

Learning the topology of graphs

Convex Group-HKUST
2018-08-16

Contents

1	Usage of the package	1
2	Explanation of the algorithms	3
2.1	learnGraphTopology: Learning the topology of graph	4
	References	4

1 Usage of the package

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

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

# Number of samples
T <- 10000
# Vector to generate the Laplacian matrix of the graph
w <- runif(10)
# Laplacian matrix
Theta <- L(w)
# Sample data from a Multivariate Gaussian
N <- ncol(Theta)
Y <- MASS::mvrnorm(T, rep(0, N), MASS::ginv(Theta))
# 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 as follows::

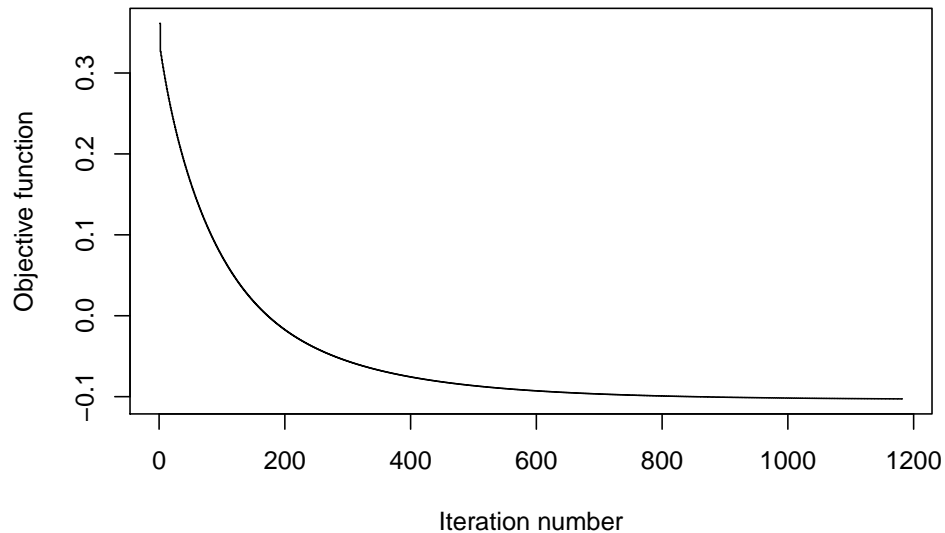
```
Theta
#>      [,1]      [,2]      [,3]      [,4]      [,5]
#> [1,]  2.3678770 -0.2875775 -0.7883051 -0.4089769 -0.8830174
#> [2,] -0.2875775  1.8017068 -0.9404673 -0.0455565 -0.5281055
#> [3,] -0.7883051 -0.9404673  3.1726265 -0.8924190 -0.5514350
#> [4,] -0.4089769 -0.0455565 -0.8924190  1.8035672 -0.4566147
#> [5,] -0.8830174 -0.5281055 -0.5514350 -0.4566147  2.4191726
res$Theta
#>      [,1]      [,2]      [,3]      [,4]      [,5]
#> [1,]  2.3384299 -0.29372890 -0.7864202 -0.38899191 -0.8692889
#> [2,] -0.2937289  1.76412935 -0.9297244 -0.05376403 -0.4869120
#> [3,] -0.7864202 -0.92972444  3.0917157 -0.81448289 -0.5610881
#> [4,] -0.3889919 -0.05376403 -0.8144829  1.74104811 -0.4838093
#> [5,] -0.8692889 -0.48691199 -0.5610881 -0.48380928  2.4010983
```

We can evaluate the performance of the learning process in a more objective manner by computing the relative error between the true Laplacian matrix and the estimated one, which can be done as follows::

```
RE <- norm(Theta - res$Theta, type="F") / max(1., norm(Theta, type="F"))
RE
#> [1] 0.02965113
```

Let's also look at the trend of the objective function per iteration:

```
k <- length(res$fun)
plot(c(1:k), res$fun, type = "s", xlab = "Iteration number",
     ylab = "Objective function")
```



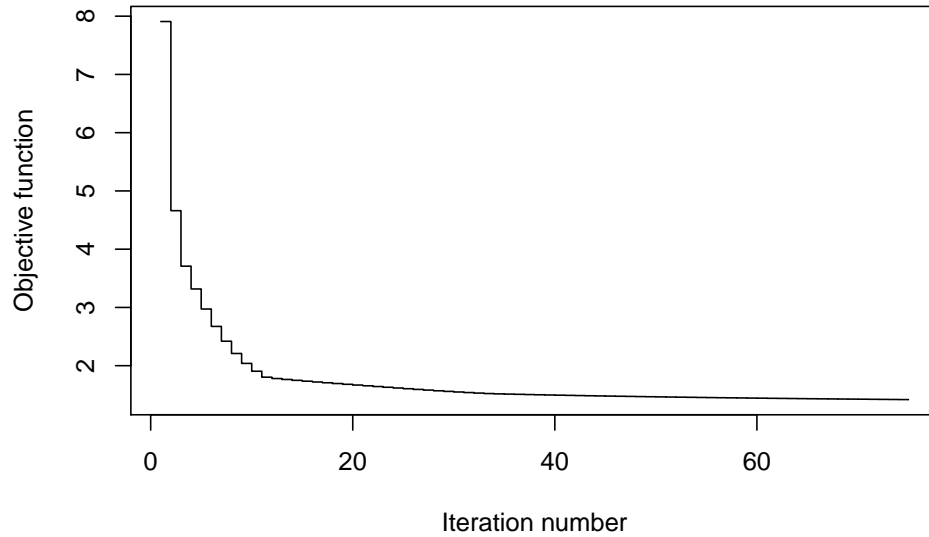
For $K > 1$, we can generate the Laplacian as a block diagonal matrix, as follows

```
library(spectralGraphTopology)
w1 <- runif(3)
w2 <- runif(3)
Theta1 <- L(w1)
Theta2 <- L(w2)
N1 <- ncol(Theta1)
N2 <- ncol(Theta2)
Theta <- rbind(cbind(Theta1, matrix(0, N1, N2)),
               cbind(matrix(0, N2, N1), Theta2))
Y <- MASS::mvrnorm(T, rep(0, N1 + N2), MASS::ginv(Theta))
K <- 2
res <- learnGraphTopology(Y, K, beta = 5, ftol = 1e-3)
RE <- norm(Theta - res$Theta, type="F") / max(1., norm(Theta, type="F"))
RE
#> [1] 0.8831134
```

```
Theta
#>           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
#> [1,]  0.96494812 -0.03715792 -0.9277902  0.0000000  0.0000000  0.0000000
#> [2,] -0.03715792  0.48514753 -0.4479896  0.0000000  0.0000000  0.0000000
#> [3,] -0.92779020 -0.44798960  1.3757798  0.0000000  0.0000000  0.0000000
```

```
#> [4,] 0.00000000 0.00000000 0.00000000 1.2716541 -0.9159023 -0.3557518
#> [5,] 0.00000000 0.00000000 0.00000000 -0.9159023 1.8312150 -0.9153127
#> [6,] 0.00000000 0.00000000 0.00000000 -0.3557518 -0.9153127 1.2710645
res$Theta
#>      [,1]      [,2]      [,3]      [,4] [,5]      [,6]
#> [1,] 1.3883562 0.0000000 -0.3594789 -0.6023027 0 -0.4265746
#> [2,] 0.0000000 0.7419500 0.0000000 -0.3003455 0 -0.4416045
#> [3,] -0.3594789 0.0000000 1.6906926 -0.5861820 0 -0.7450317
#> [4,] -0.6023027 -0.3003455 -0.5861820 1.6022717 0 -0.1134415
#> [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0 0.0000000
#> [6,] -0.4265746 -0.4416045 -0.7450317 -0.1134415 0 1.7266523
```

```
k <- length(res$fun)
plot(c(1:k), res$fun, type = "s", xlab = "Iteration number",
     ylab = "Objective function")
```



2 Explanation of the algorithms

In this section we describe in detail the algorithms designed to solve the graph topology learning problem.

2.1 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 ($\#\{\text{components}\}$), 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)

$N \leftarrow \text{ncol}(\mathbf{Y})$

while *objective function do not converged or max $\#\{\text{iterations}\}$ not reached* **do**

$k \leftarrow 0$

while *parameters do not converged or max $\#\{\text{iterations}\}$ not reached* **do**

$\mathbf{w}^{(k+1)} \leftarrow \text{w_update}(\mathbf{w}^{(k)}, \mathbf{U}^{(k)}, \boldsymbol{\lambda}^{(k)}, \beta, N, \mathbf{K})$

$\mathbf{U}^{(k+1)} \leftarrow \text{U_update}(\mathbf{w}^{(k+1)}, N)$

$\boldsymbol{\lambda}^{(k+1)} \leftarrow \text{lambda_update}(\mathbf{w}^{(k+1)}, \mathbf{U}^{(k+1)}, \alpha_1, \alpha_2, \beta, N, K)$

$k \leftarrow k + 1$

end

$\beta \leftarrow \beta(\rho + 1)$

end

return $\mathcal{L}(\mathbf{w}^{(k+1)})$

Function `w_update(w, U, λ, β, N, K):`

$\nabla_{\mathbf{w}} f \leftarrow \mathcal{L}^* \left(\mathcal{L}(\mathbf{w}) - \mathbf{U} \text{diag}(\boldsymbol{\lambda}) \mathbf{U}^T + \frac{\mathbf{K}}{\beta} \right)$

return $\max \left(0, \mathbf{w} - \frac{\nabla_{\mathbf{w}} f}{2N} \right)$

Function `U_update(w, N):`

return `eigen(L(w))$vectors[, N : 1]`

Function `U_update(w, U, α1, α2, β, N, K):`

$\mathbf{d} \leftarrow \text{diag}(\mathbf{U}^T \mathcal{L}(\mathbf{w}) \mathbf{U})$

$\boldsymbol{\lambda} \leftarrow \frac{1}{2} \left(\mathbf{d} + \sqrt{\mathbf{d} \odot \mathbf{d} + \frac{4}{\beta}} \right)$

if $\boldsymbol{\lambda}$ *has its elements in increasing order* **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 increasing order* **then**

return $\boldsymbol{\lambda}$

else

raise `Exception("eigenvalues are not in increasing order")`

end

References