Learning the topology of graphs

Convex Group-HKUST

2018-08-21

Contents

---

1 Installation

For installation instructions, please visit https://github.com/dppalomar/spectralGraphTopology

2 Problem Statement

The Laplacian matrix $\mathbf{\Theta}$ of a graph contains the information of its topology and weight connections. By definition a Laplacian matrix is positive semi-definite, symmetric, and with sum of rows equal to zero. The Laplacian linear operator $\mathcal{L}\mathbf{w}$ maps a vector of weights $\mathbf{w}$ into a valid Laplacian matrix so that the conditions are satisfied by construction.

One common approach to estimate the Laplacian matrix would be via the generalized inverse of the 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 as `MASS::ginv(cov(Y))`, where`Y` is the data matrix.

Another classical approach was proposed in [1] which incorporates a ℓ_1-norm penalty term in order to induce sparsity on the solution. The R package `glasso` provides an implementation of this estimator.

The underlying optimization problem of learning a K-component graph, solved by `spectralGraphTopology`, may be expressed as follows:

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

where $\mathcal{S}_{\mathbf{\Lambda}}$ further constrains the eigenvalues of Laplacian matrices $\mathbf{\Theta} = \mathcal{L}\mathbf{w}$ according to some topology (e.g., a K-component graph has K zero eigenvalues).

In order to solve this problem, we use a block coordinate descent algorithm to iteratively optimize each variable while helding the others fixed. For details check the paper or the algorithm description at the end of this document.

3 Usage of the package

We illustrate the usage of the package with simulated data, as follows:

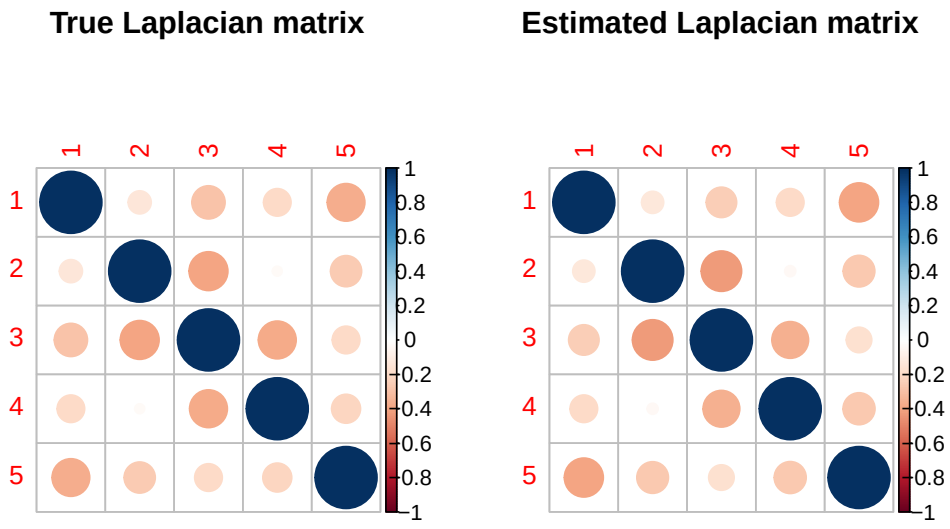
```r
library(spectralGraphTopology)
set.seed(123)

# Number of samples
T <- 200
# Vector to generate the Laplacian matrix of the graph
w <- runif(10)
# Laplacian matrix
Lw <- L(w)
# Sample data from a Multivariate Gaussian
N <- ncol(Lw)
Y <- MASS::mvrnorm(T, rep(0, N), MASS::ginv(Lw))
# Number of components of the graph
K <- 1
# Learn the Laplacian matrix
res <- learnGraphTopology(Y, K, beta = 10)
```

Let's visually inspect the true Laplacian and the estimated one:

```r
Lw
#>            [,1]       [,2]       [,3]       [,4]       [,5]
#> [1,]  2.3678770 -0.2875775 -0.7883051 -0.4089769 -0.8830174
#> [2,] -0.2875775  1.8017068 -0.9404673 -0.0455565 -0.5281055
#> [3,] -0.7883051 -0.9404673  3.1726265 -0.8924190 -0.5514350
#> [4,] -0.4089769 -0.0455565 -0.8924190  1.8035672 -0.4566147
#> [5,] -0.8830174 -0.5281055 -0.5514350 -0.4566147  2.4191726
res$Lw
#>            [,1]        [,2]       [,3]        [,4]       [,5]
#> [1,]  2.2953931 -0.27079062 -0.6305267 -0.41598450 -0.9780912
#> [2,] -0.2707906  1.95134563 -1.0254774 -0.06127561 -0.5938020
#> [3,] -0.6305267 -1.02547743  2.9706229 -0.83893311 -0.4756856
#> [4,] -0.4159845 -0.06127561 -0.8389331  1.91751386 -0.6013206
#> [5,] -0.9780912 -0.59380196 -0.4756856 -0.60132063  2.6488995
```

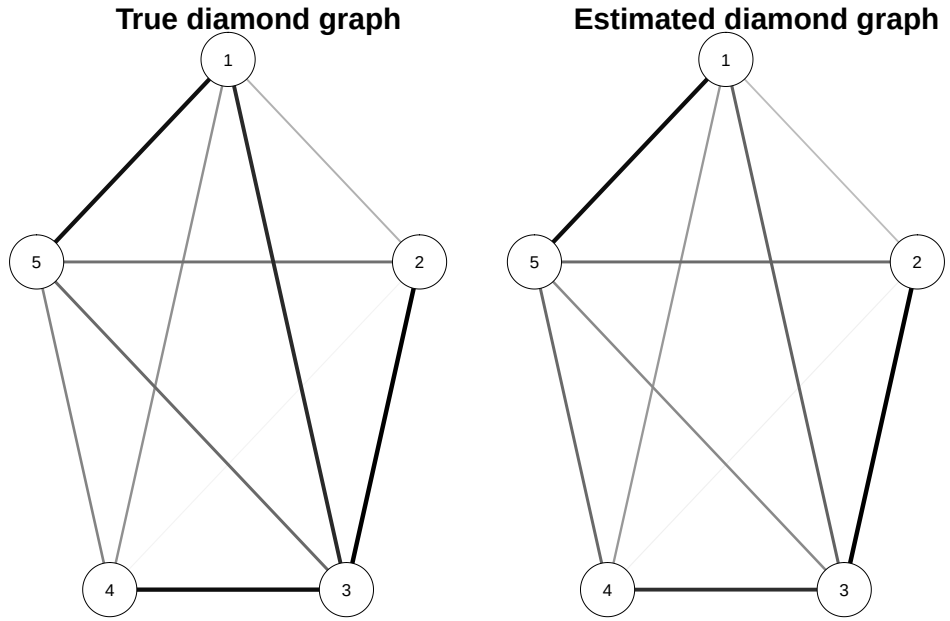
Another visual tool is the correlation matrix:

```r
library(corrplot)
par(mfrow = c(1, 2))
corrplot(cov2cor(Lw))
title("True Laplacian matrix", line = 2.5)
corrplot(cov2cor(res$Lw))
title("Estimated Laplacian matrix", line = 2.5)
```



Finally, we can also plot the graph structure:

```
# True adjancency matrix
W <- Lw - diag(diag(Lw))
library(qgraph)
par(mfrow = c(1, 2))
qgraph(W, esize = 5, edge.color = "black")
title("True diamond graph", line = 2.5)
qgraph(res$W, esize = 5, edge.color = "black")
title("Estimated diamond graph", line = 2.5)
```



We evaluate the performance of the learning process in a more objective manner by computing two criterions:

1. relative error
2. percentage improvement in average loss (PRIAL)

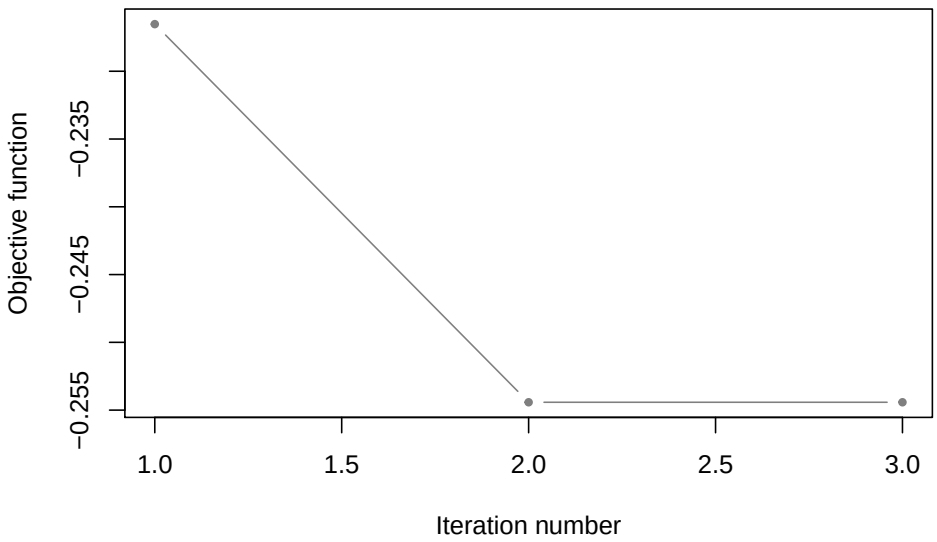
```
relativeError <- function(Xtrue, Xest) {
    return (100 * norm(Xtrue - Xest, type = "F") / norm(Xtrue, type = "F"))
}

prial <- function(Xtrue, Xest) {
    Xnaive <- MASS::ginv(cov(Xtrue))
    return (100 * (1 - (norm(Xest - Xtrue, type = "F") /
                        norm(Xnaive - Xtrue, type = "F"))^2))
}
```

```
Lw_naive <- MASS::ginv(cov(Y))
rel_err <- c(proposed = relativeError(Lw, res$Lw),
             naive = relativeError(Lw, Lw_naive))
prial_avg <- c(proposed = prial(Lw, res$Lw),
               naive = prial(Lw, Lw_naive))
rel_err
#> proposed    naive
#> 8.869914 9.224060
prial_avg
#> proposed    naive
#> 98.91276 98.82421
```

In this case, the naive estimation of the Laplacian matrix (i.e., generalized inverse of the sample covariance matrix) performs already quite well since the ratio T/N is large enough 40 for the sample covariance matrix to be accurately estimated.

Let's also look at the convergence of the objective function versus iterations:

```
N_iter <- length(res$obj_fun)
plot(c(1:N_iter), res$obj_fun, type = "b", pch=19, cex=.6, col = scales::alpha("black", .5),
     xlab = "Iteration number", ylab = "Objective function")
```



For K > 1, we can generate the Laplacian as a block diagonal matrix, as follows

```
library(spectralGraphTopology)
T <- 500
```

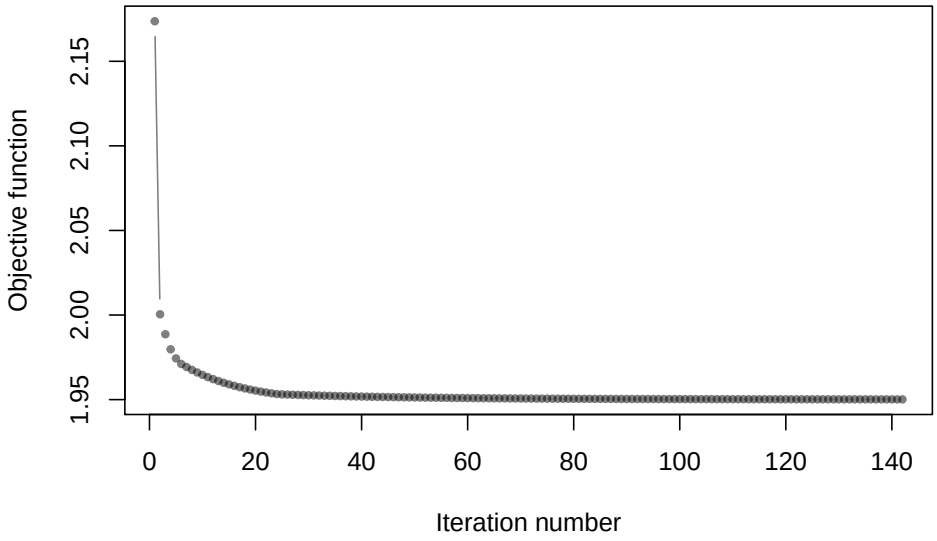
```
w1 <- runif(3)
w2 <- runif(6)
Lw1 <- L(w1)
Lw2 <- L(w2)
N1 <- ncol(Lw1)
N2 <- ncol(Lw2)
Lw <- blockDiag(Lw1, Lw2)
Y <- MASS::mvrnorm(T, rep(0, N1 + N2), MASS::ginv(Lw))
K <- 2
res <- learnGraphTopology(Y, K, beta = 10)
```

```
Lw_naive <- MASS::ginv(cov(Y))
rel_err <- c(proposed = relativeError(Lw, res$Lw),
             naive = relativeError(Lw, Lw_naive))
prial_avg <- c(proposed = prial(Lw, res$Lw),
               naive = prial(Lw, Lw_naive))
rel_err
#>  proposed     naive
#>  9.163421 12.742653
prial_avg
#> proposed    naive
#> 99.54915 99.12817
```

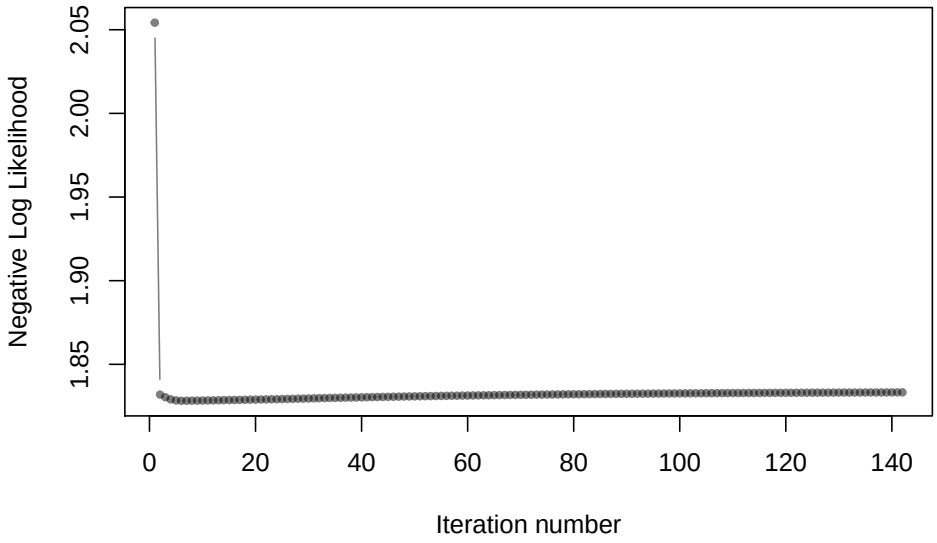
```
Lw
#>             [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
#> [1,]  1.1183094 -0.8509632 -0.2673462  0.0000000  0.0000000  0.0000000
#> [2,] -0.8509632  1.4495635 -0.5986003  0.0000000  0.0000000  0.0000000
#> [3,] -0.2673462 -0.5986003  0.8659465  0.0000000  0.0000000  0.0000000
#> [4,]  0.0000000  0.0000000  0.0000000  1.7919481 -0.6085997 -0.9921584
#> [5,]  0.0000000  0.0000000  0.0000000 -0.6085997  1.6043776 -0.7533906
#> [6,]  0.0000000  0.0000000  0.0000000 -0.9921584 -0.7533906  2.0730012
#> [7,]  0.0000000  0.0000000  0.0000000 -0.1911900 -0.2423873 -0.3274522
#>             [,7]
#> [1,]  0.0000000
#> [2,]  0.0000000
#> [3,]  0.0000000
#> [4,] -0.1911900
#> [5,] -0.2423873
#> [6,] -0.3274522
#> [7,]  0.7610295
res$Lw
#>             [,1]        [,2]       [,3]       [,4]       [,5]        [,6]
#> [1,]  1.0125573 -0.79742294 -0.2151344  0.0000000  0.0000000  0.00000000
#> [2,] -0.7974229  1.38586951 -0.5584535  0.0000000  0.0000000 -0.02999311
#> [3,] -0.2151344 -0.55845346  0.7735878  0.0000000  0.0000000  0.00000000
#> [4,]  0.0000000  0.00000000  0.0000000  1.9403468 -0.8018717 -0.99840560
#> [5,]  0.0000000  0.00000000  0.0000000 -0.8018717  1.7435502 -0.72816976
#> [6,]  0.0000000 -0.02999311  0.0000000 -0.9984056 -0.7281698  2.05285499
#> [7,]  0.0000000  0.00000000  0.0000000 -0.1400695 -0.2135088 -0.29628652
#>             [,7]
#> [1,]  0.0000000
#> [2,]  0.0000000
#> [3,]  0.0000000
```

```
#> [4,] -0.1400695
#> [5,] -0.2135088
#> [6,] -0.2962865
#> [7,]  0.6498648
```

```
N_iter <- length(res$obj_fun)
plot(c(1:N_iter), res$obj_fun, type = "b", pch=19, cex=.6, col = scales::alpha("black", .5),
     xlab = "Iteration number", ylab = "Objective function")
```

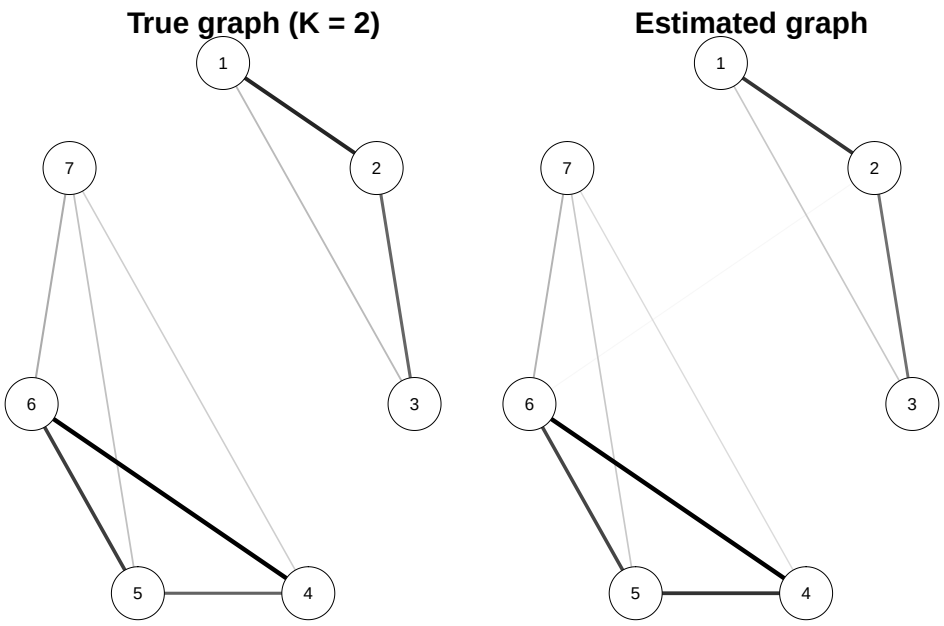



```
plot(c(1:N_iter), res$loglike, type = "b", pch=19, cex=.6, col = scales::alpha("black", .5),
     xlab = "Iteration number", ylab = "Negative Log Likelihood")
```




```
W <- Lw - diag(diag(Lw))
library(qgraph)
par(mfrow = c(1, 2))
qgraph(W, esize = 5, edge.color = "black")
title("True graph (K = 2)", line = 2.5)
```

```
qgraph(res$W, esize = 5, edge.color = "black")
title("Estimated graph", line = 2.5)
```



4 Explanation of the algorithms

In this section we present the algorithms designed to solve the graph topology learning problem.

4.1 `learnGraphTopology`: Learning the topology of graph

The goal of `learnGraphTopology` is to estimate the Laplacian matrix generated by the weight vector of a graph, $\mathbf{w}$. The algorithm for the function `learnGraphTopology` is stated as follows:

Data: $\mathbf{Y}$ (data matrix), K (#{components}), β (regularization term), $\mathbf{w}^{(0)}$, $\boldsymbol{\lambda}^{(0)}$, $\mathbf{U}^{(0)}$ (initial parameter estimates), α_1, α_2 (lower and upper bound on the eigenvalues of the Laplacian matrix), ρ (how much to increase beta per iteration).
Result: $\boldsymbol{\Theta}$ (Laplacian matrix)
$N \leftarrow \texttt{ncol}(\mathbf{Y})$
while *objective function do not converged* ***or*** *max #{iterations} not reached* **do**
  $k \leftarrow 0$
  while *parameters do not converged* ***or*** *max #{iterations} not reached* **do**
    $\mathbf{w}^{(k+1)} \leftarrow \texttt{w\_update}(\mathbf{w}^{(k)}, \mathbf{U}^{(k)}, \boldsymbol{\lambda}^{(k)}, \beta, N, \mathbf{K})$
    $\mathbf{U}^{(k+1)} \leftarrow \texttt{U\_update}(\mathbf{w}^{(k+1)}, N)$
    $\boldsymbol{\lambda}^{(k+1)} \leftarrow \texttt{lambda\_update}(\mathbf{w}^{(k+1)}, \mathbf{U}^{(k+1)}, \alpha_1, \alpha_2, \beta, N, K)$
    $k \leftarrow k + 1$
  end
  $\beta \leftarrow \beta(\rho + 1)$
end
return $\mathcal{L}(\mathbf{w}^{(k+1)})$

Function `w_update`$(\mathbf{w}, \mathbf{U}, \boldsymbol{\lambda}, \beta, N, \mathbf{K})$**:**
  $\nabla_{\mathbf{w}} f \leftarrow \mathcal{L}^{\star}\left(\mathcal{L}(\mathbf{w}) - \mathbf{U}\texttt{diag}(\boldsymbol{\lambda})\mathbf{U}^T + \dfrac{\mathbf{K}}{\beta}\right)$
  return $\texttt{max}\left(0, \mathbf{w} - \dfrac{\nabla_{\mathbf{w}} f}{2N}\right)$

Function `U_update`$(\mathbf{w}, N, K)$**:**
  return $\texttt{eigenvectors}(\mathcal{L}(\mathbf{w}))$[K+1:N] # *increasing order w.r.t. eigenvalues*

Function `lambda_update`$(\mathbf{w}, \mathbf{U}, \alpha_1, \alpha_2, \beta, N, K)$**:**
  $\mathbf{d} \leftarrow \texttt{diag}\left(\mathbf{U}^T\mathcal{L}(\mathbf{w})\mathbf{U}\right)$
  $\boldsymbol{\lambda} \leftarrow \frac{1}{2}\left(\mathbf{d} + \sqrt{\mathbf{d} \odot \mathbf{d} + \frac{4}{\beta}}\right)$ # $\odot$ *means element-wise multiplication*
  if $\boldsymbol{\lambda}$ *has its elements in nondecreasing order* ***and*** $\min(\boldsymbol{\lambda}) \geq \alpha_1$ ***and*** $\max(\boldsymbol{\lambda}) \leq \alpha_2$ **then**
    return $\boldsymbol{\lambda}$
  else
    set to α_1 the elements of $\boldsymbol{\lambda}$ whose values are less than α_1
    set to α_2 the elements of $\boldsymbol{\lambda}$ whose values are greater than α_2
  end
  if $\boldsymbol{\lambda}$ *has its elements in nondecreasing order* **then**
    return $\boldsymbol{\lambda}$
  else
    raise `Exception`(*"eigenvalues are not in increasing order"*)
  end

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.