Likelihoods of Weight Loss or: ACRONYM: Augmented degree corrected, Community Reticulately Organized Network Yielding Model

Benjamin Leinwand

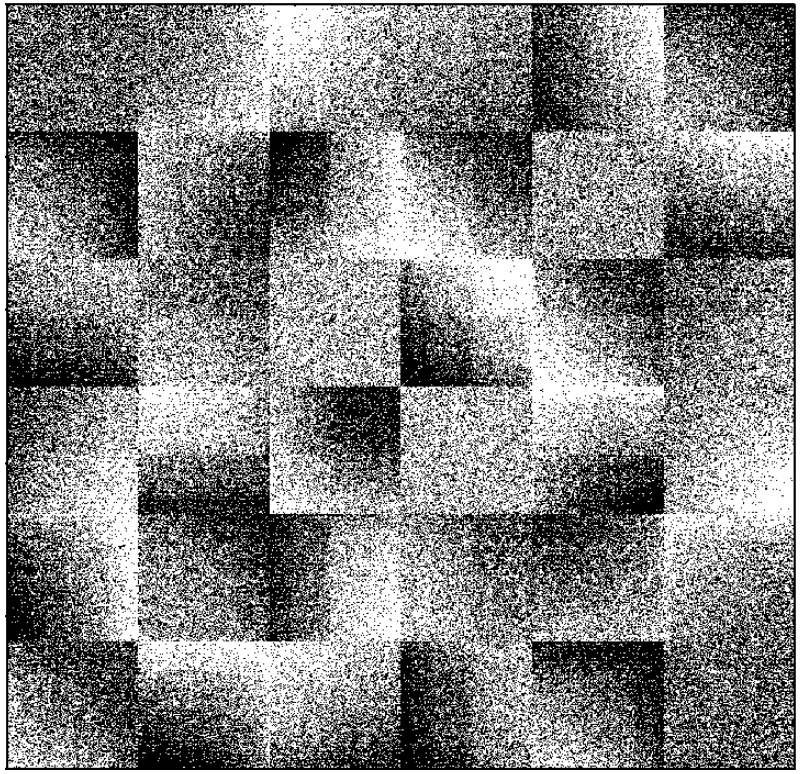
Joint work with Vince Lyzinski


WE NEED A NAME THAT'S WITTY AT FIRST,
BUT THAT SEEMS LESS FUNNY EACH TIME YOU HEAR IT.

Outline

- Motivating example, getting used to the ideas and gestalt
- Problems with existing approaches
- *H*-functions
- ACRONYM
- Parameter Estimation using Likelihood
- Simulation Results

Practice exercise

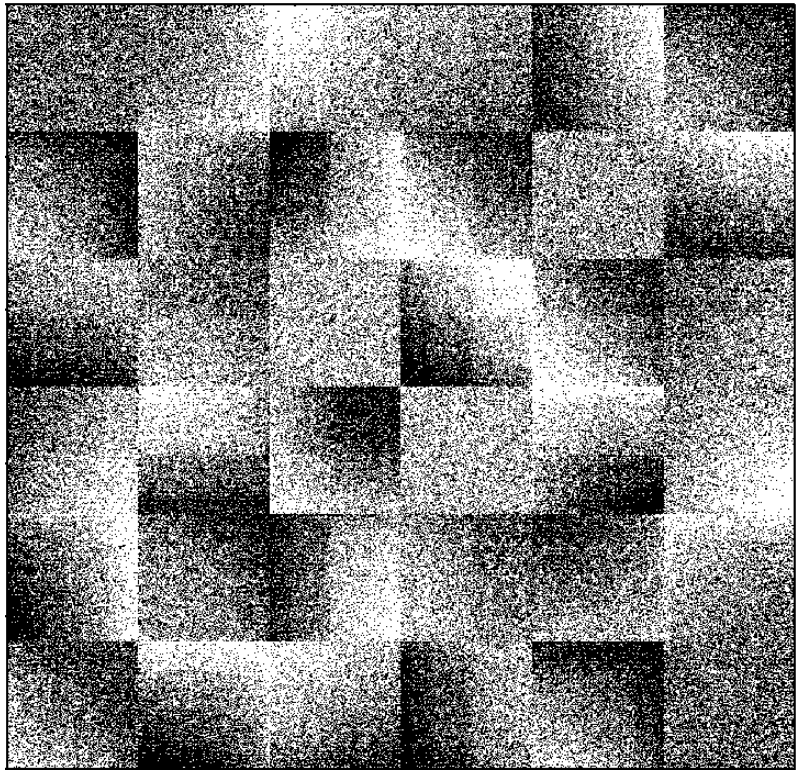


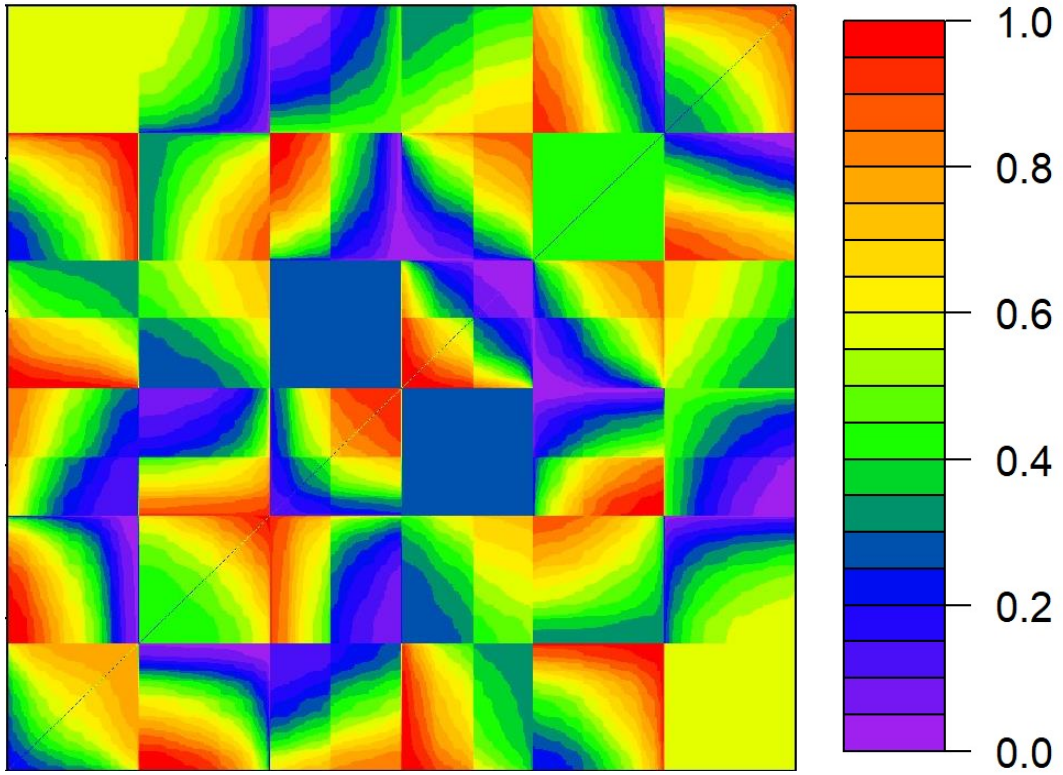
- Look at the plot on the left
- This is the adjacency matrix of a simulated network with 1200 nodes
- Black dots represent edges (1’s), white dots are non-edges (0’s)
- It is symmetric, with each node taking up both a horizontal and vertical line

Take a moment to try to visualize a heatmap of high and low probability areas for the network on the left

Practice exercise

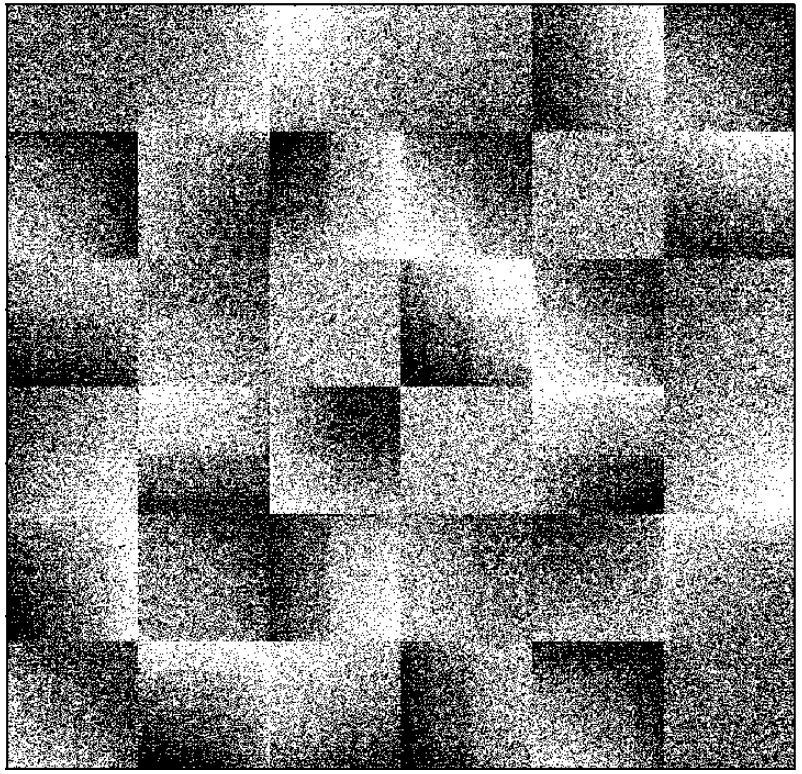
Note: diagonal runs from southwest to northeast

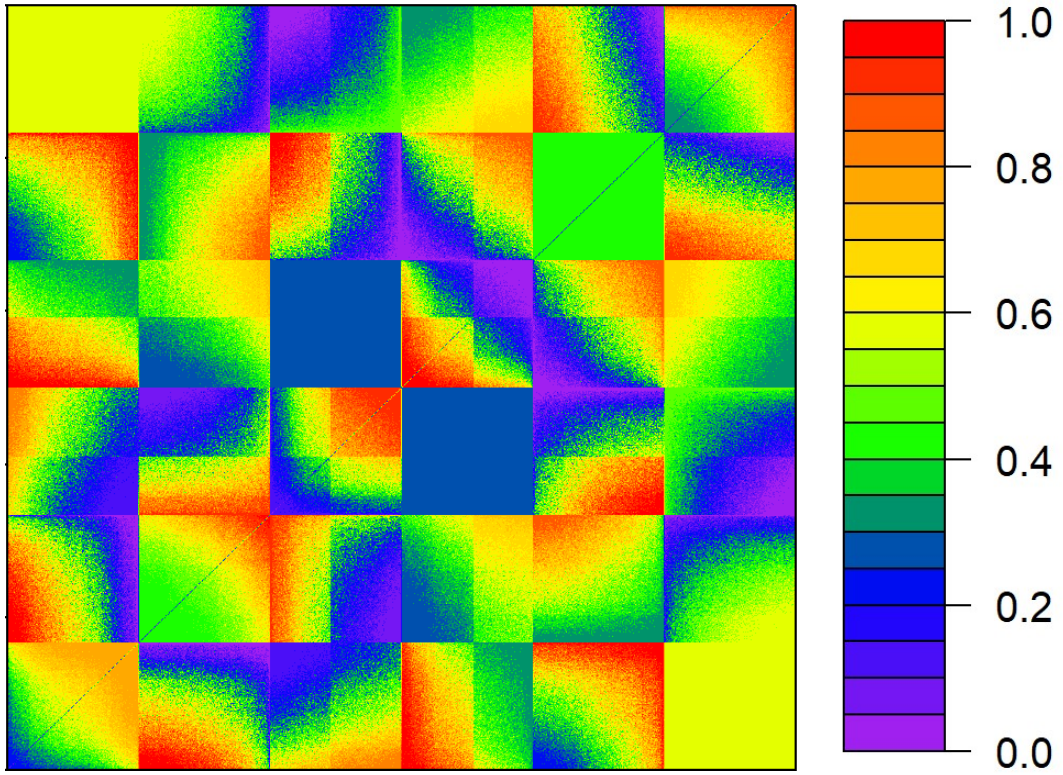




Did your mental model look like the image on the right?
Is anything different than you expected? What else do you see?

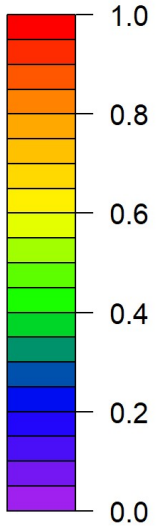
Practice exercise





The left network was actually sampled from the probability matrix on the right, which is a noisier version of the matrix shown on the previous slide

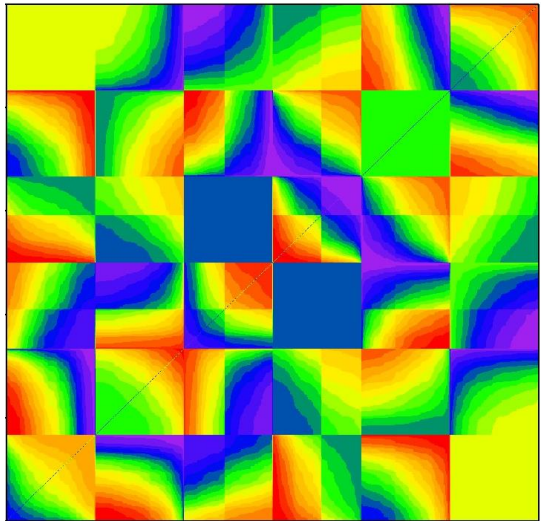
Candidate estimate 1



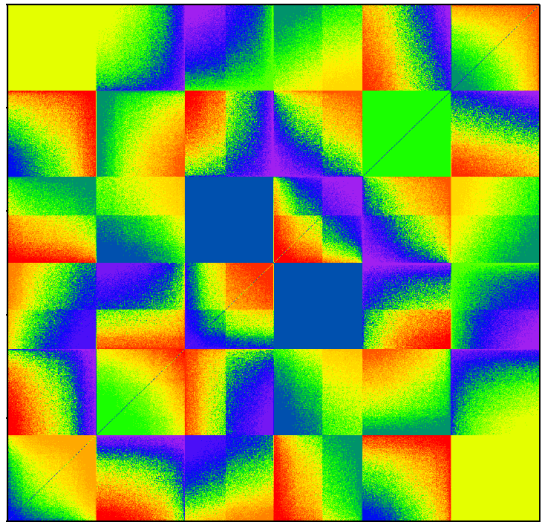
Let’s say we were trying to estimate the underlying probability matrix for the observed network

How does the image on the bottom right look?

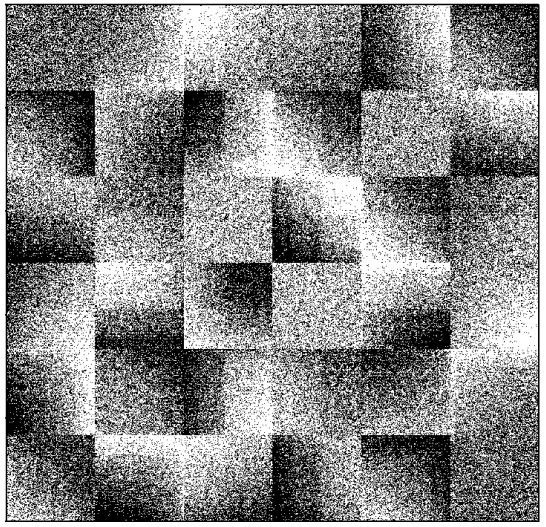
Noise-free probability matrix



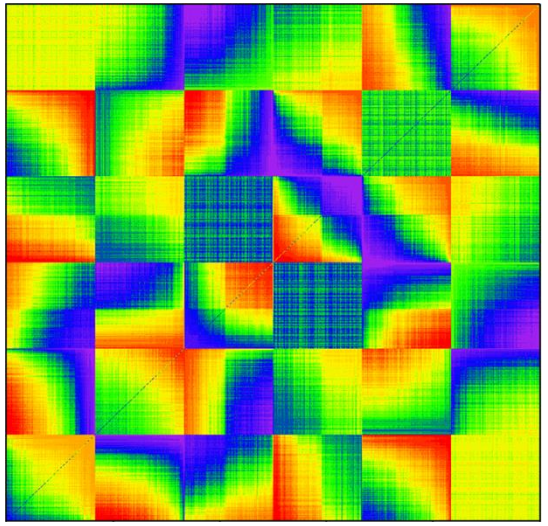
True Probability Matrix



Observed Network



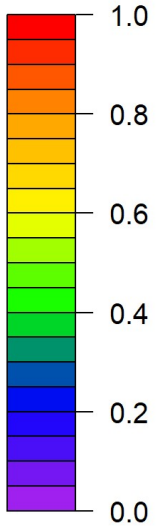
Estimated probability matrix



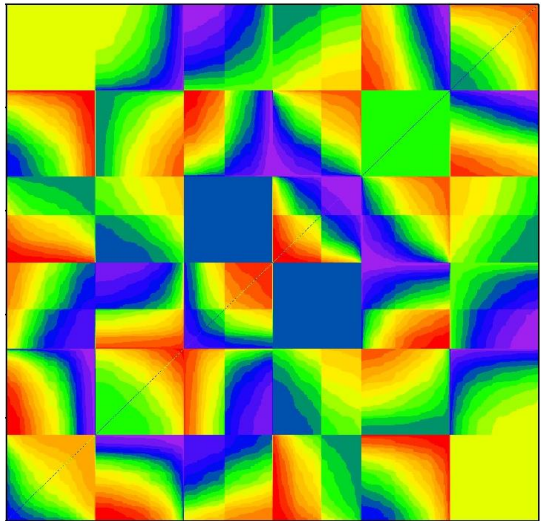
Candidate estimate 2

How about this new image? On the bottom right?

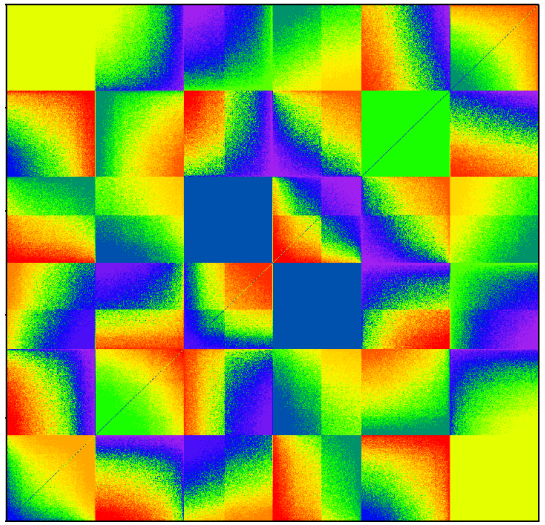
What are those white spots?



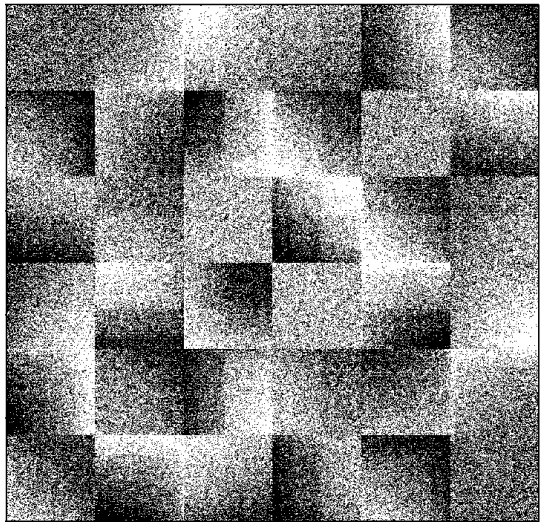
Noise-free probability matrix



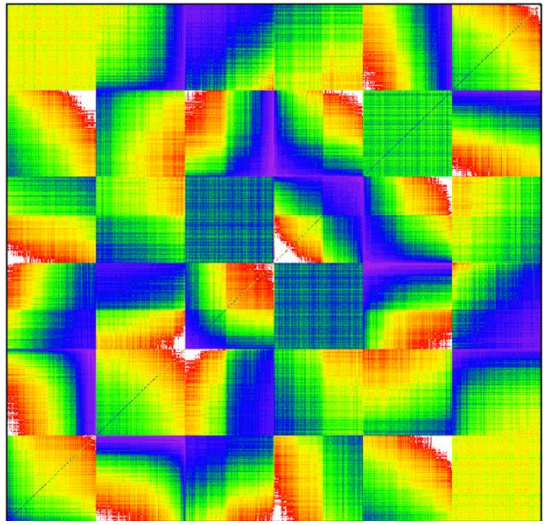
True Probability Matrix



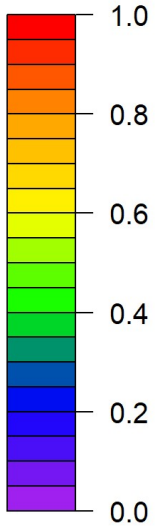
Observed Network



Estimated probability matrix



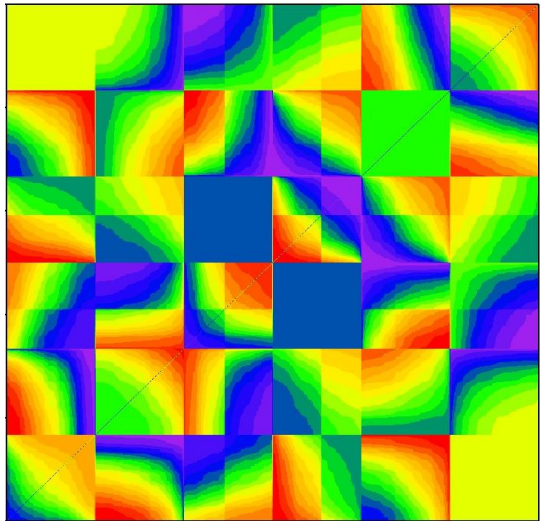
Candidate estimate 3



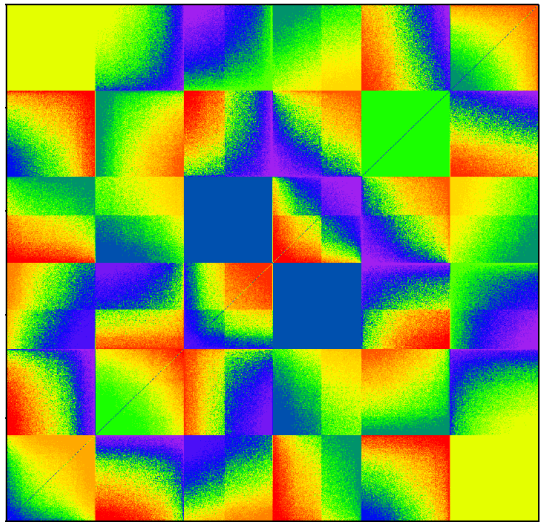
Finally, what about this estimate?

Something has gone wrong. And still white spots!

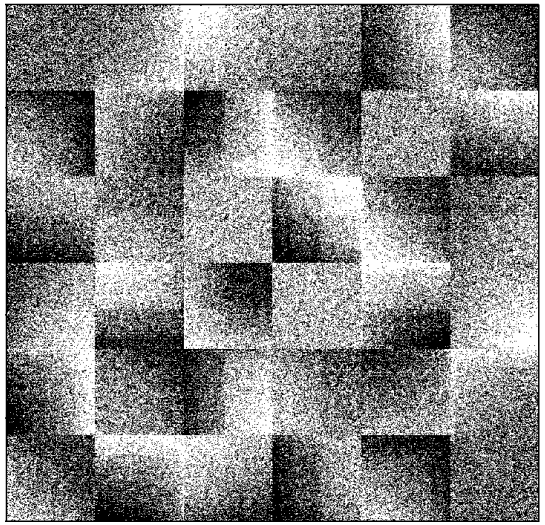
Noise-free probability matrix



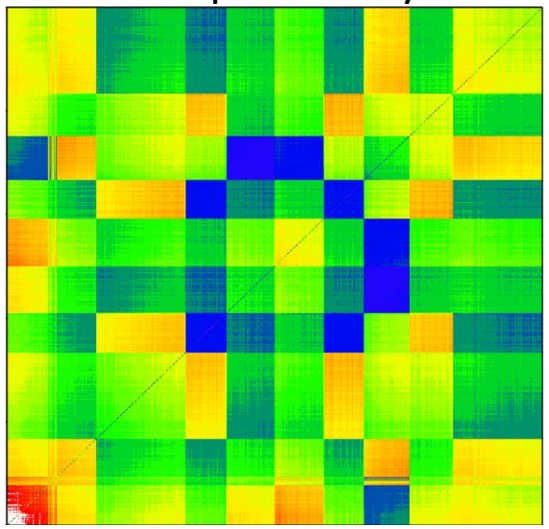
True Probability Matrix



Observed Network



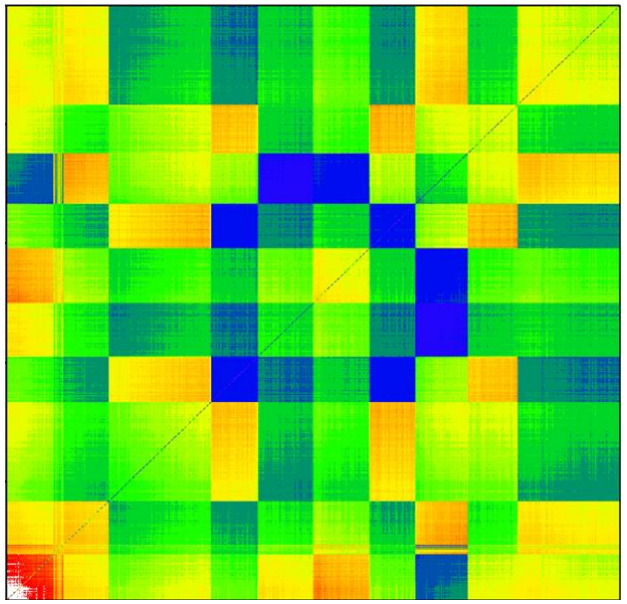
Estimated probability matrix



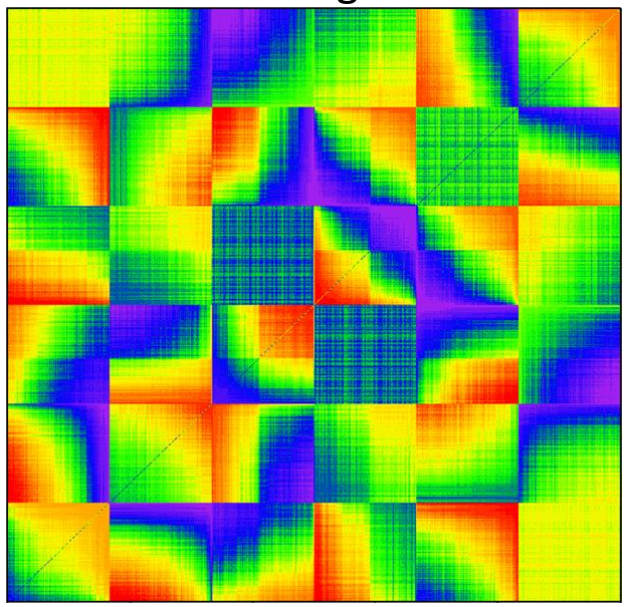
Big (self-serving!) reveal

Current approaches seem to run into issues on this example

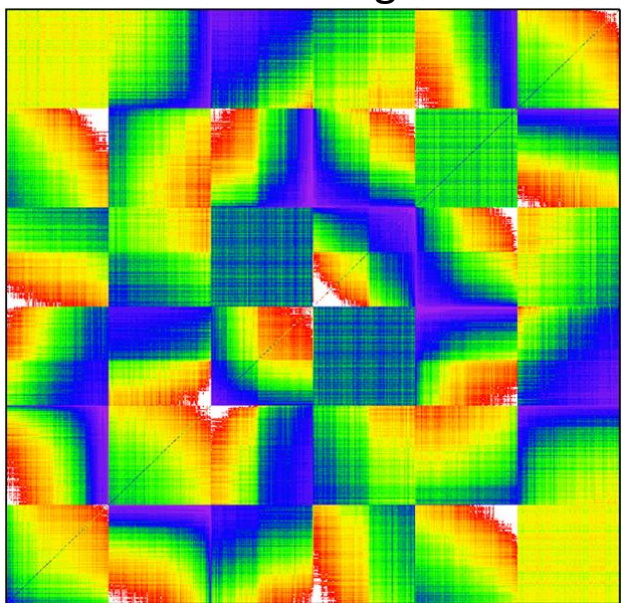
Estimated using DCBM



Estimated using ACRONYM



Estimated using PABM



What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error to get a standard Normal RV

4) Convert back to a uniform value

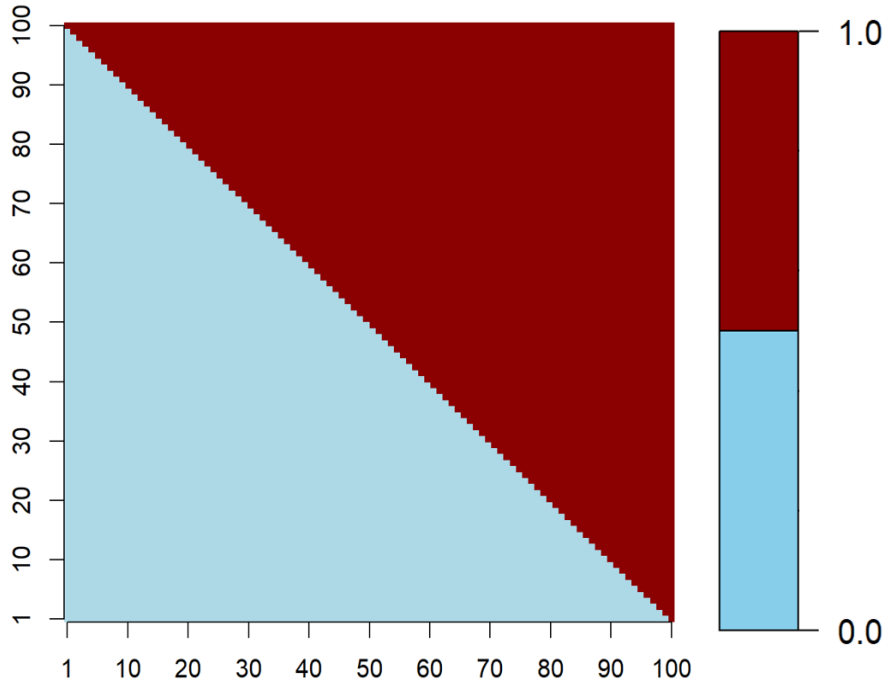
5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

Illustrative example - Threshold

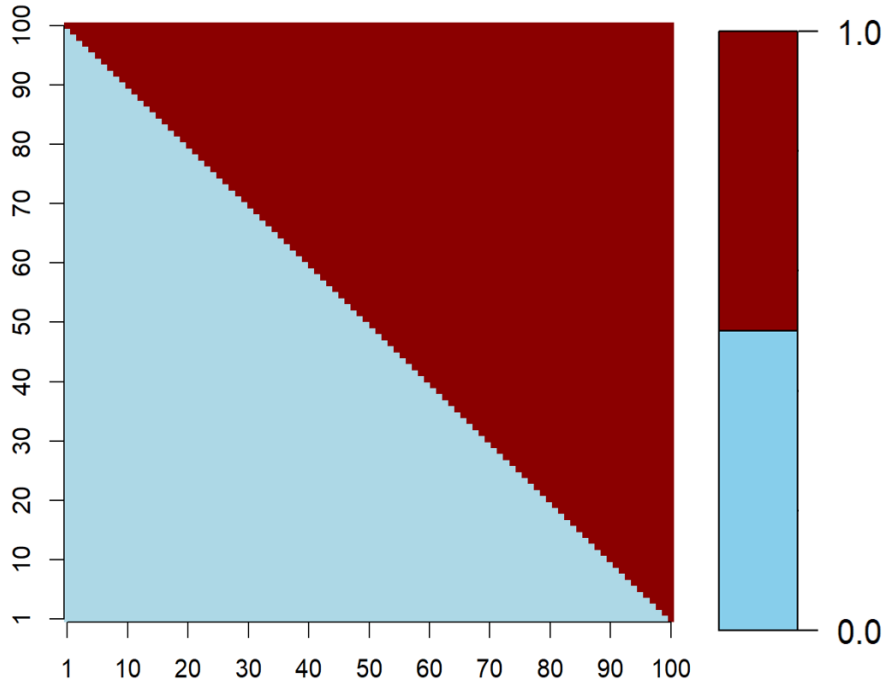
- Consider the toy **induced subnetwork** shown below
- There are 2 communities, each containing 100 nodes numbered 1 through 100; the image represents the edges between the two communities



- If two nodes in different communities have numbers whose sum exceeds 100.5, an edge is placed between these nodes
- This is **not** generated using a DCBM or PABM model

Where do other methods go wrong?

- To estimate the edges using a DCBM or PABM, each node *u* gets a parameter θ_u and a block parameter β
- There are a total of 5050 edges in the subnetwork
  - Node 100 in each community is incident to 100 of the 5050 edges,
  - Node 99 in each community is incident to 99 of these 5050 edges, etc.



The estimated edge probability between the nodes labeled 100 in each community is given by:

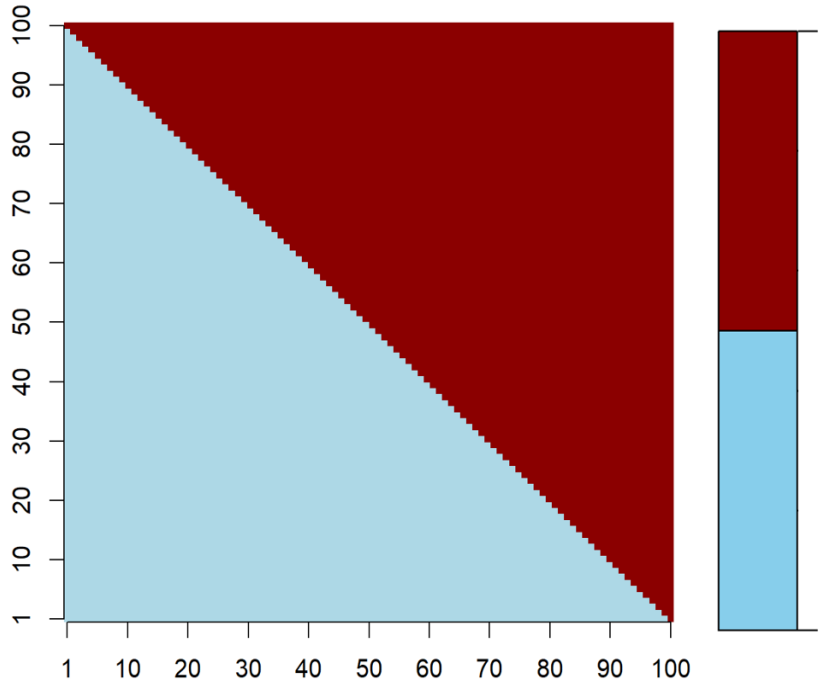
$$\hat{\theta}_u \hat{\beta} \hat{\theta}_v = \frac{100 \times 100}{5050} \times 5050 \times \frac{100 \times 100}{5050}$$

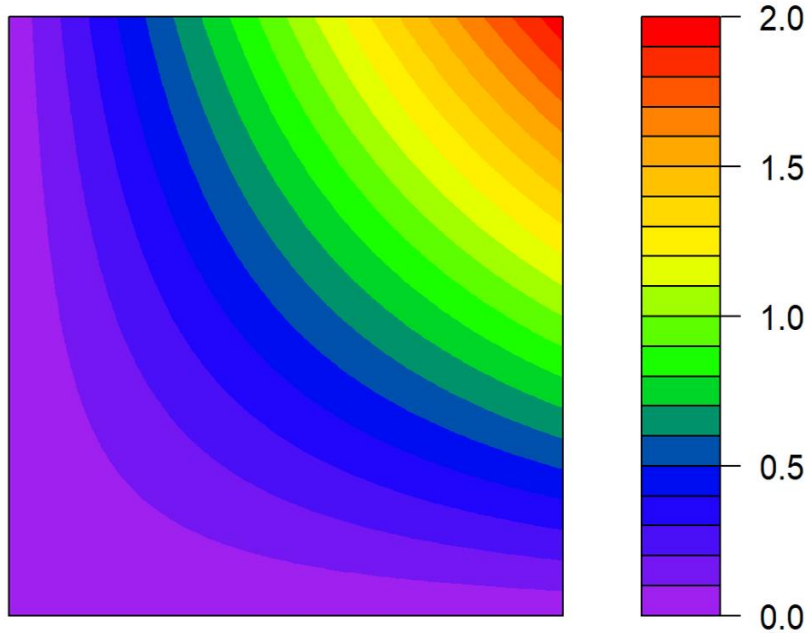
$$= 1.\overline{9801}$$

- DCBMs were initially conceived for networks with Poisson edges, where this value would make sense

Problems in the toy estimate

- We want this estimate to be the expected value of a {0, 1} valued edge, that is, a probability
- Looking at the plot on the right, many of the estimates are greater than 1. Assumptions which avoid generating results like this may not match real observed networks
- We also see convex curved contours in the estimate, even though they don't appear in the network





Objections

Why do we need another model? And this one's so complicated!

Menger: "*Entities must not be reduced to the point of inadequacy*"

We've shown failures of other approaches on simulations, real examples next slide

This seems fine for dense networks, but most networks are sparse!

Some networks are dense, and some sparse networks have dense region

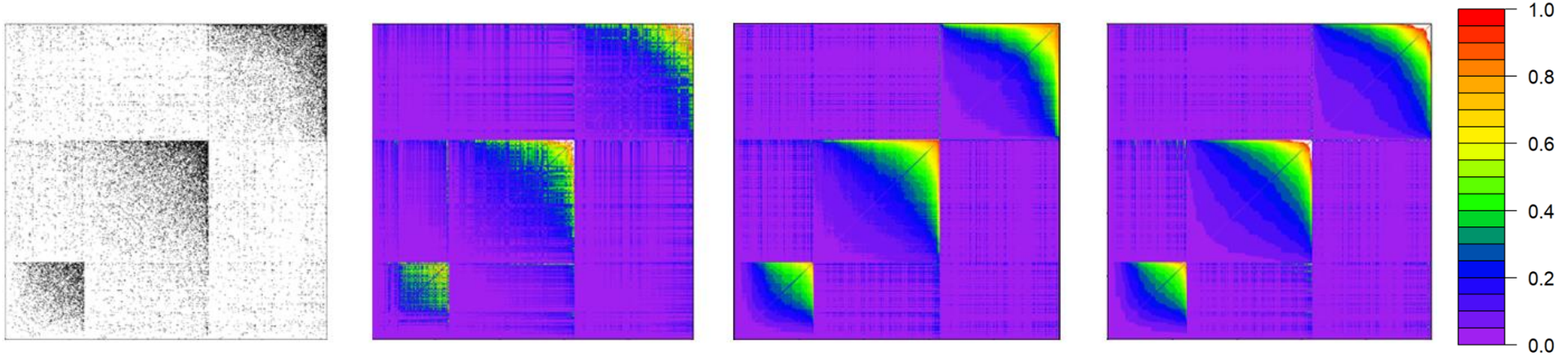
Allowing different contour structures may still be beneficial in sparse networks

So many parameters! What about Occam's razor, you're probably overfitting!

Potentially, but ACRONYM outperforms other models on link prediction in real networks using cross-validation (see paper for details)

Real Data Example: Congress on Twitter

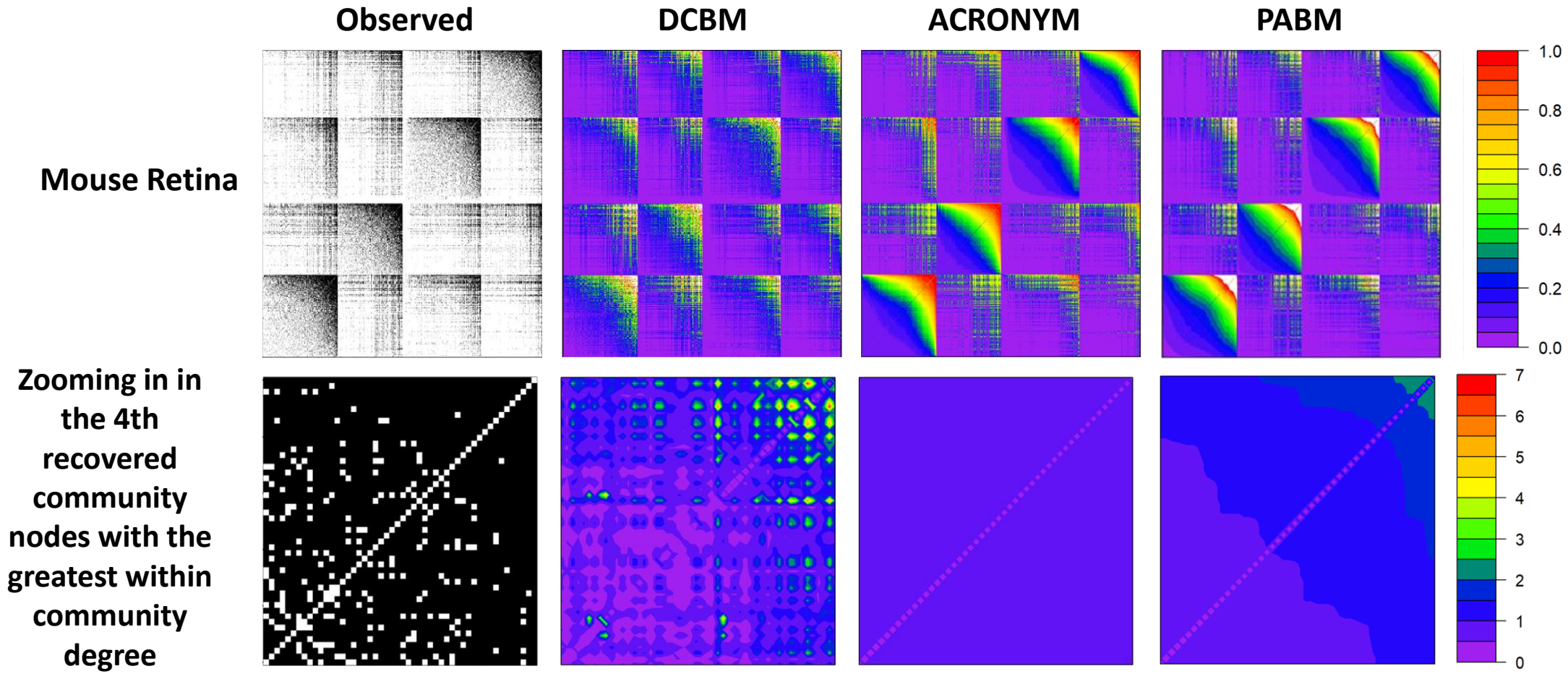
Congress on Twitter



Observed **DCBM** **ACRONYM** **PABM**

- The nodes have been reordered first by recovered community structure (according to ACRONYM) and then by within-community degree
- Notice that there are regions with different densities, but the network as a whole is only about 9% dense
- 91 of 92 senators in are assigned to community 1, which also includes 25 congresspeople.
- Community 2 contains 174 congresspeople and Senator Bernie Sanders
- Community 3 contains 184 members, all congresspeople

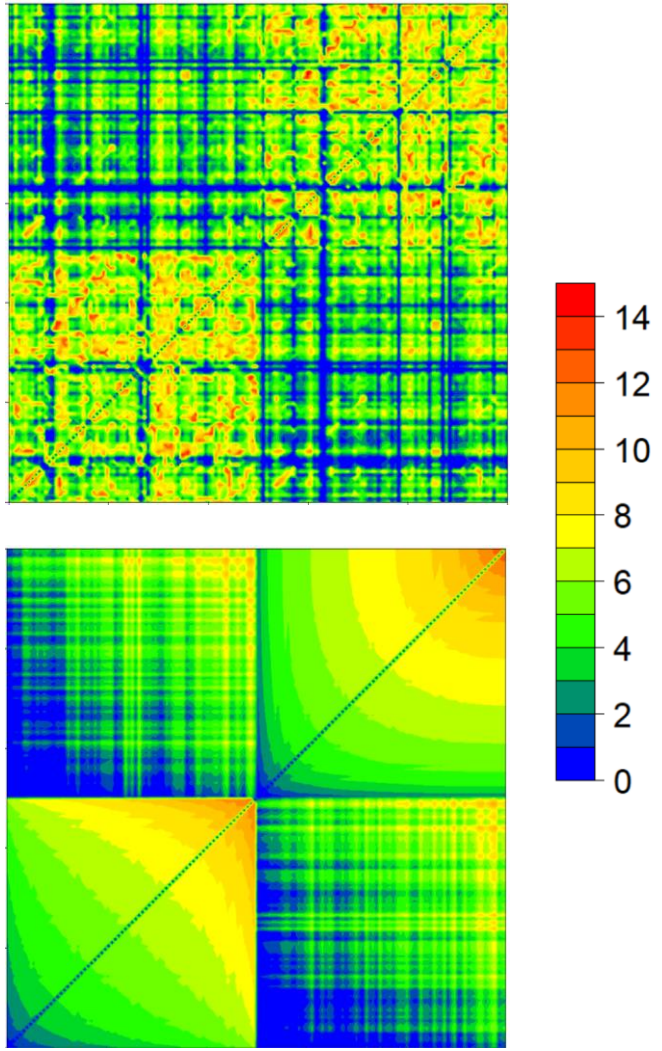
Real Data Example: Mouse Retina



Block Dense Weighted Networks with Augmented Degree Correction

- Previous work describes a model for dense **weighted** networks (like structural brain networks)
- Adapting those ideas here:

1. "Communities" are defined based on an order of preference over nodes, as well as different kinds of flexible connectivity patterns
2. As the edge values are binary here, flexible contours appear only in the probability matrix underpinning the observed network
3. These underlying probability matrices are "dense"



Leinwand, B., & Pipiras, V. (2022). Block dense weighted networks with augmented degree correction. Network Science, 10(3), 301-321. doi:10.1017/nws.2022.23

Positive *H*-function

A function *H*: (0*,* 1) × (0*,* 1) → (0*,* 1) is an *H*-function with *positive association* if:

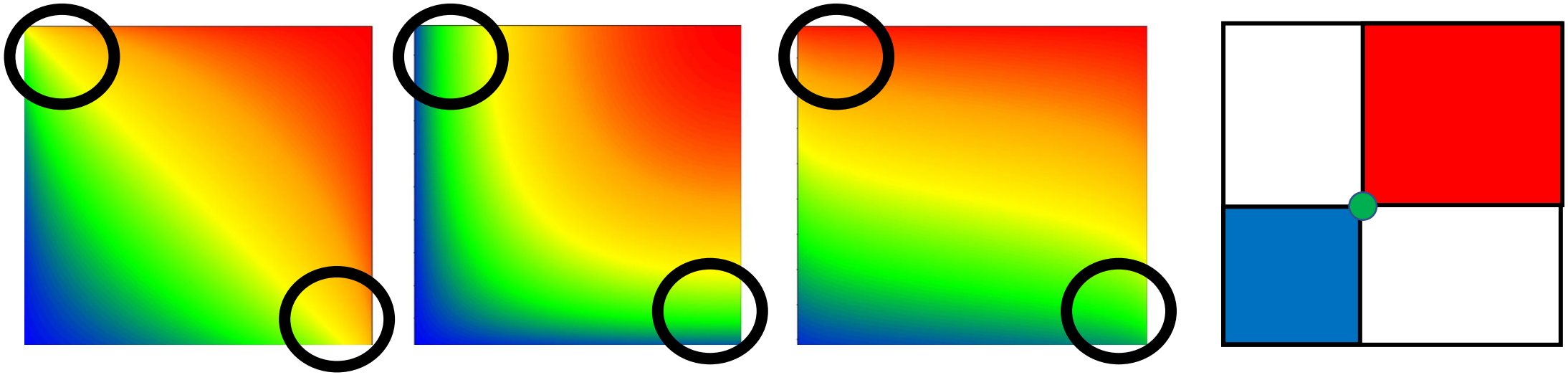
1. *H* is non-decreasing in both arguments;
2. $\iint_{H(x,y)<z} dxdy = z$, for all *z in* (0*,* 1).

Condition 2 is equivalent to saying:

"if the inputs to *H* are uniform RVs*,* so is the output of *H*"

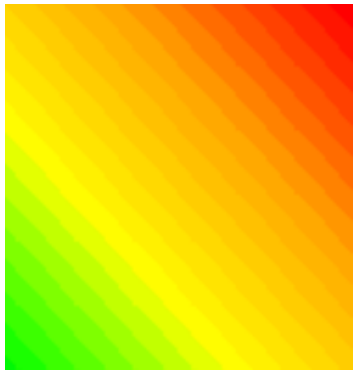
***H*-functions are not networks, but can be used in networks as maps from nodal orderings to edge orderings**

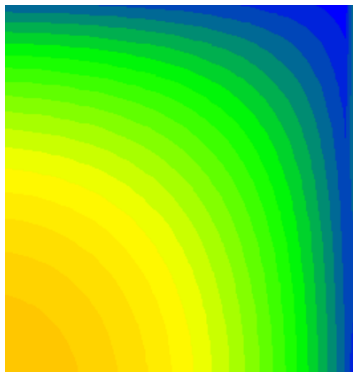
Different *H*-functions, different maps



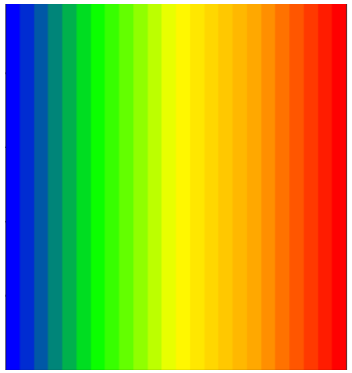
- All plots are positive *H*-functions
- Ordering is easy when both sociability parameters are large (or small)
- Different choices when one is large the other is small
  - $xy \leq H(x,y) \leq 1 - ((1-x)(1-y))$
- Different behavior than copulas, still allows for a variety of nodal effects

Negative and Simpson *H*-functions, projection


Positive association


negative association


Simpson association


Projection (onto x)

- A function H: (0, 1) × (0, 1) → (0, 1) is an *H*-function with *negative association* if $H(1-x, 1-y)$ is an *H*-function with positive association
- A function H: (0, 1) × (0, 1) → (0, 1) is an *H*-function with *Simpson association* if $H(x, 1-y)$ or $H(1-x, y)$ is an *H*-function with positive association
- A function H: (0, 1) × (0, 1) → (0, 1) such that $H(x,y) = x$ or $H(x,y) = y$ is called a *projection*

H-functions and networks

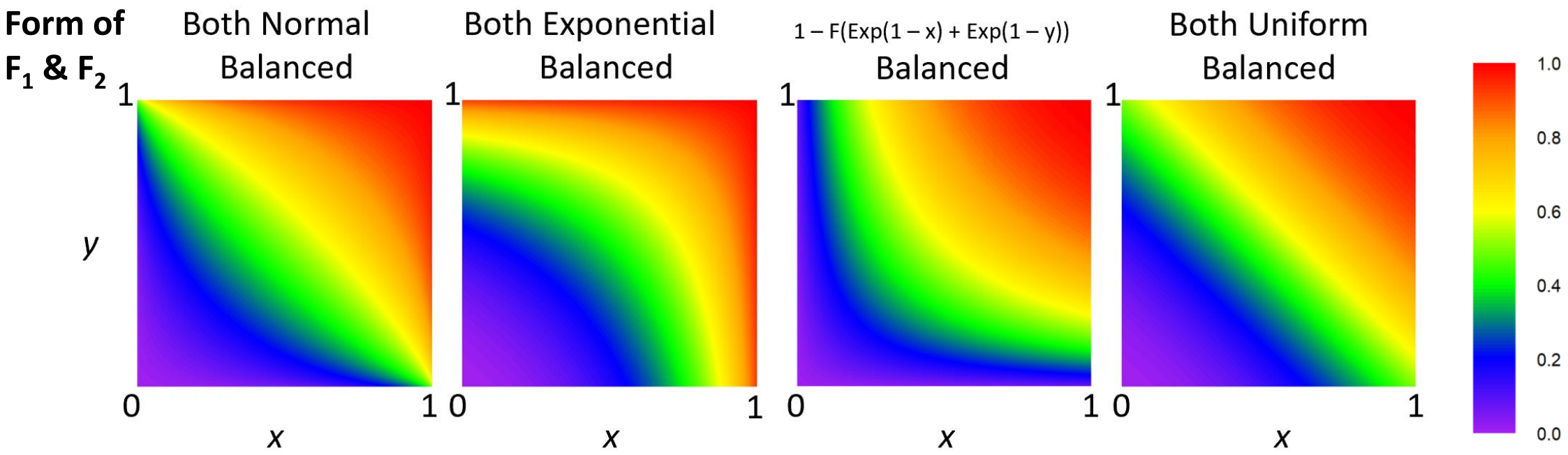
- The idea of an *H*-function in the network setting is to combine a feature from one node with a feature from a second node in a way that determines whether these nodes are linked
- In different parts of the network, different *H*-functions may govern edge probabilities
- The appropriate *H*-function depends on the community memberships of the two nodes
- These *H*-functions may differ in direction of association, functional form, and balance of community impact

Balanced *H*-functions using CDFs (ρ = 1)

Critics rave:
Closed form!
Error free!

Natural way to construct H-functions of two continuous RVs is take the CDF of the sum

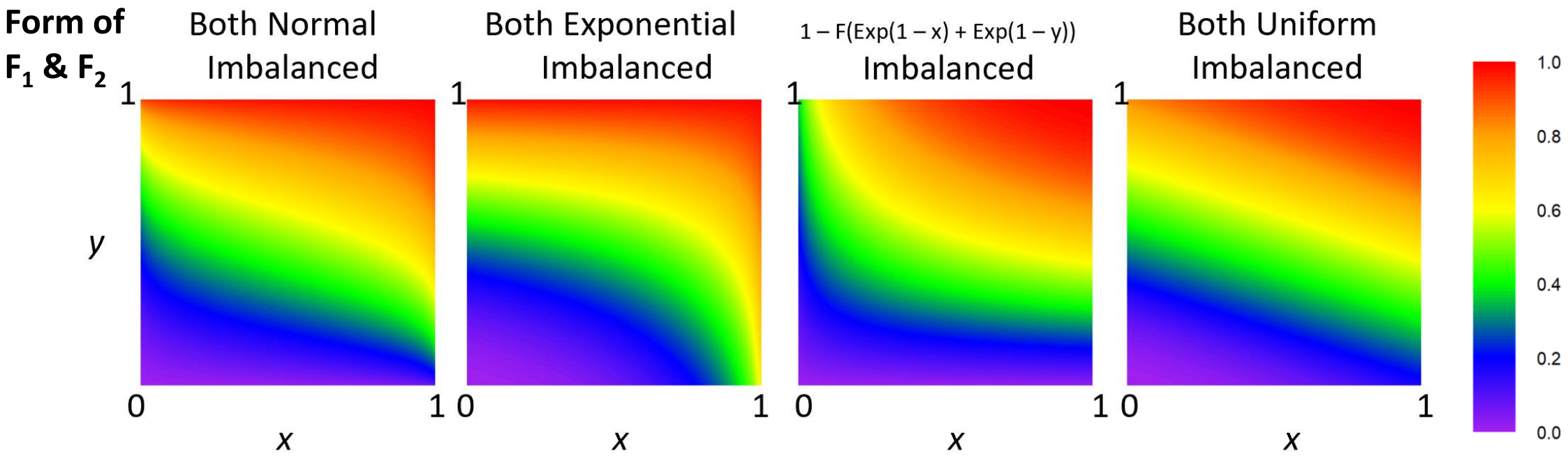
$$H(x,y) = F_{1,2}(F_1^{-1}(x) + F_2^{-1}(y))$$



Imbalanced *H*-functions using CDFs (ρ ≠ 1)

F_1 and F_2 may have different scale/rate parameters, yielding imbalanced *H*-functions

Only the ratio matters, so we can rescale F_1 to have parameter 1, and define $\rho = \frac{\text{scale}_1}{\text{scale}_2}$



Don’t use *H*-functions as a final estimate

- We need to estimate the *H*-function that determines how nodes in a subnetwork interact
- *H*-functions have a range of (0,1)

- **Why not use the value of the estimated *H*-function for a given pair of nodes as our estimate of the edge probability?**

  Because “if the inputs to *H* are uniform RVs, so is the output of *H*”

- Let’s see how this causes problems

H-functions are just orderings

1. *H*-functions have a range of (0,1)
   - There may be no edges in that subnetwork with such a high probability

H-functions are just orderings

1. *H*-functions have a range of (0,1)
   - There may be no edges in that subnetwork with such a high probability
2. The expected value of an arbitrary *H*-function in a subnetwork is ≈.5
   - Most networks are much sparser than this in all subnetworks

H-functions are just orderings

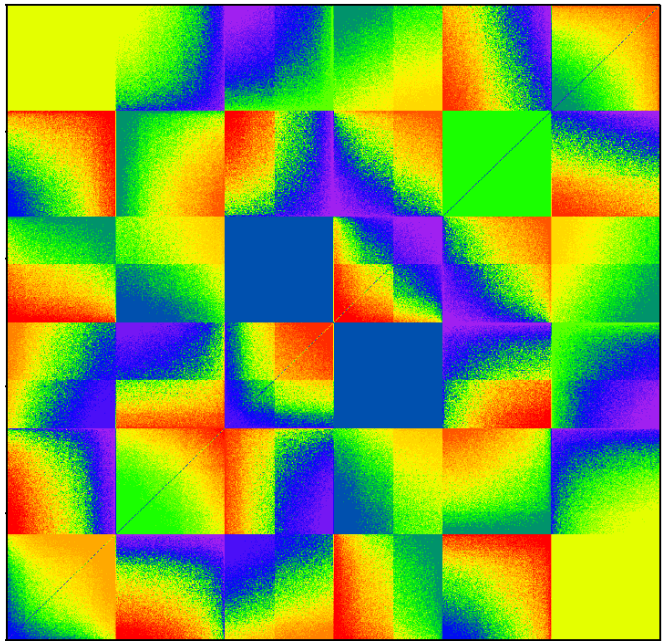
1. *H*-functions have a range of (0,1)
   - There may be no edges in that subnetwork with such a high probability
2. The expected value of an arbitrary H-function in a subnetwork is ≈.5
   - Most networks are much sparser than this in all subnetworks
3. If the subnetwork within community *i* is 25% dense, while the subnetwork within community *j is* only 15% dense, for $u_1, u_2 \in i$ and $v_1, v_2 \in j$ it could be the case that both

$$H(v_1, v_2) > H(u_1, u_2) \quad \text{AND} \quad \mathrm{P}\left(A_{v_1,v_2} = 1\right) < \mathrm{P}\left(A_{u_1,u_2} = 1\right)$$

   - Ordering isn't the same as probability, since density matters

Model needs

- Let *u, v* be vertices within communities *i* and *j* respectively
- Each vertex *u* has "sociability" parameter Ψ_u, uniformly distributed over (0,1)
  - We allow each node to have a different Ψ_u with respect to each community $i \in \{i\}$
- Need to choose the right *H*-function to model the broad pattern of interactions between nodes in *i* and nodes in *j*
  - This includes the form of the contours, but also the relative influence of each community on edge probability (balance)
- Need to incorporate noise into the model
- Need to map the *H*-function to actual probabilities



Modeling networks with *H*-functions

ACRONYM:

$$\boldsymbol{A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_u, \Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)}$$

Where $0 \leq \alpha \leq 1$, $\beta > 0$, $\alpha + \beta \leq 1$, $\varepsilon_{uv} \sim N(0,1)$

- Each subnetwork induced by a pair of communities has their own parameters inducing a unique subnetwork range of probabilities and contour patterns
- Normally distributed error is incorporated in parentheses, Bernoulli error outside

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

Let $\Psi_u = .5, \Psi_v = .8413$

Let $H(x, y)$

$= F_{1,2}(F_1^{-1}(x) + F_2^{-1}(y))$

Where F_1 and F_2 are standard normal CDFs, so $F_{1,2}$ is the CDF of a $N(0, 2)$ RV

Then
$H(x, y) = F_{1,2}(0 + 1)$

= .7602

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

$\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right)$

$= \Phi_1^{-1}(.7602) = 1/\sqrt{2}$

Letting $\sigma = 1$, we have

$\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) =$

$$\frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}} = \frac{1}{2}$$

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

ε_{uv} is an idiosyncratic standard normal RV and

$$\frac{\sigma}{\sqrt{1+\sigma^2}} = \frac{1}{2}$$

If $\varepsilon_{uv} = -.4933$

Then $\frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv} = -.2467$

Then

$$\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}$$

$$= .5 - .2467 = .2533$$

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error, to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

In the parenthesis, we add 2 normal RVs, with variances

$$\frac{1}{1+\sigma^2} \text{ and } \frac{\sigma^2}{1+\sigma^2}$$

So their sum is a standard normal RV

$$\Phi_1(.2533) = .6$$

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error, to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

Let α be .6, then

.6 * .6 = .36

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error, to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

Let β be .05, then the total range of probabilities in this subnetwork is:

(.05, .65),

So expected density of .35

And the probability here is

.36 + .05 = .41

What's happening in ACRONYM

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

1) Combine node sociabilities using an *H*-function, output Uniform RV

2) Convert that Uniform RV into "Normal Space" to avoid boundary effects

3) Add error, to get a standard Normal RV

4) Convert back to a uniform value

5) Multiply this value by α, the width of the probability range in this subnetwork

6) Add β, the minimal edge weight probability in this subnetwork

7) Draw from a Bernoulli distribution where the calculations to this point give the parameter

Drawing from a Bernoulli(.41), we get a value of 1

Therefore $A_{uv} = 1$

There is an edge between u and v

ACRONYM implications

$$A_{uv} = Bernoulli\left(\alpha * \Phi_1\left(\frac{1}{\sqrt{1+\sigma^2}}\Phi_1^{-1}\left(H\left(\Psi_{u,}\Psi_v\right)\right) + \frac{\sigma}{\sqrt{1+\sigma^2}}\varepsilon_{uv}\right) + \beta\right)$$

$$0 \leq \alpha \leq 1 \qquad \beta > 0 \qquad \alpha + \beta \leq 1$$

- If $\alpha = 1$ and $\beta = 0$, probabilities can run from 0 to 1
- If $\alpha = 0$, it's a stochastic block model, and if all β are also equal, an ER graph
  - As α shrinks, the range of probabilities also shrinks
  - As β grows, the minimal probability increases
- All probabilities are constrained to be between 0 and 1
- Contours can look different across the network, as can probability ranges
- Noise can be incorporated differently into each subnetwork

Pros and cons

- In a weighted network, edge weights carry fine-grained information
  - **Pro:** The patterns of edge weights can display an order of preferences
  - **Con:** Subnetwork edge weights have arbitrary distributions and must be estimated

- In an unweighted network, edges are either 0 or 1
  - **Con:** It is hard to discern an order of preferences from only binary values
  - **Pro:** As each edge is a Bernoulli RV that depends on node and community parameters, we can use likelihood-based techniques to estimate parameters

What do we need to estimate?

- Communities – Done separately before everything else
- **α** – width of probability range in subnetwork
- **β** – minimal probability in subnetwork
- **σ** – noise to signal ratio
- Functional form of ***H*** – We only try normal, concave, convex, linear
- **ρ** – ratio of impact of second community to first community
- $[\boldsymbol{\Psi}]$ – node sociability parameters

- We don't really care about the ε values, but as they are latent variables with assumed $N(0,1)$ distributions, estimating ε can help estimate everything else
  - At each step of an iterative algorithm, we estimate the likelihood-maximizing $[\hat{\varepsilon}_{uv}]$, but to avoid early overfitting, we instead draw a random value $\check{\varepsilon}_{uv} \sim N(\hat{\varepsilon}_{uv}, 1)$ for each edge at every step

Parameter estimation overview

- We estimate each subnetwork independently of all others
  - Better for flexibility, but can't borrow strength across the network
  - We estimate a different Ψ for each node with respect to each community
- We run an iterative algorithm for the 4 *H*-function functional forms, and take the model yielding the best likelihood as our final estimate
- At each stage, we determine which sets of parameters should be collectively updated based on whether the likelihood improves
  - When updating multiple parameters, we want broad-based improvements so as to avoid too many updates chasing random fluctuations due to sampled $[\check{\varepsilon}_{uv}]$ values
- As $[\check{\varepsilon}_{uv}]$ are random, other parameters may be updated to suit these values, but may not yield the maximum observed likelihood, so we need to also keep those parameters associated with the best observed likelihood

Iterative algorithm for each subnetwork

1. Initialize all parameters (details omitted here)

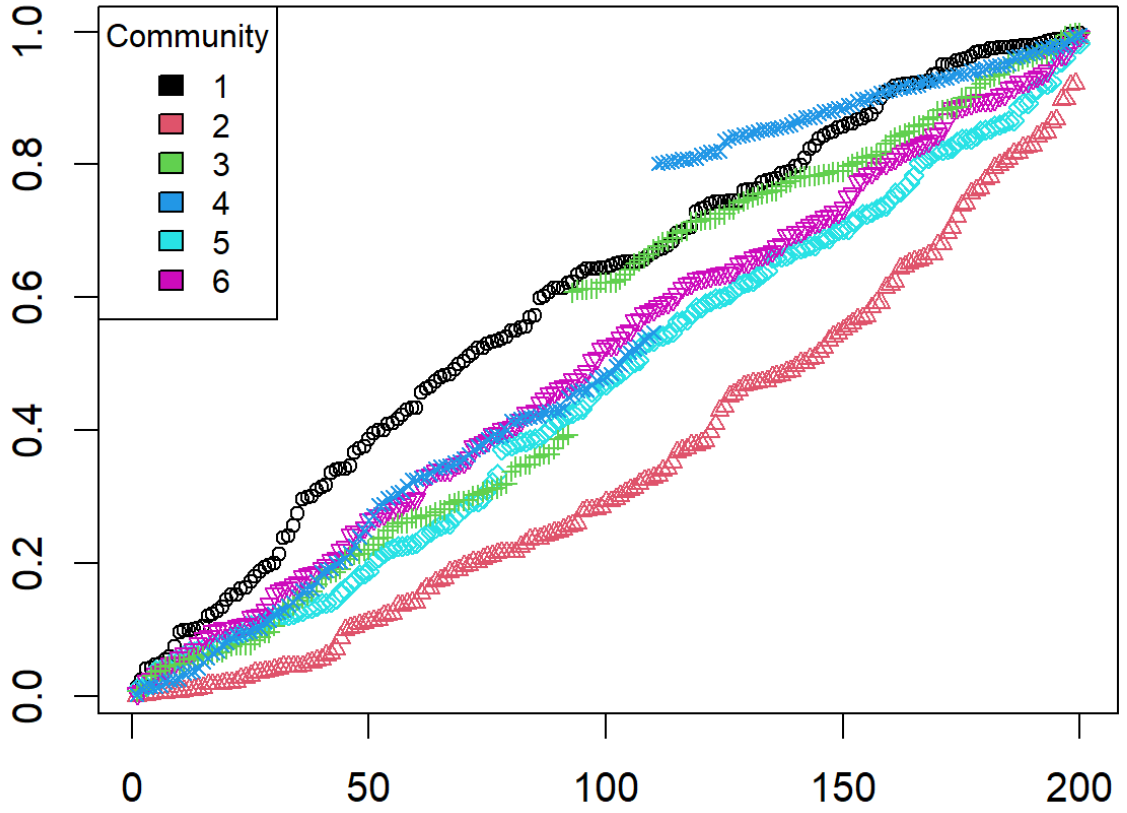
Repeat steps 2 – 4 for the specified number of iterations

2. Calculate the current log-likelihood of current parameters
3. Calculate $[\hat{\varepsilon}_{uv}]$ to maximize likelihood, draw $\check{\varepsilon}_{uv}$ ~ N($\hat{\varepsilon}_{uv}$, 1)
4. Update the following 3 parameter sets in a uniformly random order to maximize log-likelihood given $[\check{\varepsilon}_{uv}]$
   a. $\hat{\alpha}$, $\hat{\beta}$, $\hat{\rho}$, and $\hat{\sigma}$: All updated together
   b. $[\hat{\Psi}_u]$: Either all are updated, or none are
   c. $[\hat{\Psi}_v]$: Either all are updated, or none are
5. Using the best performing parameters, calculate $[\hat{\varepsilon}_{uv}]$, and re-estimate only $\hat{\sigma}$ using the $[\hat{\varepsilon}_{uv}]$ values

Simulation details

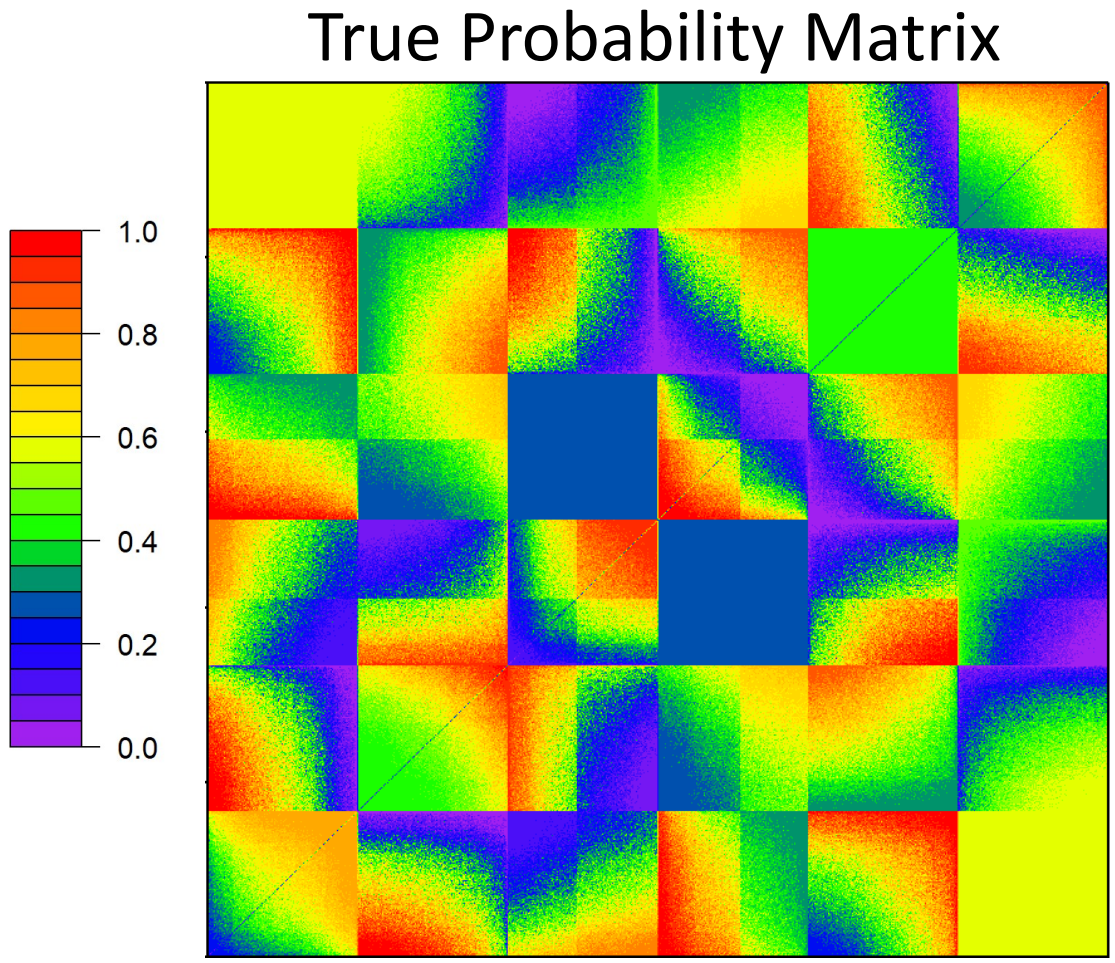
6 communities of 200 nodes each

- First 2 communities have Ψ values drawn from different Beta distributions
- 3rd and 4th communities have "gaps" in Ψ values
- 5th and 6th communities' Ψ values are drawn from a Uniform distribution



Simulation details

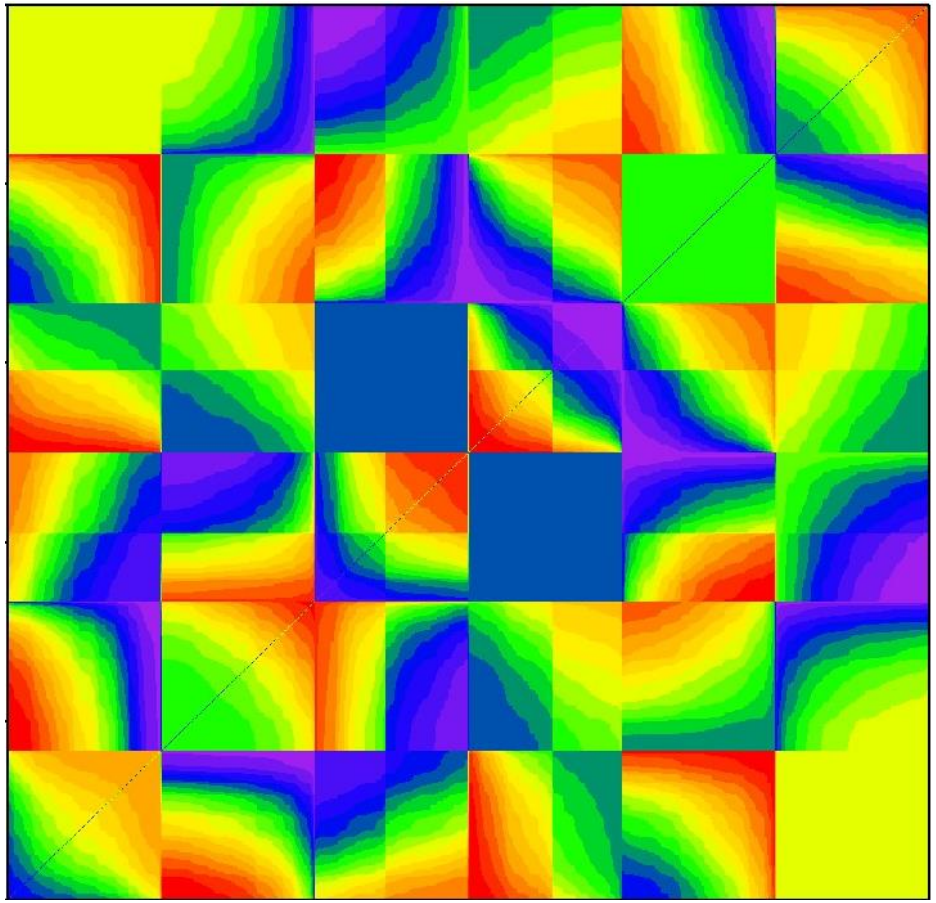
- Pairs of communities have different association directions, functional forms, α, β, ρ parameters (These are visible to the right)
- The σ values for within community edges are set to .3, and set to .4 for between community edges

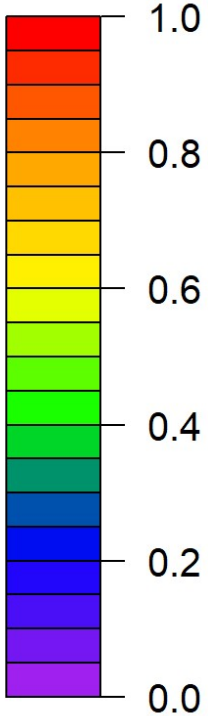


Mind the gaps!

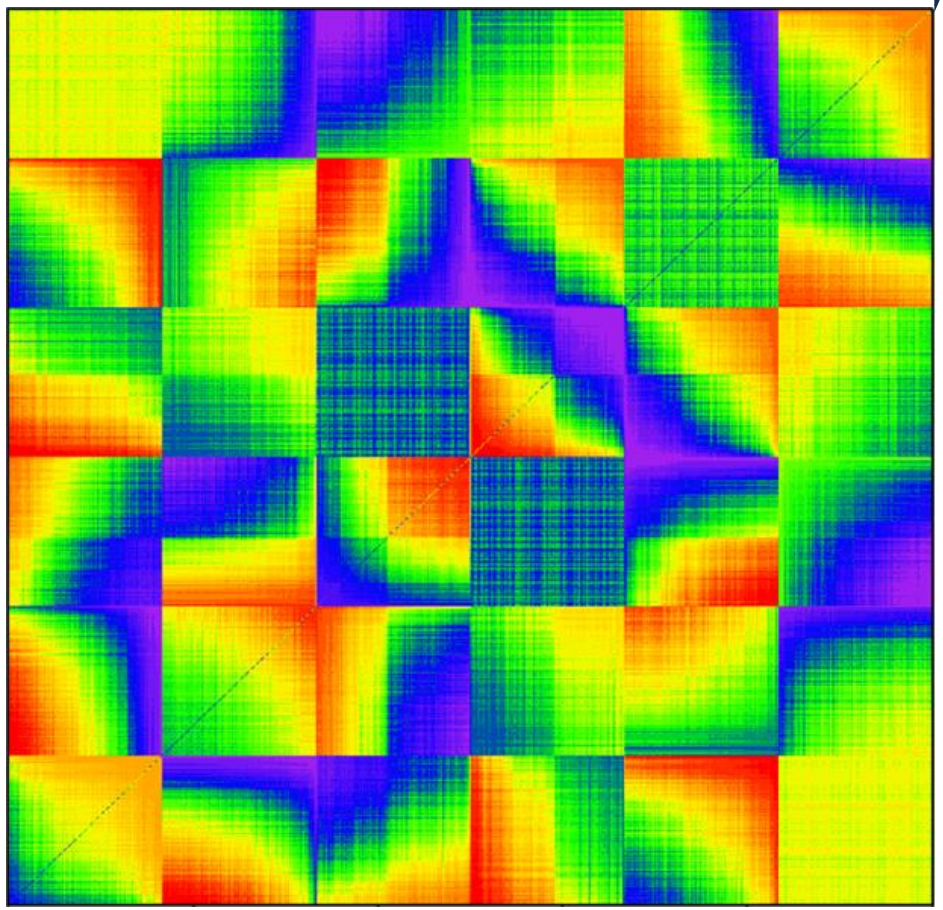
ACRONYM Estimate of simulated network

Noise-free probability matrix

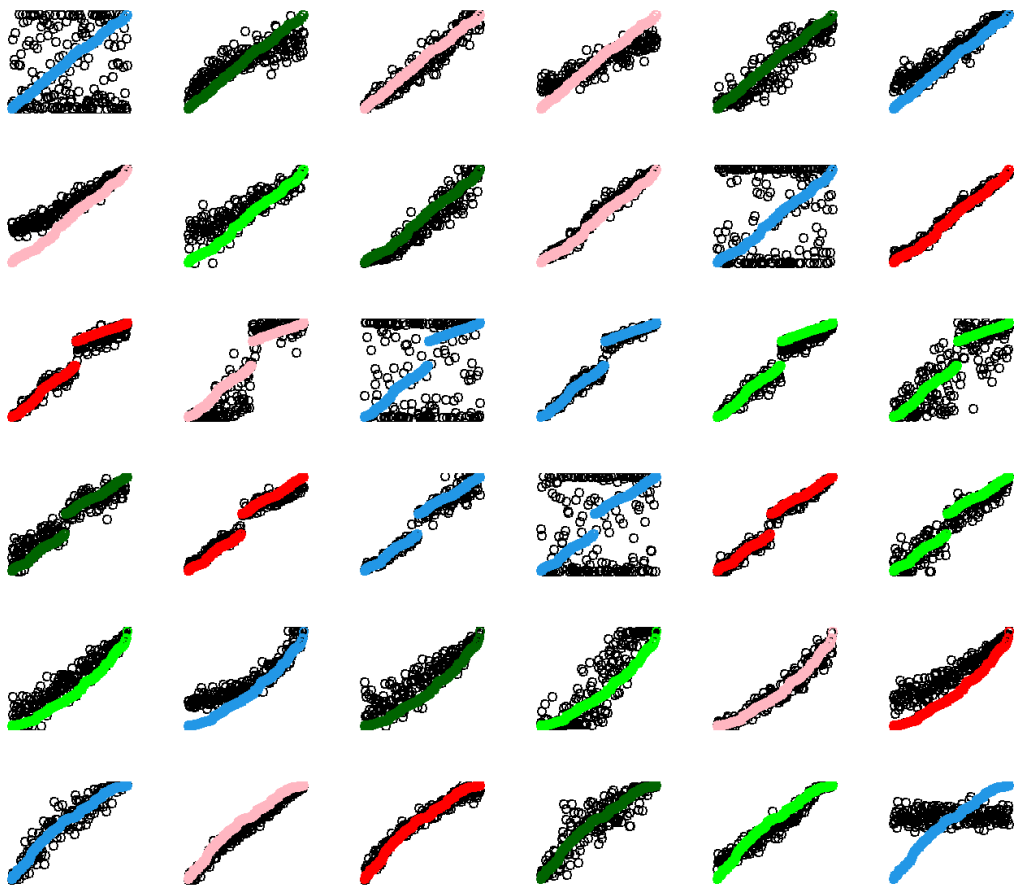




Estimated probability matrix



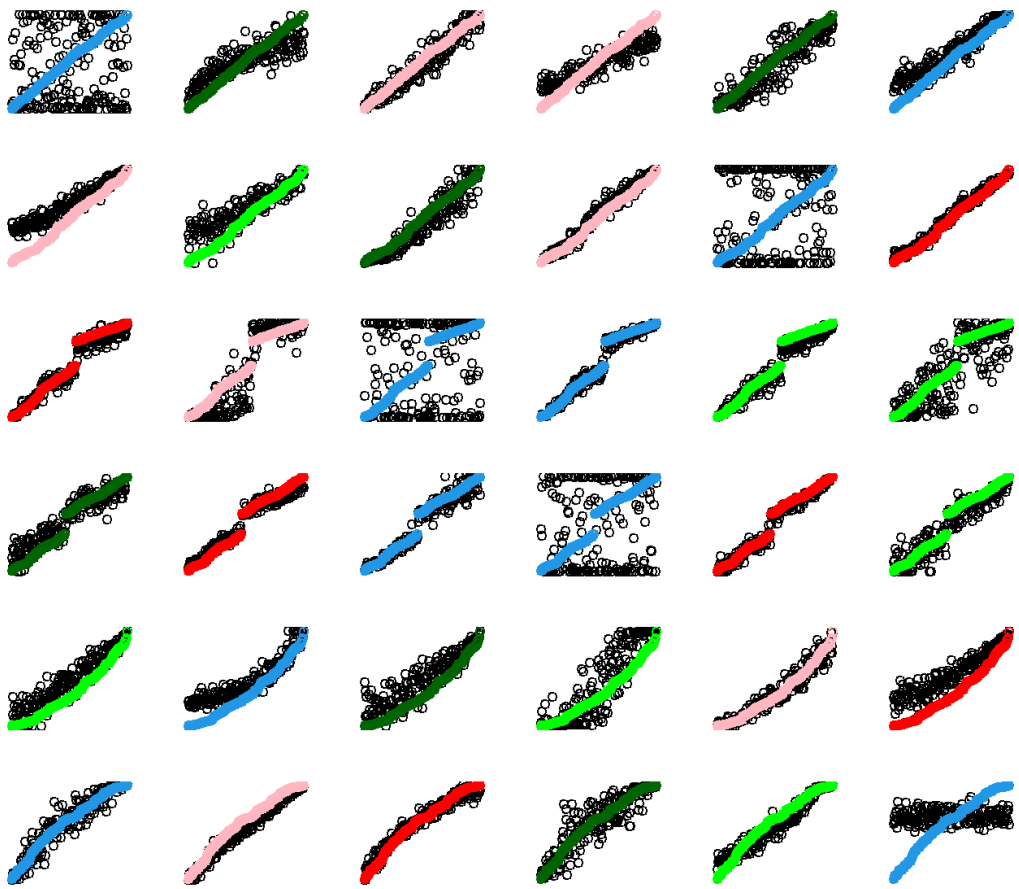
Deeper dive on Ψ estimation

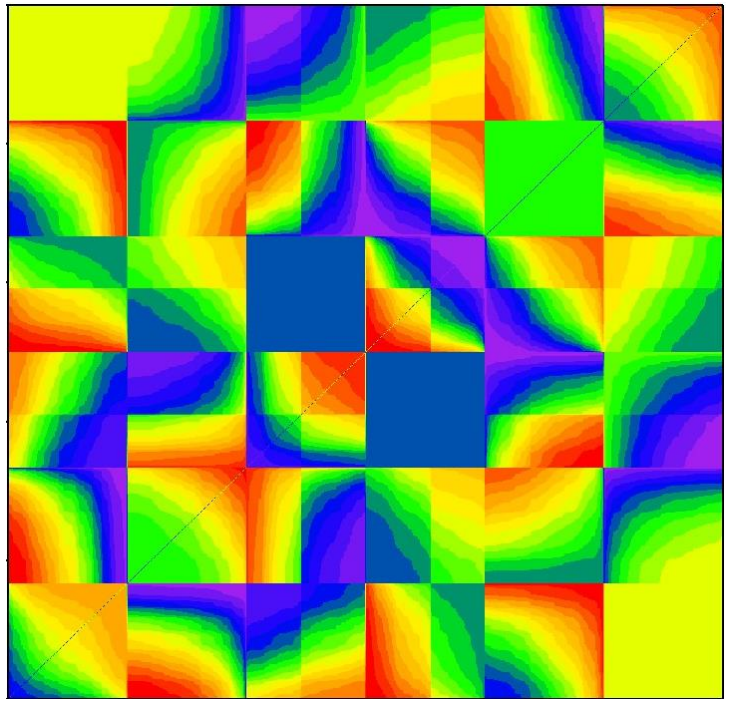


Ψ values for each node are estimated 6 times

- Each row shows estimates for Ψ for that row's community in black, with true values in color
- Bottom row depicts 6 estimates of the Ψ values for community 1, with a concave shape
- **BLUE** indicates ρ = 1
- **PINK:** the community had somewhat less influence than the other community
- **RED:** the community has far less influence
- **LIGHT GREEN:** somewhat more influential than the other community
- **DARK GREEN:** far more influence

Deeper dive on Ψ estimation





Overall, pretty good estimates except in ER subnetworks (solid squares), where the Ψ values don't matter anyway, since $\alpha = 0$

Takeaways:

Easier to estimate Ψ for nodes in communities 5 and 6 (uniform) followed by communities 3 and 4 (gaps)

Communities 1 and 2 (Beta distributions) are the most difficult

May be due to initialization

Future plans

- Error rates for community detection and estimation based on different *H*-functions and parameters
- More general classes of *H*-functions, including compositions of *H*-functions
- Extension to multiple networks
- Joint parameter estimation using Bayesian approach

Thank You!

Questions?

First Note on ε

- Usually, we view ε as idiosyncratic values relevant to only this particular network, so we don't really care about their values
- However, as latent variables with assumed $N(0,1)$ distributions, estimating ε can aid in estimating the meaningful parameters
- At each step of an iterative algorithm, we can estimate the likelihood-maximizing $[\hat{\varepsilon}_{uv}]$, but using this value tends to perform worse by overfitting to early parameter estimates, driving $\hat{\sigma}$ to 0
- Instead, at each step, we draw a random value $\check{\varepsilon}_{uv} \sim N(\hat{\varepsilon}_{uv}, 1)$

Second Note on ε

- As we assume $\varepsilon_{uv} \sim N(0,1)$, treating $\hat{H}_{\hat{\rho}}(\hat{\Psi}_u, \hat{\Psi}_v)$ and $\hat{\sigma}^2$ as fixed but ε as random then:

$$\mathbf{E}_{\varepsilon}\left[\Phi\left(\frac{1}{\sqrt{1+\hat{\sigma}^2}}\Phi^{-1}(\hat{H}_{\hat{\rho}}(\hat{\Psi}_u,\hat{\Psi}_v))+\frac{\hat{\sigma}\varepsilon_{uv}}{\sqrt{1+\hat{\sigma}^2}}\right)\right]=\Phi\left(\frac{\Phi^{-1}(\hat{H}_{\hat{\rho}}(\hat{\Psi}_u,\hat{\Psi}_v))}{\sqrt{1+2\hat{\sigma}^2}}\right)$$

- Therefore, our final estimates are given by

$$\mathbf{E}_{\varepsilon}\left[\hat{P}_{uv}\right]=\hat{\propto}\,\Phi\left(\frac{\Phi^{-1}\left(\hat{H}_{\hat{\rho}}(\hat{\Psi}_u,\hat{\Psi}_v)\right)}{\sqrt{1+\mathbf{2}\hat{\sigma}^2}}\right)+\hat{\beta}$$

Which has extra shrinkage, but during the estimation process, we don't maximize likelihood using this expectation, but instead using the $[\check{\varepsilon}_{uv}]$ values

Likelihood set up

- For an observed network with adjacency matrix $\boldsymbol{A} = [\boldsymbol{A}_{uv}]$, the log-likelihood for an estimated probability matrix $\boldsymbol{P} = [\boldsymbol{P}_{uv}]$ is given by

$$\sum_{\{u<v\}} A_{uv} \log P_{uv} + (1 - A_{uv}) \log (1 - P_{uv})$$

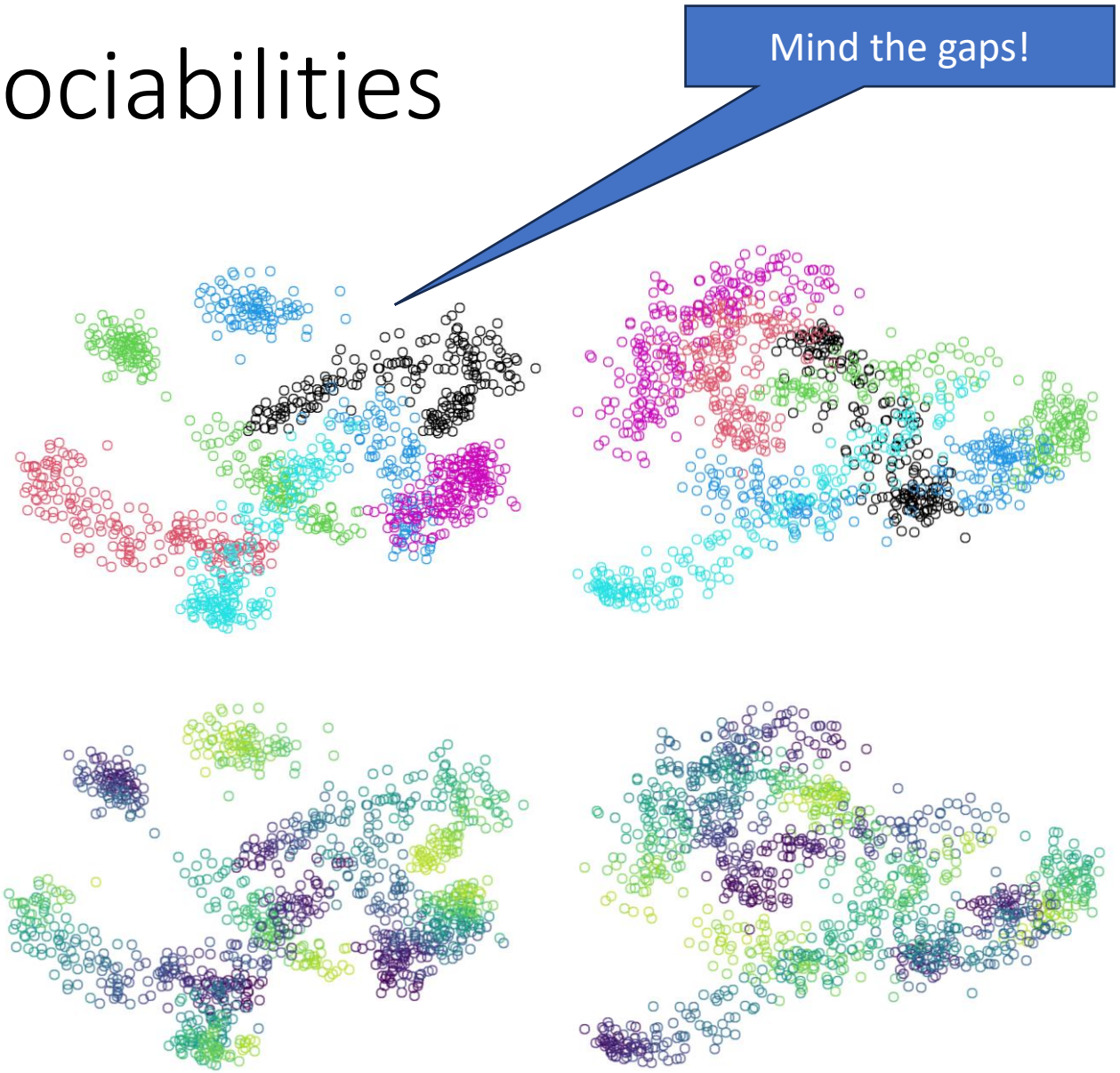
- The network's edges are Bernoulli RVs with probability

$$P_{uv} = \alpha \left(\Phi \left(\frac{1}{\sqrt{1+\sigma^2}} \Phi^{-1} (H_\rho (\Psi_u, \Psi_v)) + \frac{\sigma}{\sqrt{1+\sigma^2}} \varepsilon_{uv} \right) \right) + \beta$$

where ρ dictates the balance of the first argument and the second argument

Communities and sociabilities

- **Top left:** the first and second entries of the $\vec{u}$ vectors for each node, colored by community membership
- **Bottom left:** the first and second entries of the $\vec{u}$ vectors for each node, colored by their Ψ values
- **Top right:** the third and fourth entries of the $\vec{u}$ vectors for each node, colored by community membership.
- **Bottom right:** the third and fourth entries of the $\vec{u}$ vectors for each node, colored by their Ψ values.



Calculating $[\hat{\varepsilon}_{uv}]$

- For every edge where $\boldsymbol{A_{uv}} = 1$, the estimate simplifies to

$$\widehat{\varepsilon}_{uv} = \underset{\varepsilon}{\operatorname{argmin}} \left(\varepsilon - \left[\frac{\widehat{\alpha}\phi\left(\frac{1}{\sqrt{1+\widehat{\sigma}^2}} \Phi^{-1}(\widehat{H}_{\widehat{\rho}}(\widehat{\Psi}_u, \widehat{\Psi}_v)) + \frac{\widehat{\sigma}\varepsilon}{\sqrt{1+\widehat{\sigma}^2}} \right) \frac{\widehat{\sigma}}{\sqrt{1+\widehat{\sigma}^2}}}{\widehat{\alpha}\Phi\left(\frac{1}{\sqrt{1+\widehat{\sigma}^2}} \Phi^{-1}(\widehat{H}_{\widehat{\rho}}(\widehat{\Psi}_u, \widehat{\Psi}_v)) + \frac{\widehat{\sigma}\varepsilon}{\sqrt{1+\widehat{\sigma}}} \right) + \widehat{\beta}} \right] \right)^2$$

- For every edge where $\boldsymbol{A_{uv}} = 0$,

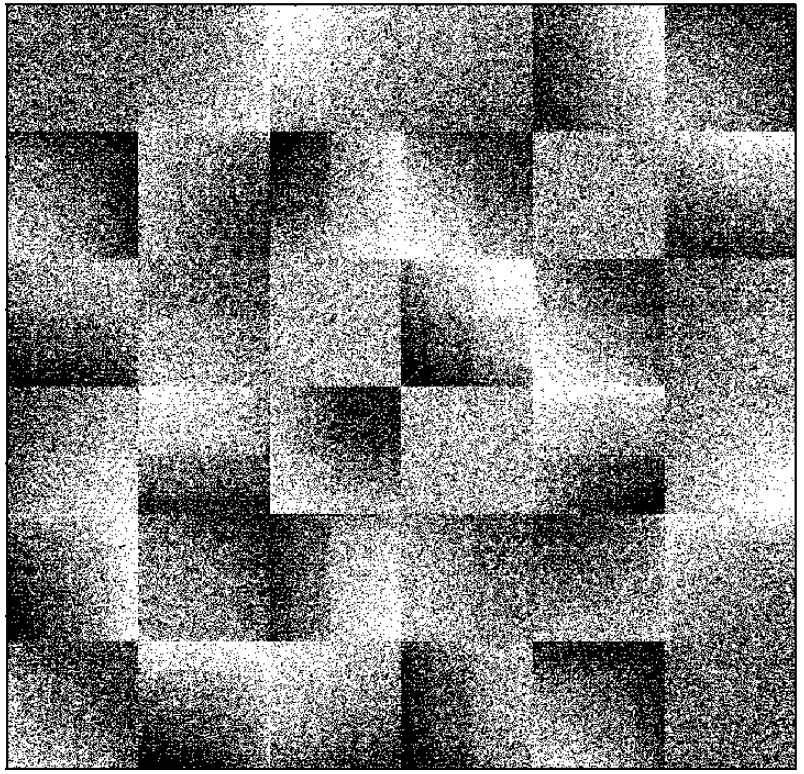
$$\widehat{\varepsilon}_{uv} = \underset{\varepsilon}{\operatorname{argmin}} \left(\varepsilon + \left[\frac{\widehat{\alpha}\phi\left(\frac{1}{\sqrt{1+\widehat{\sigma}^2}} \Phi^{-1}(\widehat{H}_{\widehat{\rho}}(\widehat{\Psi}_u, \widehat{\Psi}_v)) + \frac{\widehat{\sigma}\varepsilon}{\sqrt{1+\widehat{\sigma}^2}} \right) \frac{\widehat{\sigma}}{\sqrt{1+\widehat{\sigma}^2}}}{1 - \widehat{\alpha}\Phi\left(\frac{1}{\sqrt{1+\widehat{\sigma}^2}} \Phi^{-1}(\widehat{H}_{\widehat{\rho}}(\widehat{\Psi}_u, \widehat{\Psi}_v)) + \frac{\widehat{\sigma}\varepsilon}{\sqrt{1+\widehat{\sigma}^2}} \right) - \widehat{\beta}} \right] \right)^2$$

ε appears in the numerators and denominators

Community detection challenges

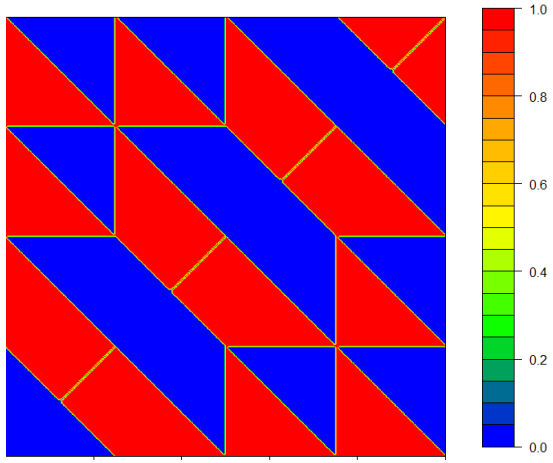
- Communities are defined by a similar ordering of preferences over other nodes, not homophily
- The network may contain positive and negative associations
- Existing methods based on e.g. modularity don’t appear to work well here
- The first eigenvector is often correlated with degree, not community

- Fortunately, the structure of **ACRONYM** guides us toward an approach based on normalizing each row of ***A***



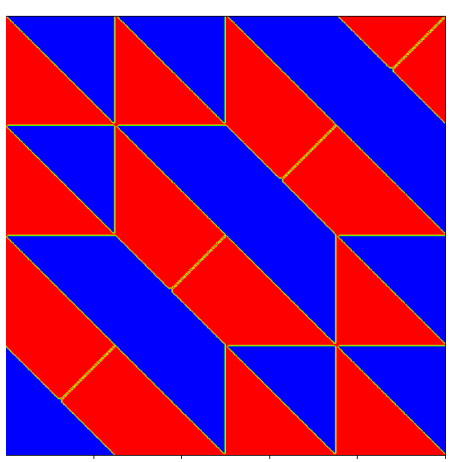
Community detection algorithm

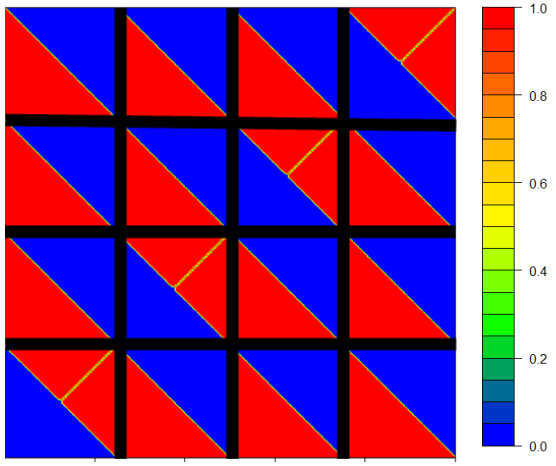
1. Row-normalize the ***A*** matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric ***N*** matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of ***N,*** and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos ($\vec{u}_v$, $\vec{u}_w$)
5. Cluster the nodes based on these cosine distances using hierarchical clustering

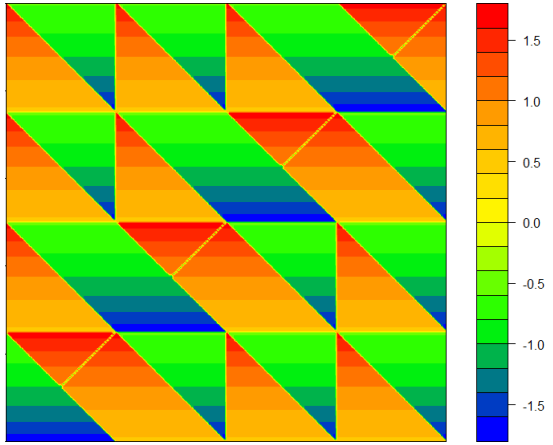


Community detection algorithm

1. Row-normalize the ***A*** matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric ***N*** matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of ***N,*** and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos ($\vec{u}_v$, $\vec{u}_w$)
5. Cluster the nodes based on these cosine distances using hierarchical clustering





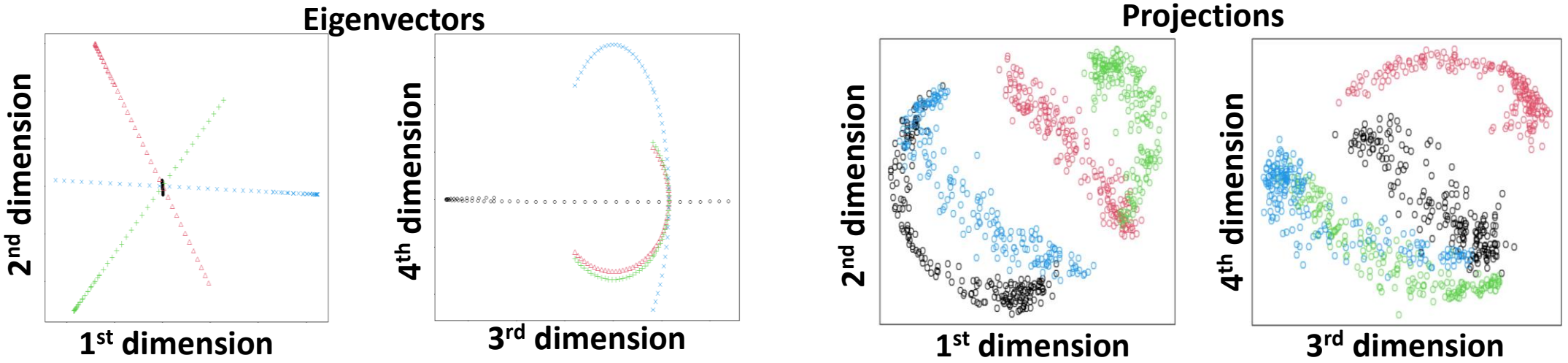


Community detection algorithm

1. Row-normalize the $\boldsymbol{A}$ matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric $\boldsymbol{N}$ matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of $\boldsymbol{N}$, and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos ($\vec{u}_v$, $\vec{u}_w$)
5. Cluster the nodes based on these cosine distances using hierarchical clustering

Community detection algorithm

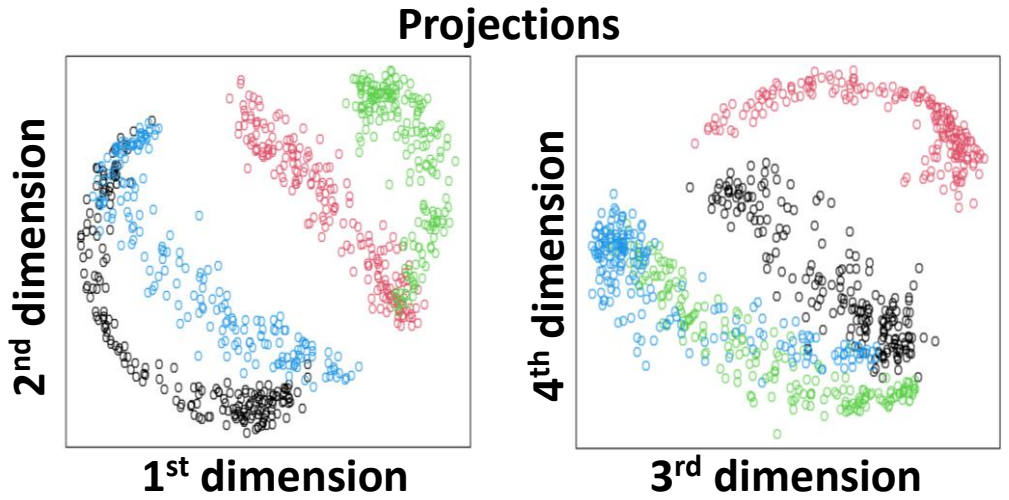
1. Row-normalize the ***A*** matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric ***N*** matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of ***N***, and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos $(\vec{u}_v, \vec{u}_w)$
5. Cluster the nodes based on these cosine distances using hierarchical clustering



Community detection algorithm

1. Row-normalize the ***A*** matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric ***N*** matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of ***N,*** and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos ($\vec{u}_v$, $\vec{u}_w$)
5. Cluster the nodes based on these cosine distances using hierarchical clustering

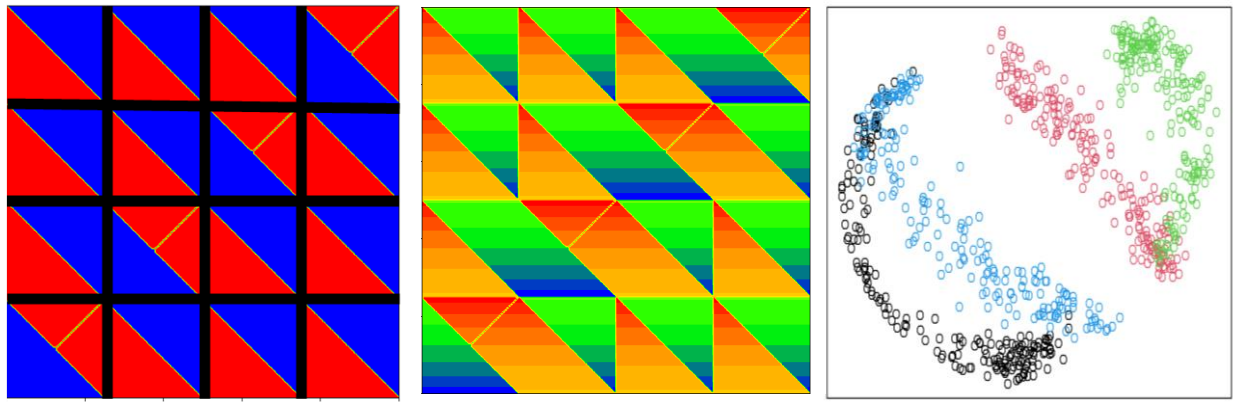
Cosine distance is not a true distance, but takes on a value between 0 and 2, with vectors pointing in the same direction getting a value of 0, and nodes pointing in opposite directions getting a value of 2



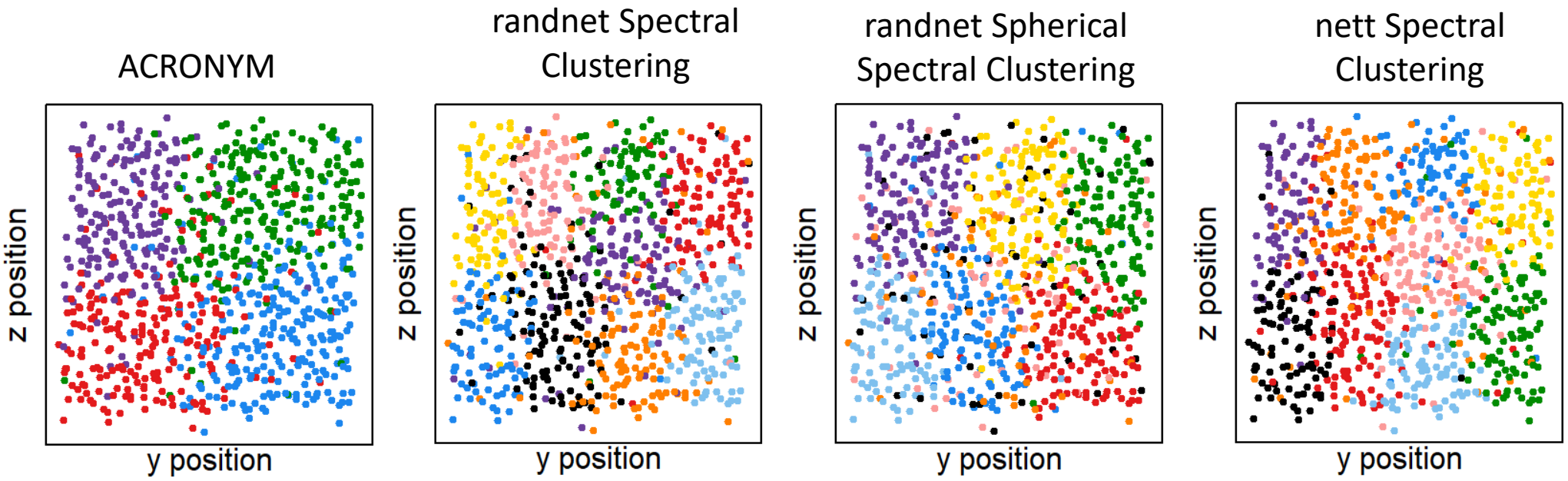
Community detection algorithm

1. Row-normalize the ***A*** matrix by taking $\boldsymbol{N_{uv}} = (\boldsymbol{A_{u\bullet}} - \overline{\boldsymbol{A_{u\bullet}}})/\mathrm{sd}(\boldsymbol{A_{u\bullet}})$
2. Plot the absolute values of the real parts of the eigenvalues of the non-symmetric ***N*** matrix and look for the elbow. Call the value of this elbow $\tilde{d}$
3. Take the $n \times \tilde{d}$ matrix made up of the first $\tilde{d}$ eigenvectors of ***N,*** and project each row of this matrix onto the surface of a hypersphere. Call the projection of the v^{th} row $\vec{u}_v$
4. Compute the cosine distances between rows, 1- cos ($\vec{u}_v$, $\vec{u}_w$)
5. Cluster the nodes based on these cosine distances using hierarchical clustering

In different contexts, we have seen different linkage criteria succeed. The network on the right would seem to do well using single-linkage clustering



Mouse retina community detection



ACRONYM finds different and fewer communities than the other approaches, but the results still appear to follow the physical structure

Probability model	**Communities**	**Edges with valid estimated probabilities**	**$-\ell(A\|\cdot)$ on edges with valid estimated probabilities**	**MSE from P^* on edges with valid estimated probabilities**
P^*	6 (true)	719,400	420,582	0
P	6 (true)	719,400	410,569	.00553
ACRONYM $\tilde{P}$	6 (true)	719,400	416,945	.00222
DCBM	6 (true)	719,400	481,696	.04088
PABM	6 (true)	707,183	414,249	.00381
PABM (Truncated at .999)	6 (true)	719,400	423,321	.00386
DCBM with spectral clustering with regularization (nett)	5	718,518	467,841	.02996
DCBM with regularized spectral clustering (randnet)	5	718,960	468,640	.03027
DCBM with regularized spherical spectral clustering (randnet)	5	719,399	469,444	.03162
DCBM with spectral clustering with regularization (nett)	16	719,277	433,656	.00882
DCBM with regularized spectral clustering (randnet)	16	719,285	434,501	.00933
DCBM with regularized spherical spectral clustering (randnet)	16	719,302	435,680	.01016

Table 1. Table describing different models fit to the simulated dataset. The first 2 rows are generated along with the observed network. Our estimate $\tilde{P}$ recovers the true communities. For the DCBM and PABM models in the next 2 rows, the true communities are assumed known, while the remaining rows use alternative methods of community detection. The third column counts the number of potential edges each model estimates to have probabilities which lie in $[0, 1]$, out of a total of 719,400 potential edges. The fourth column measures the negative log-likelihood only where the probability estimates are valid, that is, on each model's valid potential edges. The fifth column measures the average entrywise squared distance from P^* of these models, again only where the probability estimates are valid.

Probability model	**Communities**	**Potential edges with valid probabilities**	**Negative log-likelihood on potential edges with valid probabilities only**
ACRONYM $\tilde{P}$	3	112,575	24,159
ACRONYM 10 Fold CV $\tilde{P}$	NA	112,575	24,097
DCBM	3	112,328	25,925
PABM	3	112,257	24,061
PABM Truncated at .999	3	112,575	24,344
DCBM with `nett` spectral clustering (with regularization)	7	112,135	25,044
DCBM with `randnet` regularized spectral clustering	7	112,136	25,197
DCBM with `randnet` regularized spherical spectral clustering	7	112,209	25,074

Table 2. Table displaying the performance of different models fit to the Congressional Twitter dataset.

Probability model	**Communities**	**Potential edges with valid probabilities**	**Negative log-likelihood on potential edges with valid probabilities only**
ACRONYM $\tilde{P}$	4	578,350	142,346
ACRONYM 3 Fold CV $\tilde{P}$	5, 5, 4	578,350	148,237
DCBM	4	569,519	169,878
PABM	4	567,673	147,535
PABM truncated	4	578,350	154,991
`nett` spectral clustering (with regularization)	9	568,453	160,350
`randnet` regularized spectral clustering	9	568,561	159,783
`randnet` regularized spherical spectral clustering	9	569,506	157,694

Table 3. Table displaying the performance of different models fit to the Mouse Retina dataset.