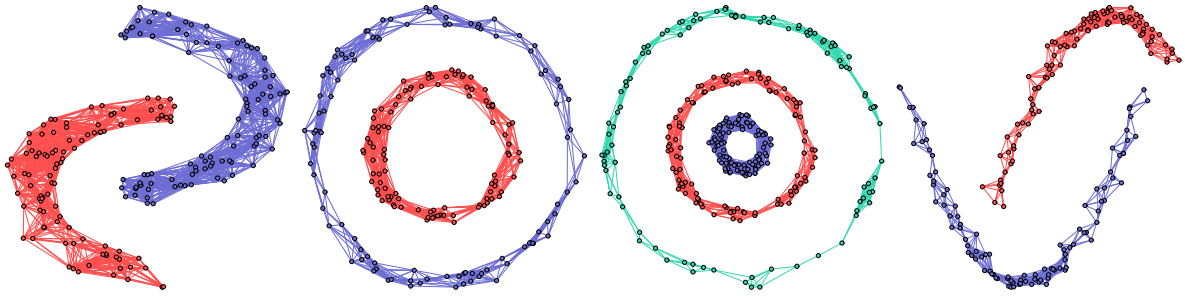
Benchmarks for spectralGraphTopology

Vinícius

December 20, 2018

1 Clustering: warm-up

To warm-up, we consider the classical problem of clustering nodes distributed in $\mathbb{R}^2$. To do that, we generate 100 nodes per cluster distributed according to structures colloquially known as *two-moons*, *two-circles*, *three-circles*, and *worms*. Figure 1 depicts the results of learning the clusters structures using the proposed algorithm denoted as *Spectral Graph Learning* (SGL). As we can note, SGL is able to perfectly distinguish the cluster membership of all the datapoints for all datasets. Additionally, Figure 2 depicts the convergence trend of the terms in the objective function for the *two-moons* structure.



(a) *two-moons.* (b) *two circles.* (c) *three circles.* (d) *worms.*

Figure 1: Clustering results learned by the SGL algorithm for synthethic datasets.

2 Grid graphs

A comprehensive performance evaluation of our spectral graph topology algorithm was performed considering grid graph models with 64 nodes denoted as $\mathcal{G}_{\text{grid}}^{(64)}$. We compare the performance of the SGL algorithm against state-of-the-art algorithms, namely CGL, CGL($\mathbf{A}$). Recall that CGL($\mathbf{A}$) stands for the CGL algorithm equipped with knowledge of the connectivity matrix $\mathbf{A}$, which gives the exact information of which nodes are connected to which ones.

The experimental setup is as follows. The edges of the graph model are sampled from $\mathsf{Uniform}(0.1, 3)$. The Laplacian matrix estimation is carried out on the basis of T samples distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{L}_{\text{grid}}^{\dagger})$. We repeat that experiment 20 times for every value of T and we average out the relative errors and F-scores[1].

Some hyperparameter tunning is required. For the SGL algorithm, we fix $\beta = 10$ for the values of T such that $T/N > 5$. Otherwise, we start with $\beta = 10^{-2}$, and we exponentially increase it up to $\beta = 4$. Additionally, we fix $\alpha = 0$.

For the CGL and CGL($\mathbf{A}$) algorithms, we follow the recipe by Eglimez, i.e., we choose α from $\{0\} \cup \left\{0.75^r \left(s_{\max}\sqrt{\log(N)/T}\right) | r = 1, 2, ..., 14\right\}$, such that the relative error between the estimated Laplacian and the ground truth is minimized, where $s_{\max}$ is the maximum off-diagonal element of the covariance matrix.

[1]For the computation of the F-score, we ignore edge weight values which are less than 10^{-1}.

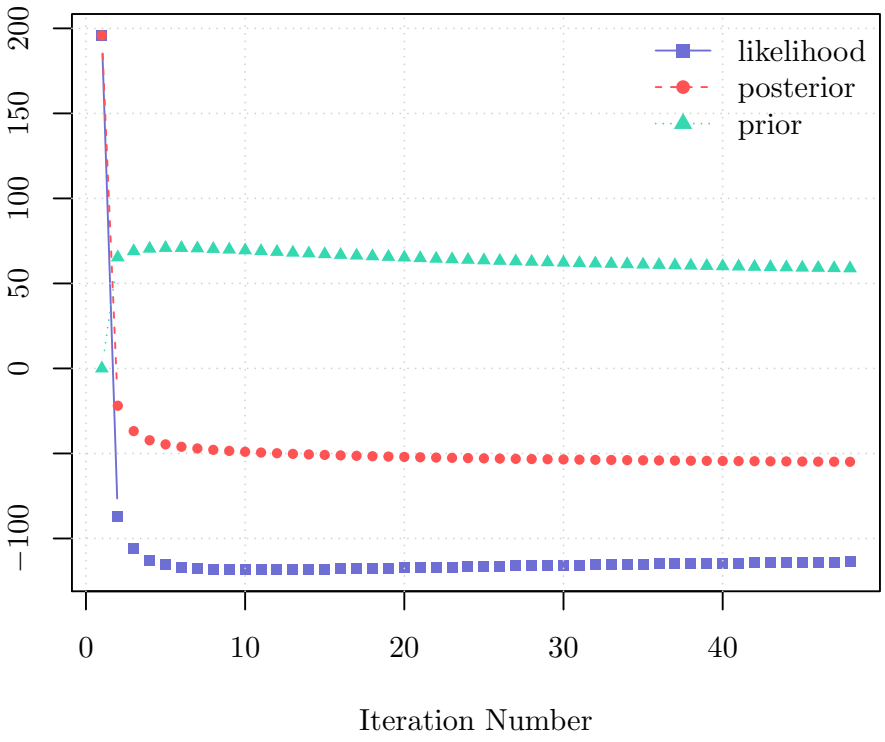


Figure 2: Convergence trend of the SGL algorithm for the *twomoon* dataset.

Figure 3 compares the performance of the algorithms for different sample size regimes for the grid graph model. As it can be noted, the SGL algorithm outperforms the CGL in both relative error and F-score senses. More precisely, for the case when the sample size is equal to the number of nodes, the difference in relative error and in F score are around 12% and 23%, respectively. As expected, with the additional prior knowledge of the connectivity matrix $\mathbf{A}$, the CGL($\mathbf{A}$) algorithm basically attains a perfect F-score for $T/N \geq 10$. However, the connectivity matrix is not always available in practical problems, especially in clustering tasks where the goal is precisely to understand the connectivity membership among the nodes. Nonetheless, the proposed SGL algorithm presents a comparable performance against CGL($\mathbf{A}$). For instance, at $T/N = 5$, the difference in relative error is only around 2.5%, and it keeps decreasing even further until virtually equal performance after $T/N = 100$, where the difference in relative error is around 1.2%. Additionally, we noted that the SGL algorithm requires far less tunning than CGL. At last, we add the relative error and the F-score for the Moore-Penrose inverse of the sample covariance matrix (ISCM) for completeness.

Figure 4 pictorially compares the graph structures learned by SGL and CGL when $T/N = 100$.

3 Erdos-Renyi graphs

The experimentation on the Erdos-Renyi model is carried out in a similar fashion as for the grid graph. We consider an Erdo-Renyi graph $\mathcal{G}_{\mathsf{ER}}^{(64,p)}$ with $p = 0.1$, where p is the probability that a particular node is connected to any other node.

On what concerns hyperparameter tunning, we fix $\beta = 1.29$ and $\alpha = 1.3 \cdot 10^{-2}$ for the values of T such that $T/N > 1$, otherwise we fix $\alpha = 0$ and start with $\beta = 10^{-2}$ and we exponentially increase it up to $\beta = 1$.

The hyperparameter tuning for the CGL and CGL($\mathbf{A}$) algorithms is done as described for the grid model.

Figure 5 reveals that the algorithms SGL and CGL obtain a similar performance across the sample size regimes. Unsurprisingly, the algorithm CGL($\mathbf{A}$) attains nearly perferct F-score for $T/N > 10$.

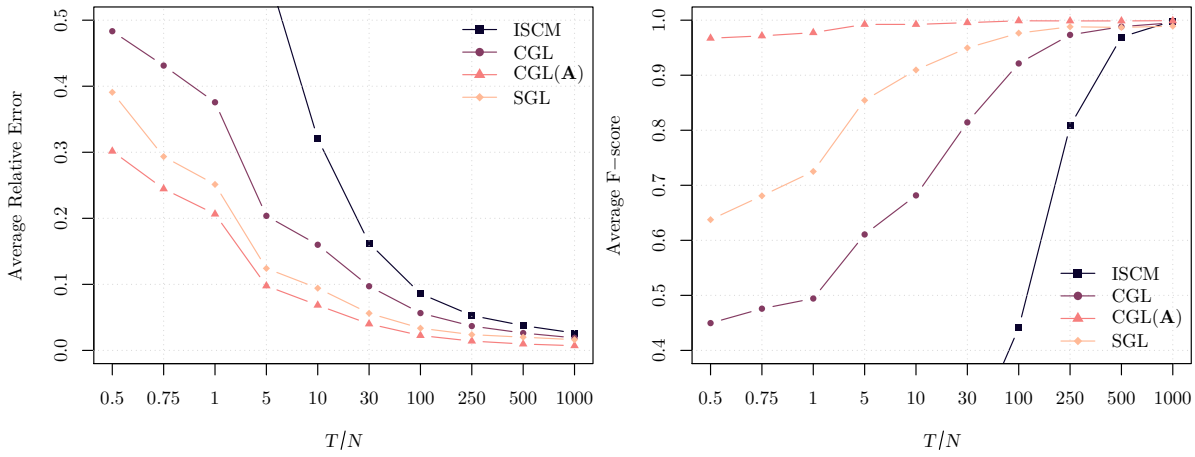
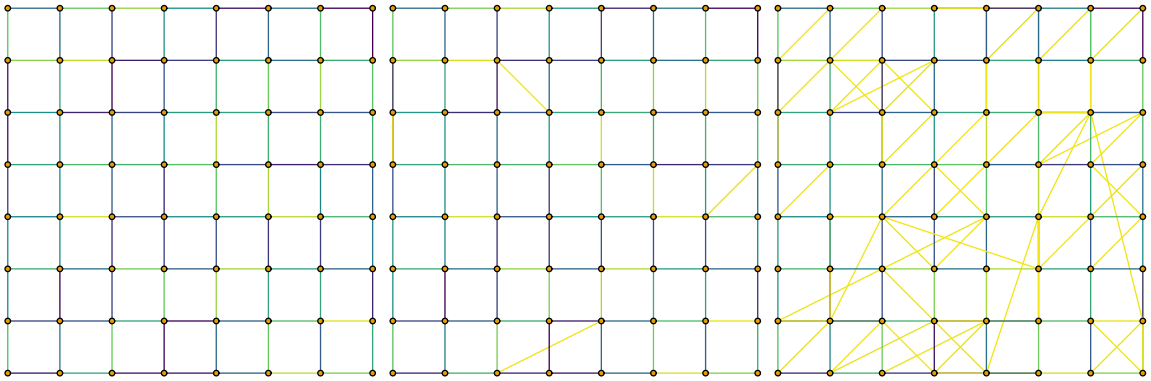


Figure 3: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{grid}}^{(64)}$.



(a) Instance of a grid graph. (b) SGL $(\mathsf{RE}, \mathsf{FS}) = (0.028, 0.991)$. (c) CGL. $(\mathsf{RE}, \mathsf{FS}) = (0.067, 0.953)$.

Figure 4: Sample results of learning $\mathcal{G}_{\mathsf{grid}}^{(64)}$ for $T/N = 100$. Edges smaller than 0.05 were removed. For SGL, we fix $\beta = 10$, whereas for CGL we perform a grid search in α to find α that maximizes the F-score. We found $\alpha = 3.6 \cdot 10^{-3}$.

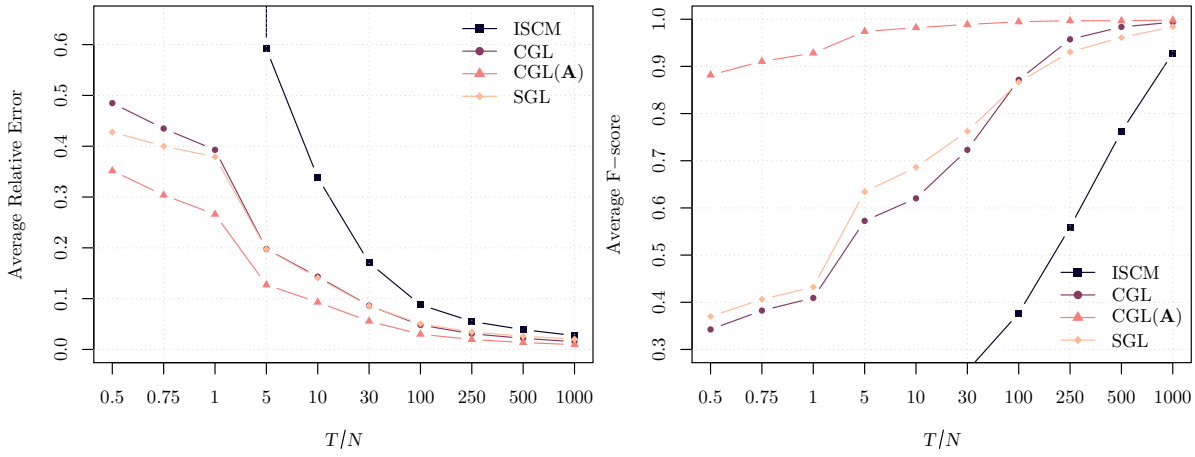


Figure 5: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{ER}}^{(64,0.1)}$.

4 Modular graphs

We consider a modular graph $\mathcal{G}_{\mathsf{M}}^{(64,p_1,p_2)}$ with four modules, $p_1 = 0.01$, and $p_2 = 0.5$, where p_1 is the probability that a particular node of a given module is connected to any other node of a different module, and p_2 is probability that a particular node is connected to any other node of the same module.

On what concerns hyperparameter tunning, we fix $\beta = 4$ for all values of T/N.

Figure 6 shows the performance of the proposed algorithms and the benchmarks.

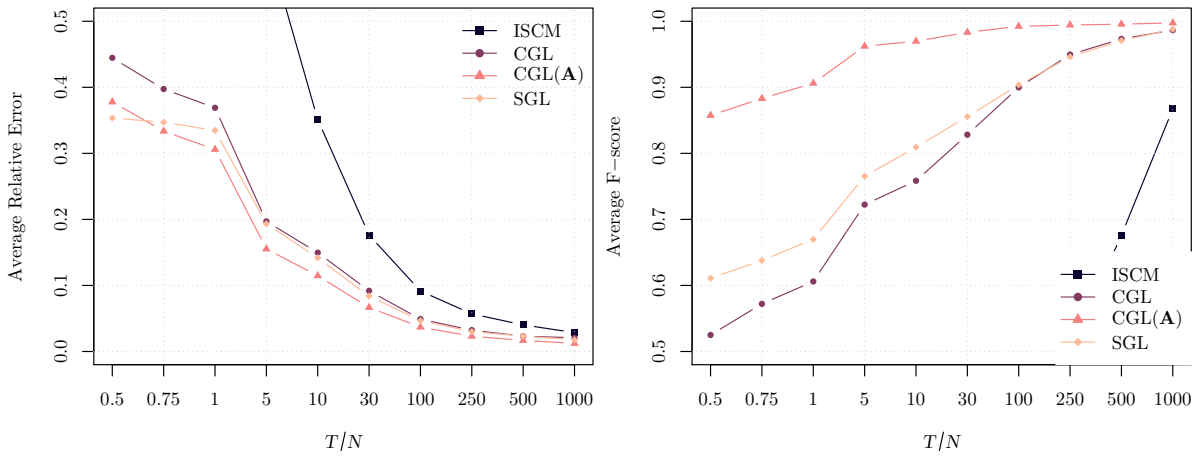


Figure 6: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{M}}^{(64,0.01,0.5)}$ with four modules.

5 Animals dataset

The animals dataset consists of binary values which are the answers to questions such as "is warm-blooded?", "has lungs?", etc. There are in total 102 such questions, which make up the features, and 33 animal categories, which are the nodes. Figure 7 shows the results of estimating the graph Laplacian of the animals dataset using the GGL algorithm with $\alpha = 0.1$ and the proposed SGL algorithm with $\beta = 1/2$ and $\alpha = 10^{-1}$. The input for all the algorithms is the sample covariance matrix plus an identity matrix scaled by $1/3$ (see Elgimez paper for an explanation of the input).

The evaluation of the estimated graphs is done by natural intuition, i.e., we expect that similar animals such as (*ant*, *cockroach*), (*bee*, *butterfly*), and (*trout*, *salmon*), would be clustered together, while presently virtually zero connection to other (group of) animals. Under this sense, it can be seen that the SGL algorithm yields a more intutive graph than the one learned by GGL.

6 Does a better initialization affect the final estimation?

The standard initialization is to take $\mathbf{w}_0$ as the off-diagonal elements of the pseudoinverse of the sample covariance matrix, denoted as $\hat{\mathbf{\Theta}}_{\mathsf{scm}}$. This is tantamount to solving the following problem

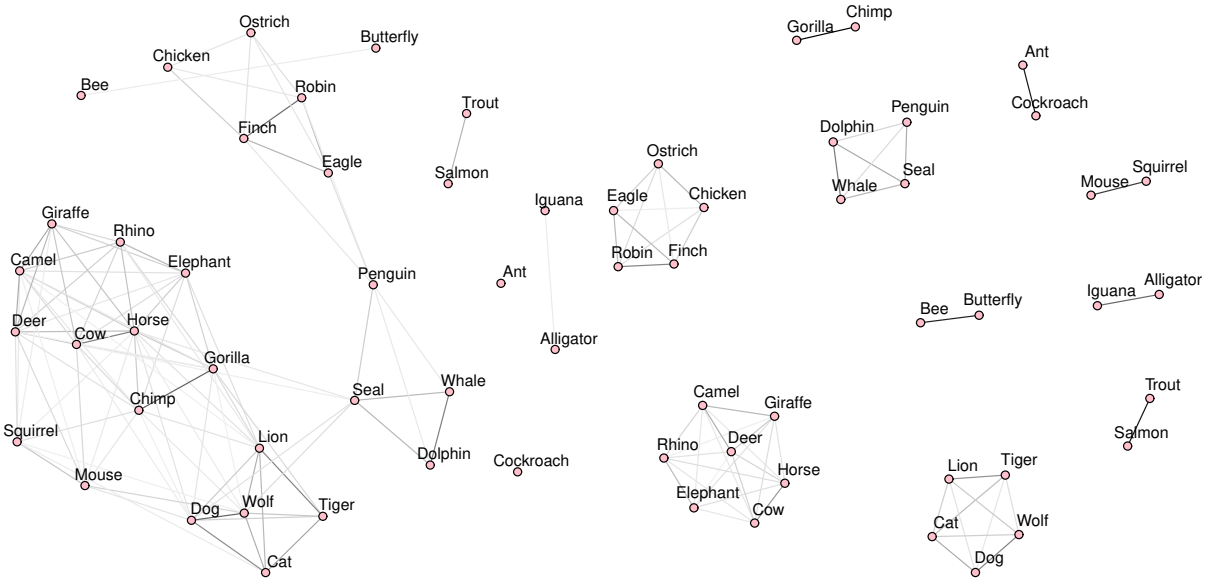
$$\begin{array}{ll} \underset{\mathbf{w}}{\mathsf{minimize}} & ||\hat{\mathbf{\Theta}}_{\mathsf{scm}} - \mathcal{L}\mathbf{w}||^2_{\mathrm{off},F} \\ \mathsf{subject\ to} & \mathbf{w} \geq \mathbf{0}. \end{array}$$

In order to improve the initial estimate, we consider the following optimization problem

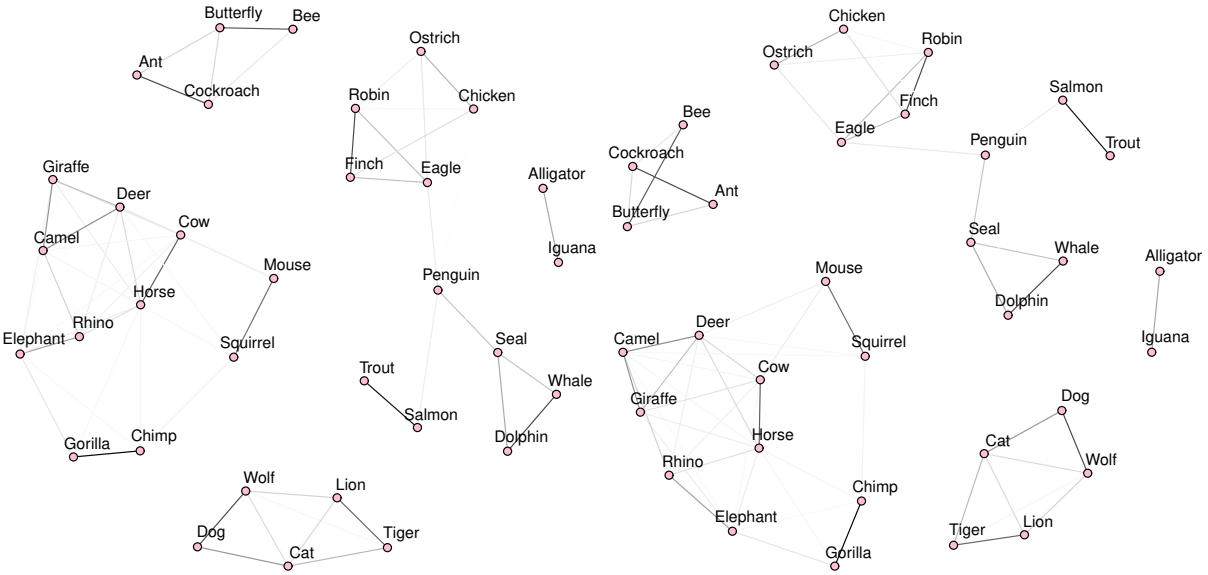
$$\begin{array}{ll} \underset{\mathbf{w}}{\mathsf{minimize}} & ||\hat{\mathbf{\Theta}}_{\mathsf{scm}} - \mathcal{L}\mathbf{w}||^2_{F} \\ \mathsf{subject\ to} & \mathbf{w} \geq \mathbf{0}. \end{array} \tag{1}$$

Note that,

$$||\hat{\mathbf{\Theta}}_{\mathsf{scm}} - \mathcal{L}\mathbf{w}||^2_F = ||\mathrm{vec}\left(\hat{\mathbf{\Theta}}_{\mathsf{scm}}\right) - \mathrm{vec}(\mathcal{L}\mathbf{w})||^2_2 = ||\mathrm{vec}\left(\hat{\mathbf{\Theta}}_{\mathsf{scm}}\right) - \mathbf{R}\mathbf{w}||^2_2, \tag{2}$$



(a) GGL($\alpha = 10^{-1}$). (b) SGL($\beta = 1/2$, $\alpha = 10^{-1}$, $K = 10$).



(c) SGL($\beta = 1/2$, $\alpha = 10^{-1}$, $K = 1$). (d) SGL($\beta = 1/2$, $\alpha = 10^{-1}$, $K = 4$).

Figure 7: Learning the connectivity of the animals dataset for different values of K.

where $\mathbf{R} \triangleq \text{vec} \circ \mathcal{L}$.

We denote the above algorithm as LLQP. Figure 9 shows the performance of the SGL algorithm when initialized by LLQP and ISCM. This experiment indicates that the SGL algorithm remains neutral to the different initilization procedures discussed here, even though it is clear that the LLQP shows a far better performance than ISCM.

7 How much does β affect the performance?

In order to investigate the performance of SGL as a function of β, we take grid graph setting and vary β for different regimes of sample size.

Figure 10 depicts the performance.

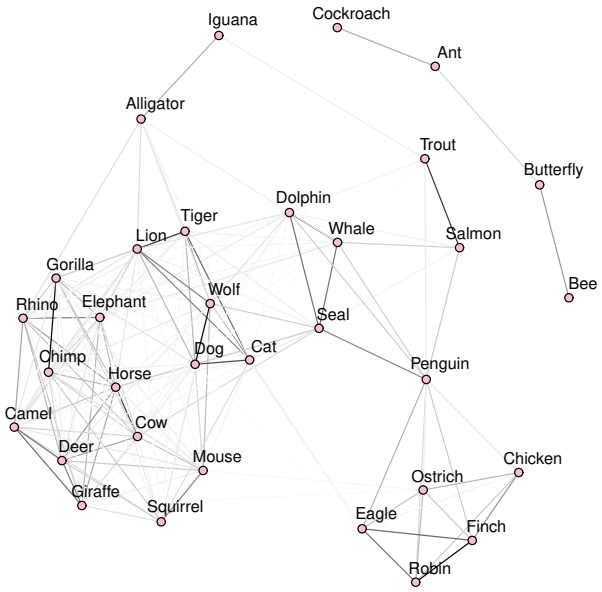


Figure 8: Learning the connectivity of the animals dataset with GLasso. Regularization parameter $\alpha = 0.05618$.

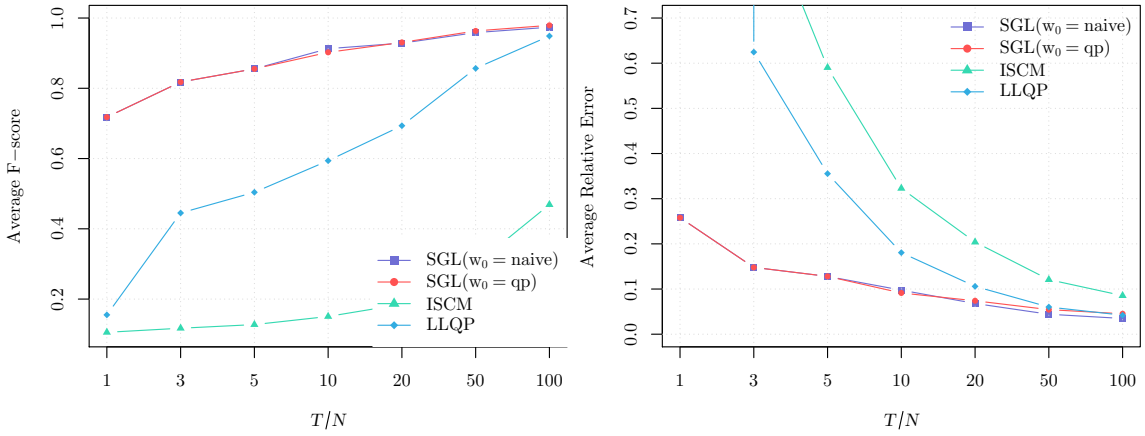


Figure 9: Relative error and F-score attained by SGL with different initialization strategies. SGL($\mathbf{w}_0 = \text{naive}$) and SGL($\mathbf{w}_0 = \text{qp}$) stand for the initialization with ISCM and LLQP, respectively.

8 4-component graph

An experiment with synthetic data generated by a 4-component graph was conducted. We consider the following model:

$$\mathbf{L}_{\text{noisy}} = \mathbf{L}_{\text{true}} + \mathbf{L}_{\text{ER}}, \tag{3}$$

where $\mathbf{L}_{\text{true}}$ represents the Laplacian matrix of a K-component graph (for this example, $K = 4$) denoted as $\mathcal{G}_K^{(p_1,p_2)}$, in which p_1 and p_2 represent the probabilities of node connections across components and within components, respectively; $\mathbf{L}_{\text{ER}}$ represents the Laplacian of an Erdos-Renyi graph $\mathcal{G}_{\text{ER}}^{(p)}$, in which p is the probability of a node connecting to any other node. The weighted edges of $\mathcal{G}_K^{(p_1,p_2)}$ were drawn from Uniform$(0, 1)$, whereas the ones of $\mathcal{G}_{\text{ER}}^{(p)}$ where drawn from Uniform$(0, \kappa)$, in which $\kappa \in (0, 1)$ controls how much noise is added to the true graph. Finally, we set $p_1 = 0$, $p_2 = 1$, $p = 0.35$, and $\kappa = 0.45$.

Then, data were sampled in the form of $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \mathbf{L}_{\text{noisy}}^{\dagger})$, where $\mathbf{A}^{\dagger}$ denotes the generalized inverse of the matrix $\mathbf{A}$. The total number of nodes N and the number of drawn samples T were set to $N = 20$ (5 nodes per component) and $T/N = 30$.

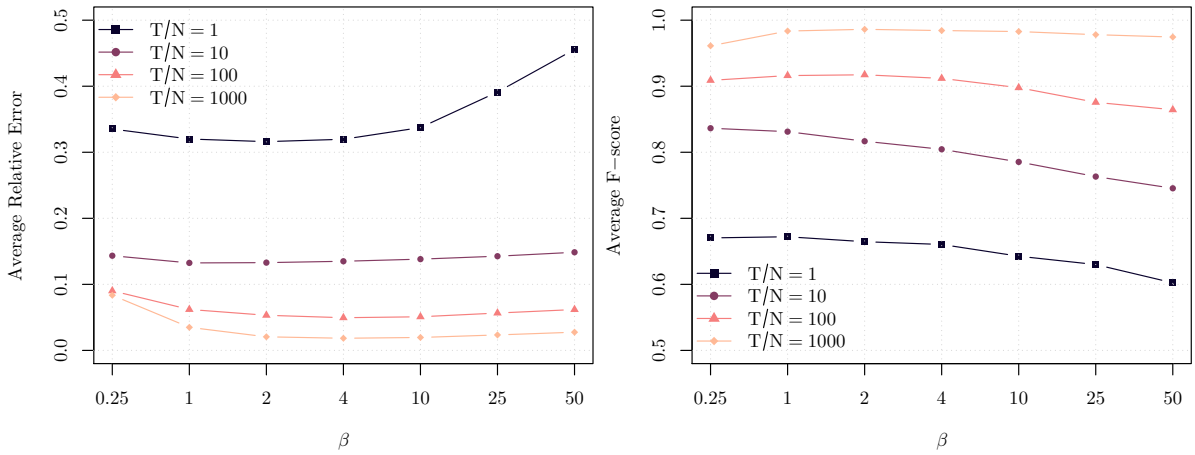
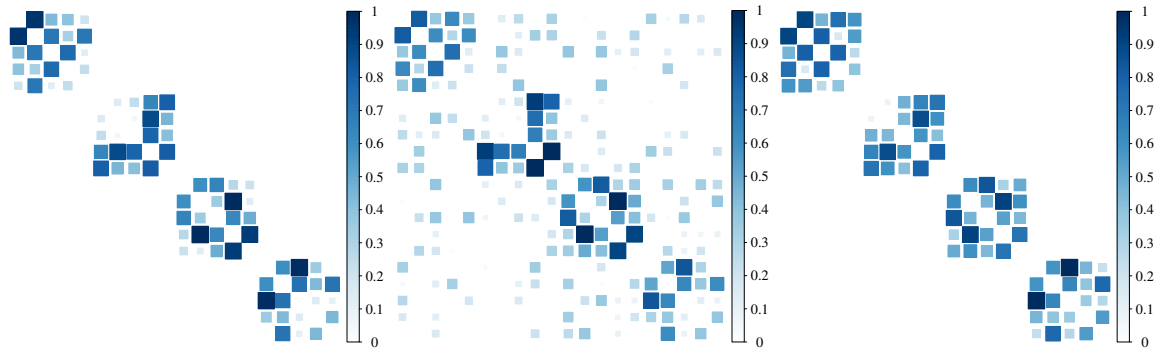
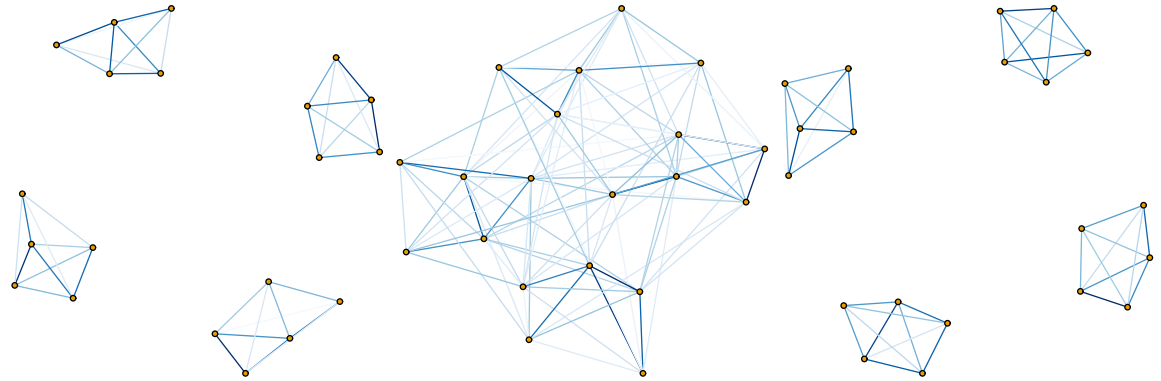


Figure 10: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{grid}}^{(64,0.1)}$.

Figure 11 illustrates the ground truth model, its noisy version, and the model learned by our spectral topology algorithm with $\beta = 20 \cdot N$ and $\alpha = 0.1$. We compute the performance of the learning process by means of the relative error (RE) and the F-score (FS). For this example, our algorithm achives $(\mathsf{RE}, \mathsf{FS}) = (0.210, 1)$ which means a perfect clustering accuracy even in a noisy model that heavily suppress the ground truth weights.



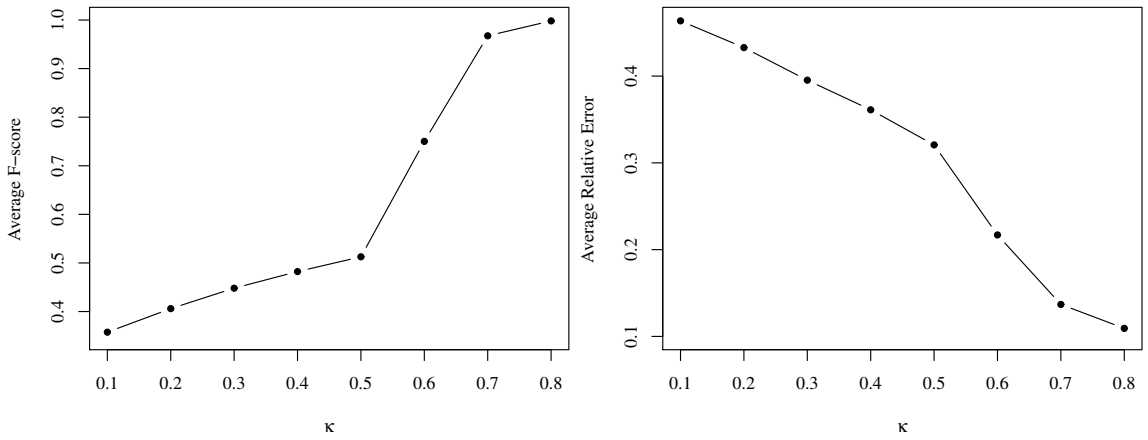
(a) Ground Truth Laplacian matrix (b) Noisy Laplacian matrix (c) Learned Laplacian matrix



(d) Ground Truth graph (e) Noisy graph (f) Learned graph

Figure 11: An example of estimating a 4-component graph. (a) the ground truth graph Laplacian matrix ($\mathbf{L}_{\mathsf{true}}$), (b) $\mathbf{L}_{\mathsf{true}}$ after being corrupted by noise, (c) the learned graph Laplacian with a performance of $(\mathsf{RE}, \mathsf{FS}) = (0.210, 1)$. The panels (d), (e), and (f) correspond to the graphs represented by the Laplacian matrices in (a), (b), and (c), respectively.

Figure 13 shows the performance of our algorithm, using the same settings as the experiment above, but for different noise regimes.



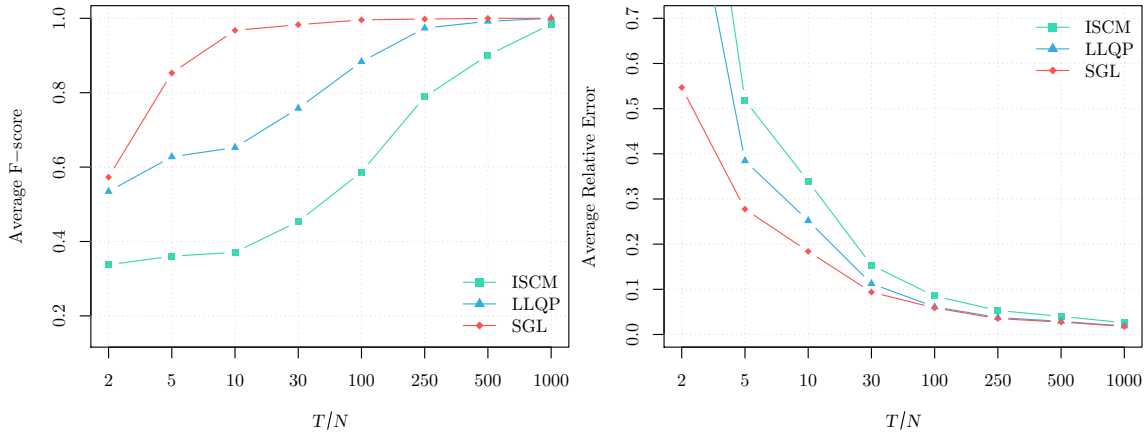
(a) Average F-score vs noise factor

(b) Relative error vs noise factor

Figure 12: Average performance results, as a function of the noise factor, for learning Laplacian matrix of a modular graph embedded in noise.

9 Performance

To infer the performance as a function of the ratio T/N, we consider a 4-component graph in which every component has four nodes connected with probability 1. The edge weights among nodes are drawn from Uniform(0.1, 3). Additionally, we perform a grid search on the sparsity parameter α, just like the one performed by Elgimez.



(a) Average F-score vs T/N

(b) Average Relative Error vs T/N

Figure 13: Average performance results as, a function of the number of samples, for learning Laplacian matrix of a 4-component graph.