LEARNING OF STRUCTURED GRAPH DICTIONARIES

This practical session will lead you to reproduce the method proposed by X. Zhang, W. Dong and P. Frossard, in "LEARNING OF STRUCTURED GRAPH DICTIONARIES", article at the Conference IEEE ICASSP, 2012.

1) Read the article attached to this tutorial. Some comments :

- The main idea is, for graph signal processing, to study how to estimate jointly a Dictionary, i.e. a set of functions which provide an overcomplete basis of graph signals (a set whose dimension is larger than the number N of nodes of the graph), and the sparse approximations on this dictionary of a collection of M graph signals that are used as training signals for learning.
- The dictionary is parametrized in the graph spectral domain, by equation (5) (also eq. (6) alternatively) in the article. How to learn the dictionary from data is described in 3.2
- The sparse approximation is done using the Orthogonal Matching Pursuit method, a classical and simple approach for that (the algorithm is recalled underneath) ; see 3.1.
- The main objective is to code Algorithm 1 and explore some examples as in 4.1.

2) Download and install the `Graph Signal Processing Toolbox` :
`https://epfl-lts2.github.io/gspbox-html/`
Reference: Perraudin Nathanaël, Johan Paratte, David Shuman, Lionel Martin, Vassilis Kalofolias, Pierre Vandergheynst and David K. Hammond, *GSPBOX: A toolbox for signal processing on graphs.* Arxiv e-print, 08-2014.
It is available in Matlab (the reference version for this TP).
A Python version is also available here: `https://github.com/epfl-lts2/pygsp`. The names used in the following are from the Matlab version, yet it should be easy to use Python if needed..

- Make yourself familiar with this toolbox (there is a long pdf book explaining how it works, with some examples at the beginning).
- Generate a random graph (for instance with `gsp_sensor`, with a moderate number of nodes $N = 16$ or 32), visualize it with `gsp_plot_graph` and explore its Graph Fourier Transform (see `gsp_compute_fourier_basis`, and `gsp_plot_signal` to display a graph signal, e.g. a Graph Fourier mode).

3) Generate a structured dictionary in the Graph spectral domain, as defined in the article in Section 2.2 and using parameters as in Sec. 4.1 (i.e.: draw random $\mathbf{\Lambda}$'s and compute the associated $\mathbf{D}$; don't forget to normalize the atoms, it will be easier).

- Then, generate sparse signal(s) on this dictionary (i.e, draw $L = \text{round}((1+\sqrt{N})/2)$ coefficients and that same number of atom labels to generate a $\mathbf{x}_m$ (with only L non zero coefficients) and compute the graph signals as $\mathbf{y}_m = \mathbf{D}\mathbf{x}_m$.

4) Code the Orthogonal Matching Pursuit algorithm (As for Section 3.1 and Step 1 of the Algorithm 1): this is a greedy method to compute an estimation of the sparse coefficients $\mathbf{x}_m$ given $\mathbf{y}_m$ and a dictionary $\mathbf{\Phi}_{tot}$ such that $\mathbf{y}_m \approx \mathbf{\Phi}_{tot}\mathbf{x}_m$.
Let $\mathbf{y}$ be your signal. Initialize the residual signal still to be approximated as $\mathbf{R}_0 = \mathbf{y}$. Then iterate, with initial step $j = 1$:

1. Compute the scalar product between $\mathbf{R}$ and each atom (=column) of $\mathbf{\Phi}_{tot}$ (code the algorithm such that a column is a function in the dictionary).
2. Select the atom associated to the maximum absolute scalar product: ϕ_j.
3. Add ϕ_j to the previous atoms selected to make a matrix $\mathbf{\Phi}_j$ with j columns.
4. Compute the orthogonal projector associated to these atoms:

$$\mathbf{P} = \mathbf{\Phi}_j(\mathbf{\Phi}_j^*\mathbf{\Phi}_j)^{-1}\mathbf{\Phi}_j^*$$

5. Apply this projector to the residual signal $\mathbf{R}$ and remove this part:

$$\mathbf{R}_j = \mathbf{R}_{j-1} - \mathbf{P}\mathbf{R}_{j-1}.$$

6. Increase j by 1 and iterate if the stopping criterion is not met.

The stopping criterion can be of various nature: achieve a given precision in the approximation (i.e, a norm of $\mathbf{R}_j$ small enough); or a set number of iterations. Here, the advice is to use L iterations as we will deal with signals that are supposed to be sparse with only L coefficients (and so, only L atoms should be necessary).
Note that the coefficients of the decomposition are then computed as:

$$\mathbf{x} = (\mathbf{\Phi}_j^*\mathbf{\Phi}_j)^{-1}\mathbf{\Phi}_j^*\,\mathbf{y}$$

using the last estimate before stopping the iterations.
Test your algorithm; most of the time, you should find exactly the same atoms and coefficients (and from time to time, a different atom obtained by OMP can be when 2 atoms are too much aligned, i.e. correlated).

5) Code the Dictionary update step, as fully justified in 3.2 of the article and described in a synthetic way as Algorithm 1, Step 2, in the article.
Advice: code it first with known sparse coefficients $\mathbf{X}$ and signals $\mathbf{Y}$ from a prescribed dictionary $\mathbf{D}$, generated as in **3)**. See if you can recover the $\mathbf{D}$ with the written code, and what is the meaning of "recovery" here.

6) Combine together the questions **4)** and **5)** to reproduce Algorithm 1.
Test your implementation of the method on synthesis signals as for some of the situations explored in the article, section 4.1 (it's not expected to reproduce all the numerical explorations).