Computing sparse eigenvectors

Konstantinos Benidis and Daniel P. Palomar

2017-11-21

Contents

This vignette illustrates the computation of sparse eigenvectors or sparse PCA with the package `sparseEigen` (with a comparison with other packages) and gives a description of the algorithms used.

1 Comparison with other packages

hulsdhiuvhsdal

2 Usage of the package

2.1 Computation of sparse eigenvectors of a given matrix

We start by loading the package and generating synthetic data with sparse eigenvectors:

```
library(sparseEigen)
set.seed(42)

# parameters
m <- 500  # dimension
n <- 100  # number of samples
q <- 3  # number of sparse eigenvectors to be estimated
sp_card <- 0.2*m  # cardinality of each sparse eigenvector
rho <- 0.6  # sparsity level

# generate non-overlapping sparse eigenvectors
V <- matrix(rnorm(m^2), ncol = m)
tmp <- matrix(0, m, q)
for (i in 1:max(q, 2)) {
  ind1 <- (i - 1)*sp_card + 1
  ind2 <- i*sp_card
```

```
  tmp[ind1:ind2, i] = 1/sqrt(sp_card)
  V[, i] <- tmp[, i]
}
# keep first q eigenvectors the same (already orthogonal) and orthogonalize the rest
V <- qr.Q(qr(V))

# generate eigenvalues
lmd <- rep(1, m)
lmd[1:q] <- 100*seq(from = q, to = 1)

# generate covariance matrix from sparse eigenvectors and eigenvalues
R <- V %*% diag(lmd) %*% t(V)

# generate data matrix from a zero-mean multivariate Gaussian distribution
# with the constructed covariance
X <- MASS::mvrnorm(n, rep(0, m), R)  # random data with underlying sparse structure
```

Then, we estimate the covariance matrix with `cov(X)` and compute its sparse eigenvectors with `spEigen()`:

```
# computation of sparse eigenvectors
res_standard <- eigen(cov(X))
res_sparse1 <- spEigen(cov(X), q, rho)
res_sparse2 <- spEigen(X, q, rho, data = TRUE)
```

We can assess how good the estimated eigenvectors are by computing the inner product with the original eigenvectors (the closer to 1 the better):

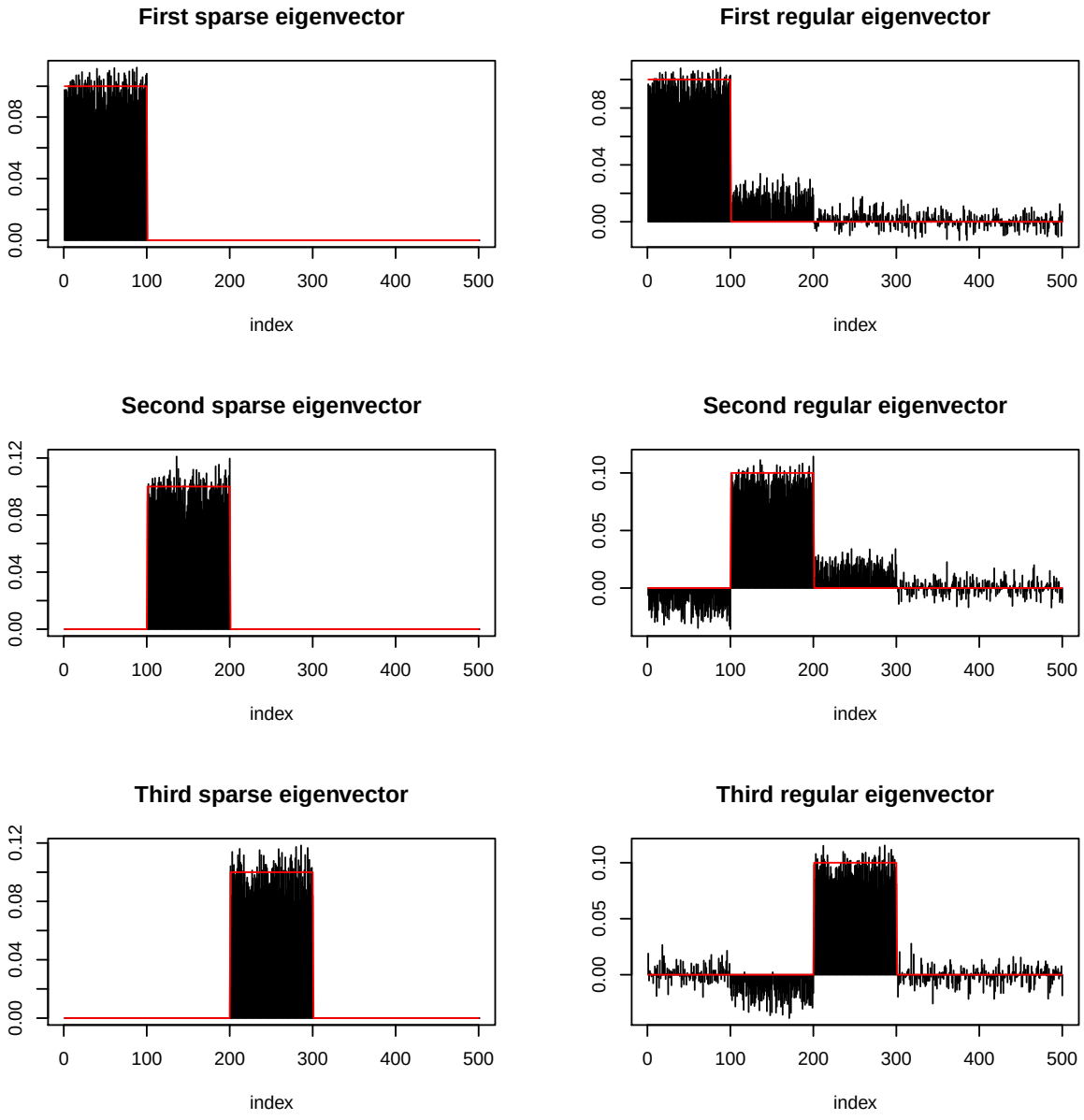
```
# show inner product between estimated eigenvectors and originals
abs(diag(t(res_standard$vectors) %*% V[, 1:q]))   #for standard estimated eigenvectors
#> [1] 0.9726306 0.9488030 0.9623054
abs(diag(t(res_sparse1$vectors) %*% V[, 1:q]))    #for sparse estimated eigenvectors
#> [1] 0.9979083 0.9968489 0.9956549
abs(diag(t(res_sparse2$vectors) %*% V[, 1:q]))    #for sparse estimated eigenvectors
#> [1] 0.9979039 0.9968422 0.9956458
```

Finally, the following plot shows the sparsity pattern of the eigenvectors (sparse computation vs. classical computation):

```
par(mfcol = c(3, 2))
plot(res_sparse1$vectors[, 1]*sign(res_sparse1$vectors[1, 1]),
     main = "First sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 1]*sign(V[1, 1]), col = "red")
plot(res_sparse1$vectors[, 2]*sign(res_sparse1$vectors[sp_card+1, 2]),
     main = "Second sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 2]*sign(V[sp_card+1, 2]), col = "red")
plot(res_sparse1$vectors[, 3]*sign(res_sparse1$vectors[2*sp_card+1, 3]),
     main = "Third sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 3]*sign(V[2*sp_card+1, 3]), col = "red")

plot(res_standard$vectors[, 1]*sign(res_standard$vectors[1, 1]),
     main = "First regular eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 1]*sign(V[1, 1]), col = "red")
plot(res_standard$vectors[, 2]*sign(res_standard$vectors[sp_card+1, 2]),
     main = "Second regular eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 2]*sign(V[sp_card+1, 2]), col = "red")
plot(res_standard$vectors[, 3]*sign(res_standard$vectors[2*sp_card+1, 3]),
```

```
     main = "Third regular eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 3]*sign(V[2*sp_card+1, 3]), col = "red")
```



2.2 Covariance matrix estimation with sparse eigenvectors

The function `spEigenCov()` requires a full-rank covariance matrix. Therefore, we generate data as previously with the only difference that we set the number of samples to be `600`.

Then, we compute the covariance matrix through the joint estimation of sparse eigenvectors and eigenvalues:

```
# computation of covariance matrix
res_sparseCov <- spEigenCov(cov(X), q, rho)
```

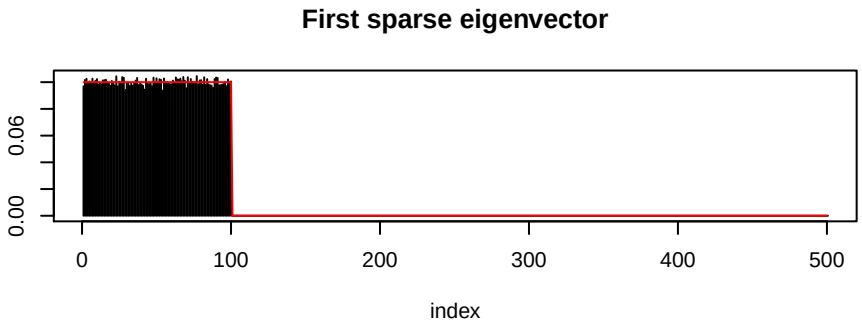
Again, we can assess how good the estimated eigenvectors are by computing the inner product with the

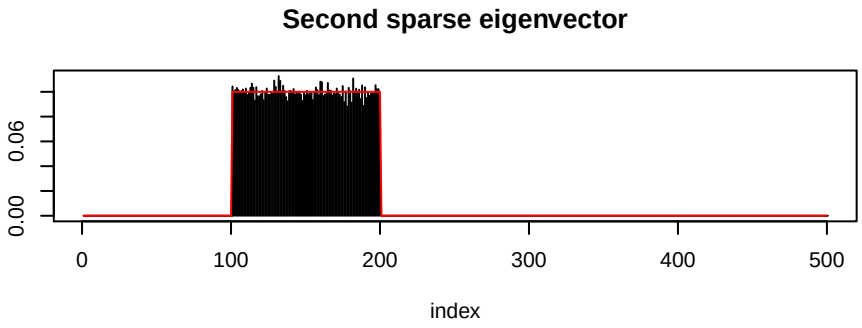
original eigenvectors:

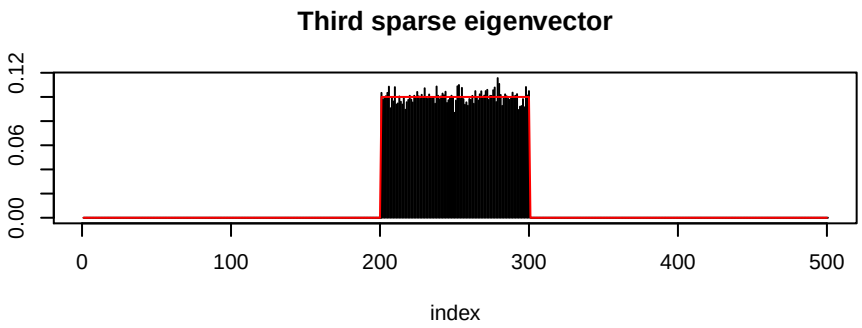
```
# show inner product between estimated eigenvectors and originals
abs(diag(t(res_sparseCov$vectors[, 1:q]) %*% V[, 1:q]))    #for sparse estimated eigenvectors
#> [1] 0.9997218 0.9990758 0.9987594
```

The following plot shows the sparsity pattern of the eigenvectors:

```
par(mfcol = c(3, 1))
plot(res_sparseCov$vectors[, 1]*sign(res_sparseCov$vectors[1, 1]),
     main = "First sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 1]*sign(V[1, 1]), col = "red")
plot(res_sparseCov$vectors[, 2]*sign(res_sparseCov$vectors[sp_card+1, 2]),
     main = "Second sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 2]*sign(V[sp_card+1, 2]), col = "red")
plot(res_sparseCov$vectors[, 3]*sign(res_sparseCov$vectors[2*sp_card+1, 3]),
     main = "Third sparse eigenvector", xlab = "index", ylab = "", type = "h")
lines(V[, 3]*sign(V[2*sp_card+1, 3]), col = "red")
```







Finally, we can compute the RMSE of the estimated covariance matrix (sparse eigenvector computation vs. classical computation):

```
# show RMSE between estimated and true covariance
norm(cov(X) - R, type = 'F') #for sample covariance matrix
#> [1] 42.60926
norm(res_sparseCov$cov - R, type = 'F') #for covariance with sparse eigenvectors
#> [1] 31.72997
```

3 Explanation of the algorithms

3.1 `spEigen()`: Sparse eigenvectors from a given covariance matrix

The goal of `spEigen()` is the estimation of the q leading sparse eigenvectors (with $q \leq \text{rank}(\mathbf{S})$) from an $m \times m$ covariance matrix $\mathbf{S}$ (typically the sample covariance matrix obtained from n samples) based on [1]. The underlying optimization problem that is solved is

$$\begin{aligned} \underset{\mathbf{U}}{\text{maximize}} \quad & \text{Tr}\left(\mathbf{U}^\top \mathbf{S}\mathbf{U}\text{Diag}(\mathbf{d})\right) - \sum_{i=1}^{q} \rho_i \|\mathbf{u}_i\|_0 \\ \text{subject to} \quad & \mathbf{U}^\top \mathbf{U} = \mathbf{I}_q, \end{aligned}$$

where $\mathbf{U} \in \mathbb{R}^{m \times q}$ is a matrix containing the q leading eigenvectors, $\mathbf{d}$ is a vector of weights to ensure that $\mathbf{U}$ contains the leading eigenvectors without an arbitrary rotation, and the ρ_i's are the regularization parameters to control how much sparsity is desired. This problem is the typical PCA formulation with an extra penalty term in the objective that penalizes the cardinality of the eigenvectors, controled by the regularization parameters ρ_i's.

The ℓ_0-"norm" is approximated by the continuous and differentiable function

$$g_p^\epsilon(x) = \begin{cases} \frac{x^2}{2\epsilon(p+\epsilon)\log(1+1/p)}, & |x| \leq \epsilon, \\ \frac{\log\left(\frac{p+|x|}{p+\epsilon}\right) + \frac{\epsilon}{2(p+\epsilon)}}{\log(1+1/p)}, & |x| > \epsilon, \end{cases}$$

where $p > 0$ and $0 < \epsilon \ll 1$ are parameters that control the approximation. This leads to the following approximate problem:

$$\begin{aligned} \underset{\mathbf{U}}{\text{maximize}} \quad & \text{Tr}\left(\mathbf{U}^\top \mathbf{S}\mathbf{U}\text{Diag}(\mathbf{d})\right) - \sum_{j=1}^{q} \rho_j \sum_{i=1}^{m} g_p^\epsilon(u_{ij}) \\ \text{subject to} \quad & \mathbf{U}^\top \mathbf{U} = \mathbf{I}_q. \end{aligned}$$

This problem can be solved via Majorization-Minimization (MM) [2] with an iterative closed-form update algorithm. For this, at each iteration (denoted by k) two key quantities are needed:

$$\mathbf{G}^{(k)} = \mathbf{S}\mathbf{U}^{(k)}\text{Diag}(\mathbf{d})$$

$$\mathbf{H}^{(k)} = \left[\text{diag}\left(\mathbf{w}^{(k)} - \mathbf{w}_{\max}^{(k)} \otimes \mathbf{1}_m\right)\tilde{\mathbf{u}}^{(k)}\right]_{m \times q},$$

where

$$w_i^{(k)} = \begin{cases} \frac{\rho_i}{2\epsilon(p+\epsilon)\log(1+1/p)}, & |\tilde{u}_i^{(k)}| \leq \epsilon, \\ \frac{\rho_i}{2\log(1+1/p)|\tilde{u}_i^{(k)}|\left(|\tilde{u}_i^{(k)}|+p\right)}, & |\tilde{u}_i^{(k)}| > \epsilon, \end{cases}$$

with $\mathbf{w} \in \mathbb{R}_+^{mq}$, $\tilde{\mathbf{u}}^{(k)} = \text{vec}(\mathbf{U}^{(k)}) \in \mathbb{R}_+^{mq}$, $\mathbf{w}_{\max} \in \mathbb{R}_+^q$, with $w_{\max,i}$ being the maximum weight that corresponds to the i-th eigenvector $\mathbf{u}_i^{(k)}$.

The iterative closed-form update algorithm is:

1. Set $k = 0$ and choose an initial point $\mathbf{U}^{(0)}$
2. Compute $\mathbf{G}^{(k)}$ and $\mathbf{H}^{(k)}$
3. Compute $\mathbf{V}_\text{L}$, $\mathbf{V}_\text{R}$ as the left and right singular vectors of $\left(\mathbf{G}^{(k)} - \mathbf{H}^{(k)}\right)$
4. $\mathbf{U}^{(k+1)} \leftarrow \mathbf{V}_\text{L}\mathbf{V}_\text{R}^\top$
5. $k \leftarrow k + 1$
6. Repeat steps 2-5 until convergence
7. Return $\mathbf{U}^{(k)}$

The initial point of the algorithm $\mathbf{U}^{(0)}$ is set by default to the q leading standard eigenvectors, unless the user specifies otherwise. Internally, all the computations of $\mathbf{G}^{(k)}$ and $\mathbf{H}^{(k)}$ are done through the eigenvalue decomposition (EVD) of $\mathbf{S}$. Since we can also retrieve the eigenvectors and eigenvalues of $\mathbf{S}$ through the singular value decomposition (SVD) of the data matrix $\mathbf{X}$, with $\mathbf{S} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}$, it becomes possible to use as an input to 'spEigen()' either the covariance matrix $\mathbf{S}$ or directly the data matrix $\mathbf{X}$.

Although $\mathbf{H}^{(k)}$ does not depend directly on $\mathbf{S}$, the parameters ρ_j are set based on its eigenvalues. In particular, each ρ_j takes a value in an interval $[0, \rho_j^{\max}]$ based on the input variable $\rho \in [0, 1]$ that the user selects, i.e., $\rho_j = \rho\rho_j^{\max}$. The uppperbound $\rho_j^{\max}$ depends, among others, on the eigenvalues of $\mathbf{S}$. Note that the theoretical upperbound is derived based on the initial problem and not the approximate. Therefore, although a suggested range for ρ is the interval $[0, 1]$, any nonnegative value is accepted by the algorithm.

3.2 `spEigenCov()`: Covariance matrix estimation with sparse eigenvectors

The function `spEigenCov()` estimates a covariance matrix through the joint estimation of its sparse (orthogonal) eigenvectors and eigenvalues [1], i.e., $\mathbf{\Sigma} = \mathbf{U}\mathbf{\Xi}\mathbf{U}^\top$, with $\mathbf{U}, \mathbf{\Xi} \in \mathbb{R}^{m\times m}$ and $\mathbf{\Xi} = \text{Diag}(\xi)$. The underlying optimization problem that is solved is

$$\begin{aligned}
\underset{\mathbf{U},\mathbf{\Xi}}{\text{minimize}} \quad & \log\det\left(\mathbf{\Xi}\right) + \text{Tr}\left(\mathbf{S}\mathbf{U}\mathbf{\Xi}^{-1}\mathbf{U}^\top\right) + \sum_{i=1}^{m}\rho_i\|\mathbf{u}_i\|_0 \\
\text{subject to} \quad & \mathbf{\Xi} \succcurlyeq 0, \\
& \xi_i \geq \xi_{i+1}, \quad i = 1, \ldots, q-1, \\
& \xi_q \geq \xi_{q+i}, \quad i = 1, \ldots, m-q, \\
& \mathbf{U}^\top\mathbf{U} = \mathbf{I}_m,
\end{aligned}$$

where $\mathbf{S} \in \mathbb{R}^{m\times m}$ is the sample covariance matrix and q is the number of eigenvectors we impose sparsity on (i.e., $\rho_i = 0$ for $i > q$). The constraints ensure that the eigenvaleus will be positive, while the first q of them will be the largest and in descending order. This is important since in case of a swap of the eigenvalues during the estimation we would impose sparsity on different eigenvectors at each iteration of the algorithm. Finally, the last constraint ensures the orthogonality of the eigenvectors.

Again, the ℓ_0-"norm" is approximated by the continuous and differentiable function $g_p^\epsilon()$ which leads to

the following approximate problem:

$$\begin{aligned}
\underset{\mathbf{U},\boldsymbol{\Xi}}{\text{minimize}} \quad & \log\det\left(\boldsymbol{\Xi}\right) + \mathrm{Tr}\left(\mathbf{S}\mathbf{U}\boldsymbol{\Xi}^{-1}\mathbf{U}^{\top}\right) + \sum_{j=1}^{m}\rho_j \sum_{i=1}^{m} g_p^{\epsilon}\left(u_{ij}\right) \\
\text{subject to} \quad & \boldsymbol{\Xi} \succcurlyeq 0, \\
& \xi_i \geq \xi_{i+1}, \quad i = 1, \ldots, q-1, \\
& \xi_q \geq \xi_{q+i}, \quad i = 1, \ldots, m-q, \\
& \mathbf{U}^{\top}\mathbf{U} = \mathbf{I}_m,
\end{aligned}$$

This problem can be solved via Majorization-Minimization (MM) [2] with an iterative semi-closed-form update algorithm. In particular, with a proper majorization, the eigenvector and eigenvalue estimation decouples. Therefore, in each iteration we need to solve the following two problems:

- Eigenvector optimization:

$$\begin{aligned}
\underset{\mathbf{U}}{\text{minimize}} \quad & \mathrm{Tr}\left(\mathbf{H}^{(k)^{\top}}\mathbf{U}\right) \\
\text{subject to} \quad & \mathbf{U}^{\top}\mathbf{U} = \mathbf{I}_m,
\end{aligned}$$

where $\mathbf{H}^{(k)} = \left[\mathrm{diag}\left(\mathbf{w}^{(k)} - \mathbf{w}_{\max}^{(k)} \otimes \mathbf{1}_m\right)\tilde{\mathbf{u}}^{(k)}\right]_{m\times m} + \left(\mathbf{S} - \lambda_{\max}^{(\mathbf{S})}\mathbf{I}_m\right)\mathbf{U}^{(k)}\left(\boldsymbol{\Xi}^{(k)}\right)^{-1}$. Again the vector $\mathbf{w}$ is given by

$$w_i^{(k)} = \begin{cases} \frac{\rho_i}{2\epsilon(p+\epsilon)\log(1+1/p)}, & |\tilde{u}_i^{(k)}| \leq \epsilon, \\ \frac{\rho_i}{2\log(1+1/p)|\tilde{u}_i^{(k)}|\left(|\tilde{u}_i^{(k)}|+p\right)}, & |\tilde{u}_i^{(k)}| > \epsilon, \end{cases}$$

with $\mathbf{w} \in \mathbb{R}_+^{m^2}$, $\tilde{\mathbf{u}}^{(k)} = \mathrm{vec}(\mathbf{U}^{(k)}) \in \mathbb{R}_+^{m^2}$, $\mathbf{w}_{\max} \in \mathbb{R}_+^m$, with $w_{\max,i}$ being the maximum weight that corresponds to the i-th eigenvector $\mathbf{u}_i^{(k)}$.

The optimal solution of this problem is $\mathbf{U} = \mathbf{V}_L\mathbf{V}_R^{\top}$ where $\mathbf{V}_L, \mathbf{V}_R$ are the lest and right singular vectors of $\mathbf{H}^{(k)}$, repsectively.

- Eigenvalue optimization:

$$\begin{aligned}
\underset{\xi}{\text{minimize}} \quad & \sum_{i=1}^{m}\left(\log\xi_i + \alpha_i^{(k)}\xi_i + \lambda_{\max}^{(\mathbf{S})}\frac{1}{\xi_i}\right) \\
\text{subject to} \quad & \xi_i \geq \xi_{i+1}, \quad i = 1, \ldots, q-1, \\
& \xi_q \geq \xi_{q+i}, \quad i = 1, \ldots, m-q,
\end{aligned}$$

where $\alpha^{(k)} = \mathrm{diag}\left(\left(\boldsymbol{\Xi}^{(k)}\right)^{-1}\mathbf{U}^{(k)^T}\left(\mathbf{S} - \lambda_{\max}^{(\mathbf{S})}\mathbf{I}_m\right)\mathbf{U}^{(k)}\left(\boldsymbol{\Xi}^{(k)}\right)^{-1}\right)$. This problem is not convex. However, it can be transformed to a convex one by the variable transformation $\phi = 1/\xi$. Solving the KKT equations of the transformed convex formulation we can derive a finite-step algorithm that gives the optimal solution of the problem.

The overall iterative semi-closed-form update algorithm is:

1. Set $k = 0$ and choose an initial point $\mathbf{U}^{(0)}$
2. Compute $\mathbf{H}^{(k)}$
3. Compute $\mathbf{V}_{\mathrm{L}}$, $\mathbf{V}_{\mathrm{R}}$ as the left and right singular vectors of $\mathbf{H}^{(k)}$
4. $\mathbf{U}^{(k+1)} \leftarrow \mathbf{V}_{\mathrm{L}}\mathbf{V}_{\mathrm{R}}^{\top}$

5. Compute $\alpha^{(k)}$

6. Get $\lambda^{(k+1)}$ from the finite-step algorithm

7. $k \leftarrow k + 1$

8. Repeat steps 2-7 until convergence

9. Return $\mathbf{U}^{(k)}$, $\lambda^{(k)}$

References

[1] K. Benidis, Y. Sun, P. Babu, and D. P. Palomar, "Orthogonal sparce PCA and covariance estimation via Procrustes reformulation," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6211–6226, Dec. 2016.

[2] Y. Sun, P. Babu, and D. P. Palomar, "Majorization-minimization algorithms in signal processing, communications, and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 794–816, Feb. 2017.