# Learning graphs from data via spectral constraints

*Ze Vinicius, Daniel Palomar, Jiaxi Ying, and Sandeep Kumar*
*Hong Kong University of Science and Technology (HKUST)*
*2019-04-28*

---

## Installation

Check out https://mirca.github.io/spectralGraphTopology for installation instructions.

## Problem Statement

Graphs are arguably one of the most popular mathematical structures that find applications in a myriad of scientific and engineering fields.

In the era of big data, graphs can be used to model a vast diversity of phenomena, including customer preferences, brain activity, genetic structures, just to name a few. Therefore, it is of utmost importance to be able to reliably estimate such structures from noisy, often sparse, low-rank datasets.

The Laplacian matrix of a graph contains the information about its topology, i.e., how nodes are connected among themselves. By definition a (combinatorial) Laplacian matrix is positive semi-definite, symmetric, and with sum of rows equal to zero.

One common approach to estimate the Laplacian matrix of a graph (without satisfying the zero row-sum property) would be via the generalized inverse of the sample covariance matrix, which is an assymptotically unbiased and efficient estimator. In this document, we call this approach the naive one. In R, this estimator can be computed simply as `MASS::ginv(cov(Y))`, where`Y` is the data matrix. However, this estimator performs very poorly when the sample size is small when compared with the number of nodes, which makes its use questionable for practical purposes.

Another classical approach, the well-known graphical lasso algorithm, was proposed in [1] where a $\ell_1$-norm penalty term was incorporated in order to induce sparsity on the solution. The R package `glasso` provides an implementation of this estimator.

We, however, begin by defining a Laplacian linear operator $\mathcal{L}$ that maps a vector of edge weights $\mathbf{w}$ into a valid Laplacian matrix. Additionally, we impose constraints on the eigenvalues and eigenvectors of the Laplacian matrix in such a way that the underlying optimization problem may be expressed as follows:

$$\begin{array}{ll} \underset{\mathbf{w},\boldsymbol{\lambda},\mathbf{U}}{\mathsf{minimize}} & -\log\det\left(\mathsf{Diag}(\boldsymbol{\lambda})\right) + \mathrm{tr}\left(\mathbf{S}\mathcal{L}\mathbf{w}\right) + \alpha h(\mathcal{L}\mathbf{w}) + \frac{\beta}{2}\left\|\mathcal{L}\mathbf{w} - \mathbf{U}\mathsf{Diag}(\boldsymbol{\lambda})\mathbf{U}^T\right\|_F^2 \\ \mathsf{subject\ to} & \mathbf{w} \geq 0, \boldsymbol{\lambda} \in \mathcal{S}_{\boldsymbol{\Lambda}}, \text{ and } \mathbf{U}^T\mathbf{U} = \mathbf{I} \end{array}$$

where $h()$ is a regularization function (e.g. to induce sparsity), $\mathbf{S}$ is the sample covariance matrix, $\mathcal{S}_{\boldsymbol{\Lambda}}$ further constrains the eigenvalues of the Laplacian matrix. For example, for a $k$-component graph with $p$ nodes, $\mathcal{S}_{\boldsymbol{\Lambda}} = \{\{\lambda_i\}_{i=1}^p | \lambda_1 = \lambda_2 = \cdots = \lambda_k = 0,\ 0 < \lambda_{k+1} \leq \lambda_{k+2} \leq \cdots \leq \lambda_p\}$.

To solve this optimization problem, we employ a block majorization-minimization framework that updates each of the variables $(\mathbf{w}, \boldsymbol{\lambda}, \mathbf{U})$ at once while fixing the remaning ones. For the mathematical details of the solution, including a convergence proof, please refer to our paper at: https://arxiv.org/pdf/1904.09792.pdf.

In order to learn bipartite graphs, we take advantage of the fact that the eigenvalues of the adjacency matrix of graph are symmetric around 0, and we formulate the following optimization problem:

$$\begin{array}{ll} \underset{\mathbf{w},\boldsymbol{\psi},\mathbf{V}}{\mathsf{minimize}} & -\log\det(\mathcal{L}\mathbf{w} + \frac{1}{p}\mathbf{1}\mathbf{1}^T) + \mathrm{tr}(\mathbf{S}\mathcal{L}\mathbf{w}) + \alpha h(\mathcal{L}\mathbf{w}) + \frac{\nu}{2}\|\mathcal{A}\mathbf{w} - \mathbf{V}\mathsf{Diag}(\boldsymbol{\psi})\mathbf{V}^T\|_F^2, \\ \mathsf{subject\ to} & \mathbf{w} \geq 0,\ \boldsymbol{\psi} \in \mathcal{S}_{\boldsymbol{\psi}}, \text{ and } \mathbf{V}^T\mathbf{V} = \mathbf{I}, \end{array}$$

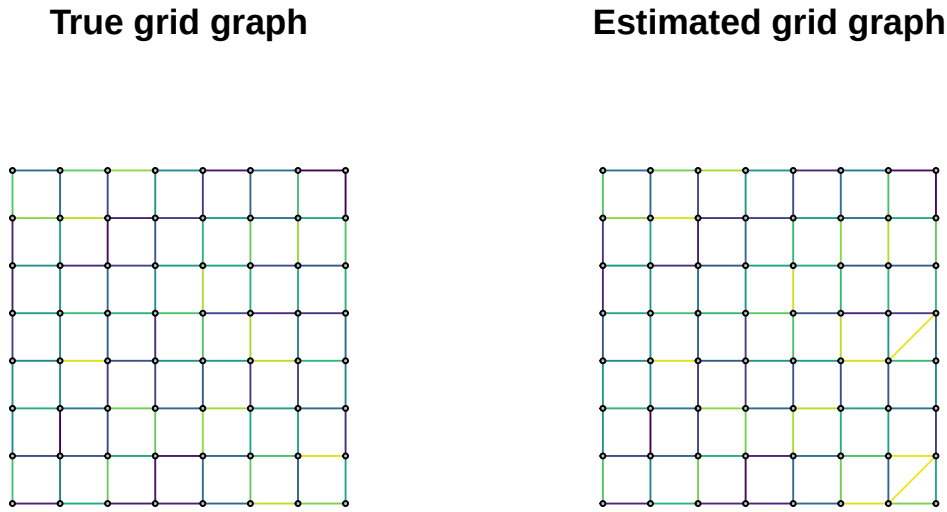
In a similar fashion, we construct the optimization problem to estimate a $k$-component bipartite graph by combining the constraints related to the Laplacian and adjacency matrices.

## Package usage

The `spectralGraphTopology` package provides three main functions to estimate k-component, bipartite, and k-component bipartite graphs, respectively: `learn_k_component_graph`, `learn_bipartite_graph`, and `learn_bipartite_k_component`. In the next subsections, we will check out how to apply those functions in synthetic datasets.

### Learning a grid graph

```
library(spectralGraphTopology)
library(igraph)
library(viridis)
set.seed(0)

# generate the graph and the data from the graph
p <- 64
grid <- make_lattice(length = sqrt(p), dim = 2)
n <- as.integer(100 * p)
E(grid)$weight <- runif(gsize(grid), min = 1e-1, max = 3)
L_true <- as.matrix(laplacian_matrix(grid))  # true Laplacian matrix
Y <- MASS::mvrnorm(n, mu = rep(0, p), Sigma = MASS::ginv(L_true))

# estimate the graph
S <- cov(Y)
graph <- learn_k_component_graph(S, w0 = "qp", beta = 20, alpha = 5e-3,
                                 abstol = 1e-5, verbose = FALSE)
graph$Adjacency[graph$Adjacency < 5e-2] <- 0
estimated_grid <- graph_from_adjacency_matrix(graph$Adjacency,
                                              mode = "undirected", weighted = TRUE)

# plots
colors <- viridis(20, begin = 0, end = 1, direction = -1)
c_scale <- colorRamp(colors)
E(estimated_grid)$color = apply(c_scale(E(estimated_grid)$weight / max(E(estimated_grid)$weight)), 1,
                          function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
E(grid)$color = apply(c_scale(E(grid)$weight / max(E(grid)$weight)), 1,
                      function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
V(estimated_grid)$color = "grey"
V(grid)$color = "grey"
la <- layout_on_grid(grid)
par(mfrow = c(1, 2))
plot(grid, layout = la, vertex.label = NA, vertex.size = 3)
title("True grid graph")
plot(estimated_grid, layout = la, vertex.label = NA, vertex.size = 3)
title("Estimated grid graph")
```

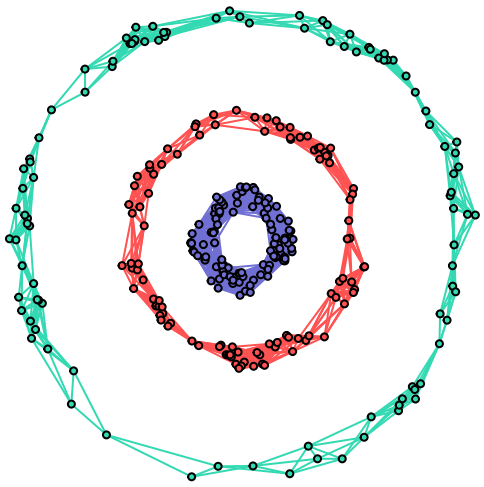



### Learning a 3-component graph

The next snippet of code shows how to learn the clusters of a set of points distributed on the plane.

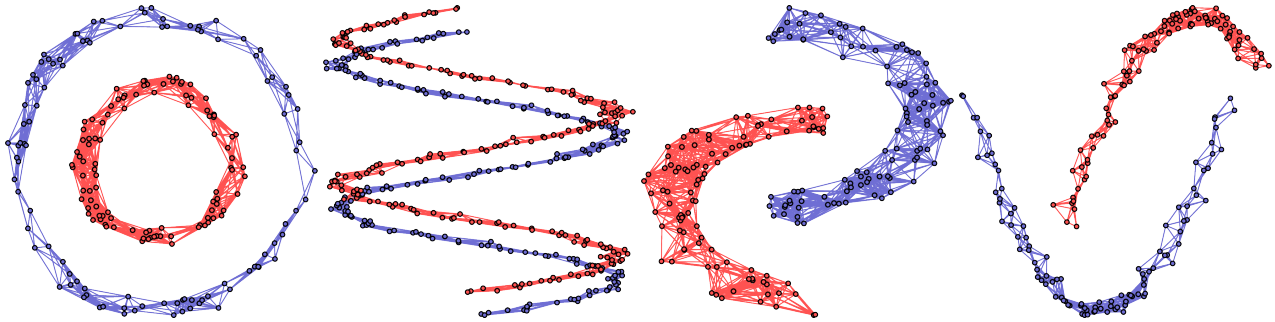
```
library(spectralGraphTopology)
library(clusterSim)
library(igraph)
set.seed(42)

# generate the graph and the data from the graph
n <- 100  # number of nodes per cluster
circles3 <- shapes.circles3(n)  # generate datapoints
k <- 3  # number of components
S <- crossprod(t(circles3$data))  # compute sample correlation matrix

# estimate the graph
graph <- learn_k_component_graph(S, k = k, beta = 1, verbose = FALSE, fix_beta = FALSE, abstol = 1e-3)

# plots
# build network
net <- graph_from_adjacency_matrix(graph$Adjacency, mode = "undirected", weighted = TRUE)
# colorify nodes and edges
colors <- c("#706FD3", "#FF5252", "#33D9B2")
V(net)$cluster <- circles3$clusters
E(net)$color <- apply(as.data.frame(get.edgelist(net)), 1,
                      function(x) ifelse(V(net)$cluster[x[1]] == V(net)$cluster[x[2]],
                                         colors[V(net)$cluster[x[1]]], '#000000'))
V(net)$color <- colors[circles3$clusters]
plot(net, layout = circles3$data, vertex.label = NA, vertex.size = 3)
title("Estimated graph on the three circles dataset")
```

## Estimated graph on the three circles dataset



Similar structures may be inferred as well. The plots below depict the results of applying `learn_k_component_graph()` on a variaety of spatially distributed points.

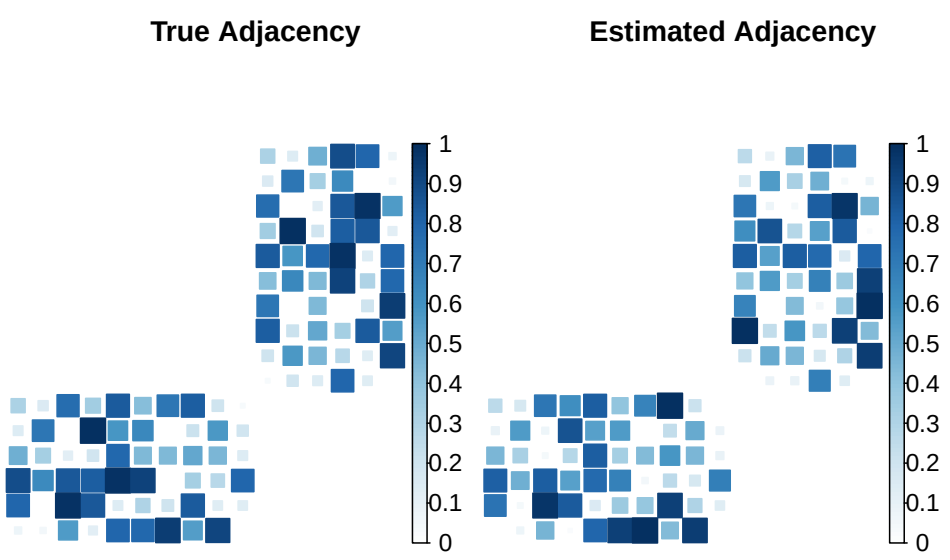
```
knitr::include_graphics(c("figures/circles2.png", "figures/helix3d.png",
                          "figures/twomoon.png", "figures/worms.png"))
```



### Learning a bipartite graph

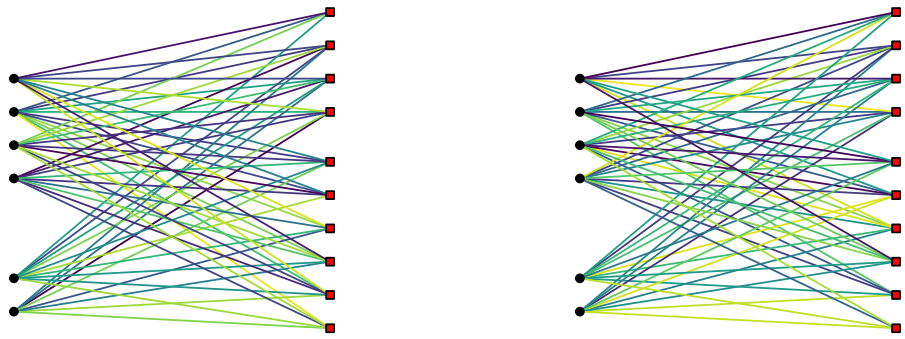
```
library(spectralGraphTopology)
library(igraph)
library(viridis)
library(corrplot)
set.seed(42)

# generate the graph and the data from the graph
n1 <- 10
n2 <- 6
n <- n1 + n2
pc <- .9
bipartite <- sample_bipartite(n1, n2, type="Gnp", p = pc, directed=FALSE)
# randomly assign edge weights to connected nodes
E(bipartite)$weight <- runif(gsize(bipartite), min = 0, max = 1)
# get true Laplacian and Adjacency
```

```r
Ltrue <- as.matrix(laplacian_matrix(bipartite))
Atrue <- diag(diag(Ltrue)) - Ltrue
# get samples
Y <- MASS::mvrnorm(100 * n, rep(0, n), Sigma = MASS::ginv(Ltrue))

# estimate graph
S <- cov(Y)  # compute sample covariance matrix
graph <- learn_bipartite_graph(S, z = 4, verbose = FALSE)
graph$Adjacency[graph$Adjacency < 1e-3] <- 0

# plots
par(mfrow = c(1, 2))
corrplot(Atrue / max(Atrue), is.corr = FALSE, method = "square", addgrid.col = NA,
         tl.pos = "n", cl.cex = 1, mar=c(0, 0, 3, 0))
title("True Adjacency")
corrplot(graph$Adjacency / max(graph$Adjacency), is.corr = FALSE, method = "square", addgrid.col = NA,
         tl.pos = "n", cl.cex = 1, mar=c(0, 0, 3, 0))
title("Estimated Adjacency")
```



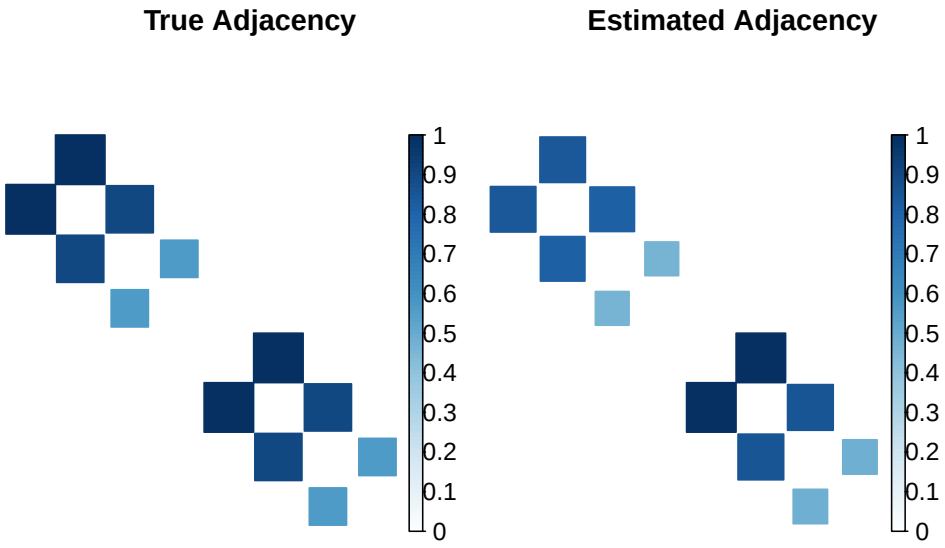

```r
# build networks
estimated_bipartite <- graph_from_adjacency_matrix(graph$Adjacency, mode = "undirected", weighted = TRU
V(estimated_bipartite)$type <- c(rep(0, 10), rep(1, 6))
la = layout_as_bipartite(estimated_bipartite)
#> Warning in layout_as_bipartite(estimated_bipartite): vertex types converted
#> to logical
colors <- viridis(20, begin = 0, end = 1, direction = -1)
c_scale <- colorRamp(colors)
E(estimated_bipartite)$color = apply(c_scale(E(estimated_bipartite)$weight / max(E(estimated_bipartite)$
                          function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
E(bipartite)$color = apply(c_scale(E(bipartite)$weight / max(E(bipartite)$weight)), 1,
                     function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
la = la[, c(2, 1)]
# Plot networks: true and estimated
par(mfrow = c(1, 2))
plot(bipartite, layout = la, vertex.color=c("red","black")[V(bipartite)$type + 1],
```

```
     vertex.shape = c("square", "circle")[V(bipartite)$type + 1],
     vertex.label = NA, vertex.size = 5)
title("True bipartite graph")
plot(estimated_bipartite, layout = la, vertex.color=c("red","black")[V(estimated_bipartite)$type + 1],
     vertex.shape = c("square", "circle")[V(estimated_bipartite)$type + 1],
     vertex.label = NA, vertex.size = 5)
title("Estimated Bipartite Graph")
```

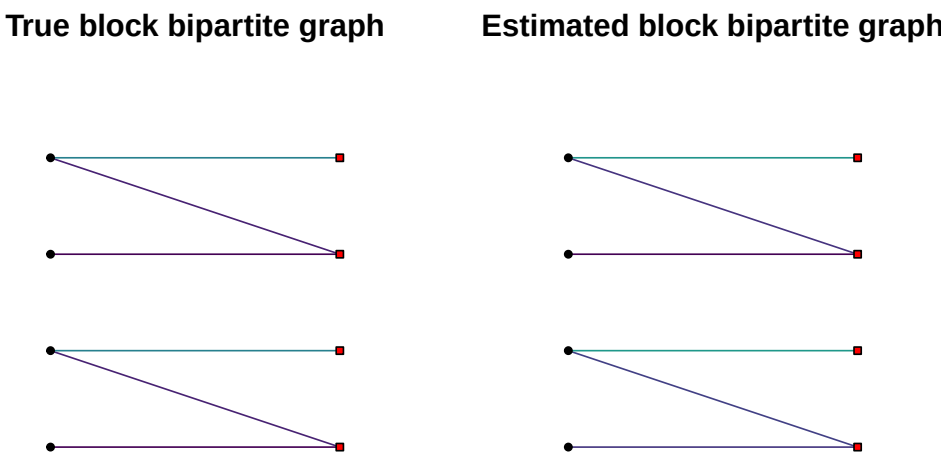



### Learning a 2-component bipartite graph

```
library(spectralGraphTopology)
library(igraph)
library(viridis)
library(corrplot)
set.seed(42)

# generate the graph and the data from the graph
w <- c(1, 0, 0, 1, 0, 1) * runif(6)
Laplacian <- block_diag(L(w), L(w))
Atrue <- diag(diag(Laplacian)) - Laplacian
bipartite <- graph_from_adjacency_matrix(Atrue, mode = "undirected", weighted = TRUE)
n <- ncol(Laplacian)

# estimate graph
Y <- MASS::mvrnorm(40 * n, rep(0, n), MASS::ginv(Laplacian))
graph <- learn_bipartite_k_component_graph(cov(Y), k = 2, beta = 1e2, nu = 1e2, verbose = FALSE)
graph$Adjacency[graph$Adjacency < 1e-2] <- 0

# plots
# Plot Adjacency matrices: true and estimated
par(mfrow = c(1, 2))
corrplot(Atrue / max(Atrue), is.corr = FALSE, method = "square", addgrid.col = NA,
         tl.pos = "n", cl.cex = 1, mar=c(0, 0, 3, 0))
title("True Adjacency")
corrplot(graph$Adjacency / max(graph$Adjacency), is.corr = FALSE, method = "square", addgrid.col = NA,
         tl.pos = "n", cl.cex = 1, mar=c(0, 0, 3, 0))
title("Estimated Adjacency")
```




```r
# Plot networks
estimated_bipartite <- graph_from_adjacency_matrix(graph$Adjacency, mode = "undirected", weighted = TRU
V(bipartite)$type <- rep(c(TRUE, FALSE), 4)
V(estimated_bipartite)$type <- rep(c(TRUE, FALSE), 4)
la = layout_as_bipartite(estimated_bipartite)
colors <- viridis(20, begin = 0, end = 1, direction = -1)
c_scale <- colorRamp(colors)
E(estimated_bipartite)$color = apply(c_scale(E(estimated_bipartite)$weight / max(E(estimated_bipartite)$
                                     function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
E(bipartite)$color = apply(c_scale(E(bipartite)$weight / max(E(bipartite)$weight)), 1,
                           function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
la = la[, c(2, 1)]
# Plot networks: true and estimated
par(mfrow = c(1, 2))
plot(bipartite, layout = la,
     vertex.color = c("red","black")[V(bipartite)$type + 1],
     vertex.shape = c("square", "circle")[V(bipartite)$type + 1],
     vertex.label = NA, vertex.size = 5)
title("True block bipartite graph")
plot(estimated_bipartite, layout = la,
     vertex.color = c("red","black")[V(estimated_bipartite)$type + 1],
     vertex.shape = c("square", "circle")[V(estimated_bipartite)$type + 1],
     vertex.label = NA, vertex.size = 5)
title("Estimated block bipartite graph")
```



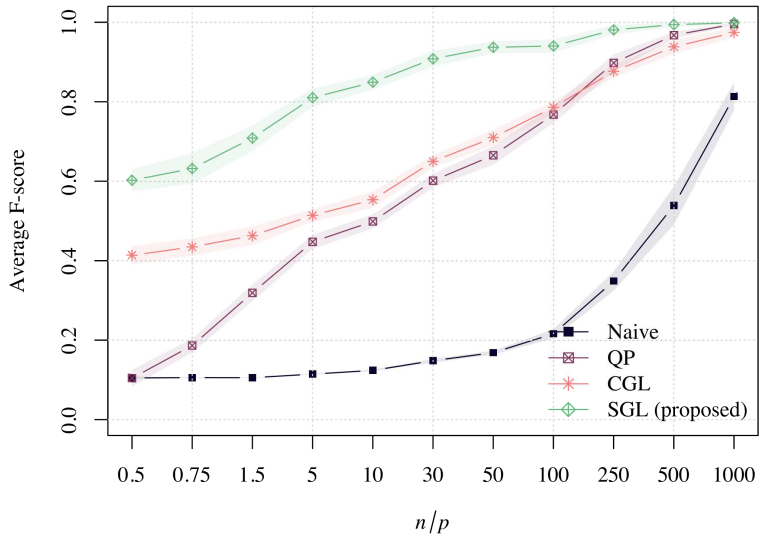

## Performance comparison

We use the following baseline algorithms for performance comparison:

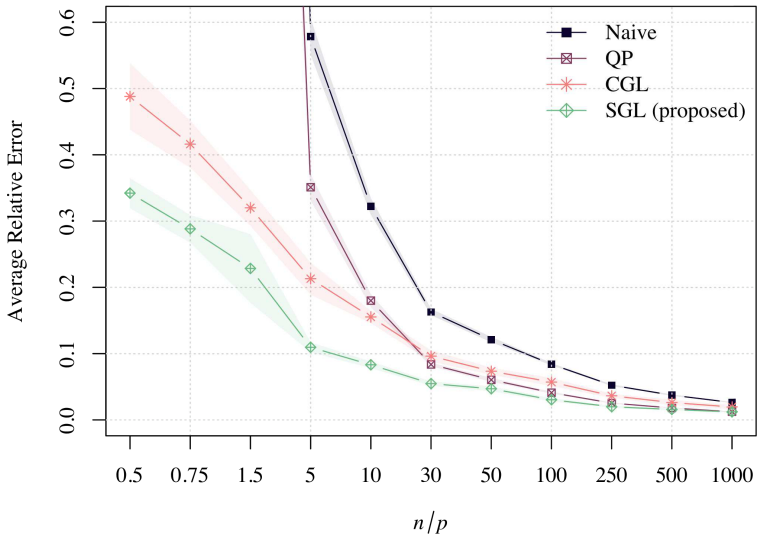
1. the generalized inverse of the sample covariance matrix, denoted as Naive;
2. a quadratic program estimator given by $\mathsf{minimize}_{\mathbf{w}} \quad \|\mathbf{S}^{\dagger} - \mathcal{L}\mathbf{w}\|_F$, where $\mathbf{S}^{\dagger}$ is the generalized inverse of the sample covariance matrix, denoted as QP;
3. the Combinatorial Graph Laplacian proposed by [2] denoted as CGL.

The plots below show the performance in terms of F-score and relative error among the proposed algorithm, denoted as SGL, and the baseline ones when learning a grid graph with 64 nodes and edges uniformly drawn from the interval [.1, 3]. For each algorithm, the shaded area and the solid curve represent the standard deviation and the mean of several Monte Carlo realizations. It can be noticed that SGL outperforms all the baseline algorithms in all sample size regimes. Such superior performance maybe be attributed to the highly structured nature of grid graphs.

```
knitr::include_graphics("figures/fscore_grid.png")
```

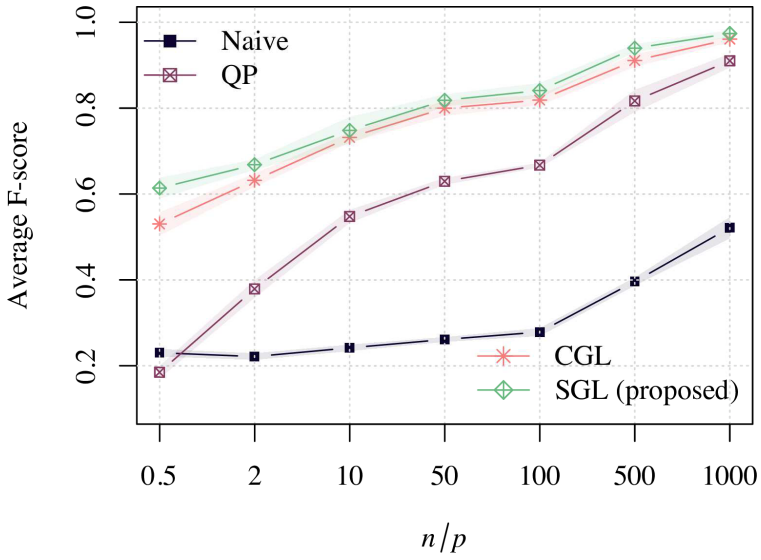



```
knitr::include_graphics("figures/relative_error_grid.png")
```

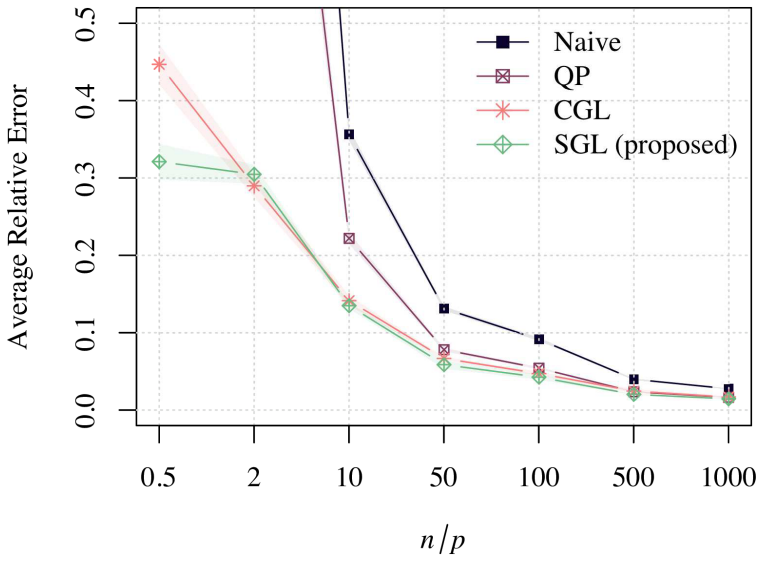



In a similar fashion, the plots below shows algorithmic performance for modular graphs with 64 nodes and 4 modules, such that the probability of connection within module was set to 50%, whereas the probability of connection accross modules was set to 1%. In this scenario, SGL outperforms the baselines algorithms QP and Naive, while having a similar performance to that of CGL. This may be explained by the fact that the edges connecting nodes do not quite have a deterministic structure like those of the grid graphs.

```
knitr::include_graphics("figures/fscore_modular.png")
```




```
knitr::include_graphics("figures/relative_error_modular.png")
```




## Clustering

One of the most direct applications of learning k-component graphs is on the classical unsupervised machine learning problem: data clustering. For this task, we make use of two datasets: the animals dataset [3] the Cancer RNA-Seq dataset [4].

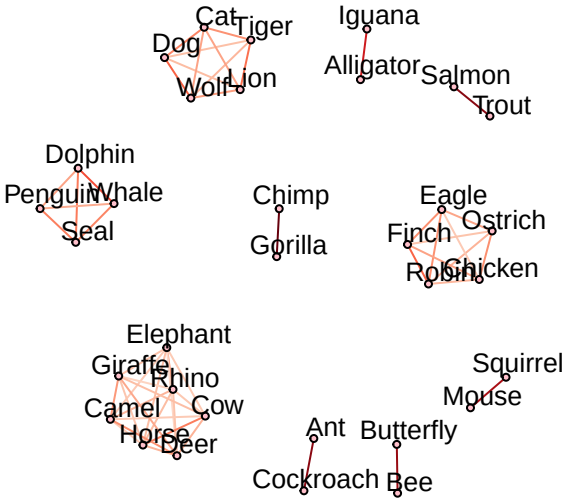
The animals dataset consists of binary answers to questions such as "is warm-blooded?," "has lungs?", etc. There are a total of 102 such questions, which make up the features for 33 animal categories.

The cancer-RNA Seq dataset consists of genetic features which map 5 types of cancer: breast carcinoma (BRCA), kidney renal clear-cell carcinoma (KIRC), lung adenocarcinoma (LUAD), colon adenocarcinoma (COAD), and prostate adenocarcinoma (PRAD). This dataset consists of 801 labeled samples, in which every sample has 20531 genetic features.

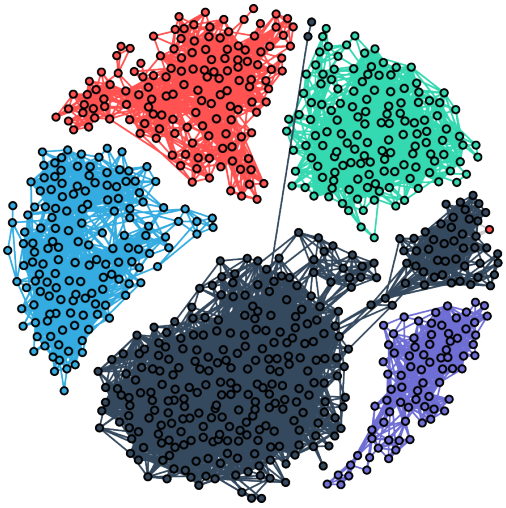
The clustering results for these datasets are shown below. The code used for the cancer-rna dataset can be found at our GitHub repo: https://github.com/dppalomar/spectralGraphTopology/tree/master/benchmarks/cancer-rna

```
library(pals)

# load data
df <- read.csv("animals-dataset/features.txt", header = FALSE)
names <- matrix(unlist(read.csv("animals-dataset/names.txt", header = FALSE)))
Y <- t(matrix(as.numeric(unlist(df)), nrow = nrow(df)))
p <- ncol(Y)

# estimate graph
graph <- learn_k_component_graph(cov(Y) + diag(1/3, p, p), w0 = "qp", beta = 1, k = 10, verbose = FALSE)

# plots
net <- graph_from_adjacency_matrix(graph$Adjacency, mode = "undirected", weighted = TRUE)
colors <- brewer.reds(100)
c_scale <- colorRamp(colors)
E(net)$color = apply(c_scale(abs(E(net)$weight) / max(abs(E(net)$weight))), 1,
                     function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
V(net)$color = "pink"
plot(net, vertex.label = names,
     vertex.size = 3,
```

```
    vertex.label.dist = 1,
    vertex.label.family = "Helvetica",
    vertex.label.cex = .8,
    vertex.label.color = "black")
```




```
knitr::include_graphics("figures/cancer-rna-graph.png")
```



## References


[1] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[2] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under laplacian and structural constrints," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 825–841, 2017.

[3] D. N. Osherson, J. Stern, O. Wilkie, M. Stob, and E. E. Smith, "Default probability," *Cognitive Science*, vol. 15, no. 2, pp. 251–269, 1991.

[4] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository." University of California, Irvine, School of Information; Computer Sciences, 2017.