Social Networks*, August, S0378873323000497. https://doi.org/10.1016/j.socnet.2023.07.003.

Wasserman, Stanley, and Philippa Pattison. 1996. "Logit models and logistic regressions for social networks: I. An introduction to Markov graphs and p*." *Psychometrika* 61 (3): 401–25. https://doi.org/10.1007/BF02294547.