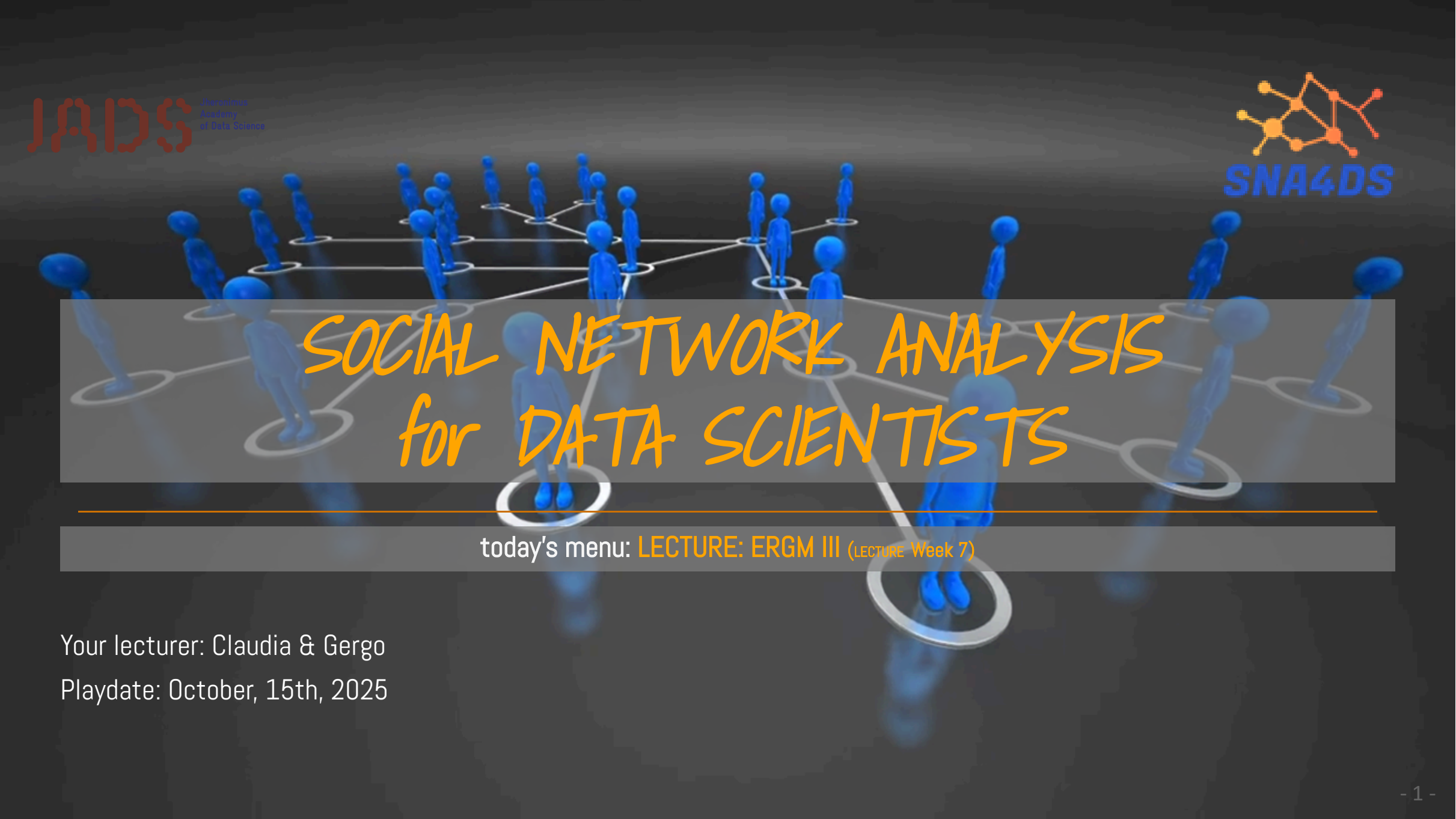


SOCIAL NETWORK ANALYSIS for DATA SCIENTISTS

today's menu: LECTURE: ERGM III (LECTURE Week 7)

Your lecturer: Claudia & Gergo

Playdate: October, 15th, 2025

Menu'

- ERGMs Estimation
- Model diagnostics
- Curved Terms (to solve degeneracy problem)
- Goodness of fit

ERGMs Estimation

Estimation

The estimation allows for the identification of the parameters that maximize the likelihood of a graph.

Finding the θ of the terms that maximize the probability of simulating graphs like the observed real one

- We discussed how to insert terms and how to specify the models
- We still need to check if the specification we inserted is doing its job
- Since the algorithm takes care of the estimation we need to understand it a bit better

ERGM algorithms

SNA4DS

Dyadic independent ERGM: estimated as a logit model with Maximum (pseudo) Likelihood Estimation (MLE or MPLE)

- We need to check the goodness of fit as we did with logit models
- We do that using the `ergm::gof()` function

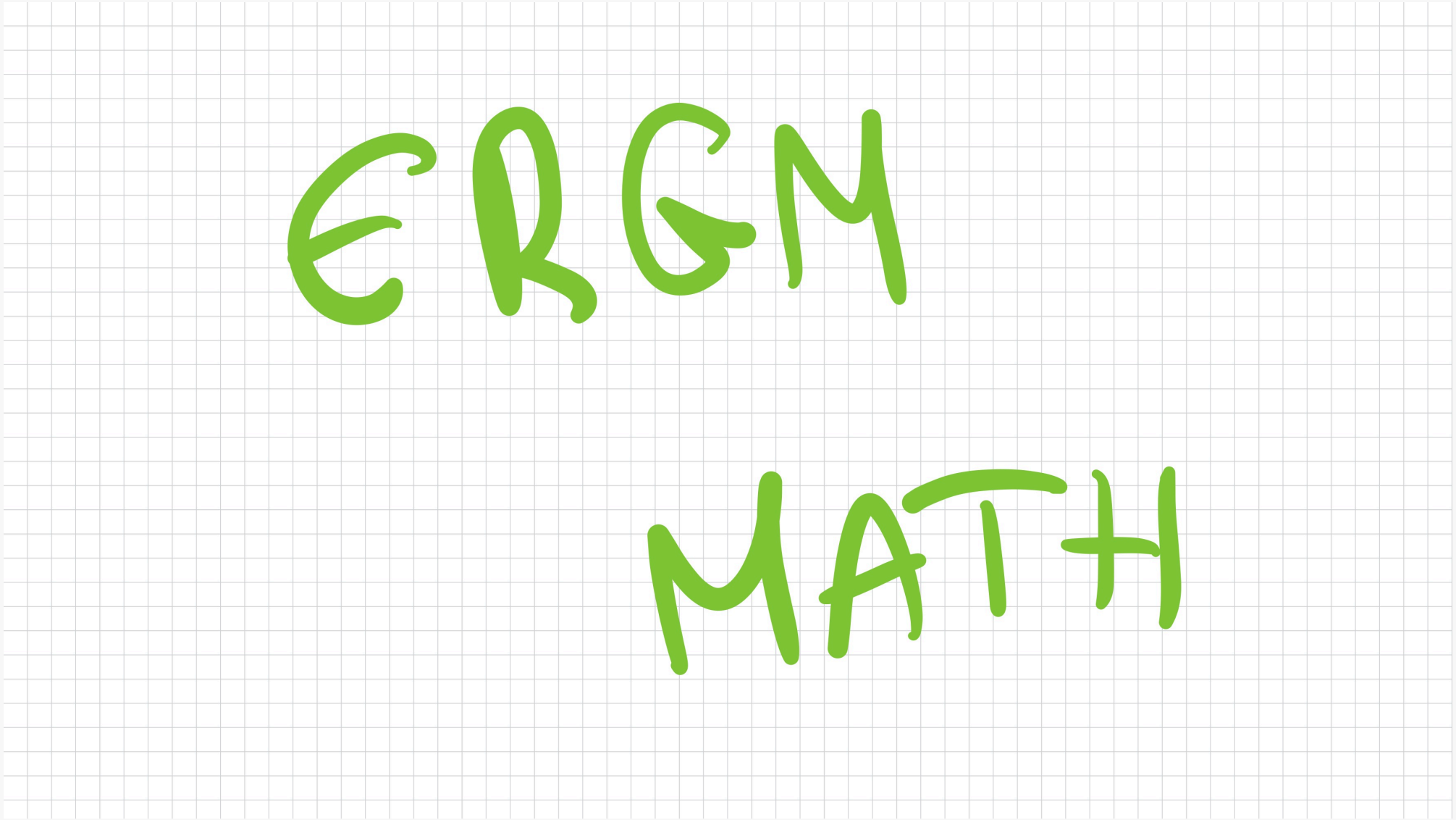
Dyadic (inter)dependent: we need to estimate parameters using Markov chains Monte Carlo MCMC simulations

- We still need to check the goodness of fit but also we need to check whether the simulation worked as we wanted to. We need to check two indicators
  - `ergm::gof()`
  - `ergm::mcmc.diagnostics()`

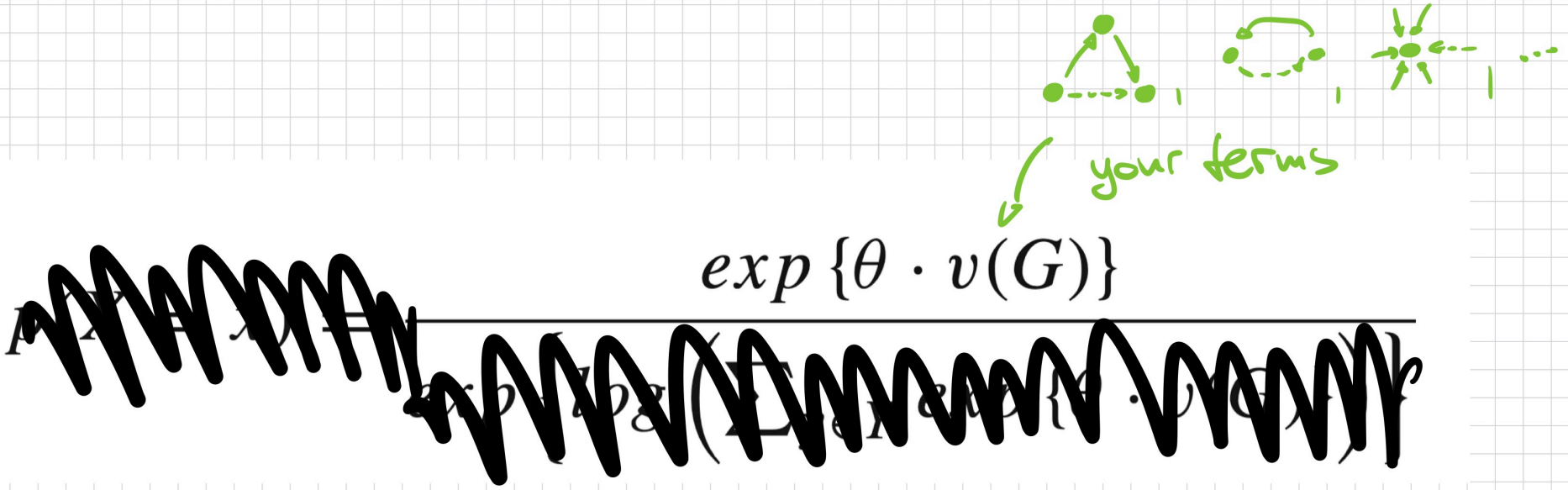
Intuitive estimation -Dyadic dependent

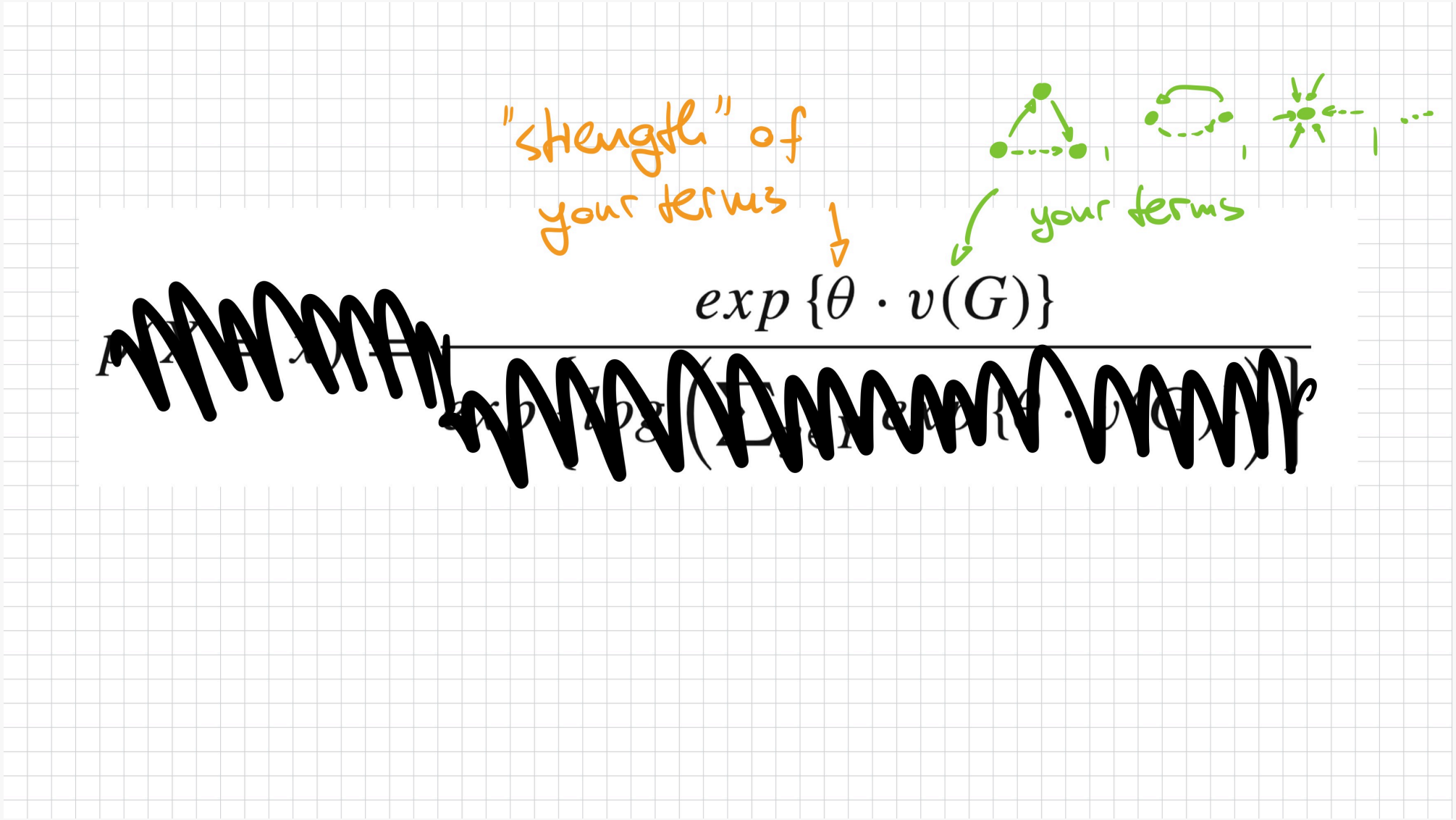
Explaining a Markov Chain Maximum Likelihood estimation

- We simulate Markov Chains using a Monte Carlo (random) method.
- A Markov chain is a sequence of random variables such that the value taken by the random variable only depends upon the value taken by the previous variable.
- By switching the entries in the matrix representing the network (0-1) we generate a series of graphs that differ from each other only by one edge and that depend from the previous graph.
- This sequence of generated graphs is a Markov chain.
- Within this generation of chains the Metropolis-Hastings algorithm or the Gibbs sampler decide which entries (0-1) change state.
- These algorithms start picking a variable at random and change its state
- They retain the change of state only if the new graph increases the likelihood
- Finally the Robbins-Monroe algorithm estimates the parameters in the model.


ERGM
MATH

$$p(X = x) = \frac{exp\{\theta \cdot v(G)\}}{exp\left\{log\left(\sum_{y \in Y} exp\{\theta \cdot v(G)\}\right)\right\}}$$


your terms
$exp\{\theta \cdot v(G)\}$


"strength" of your terms
your terms
$exp\{\theta \cdot v(G)\}$

What are your terms?

Remember lab ERGM 1 !

(Markovian dependence, neighborhood, etc)

probability of your specific network forming

$$p(X = x) = \frac{exp\{\theta \cdot v(G)\}}{exp\left\{log\left(\sum_{y \in Y} exp\{\theta \cdot v(G)\}\right)\right\}}$$

probability of your specific network forming

(numerical representation of) your observed network

$$p(X = x) = \frac{exp\{\theta \cdot v(G)\}}{exp\left\{log\left(\sum_{y \in Y} exp\{\theta \cdot v(G)\}\right)\right\}}$$

probability of your specific network forming

(numerical representation of) your observed network

$$p(X = x) = \frac{exp\{\theta \cdot v(G)\}}{exp\left\{log\left(\sum_{y \in Y} exp\{\theta \cdot v(G)\}\right)\right\}}$$

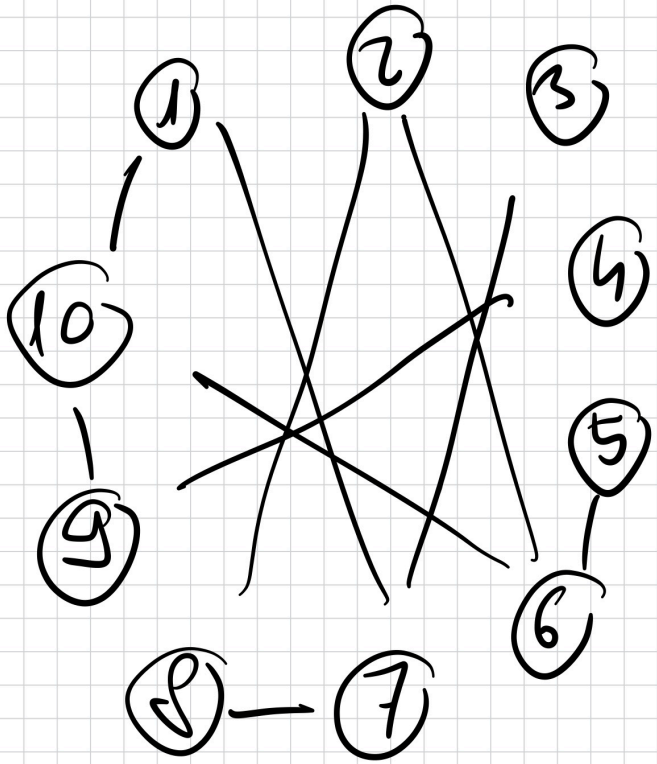
(numerical representation of) all possible networks

PROBLEM!

"all possible networks" is an

insanely large number of networks.

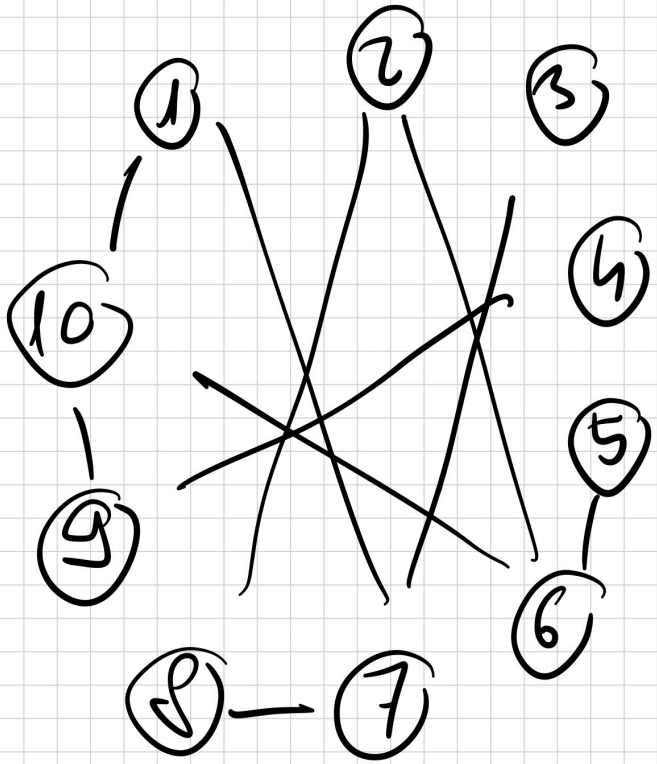
For nerds: n nodes define $2^{\frac{n(n-1)}{2}}$ graphs.



35 184 372 088 832

different graphs

are possible



35 184 372 088 832

different graphs are possible

same magnitude as:

- number of ants alive on Earth
- population of the Star Wars and Asimov universes

(- more than how many cells you have)

TLDR:

Computing an ERGM is impossible.

(except in case of dyadic independence,

=

network doesn't matter

$$p(X = x) = \frac{exp\{\theta \cdot v(x)\}}{exp\left\{log\left(\sum_{y \in Y} exp\{\theta \cdot v(y)\}\right)\right\}}$$

no network = no 10^{15} combinations ✓)

Unless you know a clever trick:

MCMC

Markov Chain Monte Carlo

Goal:

Identify Θ that maximizes probability of observed graph.

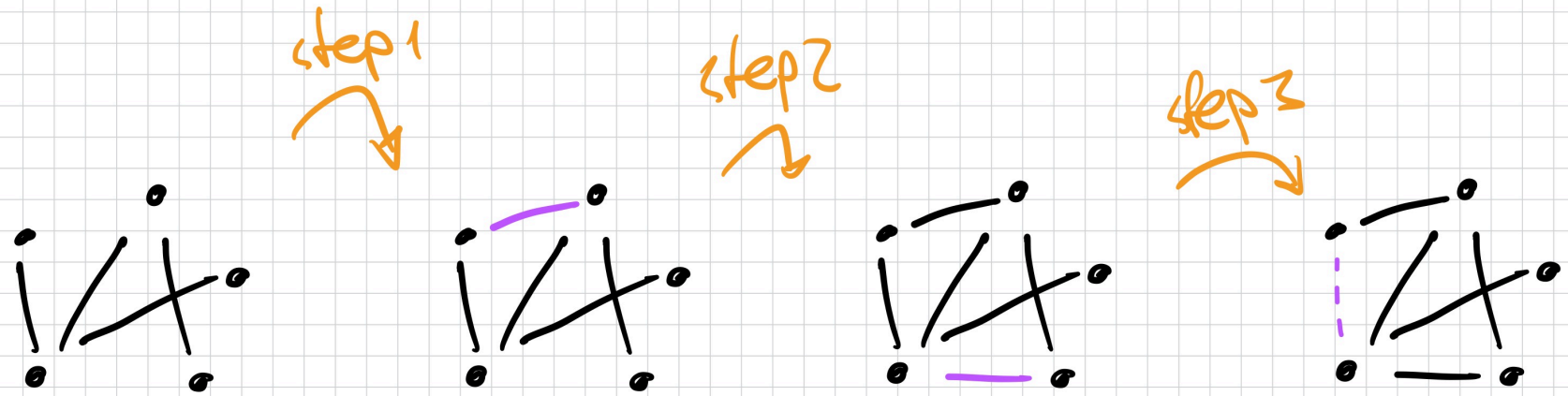
this is what you can interpret to examine hypotheses.

Need:

Sample of networks from the same probability distribution. (described by θ)

let's simulate!

Markov-chains = series of networks, 1 edge changes



This leads to a sample of networks, but we don't know the distribution (random change)

HOW to PICK WHICH EDGE TO CHANGE?

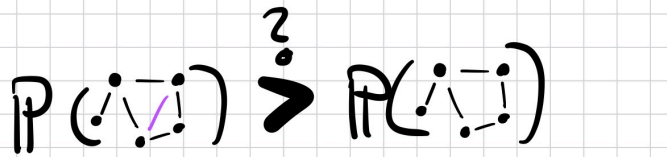
METROPOLIS - HASTINGS:

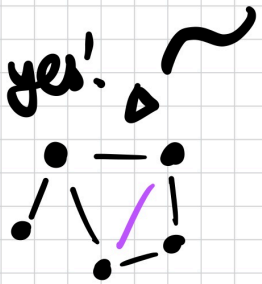
1. change edge at random

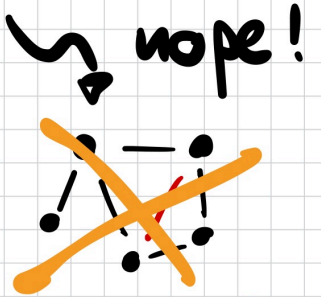
2. Assess if "good change"

   yes = changed network is base

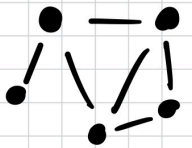
   no = old network is base





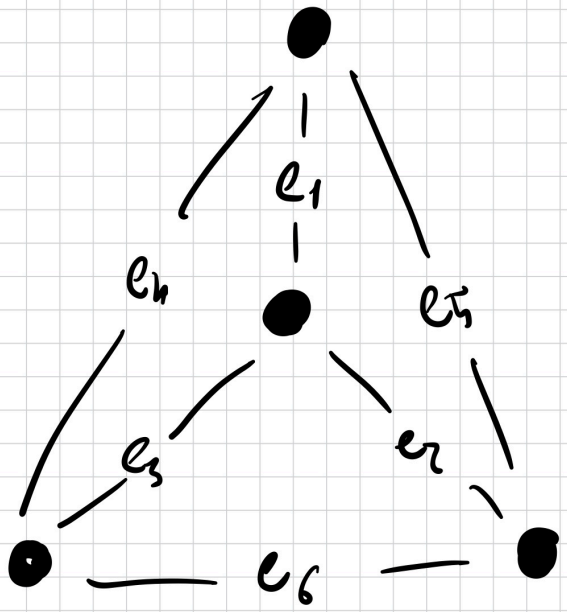


3. Rinse and repeat!



ALTERNATIVE WAY:

Gibbs-sampler → making sure the steps go in the right direction



1. calculate $\mathbb{P}(e_1 \mid \text{rest of net})$
   - ↳ $\mathbb{P}$ small? → set e_1 to $\emptyset$
2. calculate $\mathbb{P}(e_2 \mid \text{rest of net without } e_1)$
   - ↳ $\mathbb{P}$ big → set e_2 to 1
3. ... rinse and repeat

HOW DOES THIS GET US A BETTER Θ?

Robbins-Monro: Example: getting a better Θ value for triadic closure

1. Observed network has 4 closed triads
2. Using current Θ_t and MCMC, you make a sample network
3. Count number of triads in sample: 3
4. New $\Theta_{t+1} = \Theta_t + \alpha_t (4 - 3)$ ← here is where you make it better

α is a learning rate

Convergence

SNA4DS

- Chains should evolve toward increasing the likelihood of producing networks similar to the observed one.
- If the process goes according to plans, we say that the model converged
- If you look at the simulation output you will see a series of lines with different trends that chain after chain become similar and overlap (We will see it in a few minutes)
- If the model does not converge the chains never become similar and parallel universe continue to exist until the model fails (We will also see that in a few minutes)

Burn-in

- In the beginning, the model will have a time for the Metropolis-Hasting algorithm or the Gibbs sampler to increase the likelihood of the chains to resemble the real network
- This is the burn-in
- The burn-in represents the number of simulations that are removed from the MCMC in order to make the chain "forget" the (random) starting point

MCMC sample size

- If the model struggles at convergence it is a good idea to give it the opportunity to be more accurate.
- Increasing the sample size, the Metropolis-Hasting algorithm or the Gibbs sampler can source more information about the previous chain and increase the accuracy of the following one.
- It does not always work. If your terms are not capturing the dynamics of the real network it will never work.
- It increases your computational time, so careful when you use it

Model diagnostics

Checking the simulation in Dyadic dependent Models

Model using Florentine Business network

```
fit <- ergm::ergm(floB ~ edges + degree(1))
ergm::mcmc.diagnostics(fit)
```

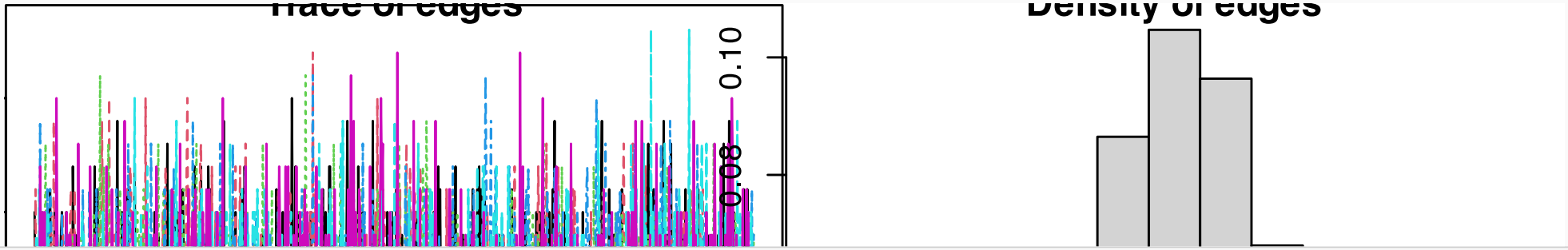
This function outputs three things:

- Text that explains the diagnostics
- A series of relevant measures
- Plots the MCMC diagnostics

MCMC Diagnostics

```
fitFlo.01 <- ergm::ergm(floB ~ edges + degree(1),
                        control = ergm::control.ergm(
                       MCMC.samplesize = 5000,
                       MCMC.burnin = 1000,
                       MCMLE.maxit = 10,
                       parallel = 6,
                       parallel.type = "PSOCK"
                     ))

ergm::mcmc.diagnostics(fitFlo.01)
```



Understanding the `mcmc.diagnostics` output

You need to know that

- your diagnostic analyses the iterations of the last chain produced
- ERGMs, as OLS, is predicting mean values of parameters
- The mean in the diagostics should be 0
- the standard deviation should form a bell curve

"Quantiles for each variable": 2.5% 25% 50% 75% 97.5%

-8, -2, 0, 2, 8 Would be an ideal scenario

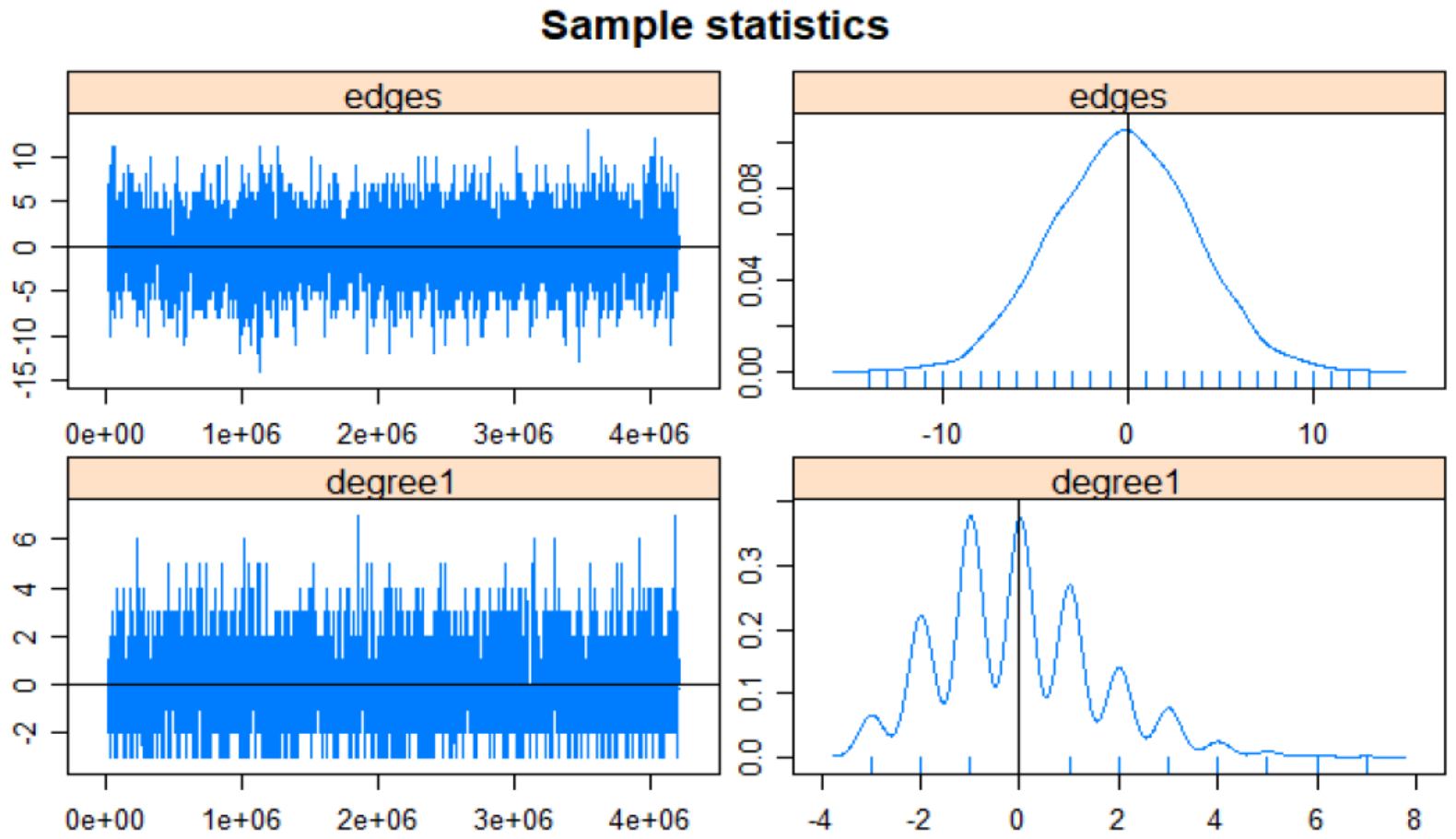
"Are sample statistics significantly different from observed?"

You want your p-value here to be as high as possible

Plot of the MCMC diagnostics

statistics from the final iteration of the MCMC chain

THIS IS GOOD!



"traceplot" (the deviation of the statistic value in each sampled network from the observed value)

distribution of the sample statistic deviations should approximate a bell curve

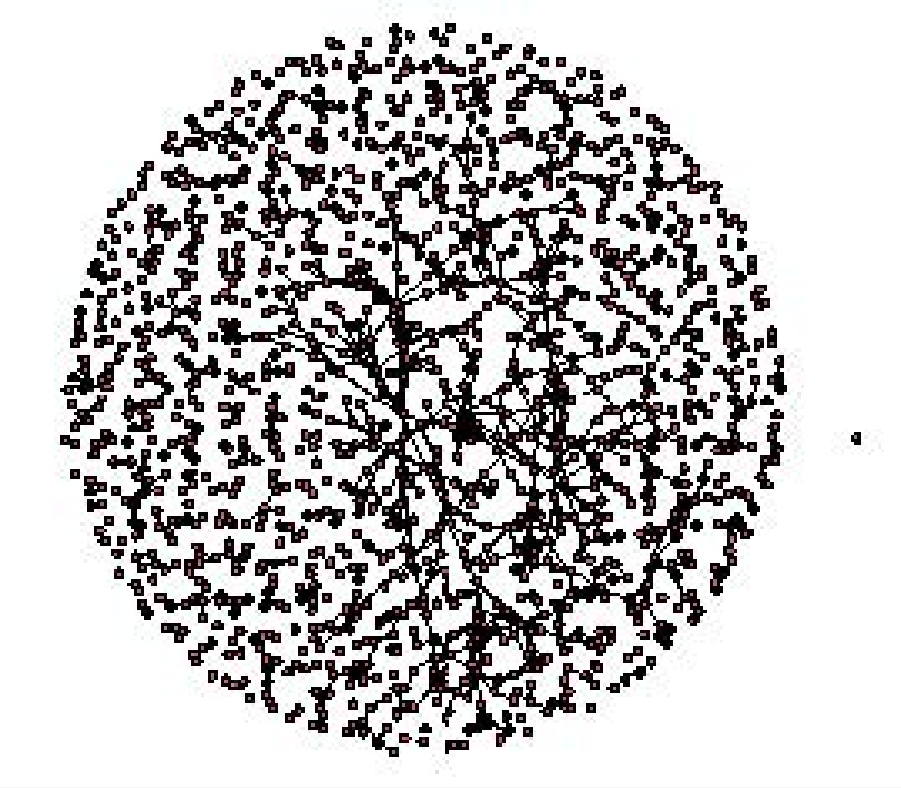
This ok since this is a small network

Trace Plot shape: fuzzy caterpillar

Bad model example

Faux Magnolia High. Data simulated from an American High School

- Nodes: 1461
- Edges: 974
- Sparse
- Undirected



Bad model example

Running this model we will get an error

```
# fitMag.01 <- ergm::ergm(magnolia ~ edges + triangles,
#                     control = ergm::control.ergm(
#                       MCMC.samplesize = 5000,
#                       MCMC.burnin = 1000,
#                       MCMLE.maxit = 10,
#                       parallel = 6,
#                       parallel.type = "PSOCK"
#                     ))
```

```
Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL), :
```

```
Number of edges in a simulated network exceeds that in the observed by a factor of more than 20. This is a strong indicator of model degeneracy or a very poor starting parameter configuration. If you are reasonably certain that neither of these is the case, increase the MCMLE.density.guard control.ergm() parameter.
```

This is called model DEGENERACY

The term `triangles` caused Degeneracy

Instead of converging, the algorithm heads off into networks that are much much more dense than the observed network. This is such a clear indicator of a degenerate model specification that the algorithm stops after 3 iterations, to avoid heading off into areas that would cause memory issues.

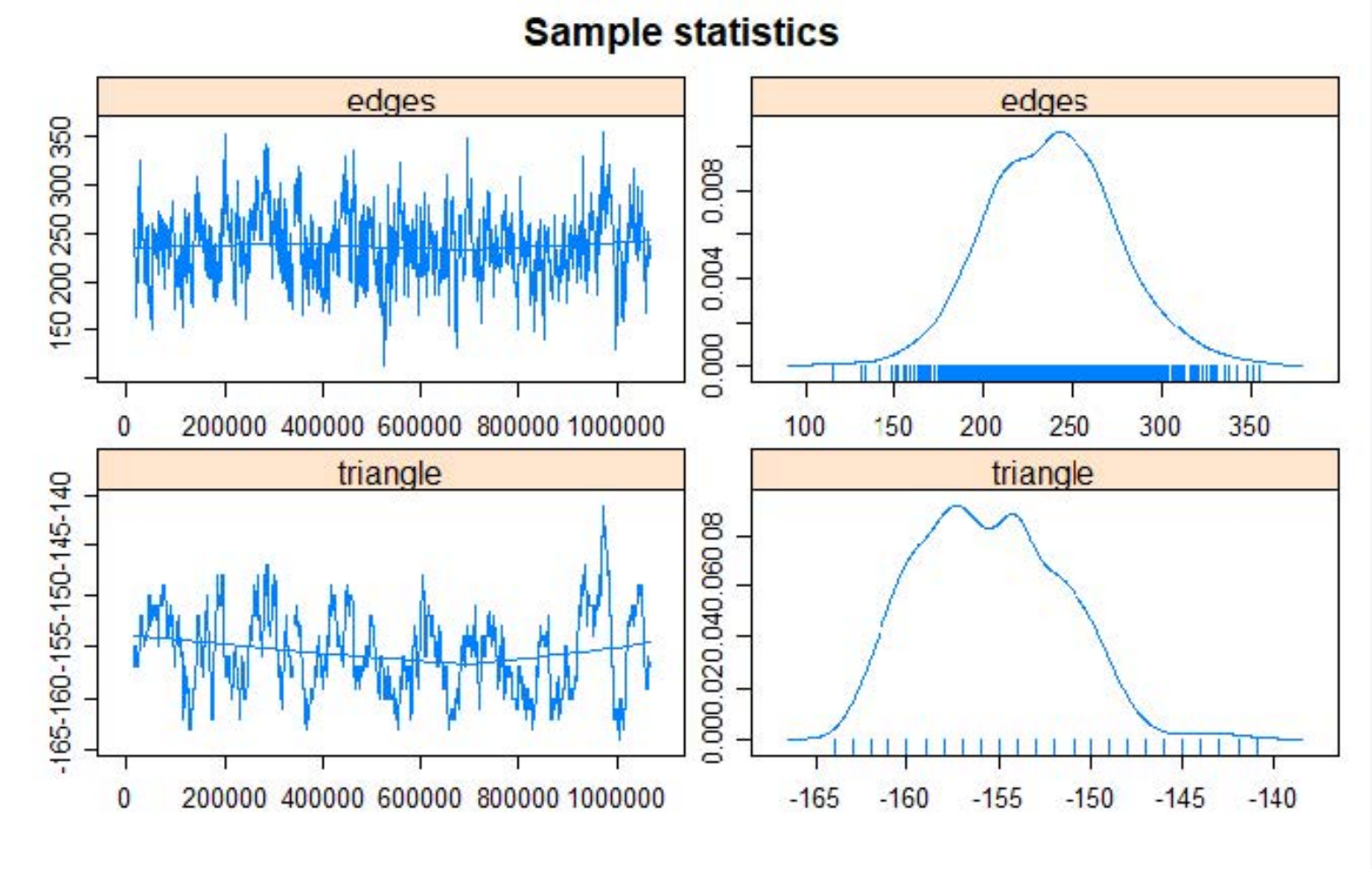
We can run the model limiting the number of iterations so it will converge and we can see what are the problems
Since the 3rd iteration was the last one in the model output we stop it at 1

```
# fitMag.01 <- ergm::ergm(magnolia ~ edges + triangles,
#                     control = ergm::control.ergm(
#                       MCMC.samplesize = 5000,
#                       MCMC.burnin = 1000,
#                       MCMLE.maxit = 1,
#                       parallel = 6,
#                       parallel.type = "PSOCK")
#                     )
#
# ergm::mcmc.diagnostics(fitMag.01)
```

Since sometimes this does not converge I'm not running the code in the slides

Bad model trace plot (where is the o?!)

Chains are not mixing well!



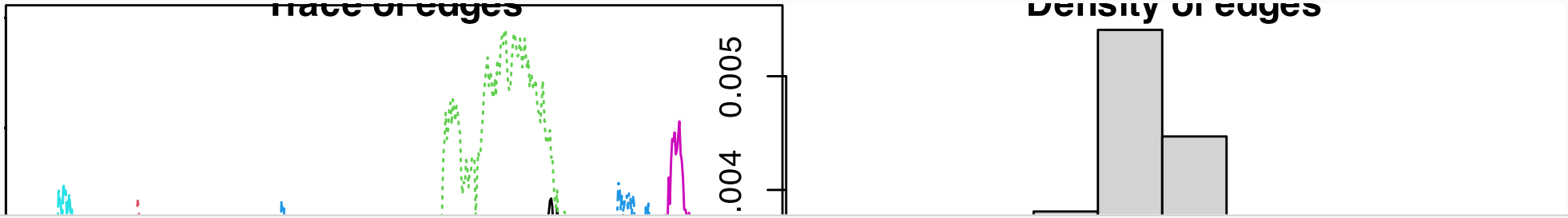
Curved Terms (to solve degeneracy problem)

How do we make the model converge?

We substitute the Markovian term with a curved one

```
set.seed(000023)
Magfit.02 <- ergm::ergm(magnolia ~ edges + gwesp(0.25, fixed = TRUE),
                    control = ergm::control.ergm(
                      MCMC.samplesize = 5000,
                      MCMC.burnin = 1000,
                      MCMLE.maxit = 1,
                      parallel = 6,
                      parallel.type = "PSOCK")
                    )

ergm::mcmc.diagnostics(Magfit.02)
```



Curved terms

- do the same job as the Markovian terms
- but they have been mathematically improved to avoid degeneracy and improve model fit

When you use a Markovian term that does not work swap it for the curved equivalent.

- DISCLAIMER: Markovian terms will work only for very small and simple networks.

Keep your Curved equivalent handy.

Most common Curved terms

SNA4DS

- `gwesp(decay = 0.25, fixed = FALSE)` - it can be used in place of `triangles` or `esp`
- `gwdsp(decay = 1, fixed = FALSE)` - it can be used in place of `dsp`
- `gwnsp(decay = 1, fixed = FALSE)` - it can be used in place of `nsp`
- `gwdegree(decay = 0.5, fixed = FALSE)` - it can be used in place of `degree`, but careful, this is degree distribution.
- `gwidegree(decay = 0.5, fixed = FALSE)` - it can be used in place of `idegree`, but careful, this is degree distribution.
- `gwodegree(decay = 0.5, fixed = FALSE)` - it can be used in place of `odegree`, but careful, this is degree distribution.
- there are also curved terms that don't gave a direct correspondence to the Markovian one. E.g., `dgwdsp(decay, fixed=FALSE` - it can be used as a sum of `gwnsp` and `gwesp`

The optional argument fixed indicates whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential family model. AKA: it uses different algorithms. At this stage, try both and keep the best one.

How to find curved terms?

```
ergm::search.ergmTerms(search = "curved")
```

```
## Found  16  matching ergm terms:
## altkstar(lambda, fixed=FALSE) (binary)
##     Alternating k-star
##
## Curve(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf, cov=NULL) (binary, valued)
## Parametrise(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf, cov=NULL) (binary, valued)
## Parametrize(formula, params, map, gradient=NULL, minpar=-Inf, maxpar=+Inf, cov=NULL) (binary, valued)
##     Impose a curved structure on term parameters
##
## gwb1degree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL) (binary)
##     Geometrically weighted degree distribution for the first mode in a bipartite network
##
## gwb1dsp(decay=0, fixed=FALSE, cutoff=30) (binary)
##     Geometrically weighted dyadwise shared partner distribution for dyads in the first bipartition
##
## gwb2degree(decay, fixed=FALSE, attr=NULL, cutoff=30, levels=NULL) (binary)
##     Geometrically weighted degree distribution for the second mode in a bipartite network
```

Goodness of fit

GOF

Let's get back to our pretty Florentine Business network

```
 FloFit.01 <- ergm::ergm(floB ~ edges + degree(1))
 fit.gof<- ergm::gof(FloFit.01)
 fit.gof


##
## Goodness-of-fit for degree
##
##         obs min mean max MC p-value
## degree0   5   0 2.82   7       0.32
## degree1   3   0 3.15   8       1.00
## degree2   2   0 5.51  10       0.08
## degree3   2   0 2.92   8       0.88
## degree4   3   0 1.34   5       0.30
## degree5   1   0 0.19   3       0.28
## degree6   0   0 0.07   1       1.00
##
## Goodness-of-fit for edgewise shared partner
##
##      obs min  mean max MC p-value
```

GOF output

Output: 4 Parts

1. Goodness of fit for degree (always)
2. Goodness of fit for edgewise shared partner (always)
3. Goodness of fit for minimum geodesic distance (always)
4. Goodness of fit for model statistics (changes with model specifications)

It is possible to customize some other parameters

GOF 1. Goodness-of-fit for degree

```
fit <- ergm::ergm(flobusiness ~ edges + degree(1))
fit.gof <- ergm::gof(fit)
plot(fit.gof)
```

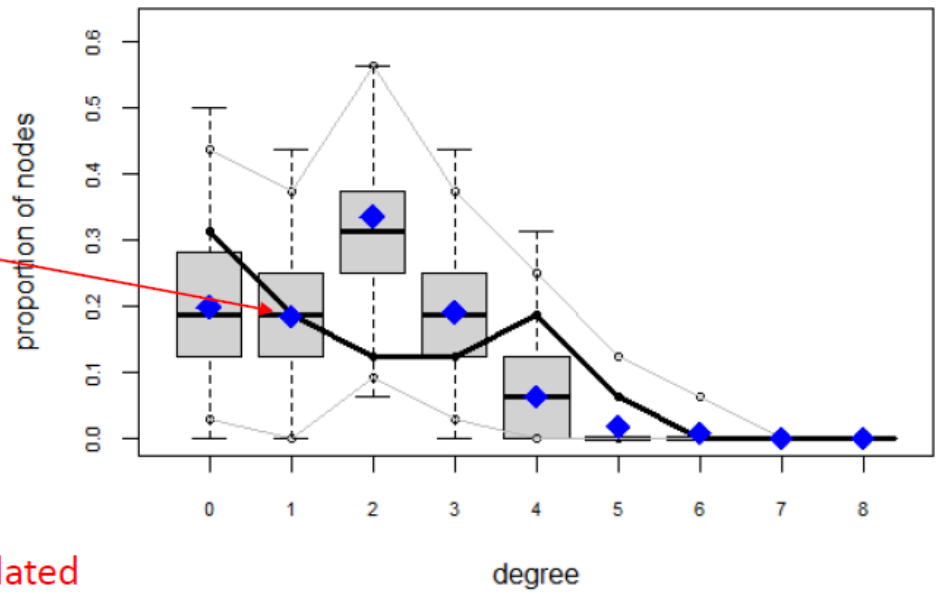
Good! Real network == Network created with model parameters!

```
Goodness-of-fit for degree

   obs min mean max MC p-value
0    5   0 3.19   9       0.38
1    3   1 3.50   7       1.00
2    2   2 4.98   9       0.12
3    2   0 2.92   7       0.88
4    3   0 1.07   4       0.30
5    1   0 0.28   3       0.48
6    0   0 0.06   1       1.00
```

One row for each possible node degree

simulated



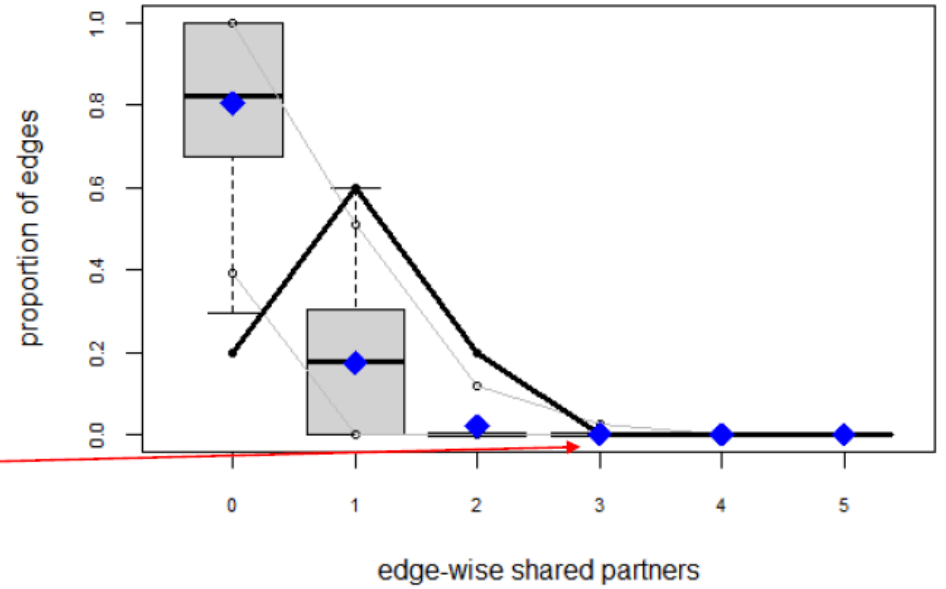
GOF 2.Goodness-of-fit for edgewise shared partner

```
fit <- ergm::ergm(flobusiness ~ edges + degree(1))
fit.gof <- ergm::gof(fit)
```

Output: 4 Parts

```
Goodness-of-fit for edgewise shared
partner

      obs min  mean max MC p-value
esp0    3   2 11.33  19       0.04
esp1    9   0  2.53  12       0.10
esp2    3   0  0.25   6       0.06
esp3    0   0  0.02   1       1.00
```



GOF 3. Goodness-of-fit for minimum geodesic distance

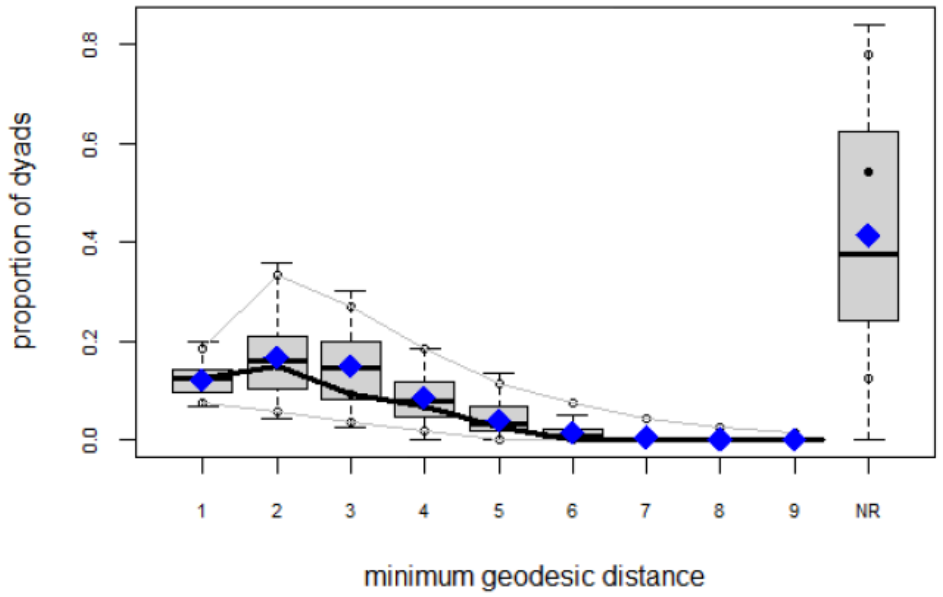
```
fit <- ergm::ergm(flobusiness ~ edges + degree(1))
fit.gof <- ergm::gof(fit)
```

Output: 4 Parts

```
Goodness-of-fit for minimum geodesic
distance

     obs min  mean max MC p-value
1     15   6 14.13  23       0.92
2     18   4 18.77  38       1.00
3     11   0 16.53  38       0.72
4      8   0  9.83  22       0.90
5      3   0  4.43  15       0.88
6      0   0  1.75  11       0.90
7      0   0  0.59   6       1.00
8      0   0  0.17   4       1.00
9      0   0  0.06   2       1.00
10     0   0  0.01   1       1.00
Inf   65   0 53.73 108       0.70
```

isolates



GOF 4. Goodness-of-fit for model statistics

```
fit <- ergm::ergm(flobusiness ~ edges + degree(1))
fit.gof <- ergm::gof(fit)
```

Output: 4 Parts

```
Goodness-of-fit for model statistics

         obs min  mean max MC p-value
edges     15   6 14.13  23       0.92
degree1    3   1  3.50   7       1.00
```

One box plot (estimate) for every stat in the model