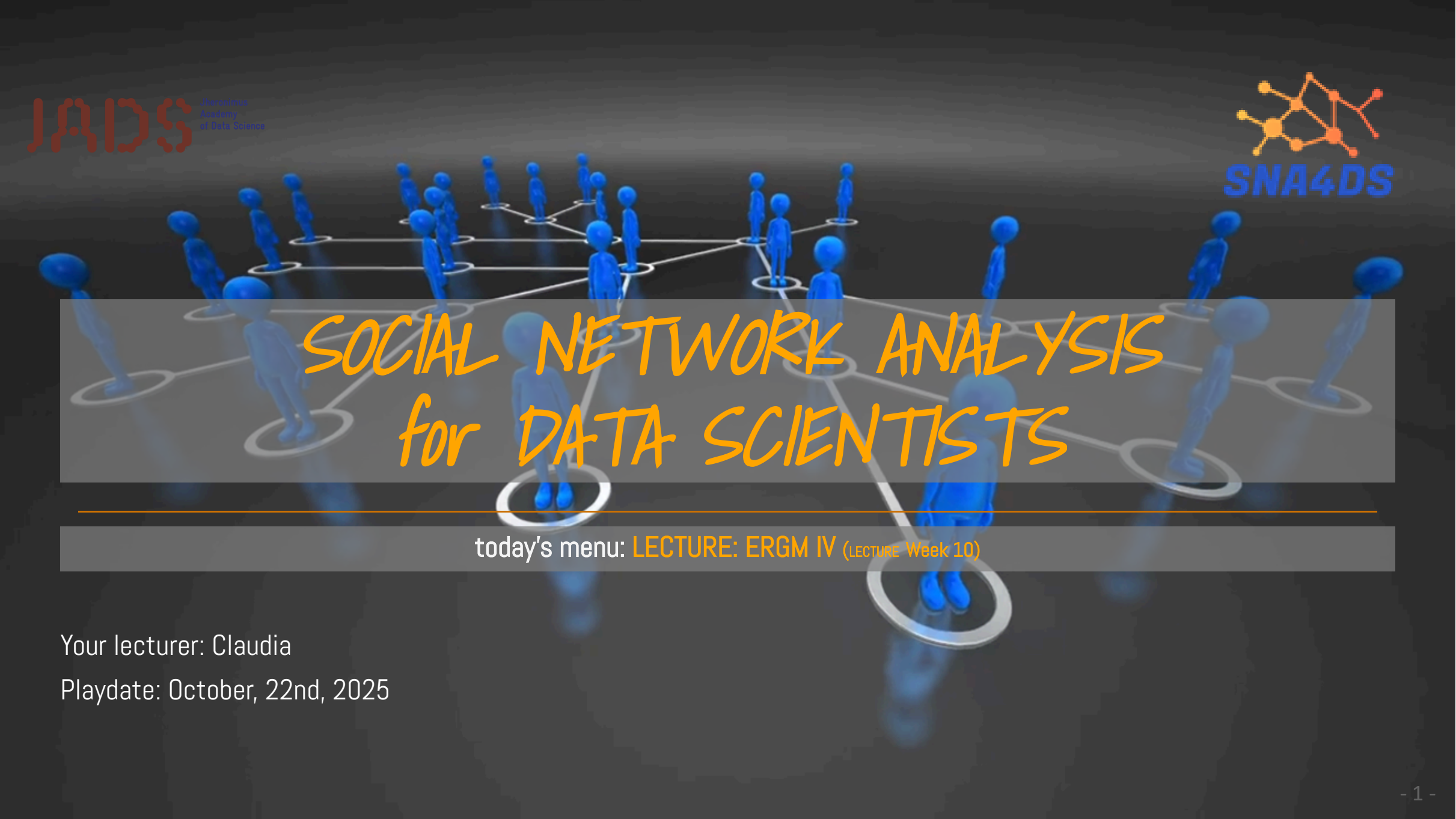


SOCIAL NETWORK ANALYSIS for DATA SCIENTISTS

today's menu: LECTURE: ERGM IV (LECTURE Week 10)

Your lecturer: Claudia

Playdate: October, 22nd, 2025

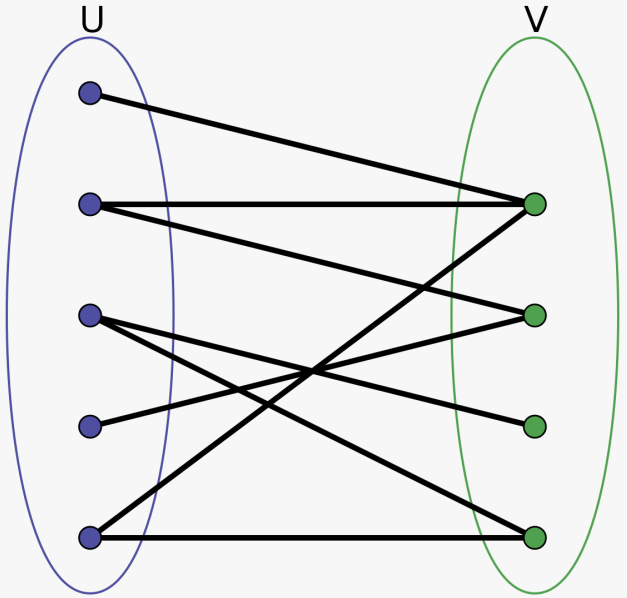
Menu'

- Bipartite ERGMs
- Weighted ERGMs

Bipartite ERGMs

Recap: What's a bipartite graph?

- 2 sets of objects
- connections are allowed across sets
- connections are forbidden within set



Classic Examples

People --- Events

- Kids to birthday parties
- Men to knitting club meetings
- Women to football practices
- ...

Less Common Examples

SNA4DS

Two groups of people

- heterosexual relationships
- consultants and customers
- doctors and patients
- manager and employees
- professors and students

It is about connections that are forbidden or that make no sense for us in that moment

Unimodal (regular) ERGMs

Deal with one set of nodes (each one can connect to each other)

to make it work for bipartite data you can project it

- set 1 are the nodes and being pairwise connected to a node in set 2 creates an edge on this one projection
- set 2 are the nodes and being pairwise connected to a node in set 1 creates an edge on the other projection

Both projections would be undirected by definition

(also, usually bipartite connections go in one direction only -- kinds --> parties)

If you don't want/can't project it, you can use Bipartite ERGMs

Bipartite ERGMs

Conceptually the same as the unimodal

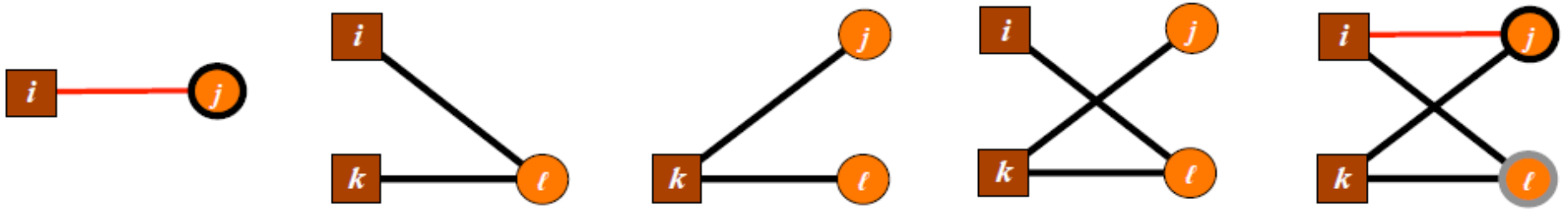
- It works as a unimodal with constraints
- Same formula
- Special terms same concepts

They are cumbersome ergm

- It takes much longer to run
- You need to use the advanced options (control ergm) to make the model work

Bipartite terms

$$\log \Pr(X = x) = \theta_1 z_1(x) + \theta_2 z_2(x) + \cdots + \theta_p z_p(x) + \psi(\theta)$$



Edges

"people" 2-stars

"affiliation" 2-stars

3-paths

4-cycles

4 cycle: Bi-cliques

Bipartite terms

```
ergm::search.ergmTerms(search = 'bipartite')
```

Found 36 matching ergm terms

Some examples

- `nodecov` --> `b1cov` & `b2cov`
- `nodefactor` --> `b1factor` & `b2factor`
- `nodematch` --> `b1nodematch` & `b2nodematch`
- `star(k)` --> `b1star(k)` & `b2star(k)`
- `gwdsp()` --> `gwb1dsp()` & `gwb2dsp()` - no esp since triangles cannot close!
- etc., ...

Toy Example

convergence time is higher than the unimodal ERGM so, tiny data set for the slides

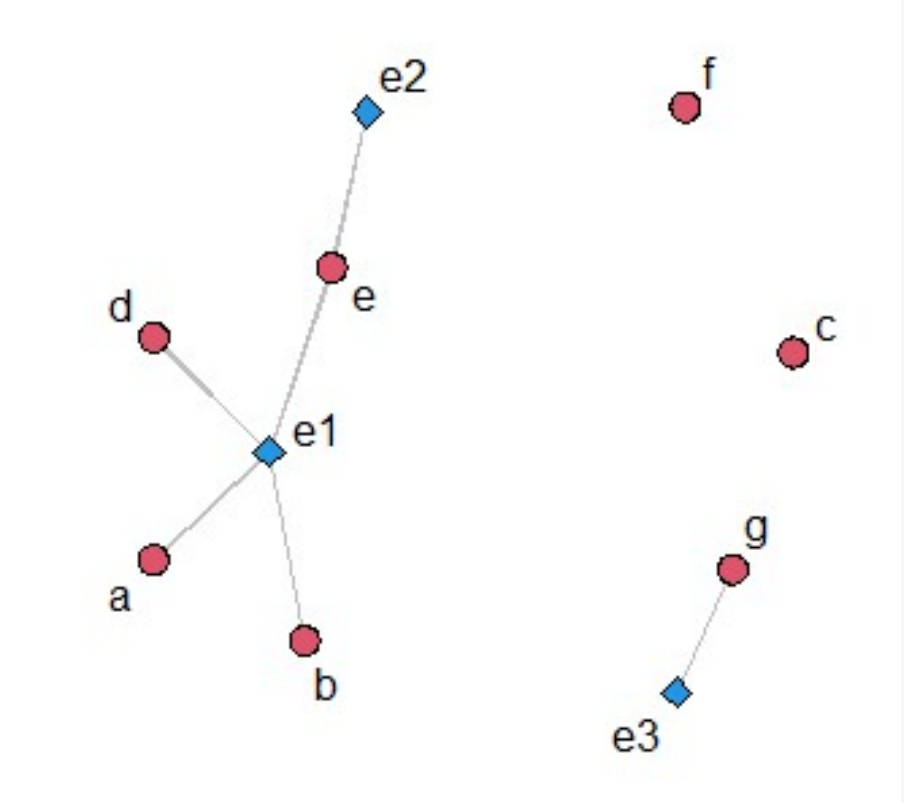
Kids going to events

- 7 kids
- 3 events
- 6 edges
- 1 attribute (clubs red and yellow)

Let's assume we already have

- RQs
- Hs
- chosen model terms

Let's run



Baseline model

```
m0 <- ergm::ergm(kids ~ edges,
                 control = ergm::control.ergm(
                    MCMC.samplesize = 5000,
                    MCMC.burnin = 1000,
                    MCMLE.maxit = 10,
                    parallel = 10,
                    parallel.type = "PSOCK"
                  ) )
summary(m0)
```

```
## Call:
## ergm::ergm(formula = kids ~ edges, control = ergm::control.ergm(MCMC.samplesize = 5000,
##     MCMC.burnin = 1000, MCMLE.maxit = 10, parallel = 10, parallel.type = "PSOCK"))
##
## Maximum Likelihood Results:
##
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges  -0.9163     0.4830      0  -1.897   0.0578 .
## ---
```

Dyadic independent term

```
m1 <- ergm::ergm(kids ~ edges  + b1factor("color", levels = -1),
                 control = ergm::control.ergm(
                    MCMC.samplesize = 5000,
                    MCMC.burnin = 1000,
                    MCMLE.maxit = 10,
                    parallel = 10,
                    parallel.type = "PSOCK"
                  ) )
summary(m1)
```

```
## Call:
## ergm::ergm(formula = kids ~ edges + b1factor("color", levels = -1),
##     control = ergm::control.ergm(MCMC.samplesize = 5000, MCMC.burnin = 1000,
##         MCMLE.maxit = 10, parallel = 10, parallel.type = "PSOCK"))
##
## Maximum Likelihood Results:
##
##                       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                  -0.6931     0.6124      0  -1.132    0.258
```

Dyadic dependent terms

```
m2 <- ergm::ergm(kids ~ edges  + b1factor("color", levels = -1) + b1star(2),
                 control = ergm::control.ergm(
                    MCMC.samplesize = 5000,
                    MCMC.burnin = 1000,
                    MCMLE.maxit = 10,
                    parallel = 10,
                    parallel.type = "PSOCK"
                  ) )
summary(m2)
```

```
## Call:
## ergm::ergm(formula = kids ~ edges + b1factor("color", levels = -1) +
##     b1star(2), control = ergm::control.ergm(MCMC.samplesize = 5000,
##     MCMC.burnin = 1000, MCMLE.maxit = 10, parallel = 10, parallel.type = "PSOCK"))
##
## Monte Carlo Maximum Likelihood Results:
##
##                      Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                 0.08555    1.13800      0   0.075    0.940
```

Are we done?

SNA4DS

We inserted all the terms, are we done?

In this toy example, we might, but in a real bipartite ergm, hardly.

The function `ergm::ergm` is set to run simulation on every possible combination (ignoring the forbidden ones)

But many more combinations are not possible here, then why not saving some computational time?

We need to constraint the model

Constraints

`ergmConstraint -- Sample Space Constraints for Exponential-Family Random Graph Models`

p. 122, ergm user manual https://cran.r-project.org/web/packages/ergm/ergm.pdf

Model the distribution telling to the simulation what needs to stay stable or vary

- "+" increase the constraint
- "-" decrease the constraint

Several options there. Example:

bd <- sets a min and a max for degree

Constraining for degree

- min 0 -- isolates
- max 7 -- number of nodes that can attend party

```
# network::get.vertex.attribute(kids, "color")

m3 <- ergm::ergm(kids ~ edges  + b1factor("color", levels = -1) + b1star(2),
                 constraints= ~ bd(minout = 0, maxout = 10),
                 control = ergm::control.ergm(
                    MCMC.samplesize = 5000,
                    MCMC.burnin = 1000,
                    MCMLE.maxit = 10,
                    parallel = 10,
                    parallel.type = "PSOCK"
                  ) )
summary(m3)
```

```
## Call:
## ergm::ergm(formula = kids ~ edges + b1factor("color", levels = -1) +
##     b1star(2), constraints = ~bd(minout = 0, maxout = 10), control = ergm::control.ergm(MCMC.samplesize
##     MCMC.burnin = 1000, MCMLE.maxit = 10, parallel = 10, parallel.type = "PSOCK"))
##
## Monte Carlo Maximum Likelihood Results:
```

Another approach - blocking connections

Example: you want to model relationships between bosses and employees as a bipartite

- Connections not allowed within the same work category for your study purposes
- This is bipartite in nature
- Rather than model it as a bipartite though, you insert the network as a unimodal
- Then you constraint for impossible relationships between bosses and employees
- you do that using the argument `blocks` e.g., `ergm::ergm(formula, constrainsts = ~ blocks(...)`

offset

Another advanced option in ergm specification

You can use it to improve convergence.

Rather than removing a term that messes with your simulation, you fix its coefficient to something

Enclosing a model term in offset() fixes its value to one specified in offset.coef. (A second argument—a logical or numeric index vector—can be used to select which of the parameters within the term are offsets.)

Control ergm

Since your simulation very likely is still clunky, you need to set up these parameters too

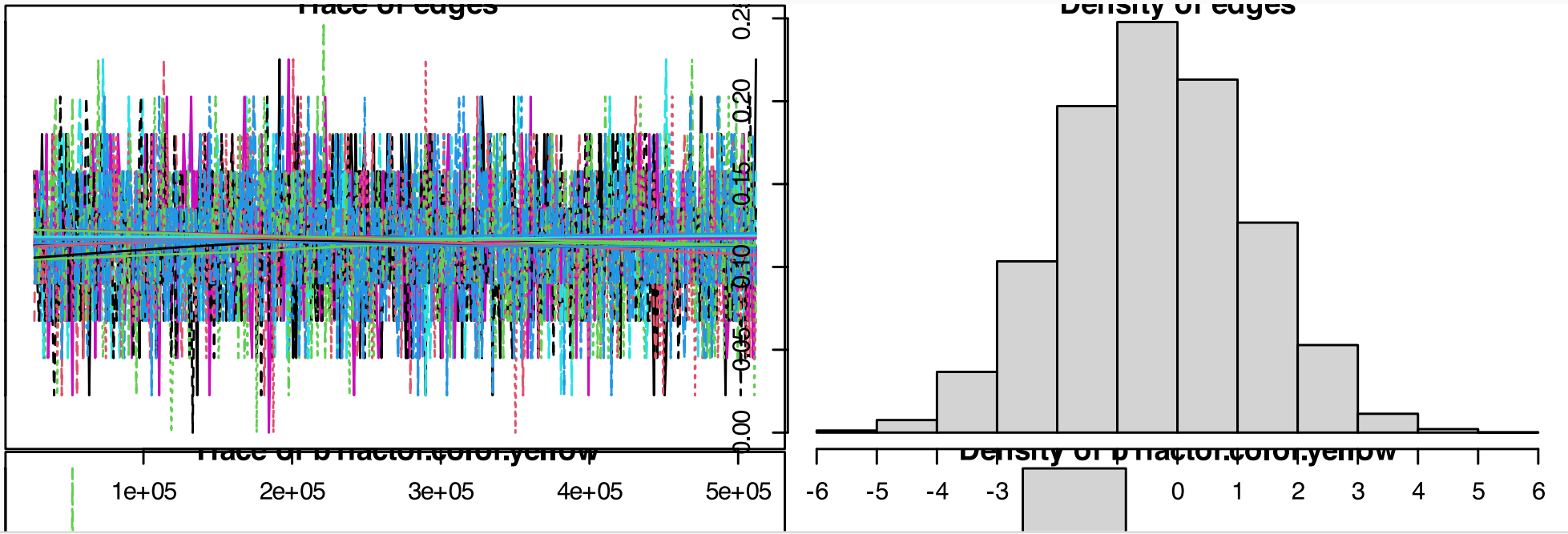
```
# network::get.vertex.attribute(kids, "color")

m4 <- ergm::ergm(kids ~ edges  + b1factor("color", levels = -1) + b1star(2),
                 constraints= ~ bd(minout = 0, maxout = 10),
                 control = ergm::control.ergm(
                    MCMC.samplesize = 5000,
                    MCMC.burnin = 1000,
                    MCMLE.maxit = 10,
                    parallel = 10,
                    parallel.type = "PSOCK"
                  ) )
summary(m4)
```

```
## Call:
## ergm::ergm(formula = kids ~ edges + b1factor("color", levels = -1) +
##     b1star(2), constraints = ~bd(minout = 0, maxout = 10), control = ergm::control.ergm(MCMC.samplesize
##     MCMC.burnin = 1000, MCMLE.maxit = 10, parallel = 10, parallel.type = "PSOCK"))
##
## Monte Carlo Maximum Likelihood Results:
```

MCMC.diagnostics

The usual way

```
ergm::mcmc.diagnostics(m4)
```



The usual way

```
 ergm::gof(m4)


##
## Goodness-of-fit for model statistics
##
##                        obs min mean max MC p-value
## edges                    6   1 5.84  10          1
## b1factor.color.yellow    2   0 1.90   5          1
## b1star2                  1   0 0.99   4          1
##
## Goodness-of-fit for minimum geodesic distance
##
##     obs min  mean max MC p-value
## 1     6   1  5.84  10       1.00
## 2     7   0  5.56  18       0.60
## 3     3   0  2.20  11       0.76
## 4     0   0  1.31   8       1.00
## 5     0   0  0.26   4       1.00
## 6     0   0  0.07   3       1.00
```

Interpreting Results - Final model

The advanced `ergm` options improved the IAC and BIC

```
 texreg::screenreg(m4)
```

```
##
## ==============================
##                         Model 1
## ------------------------------
## edges                    0.02
##                         (1.25)
## b1factor.color.yellow   -0.82
##                         (1.42)
## b1star2                 -1.23
##                         (1.62)
## ------------------------------
## AIC                      0.87
## BIC                      4.01
## Log Likelihood           2.56
## ==============================
## *** p < 0.001; ** p < 0.01; * p < 0.05
```

Weighted ERGMs

Weighted Networks

A weighted network is a network where the edges express the weight or intensity of the relationship.

It is still unimodal, but it contains more information.

Different kinds of weight

Example 1: Friendship network

Built using two questions

- Who are your closest friends? --> ties
- How many times per week do you see them? --> weight

In this case if you don't consider the weight you can still predict tie formation.

It makes sense.

The weight is more like an attribute.

An helpful but not essential extra info

The network is not fully connected

Different kinds of weight

Example 2: Financial Exhange Network

- Nodes: countries
- Edges: whether money moves from one to the other

This network is fully connected

The weight ($$$) makes all the difference.

Type two

How to ask causal questions (edge formation) on this type 2 weighted networks?

Option 1: You find a meaningful threshold and remove the weight.

- Above N (dollars) -- edge between 2 countries
- Below N (dollars) -- no edge between 2 countries

It might work, but you are wasting lot of information

The right thing to do would be explaining edge formation with a ERGM specifically designed for weighted data.

Yes, How?

Mathematically, the weight adds complexity to the problem.

The ERGM formula needs to be expanded to include this new bit.

How?

It depends on how the problem is formulated in the first place

DUH!

Get to know your weight!

Variables.

What kind of variable is your weight?

- Count
- Continuous (either bounded, bounded on one side, or unbouded)

Count

Sampson Monks, 3 networks:

- Like each other in time 1
- Like each other in time 2
- Like each other in time 3

"Like" can have 3 values (if connected at all)

- 1 liked at some point (so not much overall)
- 2 liked most of the time (likelihood with some issues)
- 3 real friendship (likelihood lasts over time)

If you have a case like this you have a count weight [1,2,3, ..., N) -- bounded on one side

May `statnet` be with you (`ergm.count` packages)

Continuous

Financial exchange

Continuous (...Dollars...) Unbounded.

Any amount of money (also negative)

This is a more general case that encompasses the one before as well.

`statnet` does not help us anymore

We need a new environment to estimate the probability of forming "financial edges" between pairs of countries.

We have the Generalized Exponential Random Graph Model (GERGM)

[introduced by the authors of your text book]

Desmarais, B. A., Cranmer, S.J. (2012), Statistical Inference for valued-edge networks: The Generalized Exponential Random Graph Model, PLOS ONE

GERGM

Even if the analogy with the GLM is obvious, it is not really the same thing.

In the GERGM the network is always a fully connected one. Hence it cannot handle every type of network as the name suggests.

A GERGM is defined by two equations respectively representing:

1. Effect of covariates on the edge formation together with the marginal distribution properties of the edges (the old familiar equation)
2. The relationships among the edges

The probability distribution is conditioned on two parameters θ and γ

GERGM Math

Original ERGM equation

$$P(N,\theta) = \frac{exp\{\theta^T h(N)\}}{\sum_{N^* \in \mathcal{N}} exp\{\theta^T h(N^*)\}^T}$$

GERGM

$$fX(x|\theta) = \frac{exp\{\theta^T h(x)\}}{\int_{[0,1]^m} exp(\theta^T h(z))dz^T}$$

With $x \in [O, I]^m$ that represents the way the edges values relate to each other.

$\mathcal{N}$ does not denote anymore the adjancency matrix since the network is fully connected.

We refer at this network with X instead.

GERGM Math: bonus equation

Up to this point we can handle

- a fully connected network with weights bounded between 0 and 1
- parameters ranging from 0 and 1

We want more! Hence, here is your bonus equation!

$$f_Y(y|\theta,\gamma) = \frac{exp(\theta^T h(T(y|\gamma))}{\int_{[0,1]^m} exp(\theta^T h(z))dz^T} \prod_{i,j} t_{i,j}(y_{i,j}|\gamma)$$

This equation models the distribution of the edge weights and the parameters.

Nice to meet (see) you GERGM package

Installation:

```
remotes::install_github("matthewjdenny/GERGM", dependencies = TRUE)
```

User manual

`https://github.com/matthewjdenny/GERGM`

Vignette

in the R console `browseVignettes("GERGM")`

Intercept

`edges` - same as in `ergm` - still used as intercept.

More options available:

- edges(method = "endogenous") - to include the intercept as in a traditional ERGM
- edges(method = "regression") - to include an intercept in the lambda transformation of the network

GERGM Endogenous model terms:

- `twostars`, equivalent to `star(2)`
- `out2stars`, equivalent to `ostar(2)`
- `in2stars`, equivalent to `istar(2)`
- `ctriads`, same as in `ergm`
- `mutual`, same as in `ergm`
- `ttriads`, same as in `ergm`

Convergence problems? Use exponential down-weighting

E.g., `out2stars(alpha = 0.25)` - default = 1

GERGM Exogenous model terms:

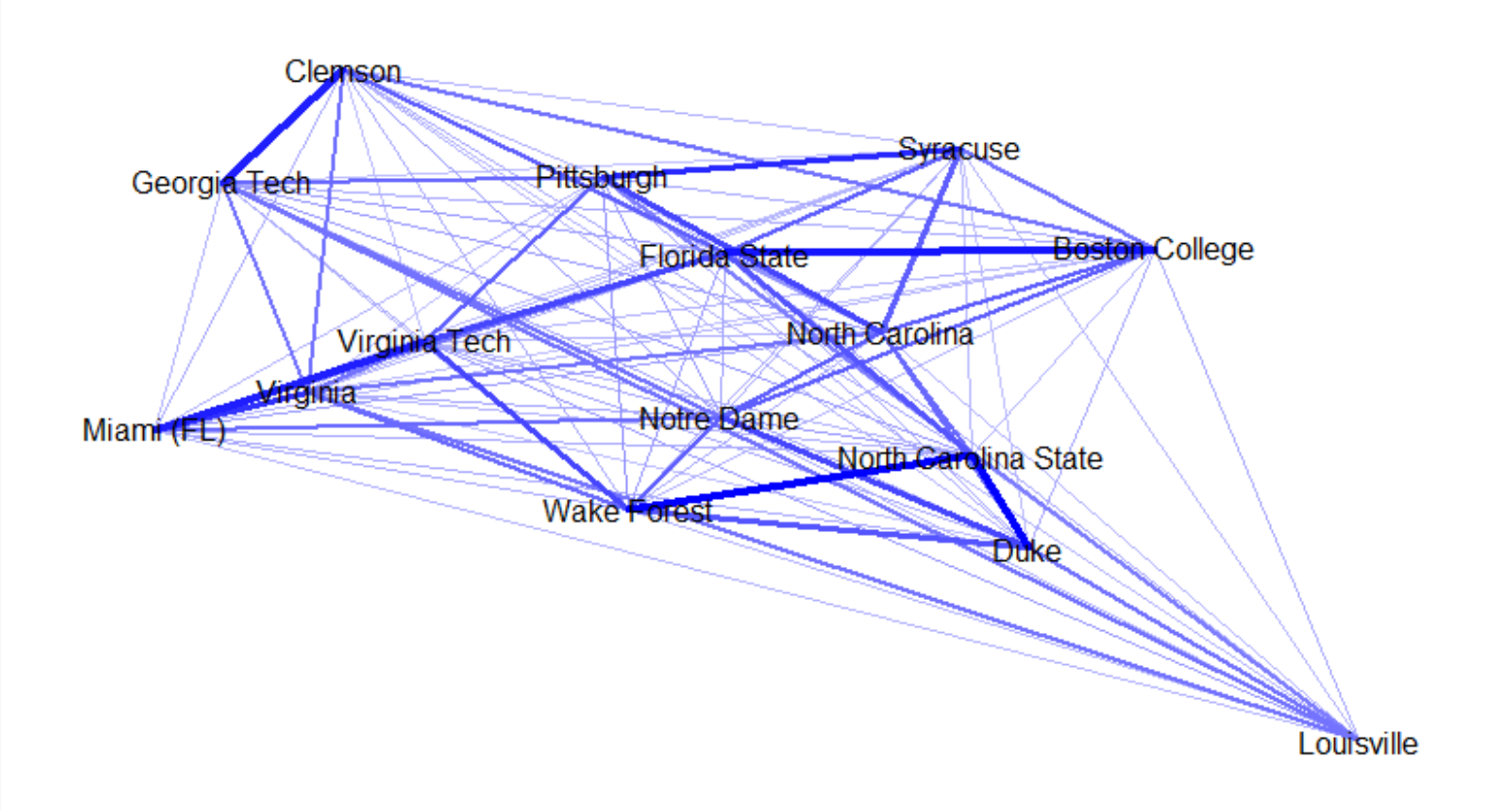
SNA4DS

Very similar to the `ergm` package

- `absdiff(covariate = "MyCov")`
- `sender(covariate = "MyCov")` -- != from `ergm` here attribute - nodeocov
- `receiver(covariate = "MyCov")` -- != from `ergm` here attribute - nodeicov
- `nodematch(covariate = "MyCov", base = "Ref.cat")`
- `nodemix(covariate = "MyCov", base = "Ref.cat")`
- `netcov(network)` -- like `edgecov` in `ergm`

GERGM example from your book

- nodes = Basketball teams
- edge-weight = N of points that i scored to j during the season



Data Format

NO NETWORK or IGRAPH class

- adjacency weighted matrix
- data frame with attributes (all together)

```
 class(adjacencyMatrix)

## [1] "matrix" "array"

 dim(adjacencyMatrix)

## [1] 15 15

 class(covariateData)

## [1] "data.frame"

 dim(covariateData)
```

Model Specification

SNA4DS

Let's assume that

- we have our solid theory ready and we can specify the model already
- we run baseline models and other control models already

We are running our final model

Nope! we specify the formula first!

```
formula <- adjacencyMatrix ~ edges +
  sender("enrollment") +
  receiver("enrollment") +
  netcov(nGames) +
  mutual(alpha = .9)
```

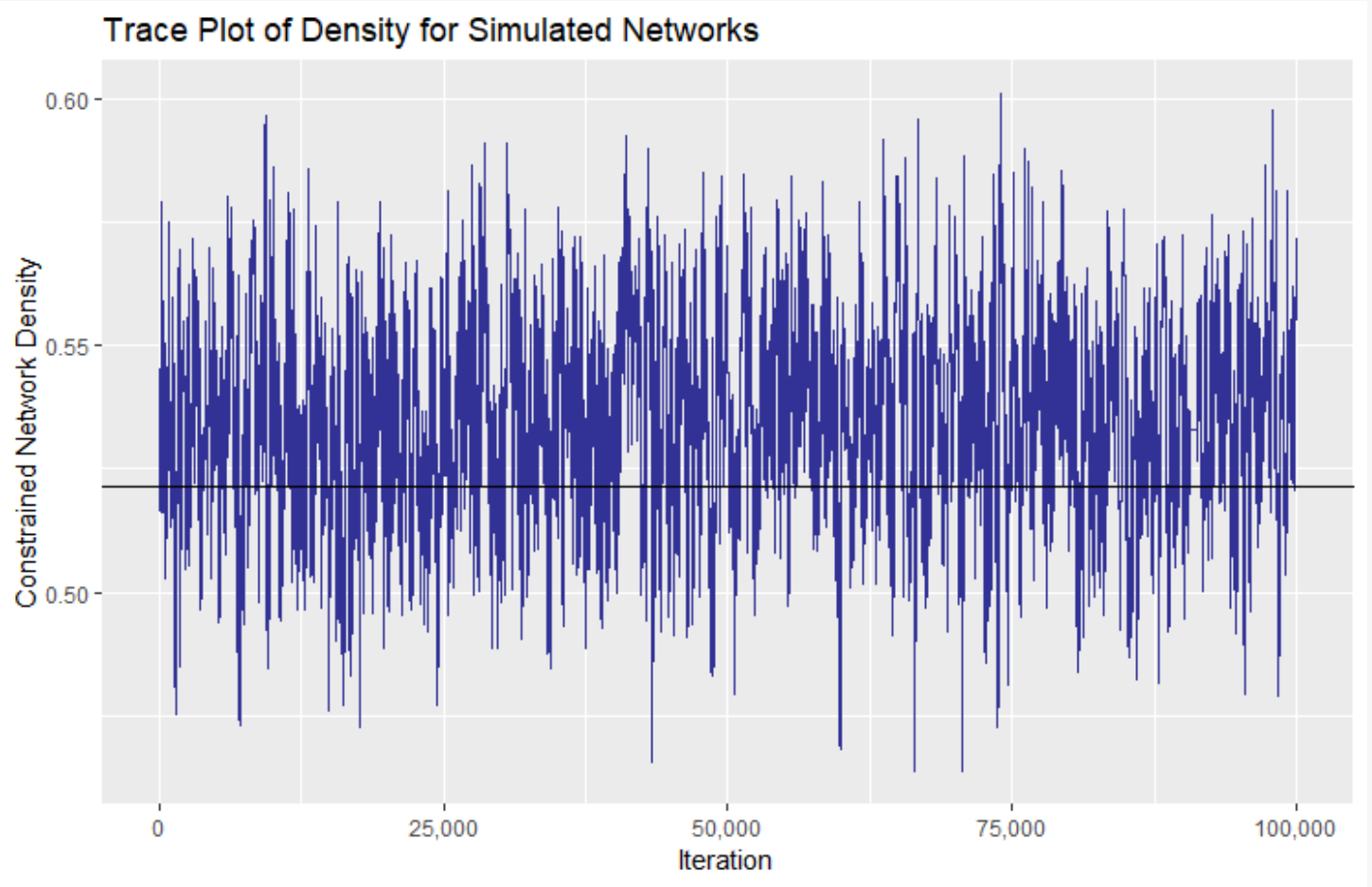
Now we are running the model

SNA4DS

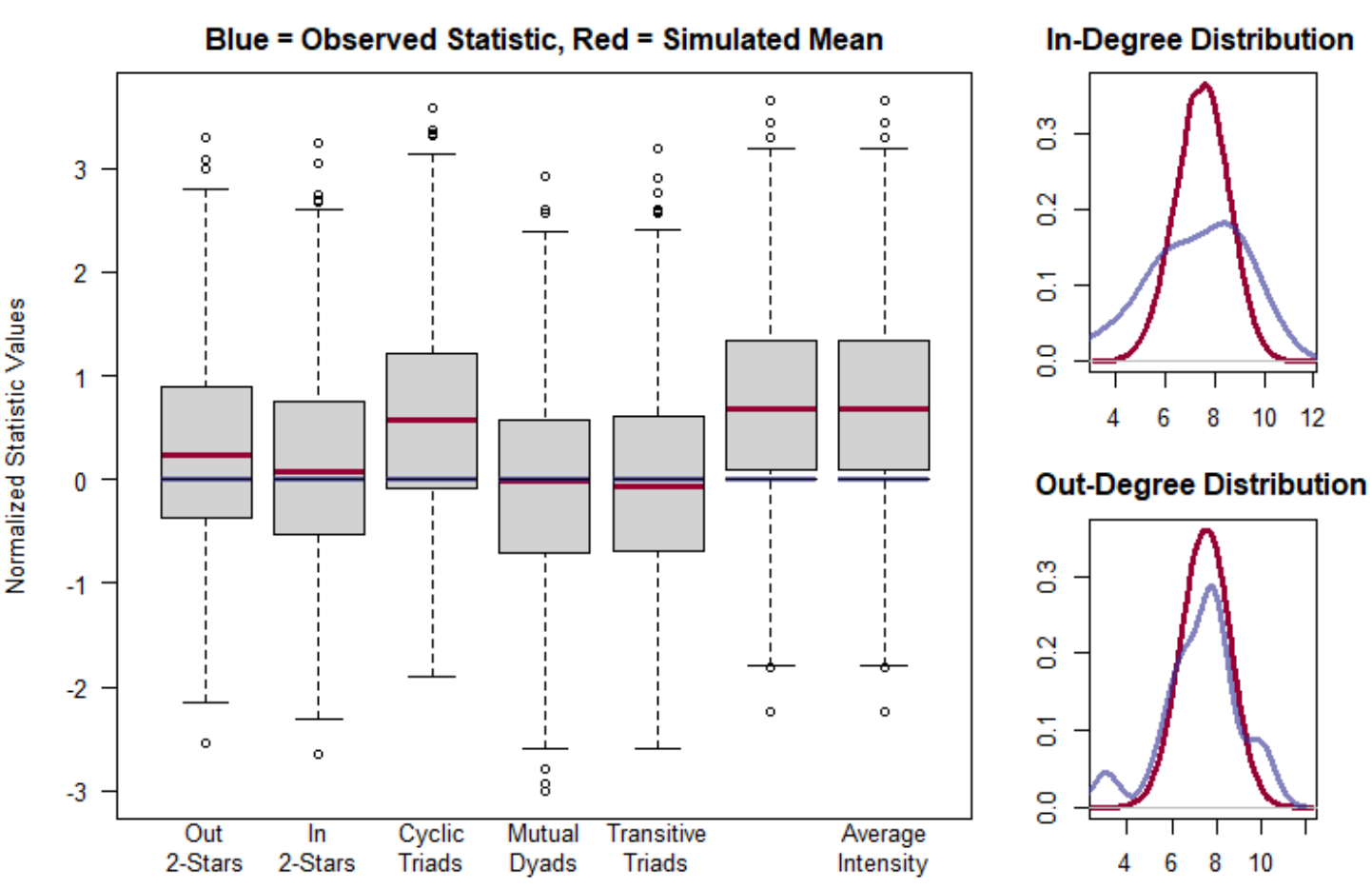
```
set.seed(5)

gergmResults <- GERGM::gergm(formula,
                             covariate_data = covariateData, # passing attributes on
                             number_of_networks_to_simulate = 100000,
                             MCMC_burnin = 10000,
                             thin = 1/10, # retaining only a small number of simulated Networks in the c
                             transformation_type = "Cauchy", # it depends on the distribution of your we
                             parallel = TRUE,
                             cores = 6)


## You have specified 2 node level covariate effects.
## You have provided 1 network covariates.
## [1] 1 1 1 1 1 1 1
## Updating Estimates -- Iteration: 1
## Initial Lambda Estimates:
##    enrollment_sender  enrollment_receiver           intercept       nGames_netcov dispersion paramete
##            0.0000000           0.0000000           0.9474667           0.0000000          -0.659823
## initial  value 327.330101
```

Now we are exploring the results

```
GERGM::Trace_Plot(gergmResults)
```

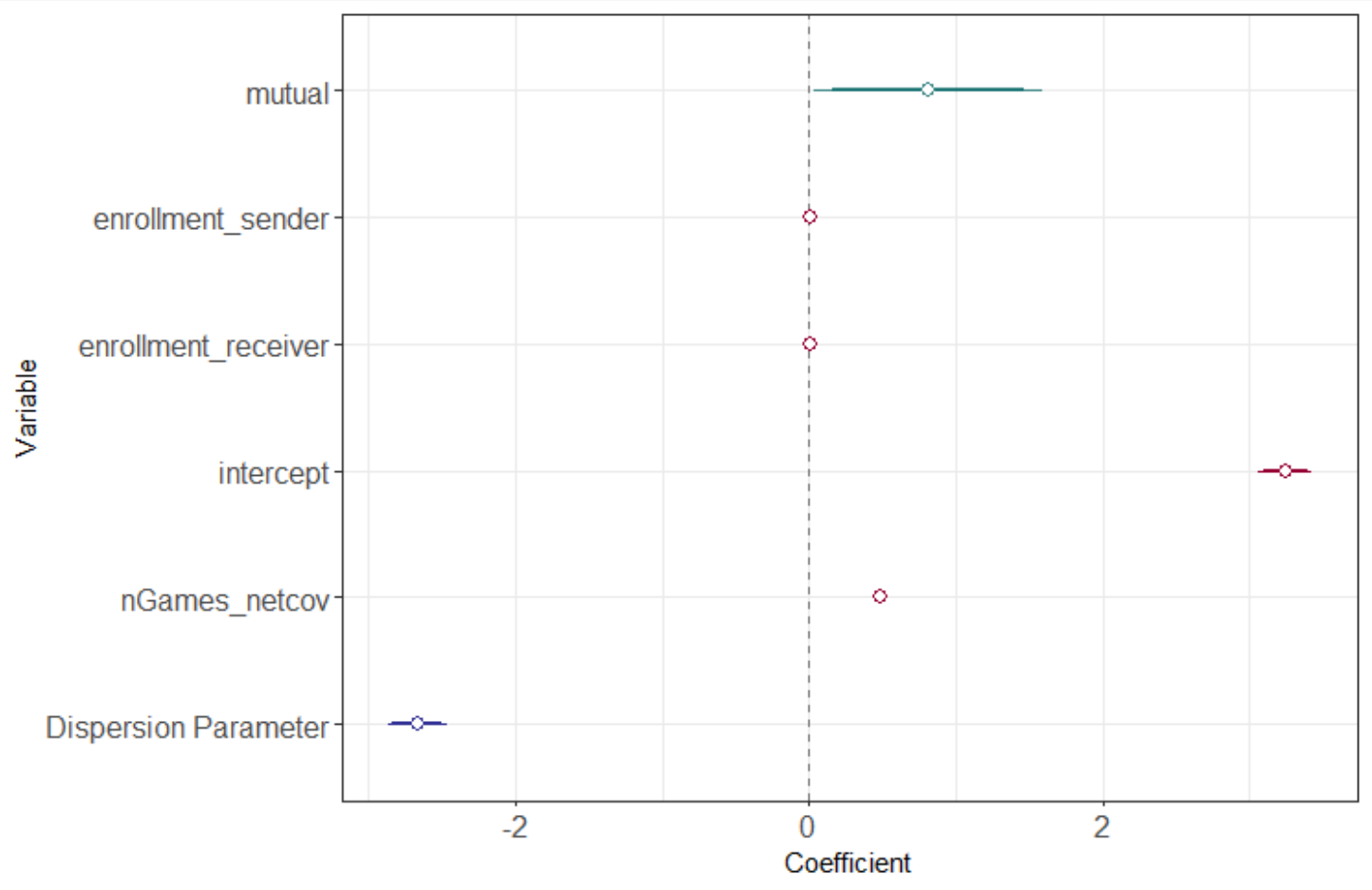


Now we are exploring the results



Now we are exploring the results

```
GERGM::Estimate_Plot(gergmResults)
```



Now we are exploring the results


```
GERGM::Trace_Plot(gergmResults)
```

```
(EstSE <- rbind(t(attributes(gergmResults)$theta.coef),
                t(attributes(gergmResults)$lambda.coef)))
```

```
##                              est          se
## mutual                0.811425685 0.398744983
## enrollment_sender     0.003329245 0.001086689
## enrollment_receiver   0.004628842 0.011306715
## intercept             3.245288935 0.092024635
## nGames_netcov         0.482791749 0.015208010
## dispersion           -2.669664798 0.101491601
```


It's Over