Master Project title

Final master project

Cristina Gava

July 28, 2018

Contents

Chapter 1

The emerging field of graph signal processing

Nowadays huge amounts of data are collected every day, from engineering sciences to our personal data regarding health monitoring, to financial data or political influences. The analysis and the process of these huge datasets has become a non indifferent challenge, since, in fact, data tends to reside on complex and irregular structures, which progressively required the introduction of innovative approaches to better handle and utilize them. [1] [2]

One of the central entities we will use in this work is the concept of *graph signal*. Graphs are generic data representation structures, which are useful in a great variety of fields and applications for everyday life and technology in general. These structures are composed by two main elements: the nodes and the edges; the former are a set of points, identifying an aspect of the graph structure itself, while the latter are connections between the nodes. Several structures we encounter in natural entities and abstract constructions can be represented by a graph structure, and when we associate values to their set of nodes and edges we obtain a graph signal. To be specific, a graph signal is seen as a finite collection of samples located at each node in the structure and that are interconnected among themselves through the edges. To them we can associate numerical values representing the weights of the connections. The metric for these weights is not unique, as it depends on which relation between the nodes we are looking at: a typical metric for graph weights may be, for example, inversely proportional to the distance (physical or not) between two different nodes, but

other options are possible.

As previously mentioned, graph structures appear to be significantly helpful when they are used to represent signals and their applications are the most varied: we could focus on transportation networks, where we could want to work with data describing human migration patterns or trade goods; we could also apply graph theory over social networks, in which users can be seen as our nodes, for example, while their connections to friends are our edges. [1] Brain imaging is another interesting application of graph signals: through it we could infer the inner structure of some regions of the brain, in a way that permits us to better understand physiological dynamics. [3] We could go on listing a considerable amount of valuable applications, from genetic and biochemical research to fundamental physical experiments, all of them benefitting in large part from the new representation structure offered by a graph signal.

1.1 Mathematical description of the graph structure

In the field of Graph Signal Processing (GSP) spectral graph theory plays an important role, it focuses on analysing, constructing and manipulating graphs and makes use of well know tools and concepts as frequency spectrum and the graph Fourier transform. In our work, the signals we are considering are defined on an undirected, connected, weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, W\}$ consisting on a finite set of vertices $\mathcal{V}$, with $|\mathcal{V}| = N$, a set of edges $\mathcal{E}$ and a weighted adjacency matrix $\mathcal{W}$. The adjacency matrix contains the information of the connections between two nodes: if an edge e is present between two vertices i and j, then the entry $W_{i,j}$ represents the weight of that edge, while if there is no connection between two nodes, the value of the edge is null ($W_{i,j} = 0$). Moreover, if the graph was not connected and had K connected components, then we could decompose the graph signal over $\mathcal{G}$ into M sections which can be processed independently of each other.

Over this structure, we can then spread the signal we need to analyse: a signal or function $f : \mathcal{V} \to \mathbb{R}$ defined on the vertices of our graph can be described as a vector $f \in \mathbb{R}^N$ that has the signal value at the i^{th} vertex in $\mathcal{V}$ as the i^{th} component of the vector f. Figure 1.1 shows an example of graph signal: the colours of the vertices represents the values the signal has on them.

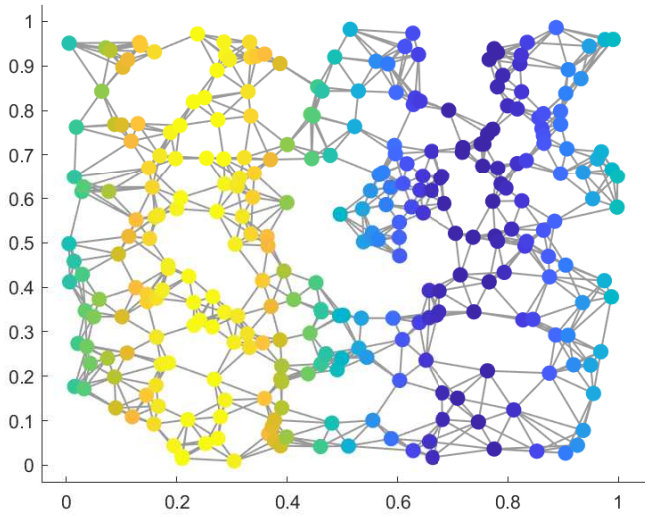


Figure 1.1: Random sensor graph

1.2 The graph Laplacian

To go from the graph vertex domain (which plays the same role of the time domain in the classical signal processing theory) to the spectral domain the *graph Laplacian operator*, or *Combinatorial graph Laplacian*, has been introduced. This operator is defined as $L := D - W$, where D represents a diagonal matrix whose i^{th} diagonal element is equal to the sum of weights of all the edges incident to the vertex i. This entity is a difference operator, since it satisfies the condition:

$$((Lf)(i) = \sum_{j \in \mathcal{N}_i} W_{i,j}[f(i) - f(j)] \tag{1.21}$$

The element $\mathcal{N}_i$ represents the set of vertices connected to vertex i by an edge.

From its definition, the graph Laplacian L is a real symmetric matrix, thus it has a complete set of orthonormal eigenvectors, here denoted by $\{u_l\}_{l=0,1,\ldots,N-1}$, to which real and non-negative eigenvalues are associated ($\{\lambda_l\}_{l=0,1,\ldots,N-1}$). Together with the eigenvectors, they satisfy the condition:

$$Lu_l = \lambda_l u_l, \qquad \text{for } l = 0, 1, \ldots, N-1 \tag{1.22}$$

In the eigenvalues set, the one corresponding to 0 has a multiplicity equal to the number of connected components of the graph; from this, since we are considering only connected

graphs, we can assume the Laplacian eigenvalues have the distribution: $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \cdots \leq \lambda_{N-1} = \lambda_{max}$ and, of course, the set $\sigma(L) = \{\lambda_0, \lambda_1, \ldots, \lambda_{N-1}\}$ is the entire spectrum of our signal.

Eigenvalues and eigenvectors are then used to build up the graph version of a well known and useful transform, which is the *classical Fourier Transform* and that we recall here being defined as:

$$\hat{f}(\xi) = \langle f, e^{2\pi i \xi t} \rangle = \int_{\mathbb{R}} f(t) e^{-2\pi i \xi t} dt. \tag{1.23}$$

We can see the transform as the expansion of a function f in terms of the complex exponentials, elements which represent the eigenfunctions of the one-dimensional Laplace operator:

$$-\Delta(e^{2\pi i \xi t}) = -\frac{\partial^2}{\partial t^2} e^{2\pi i \xi t} = (2\pi\xi)^2 e^{2\pi i \xi t} \tag{1.24}$$

Therefore, from the observation of the classical Fourier Transform, we can analogously define the *Graph Fourier Transform* $\hat{f}$ of any function $f \in \mathbb{R}^N$ on the vertices of $\mathcal{G}$ as the expansion of f in terms of the eigenvectors of the graph Laplacian [3]:

$$\hat{f}(\lambda_l) = \langle f, u_l \rangle = \sum_{i=1}^{N} f(i) u_l * (i) \tag{1.25}$$

while, at the same time, the *inverse Graph Fourier Transform* is given by:

$$f(i) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l) u_l(i) \tag{1.26}$$

Chapter 2

The Graph learning issue

For the most part, the research regarding graph signal processing has been focusing on working with the signals spanned onto a graph manifold, assuming the graph structure to be already known. However, the choice of the underlying graph for the signal does not necessarily represent faithfully the ground truth intrinsic manifold over which the signal is structured. In these cases, a more suitable approach to the problem could be trying to learn the graph structure underneath, such that the processing of the signal can rely on assumptions that better capture the intrinsic relationships between the entities or make them clear when otherwise it would have not been so.[4] Clearly the question is not trivial, since in general the action of learning a graph from sample data represents an ill-posed problem, with the consequence that there may be many solutions to the structure associated to a data. [5] To overcome this obstacle several approaches have been designed in the recent years, and to some of them we will give a more detailed description in the next chapters, when we will address the basis of our work.

Chapter 3

The smoothness assumption

Going further with the spectral analysis, some other intuitions about the graph spectrum can be conveyed from the classical setting to the graph setting.

One of them is that in the classical setting the eigenvalues carry a specific notion of frequency: complex exponentials related to lower frequencies are relatively smooth, slowly oscillating functions whereas for high frequencies the associated complex exponentials oscillate more rapidly. Reporting this onto the graph spectral settings, we see that the Laplacian eigenfunctions associated with lower eigenvalues can be seen as smooth, while the ones related to higher eigenvalues tend to oscillate more intensively. [6] In the specific case of the null eigenvalue the related eigenvector (χ_0) is constant and equal to $\frac{1}{\sqrt{N}}$ [3].

3.1 The smoothness in the graph signal domain

Taking a step back, one of the common simple assumptions about data residing on graphs is that the signal changes smoothly between connected nodes; in this, a simple way to quantify how smooth is a set of vectors residing on a weighted undirected graph is through the function:

$$\frac{1}{2}\sum_{i,j} W_{i,j}||y_i - y_j||^2 \tag{3.11}$$

with W_ij representing the weight between node i and node j, as usual, and y_i and y_j representing two vectors in $y_1, y_2, \ldots, y_N \in \mathbb{R}$. This means that if two vectors y_i and y_j reside on two nodes connected by a high W_{ij}, then they are expected to have a small distance; on

the other hand, if two nodes are considered to have a big distance, then the weight of the edge connecting them should be small.

3.2 Smoothness in the vertex domain

The same concept can be translated into the vertex domain under the form:

$$\sum_{i,j} W_{i,j}||\chi_i - \chi_j||^2 \tag{3.22}$$

Where χ_i and χ_j are the i^{th} and the j^{th} column of the matrix of eigenvectors $\boldsymbol{\chi}$ related to the graph Laplacian L.
Rewriting Equation 3.11 in matrix form, we obtain[7]:

$$\text{tr}(\chi^T L \chi) \tag{3.23}$$

and, recalling that in the Laplacian matrix we can explicit the diagonal matrix containing the associated eigenvalues Λ:

$$L = \chi \Lambda \chi \tag{3.24}$$

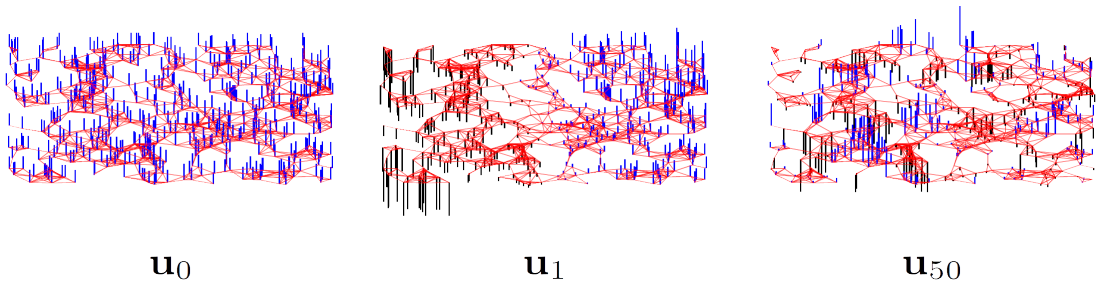
we can thus arrive to the relation between the smoothness and the graph signal spectrum:

$$\text{tr}(\chi^T L \chi) = \text{tr}(\chi^T \chi \Lambda \chi^T \chi) = \text{tr}(\Lambda) \tag{3.25}$$

This result clearly explains the relation between the signal smoothness and the eigenvalues of the Laplacian that we presented at the beginning of the chapter.Figure 3.1 shows different graph Laplacian eigenvectors for a random graph. The eigenvectors associated to larger eigenvalues tend to oscillate more rapidly and to have dissimilar values on vertices connected by a high weight edge, this moreover bringing to higher occurrences of zero crossing associated to the graph Laplacian. [3].

3.3 Smoothness and sparsity

In [7] the previous notion of smoothness is directly connected with the one of graph sparsity: in fact they show that the smoothness term can be seen equivalently as a weighted l^1 norm of the adjacency matrix. Minimizing this matrix can lead to a graph with a sparse set of



$\mathbf{u}_0$ $\mathbf{u}_1$ $\mathbf{u}_{50}$

Figure 3.1: three graph Laplacian eigenvectors for a random sensor network graph. It can be seen how χ_{50} contains definitely more zero crossing than the other two vectors χ_0 and χ_1

edges that prefers only the ones associated to small distances in the *pairwise distances matrix* $\mathbb{Z} \in \mathbb{R}_+^{N \times N}$ defined as $\mathbb{Z}_{i,j} = ||\chi_i - \chi_j||^2$.

This concept is strictly connected with the compressibility property of a signal, since previous studies validate the hypothesis that smooth signals are likely to be compressible in the frequency domain. In [8] it has been shown that the liner approximation error is upper bounded by the overall variation of the signal, this bringing to the conclusion that if the total variation of a signal is small, then the approximation error is small. This resulting in the fact that the signal can be well approximated by a small portion of the Fourier coefficients.

Chapter 4

Representing a signal: graph learning techniques and dictionary representation

In chapter 1 we presented the main reason why signal processing field expressed the necessity of having new different structures in order to better represent the amount of information we can collect from a phenomena: what we did not focused on yet, is the fact that these amount of data are usually largely redundant, since they are a densely sampled version of the signal and they may represent multiple correlated versions of the same physical event. So normally the significative information regarding the underlying processes is largely reducible in dimensionality with respect of the collected dataset. [9] Thus, we can obtain the data representations starting from the idea that our observations can be described by a sparse subset of elementary signals - so called *atoms* - taken from an *overcomplete dictionary*. When the dictionary forms a basis, then every signal can be univocally represented through the linear combination of the dictionary atoms, moreover the overcompleteness property implies that atoms are linearly dependent. [9] [10]

With these premises, we consider a dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_L] \in \mathbb{R}^{N \times L}$, in which the columns represent the dictionary atoms and, for the overcompleteness, $L \geq N$. Through this entity, the signal representations can be done through two main paths, either the *synthesis* path, or the *analysis* path, and the two can significantly differ in the overcomplete case.
In the synthesis path, the signal $\mathbf{x} \in \mathbb{R}^N$ is represented as a linear combination of the

dictionary atoms:

$$\mathbf{x} = \mathcal{D}^T \gamma_s \tag{4.01}$$

while in the analysis path it is represented through its inner product with the atoms:

$$\gamma_a = \mathcal{D}^T \mathbf{x} \tag{4.02}$$

Where $\mathbf{x}$ accounts for the sparsity concept and is called *sparsity matrix*: as the name suggests, this matrix is a sparse matrix having in its columns a number of non-zero elements equal to the sparsity coefficient we impose. The sparsity coefficients indicates the number of atoms in the dictionary concurring to reproduce the signal in the vertex corresponding to the dictionary row, while the positions of these non-zeros elements in $\mathbf{b}$ indicate which are the sources of this generated signal.

The representation in Equation 4.01 has the consequence that, when the dictionary is overcomplete, the set of representations $\gamma_s$satisfying the equation is *infinitely large*, allowing us to look for the most informative representation of the signal with respect of a certain cost function $\mathcal{C}(\gamma)$. In this way we arrive to a first general optimization problem in the form:

$$\gamma_s = \underset{\gamma}{\operatorname{argmin}}\ \mathcal{C}(\gamma) \quad \text{Subject To } \mathbf{x} = \mathcal{D}\gamma \tag{4.03}$$

The way we choose the form of the cost function obviously influences the structure of our solution. One of our goals is to achieve sparsity in the representation of the signal, such that the signal reconstruction is reduced in dimensionality as we are trying to achieve from the beginning. Problem stated in Equation 4.03 becomes what is commonly referred as *sparse coding*, and there are different functions we can apply in order to obtain it: these functions have the characteristic of being tolerant to large coefficients and, at the same time, importantly penalize small non-zero ones. Among these functions, our choice fell on the l^1 norm, which is one of the simplest and most effective functions of this type.

4.1 Choosing the right dictionary

What we did not focused on yet is the choice of the proper dictionary for out task. In the research there has been so far, different models of dictionaries have been defined and used for the most different purposes, in the beginning the attention was mainly on traditional

dictionaries, such as wavelet and Fourier dictionaries, which are simple to use and perform well for 1-dimensional signals. However, these structures were too simple to properly describe more complex and high-dimensional data, so the focus slowly moved to seeking solutions that better performed in this environment. Dictionaries emerged from this need were coming from two main sources:

- As an *analytical model*, which allows to extract the dictionary straight form the data for a fast implicit implementation that does not require multiplication by the dictionary matrix;

- As a *set of realizations* of the data, which offers an increased flexibility and adaptability to data structure;

The first model prefers speed over adaptability, since its success depends on the chosen underlying model, sometimes resulting in a over-simplistic solution for the proposed problem. From this, the necessity to also focus on the second type of dictionary, also defined as *trained dictionary*.
Machine learning techniques of the period between 1980's and 1990's allowed to approach this problem under the new assumption that the structure of a natural phenomena can be accurately extracted *straight form the data* with better results than using a mathematical formulation. The most recent training methods focus on the l^0 and l^1 sparsity measures, which have a simpler formulation and at the same time can use the more recent sparsity coding techniques. [11] [12] Among these learning methods, particular relevance acquired the *parametric training methods*, such as *translation-invariant dictionaries*, *multiscale dictionaries* or *sparse dictionaries*. These implementations involve the reduction of parameters' number and the assumption of several desirable properties on the dictionary, in the end leading to an accelerate convergence, reduced density of the local minima and the convergence to a better solution. Moreover, the generalization of the learning process is improved thanks to the smaller number of parameters, as much as the reduction in number of examples needed. Finally, parametric dictionaries bring to a more efficient implementation, due to the fact that parametrization typically has a more compact representation and, not less important, a parametric dictionary may be designed to represent infinite or arbitrary-sized signals. [10]
(Cri says: forse aggiungi parte sugli sparse dictionaries sempre presa dallo stesso paper. Poi: vedi se aggiungere un excursus storico sui dizionari ad apprendimento).

The sparse approximation

The intention of sparse approximations is to represent a certain signal y of dimension n as the linear combination of a small amount of signals selected from the source database, which is the dictionary. In the dictionary the elements are typically unit norm functions, called *atoms*, that can be denoted through ϕ_k (with $k = 1, \ldots, N$ and being N the size of the dictionary) and span the entire space the signals live in.
When a dictionary is overcomplete, then every signal can be represented with the aforementioned linear combination:

$$\mathbf{y} = \phi\mathbf{a} = \sum_{k=1}^{N} a_k \phi_k \tag{4.14}$$

and the value that $\mathbf{a}$ can assume is not unique.
To achieve sparse and efficient representations, the requirement for finding the exact representation is in general relaxed: starting from the NP-hard problem which is the minimization of the l^0 norm of $\mathbf{a}$, we can approach the issue through convex relaxation methods that solve a problem in the form:

$$\min_{\mathbf{a}} \ (||\mathbf{y} - \phi\mathbf{a}||_2^2 + \lambda||\mathbf{a}||_1) \tag{4.15}$$

in which the relaxation allowed to replace the nonconvex l^0 norm in the original problem with a more meaningful l^1 norm. [9]

Chapter 5

Problem Presentation

Given the premises presented in the previous chapters, our work mainly focused on two open issues regarding both the dictionary learning and the graph learning problem. The starting point were the works of Dorina Thanou - regarding the dictionary learning approach for graph signals - and the work of Hermina Petric Maretic - regarding the graph learning approach.
In the related work, Thanou et al. proposed a learning algorithm which was able to retrieve the kernel functions for a certain dictionary, alternatively optimizing over the kernel coefficients α and the sparsity matrix X, while Maretic et al. started from the complementary assumption that the entities known in their case where exactly the kernel coefficients, while they alternatively optimized over the sparsity matrix and the graph weight matrix.

5.1 State of the art

There has already been a discrete amount of work around graph inference for signals that are expected to be smooth on the graph. In [5] Dong et al. addressed the problem adopting a factor analysis model for graph signals and imposing a Gaussian probabilistic prior on the independent variables which model the signal. Their results showed that smoothness property of a graph signal is favoured by the efficient representation obtained through these priors. In particular, in the algorithm they presented they deployed the use of l^1 and Frobenius norm, the former is expressed by a constraint on the Trace of the Laplacian, and eventually accounts for a sparsity term, while the latter is added as a penalty term in the

objective function in order to control the distribution of the off-diagonal entries in L.

At the same time, in [7] Kalofolias proposes another framework to learn a graph structure under the smoothness assumption. They also use the minimization of the trace term for the Laplacian matrix, clinching that the minimizations of it brings to naturally sparse solutions.

However, this assumption is not always exhaustively descriptive of the real problem we are considering, so we might want to add other reinforcements to the priors in order to better describe the situation. For example, we could imagine a real world network to show a localized behavior or a more organized one having a certain pattern repeating over the graph. In [4], this was an assumption for the signal, bringing it to be a sparse combination of a small number of atoms in a polynomial graph dictionary. This in the end was creating atoms localized in every node on the graph, repeating the same pattern over it with respect to every node as a source, in the same way it is described in [13]. However Maretic's work is limited by the fact that, as we already mentioned, it assumes the kernels to be know functions while it concentrates only on the graph dictionary and thus fixing the spectral behavior of the atoms.

5.2 Problem structure

As previously mentioned, this work deals with the general class of signals that can be represented in a sparse way through atoms of a polynomial dictionary [13], such that it can give a natural segregation of each signal into localised components, as shown in Figure 5.1. The figure shows how each of these components derive from one atom, which is directly dependent of the graph dictionary. This is well explained if, for polynomial kernels of degree k, we see our atoms as an entity spanning only the k-hop distance from the node that represents the center of the atom (source), as we will describe in the next section.

In continuity with the prior work, here we assume the sources of our atoms to be unknown and not necessarily the same for all the different generated parts of the signal. This means, in mathematical terms, assuming that we do not know *a priori* the position, in the columns of the sparsity matrix X, of the non-zero entries that indicate which are the nodes acting as sources for the atoms.

Closely related to the segregation assumption then, is the fact that we could imagine our signal as having a sort of community structure. This means that the graph can be clustered

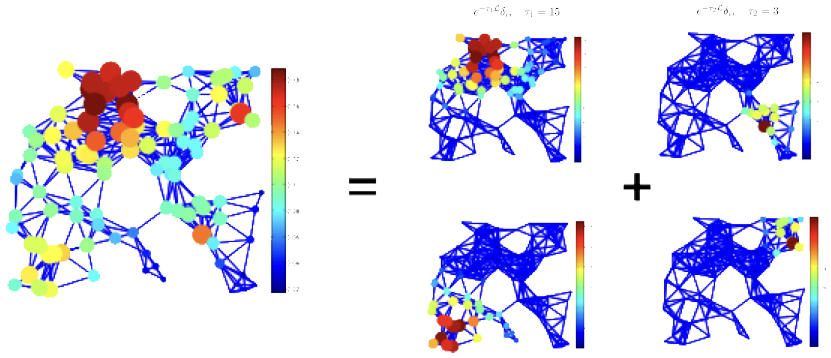


Figure 5.1: Example of signal decomposition using the polynomial dictionary

into communities that are poorly related among themselves when talking about the signal variation, thus imagining very different node signals to be sufficiently distant, and so, being part of separated communities.

5.2.1 The translation operator and the smoothness priors

To better describe the structure of our dictionary, we start recalling that in section 1.2 we introduced the Laplacian operator as $L = D - W$ and here we focus on a modified version of this operator, that is the *normalized Laplacian* $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$. The normalized Laplacian is also a real symmetric and positive semidefinite matrix, with a complete set of orthonormal eigenvectors $\chi = [\chi_0, \chi_1, \ldots, \chi_{N-1}]$ and associated non negative eigenvalues $\sigma(\mathcal{L}) := \{0 = \lambda_0, \lambda_1, \ldots, \lambda_{N-1} \leq 2\}$ (where the upper bound 2 comes from the normalization). The normalized Laplacian eigenvectors are Fourier basis, this bringing any function y defined on the vertices of the graph to be represented in his Fourier transform version $\hat{y}$ at frequency λ_l as:

$$\hat{y}(\lambda_l) = \langle y, \chi_l \rangle = \sum_{n=1}^{N} y(n) \chi_l^*(n) \tag{5.21}$$

and its relative inverse transform to be:

$$y(n) = \sum_{l=0}^{N-1} \hat{y}(\lambda_l) \chi_l(n), \quad \forall n \in \mathcal{V} \tag{5.22}$$

This transform plays a major role not only in the harmonic analysis, but also in defining the signal translation onto a graph, which actually is the main transformation we are interested in when we try to construct a graph dictionary. It is worth to notice here that in the classical signal processing field the translation operator is defined through the change of variable $(T_n f)(t) := f(t-n)$, but when it comes to graph signal processing, this concept looses meaning since there is no significate to $f(\circ - n)$ in the graph environment. However, from transform theory we can recall that the classical translation operator T_n can also be seen as a convolution with a Kronecker delta δ centered in n[3] [13]:

$$T_n g = \sqrt{N}(g * \delta_n) = \sqrt{N} \sum_{l=0}^{N-1} \hat{g}(\lambda_l)\chi_l^*(n)\chi_l \quad . \tag{5.23}$$

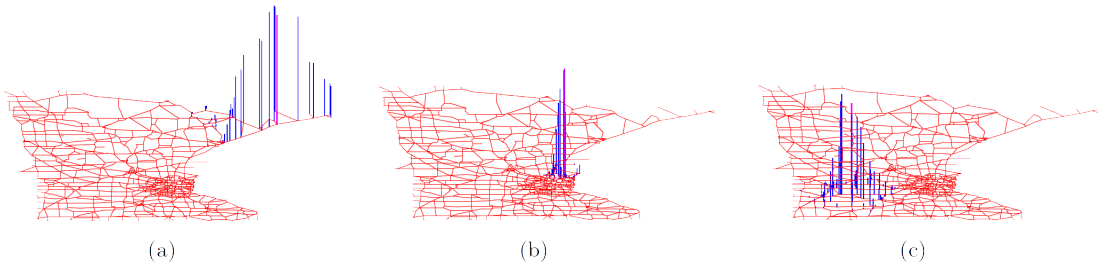
This leads to think about Equation 5.23 as an operator acting on the kernel $g(\cdot)$ directly defined in the spectral domain and from which we can obtain the effective translation to kernel n through inverse graph Fourier transform. Moreover the term $\sqrt{N}$ in the Equation 5.23 is a guarantee for the translation operator to preserve the mean value of the signal.

Now, in all of this we can assume the signal to have a smooth support as the one described in chapter 3; this prior is carried by the kernel entity $g(\cdot)$ and can be seen as a measure to control the localization of $T_n g$ centered at vertex n (meaning that the magnitude $(T_n g)(\mathrm{i})$ of the translated kernel at vertex i decays with the increasing distance from the center of the kernel). We can thus compose our atoms $T_n g$ as smooth entities, coming from the assumption that the kernel function in Equation 5.23 is a smooth polynomial function of degree K:

$$\hat{g}(\lambda_l) = \sum_{k=0}^{K} \alpha_k \lambda_l^k, \quad l = 0, \ldots, N-1 \tag{5.24}$$

From this fact, we can so obtain a global expression for a translation operator as:

$$\begin{aligned} T_n g = \sqrt{N}(g * \delta_n) &= \sqrt{N} \sum_{l=0}^{N-1} \sum_{k=0}^{K} \alpha_k \lambda_l^k \chi_l^*(n)\chi_l \\ &= \sqrt{N} \sum_{k=0}^{K} \alpha_k \sum_{l=0}^{N-1} \lambda_l^k \chi_l^*(n)\chi_l = \sqrt{N} \sum_{k=0}^{K} \alpha_k (\mathcal{L}^k)_n \end{aligned} \tag{5.25}$$



(a) (b) (c)

Figure 5.2: Example of graph signal translation: translation for $T_{200}g$ (a), $T_{1000}g$ (b) and $T_{2000}g$ (c)

where $(\mathcal{L})_n$ represents the n^{th} column of the k^{th} power of the Laplacian matrix $\mathcal{L}^k$. The final concatenation of N such columns brings us to generate a set of N atoms, corresponding to the columns of

$$Tg = \sqrt{N}\hat{g}(\mathcal{L}) = \sqrt{N}\chi\hat{g}(\Lambda)\chi^T = \sqrt{N}\sum_{k=0}^{K}\alpha_k\mathcal{L}^k \tag{5.26}$$

Where the quantity Λ corresponds to the diagonal matrix of the eigenvalues in such a way that the following relation holds: $\mathcal{L} = \chi\Lambda\chi^T$ [5]. This brief section has the intrinsic meaning that if the kernel $g(\cdot)$ is a polynomial of degree K, then the translation operator is 0 in all the vertices i that are more than K hops far away from the center vertex n, which means that the vertex domain support of the translated vertex is contained in a sphere of radius K and center in n and its magnitude decades with the increasing distance from n to i. In Figure 5.2 there is an example of an atom translation in three different ways.

5.2.2 Dictionary learning section

Gives the previous background, this part of the learning problem mainly tries to identify useful criteria for the way each kernel function spreads over the surrounding nodes. If, in fact, we were able to define the behavior of an atom inside its surroundings, we could use this information to sum up the description of a graph's portion in a more accurate way.

Since experimental description of real-life graph structures gives us reasons to believe that they tend to rely on smooth patterns (and so, low frequencies), assuming a certain kernels structure could be a way to relate the manifold macrostructure with its inner structure. From this, we arrive to the main assumption in our work, which is kernels smoothness: what

we wish is that our atoms were representative of the portion of the graph they are covering, so that they should spread with non trivial values over all nodes of the subgraph. For this reason we constrained our dictionary kernels in such a way that they have mostly smooth support in the surroundings of the source node. To be precise, this concept is something different form the smoothness assumption we widely examined in the previous sections: it adds some more information strictly related to the behavior of the kernel itself. In fact, if we learned a smaller order polynomial with constraints in high frequencies, we would not have a large spread of our atoms, things we want to have so that our atoms can represent a large portion of the graph.

To have this new smoothness constraint we look at the kernel polynomial:

$$g(\lambda) = \sum_{k=0}^{K} \alpha_k \lambda_k \tag{5.27}$$

and we try to constraint the roots to correspond to the highest eigenvalues.
This would change the expression of the kernel and turn our learning problem into a simpler one.

In order to include this new smoothness notion we could go along two different paths:

- We could add this information into the objective function;
- Or we could add the information to the constraints;

Smoothness prior in the objective function

In the first case the problem becomes trying to learn a smaller number of coefficients in the kernel function:

$$g(\lambda) = h(\lambda)(\lambda - \lambda_N)(\lambda - \lambda_{N-1}) \cdot \cdots \cdot (\lambda - \lambda_{N-M+1}) \tag{5.28}$$
$$\textit{where} \qquad h(\lambda) = \sum_{n=0}^{K-M} \gamma_n \lambda^n$$

With this statement as start, we could try to give a first formulation of our problem, where we try to also include the fact that the atoms in the dictionary are actually coming from a graph dictionary. In fact, if we assumed these atoms to be only vectors describing the signal, we would keep this part separated from the graph notion.

With the previous formula we can thus arrive to a dictionary problem of this type:

$$\underset{\gamma_0,\dots,\gamma_{K-M}X}{\operatorname{argmin}} \quad ||Y - \mathcal{D}X||_F^2 + \alpha||\mathcal{D}||_1 \tag{5.29}$$

$$\begin{aligned} with \qquad g(\lambda) &= h(\lambda)(\lambda - \lambda_N)(\lambda - \lambda_{N-1}) \cdot \dots \cdot (\lambda - \lambda_{N-M+1}) \\ h(\lambda) &= \sum_{n=0}^{K-M} \gamma_n \lambda^n \\ \mathcal{D}_i &= [g(L_i)]_{\text{source}_i} \\ 0 \leq g(\lambda) &\leq c \qquad (5.210) \\ (c-\epsilon)I \preceq \sum_{s=1}^{S} \mathcal{D}_s &\preceq (c+\epsilon)I \qquad (5.211) \end{aligned}$$

Where the second term in Equation 5.29 accounts for the graph sparsity, while the positions of the non-trivial values of X are known, but the values are to be learned. Moreover, the last two constraints Equation 5.210 and Equation 5.211 are taken from [13] and ensure the kernels are bounded and span the entire spectrum.

Smoothness prior in the constraints

On the other hand, we could be interested in adding the prior to the constraints of the minimization problem presented above. In this way the problem would assume a form like the following one:

$$\underset{\gamma_0,\dots,\gamma_K X}{\operatorname{argmin}} \quad ||Y - \mathcal{D}X||_F^2 + \alpha||\mathcal{D}||_1 \tag{5.212}$$

$$\begin{aligned} with \qquad g(\lambda) &= \sum_{k=0}^{K} \alpha_k \lambda^k \\ \mathcal{D}_i &= [g(L_i)]_{\text{source}_i} \\ 0 \leq g(\lambda) &\leq c \qquad (5.213) \\ (c-\epsilon)I \preceq \sum_{s=1}^{S} \mathcal{D}_s &\preceq (c+\epsilon)I \\ 0 \leq g(\lambda') &\leq \varepsilon \\ \lambda' \in [\lambda_N, \lambda_{N-1}, \dots, \lambda_{N-M+1}] \quad and \quad \varepsilon &\approx 0 \qquad (5.214) \end{aligned}$$

Where also in this case M represents the number of eigenvalues we want to send the kernel functions to 0.

5.2.3 Joint large graph and dictionary learning

Come to this point, an interesting aspect we could focus on is the attempt to joint both the graph and the dictionary learning problem; in fact, if we assume that the graph is unknown we could imagine adding a graph learning part to the optimization. In doing this, we have to take into account two main complications: first the fact that we do not have fixed eigenvalues, since we would learn the Laplacian again at every new optimization step, and second the fact that as we are learning the eigenvalues, we do not have their initial values either. To this aspects we still have to take into account that the structure of the kernels should be incorporated in our learning problem. For this part, in the end we assumed only one kernel for simplicity so that the dictionary atoms will come all from the same kernel but will be localized in different sources spreading throughout different parts of the graph. This meaning that the pattern has to be followed by all the atoms, but it will be adapted to small different graphs.

Therefore the overall problem could become something similar to:

$$\underset{X,W_1,\dots,W_m,\gamma_0,\dots,\gamma_{K-M}}{\operatorname{argmin}} \quad ||Y - \mathcal{D}X||_F^2 + \alpha||\mathcal{D}||_1 \tag{5.215}$$

$$\begin{aligned}
\text{where} \quad g(\lambda) &= h(\lambda)(\lambda - \lambda_N)(\lambda - \lambda_{N-1}) \cdot \dots \cdot (\lambda - \lambda_{N-M+1}) \\
h(\lambda) &= \sum_{k=0}^{K-M} \gamma_n \lambda^n \\
\mathcal{D}_i &= [g(L_i)]_{\text{source}_i} \\
L_i &= \text{normalised Laplacian}(W_i) \\
W_{ij} &= W_{ji} \geq 0, \quad \forall i,j \\
W_{ii} &= 0, \quad \forall i \\
0 &\leq g(\lambda) \leq c \\
(c-\epsilon)I \preceq \sum_{s=1}^{S} \mathcal{D}_s &\preceq (c+\epsilon)I
\end{aligned} \tag{5.216}$$

in the case the smoothness prior is carried by the objective function, while:

$$\underset{X,W_1,\dots,W_m,\gamma_0,\dots,\gamma_K}{\operatorname{argmin}} \quad ||Y - \mathcal{D}X||_F^2 + \alpha||\mathcal{D}||_1 \tag{5.217}$$

$$\begin{aligned}
\text{where} \qquad g(\lambda) &= \sum_{k=0}^{K} \gamma_k \lambda^k \\
\mathcal{D}_i &= [g(L_i)]_{\text{source}_i} \\
L_i &= \text{normalised Laplacian}(W_i) \\
W_{ij} &= W_{ji} \geq 0, \quad \forall i,j \\
W_{ii} &= 0, \quad \forall i \\
0 &\leq g(\lambda) \leq c \\
(c-\epsilon)I \preceq \sum_{s=1}^{S} \mathcal{D}_s &\preceq (c+\epsilon)I \\
0 &\leq g(\lambda') \leq \varepsilon \\
\lambda' &\in [\lambda_N, \lambda_{N-1}, \dots, \lambda_{N-M+1}] \quad \textit{and} \quad \varepsilon \approx 0
\end{aligned} \tag{5.218, 5.219}$$

in the case the smoothness is carried by the constraints.
We also remember that the constraint in Equation 5.216 accounts for the fact that we are assuming to work with undirected graphs.

5.2.4 Work structure

To summarize, the main steps in which the following work is articulated are the following:

- We implement some constraints related to the smoothness of the kernels in the dictionary learning approach, and we show the improvements deriving from it;
- We present a solution to solve both the dictionary and the graph learning problem within the same framework;
- We underline how the smoothness constraints on the kernels can still provide improvements even in the case of double learning frameworks;
- We briefly present the possible next steps in this work, in order to obtain further implementations;

Chapter 6

The dictionary learning improvement under smoothness constraints

As we previously hinted, in the work of Thanou, Shuman and Frossard the focus was on learning the parametric functions (*kernels*) through which it was possible to reconstruct the graph signal. Moreover, in their work they modelled the graph signal as a combination of overlapping local patterns, describing localized events on the graph which can appear in several vertices. This representation of the graph signal brought to a representation of the dictionary structure as a concatenation of subdictionaries, which are polynomials of the graph Laplacian. Under this assumption they then learned the coefficients of the kernels through a numerical optimization.

6.0.1 The dictionary structure and main assumptions

The dictionary structure we used, presented in [13], is in the following form: we designed a graph dictionary $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_S]$, or a concatenation of a set of S subdictionaries, as:

$$\mathcal{D}_s = \hat{g}_s(\mathcal{L}) = \chi(\sum_{k=0}^{K} \alpha_{sk}\Lambda^k)\chi^T = \sum_{k=0}^{K} \alpha_{sk}\mathcal{L}^k \tag{6.01}$$

where $\hat{g}_s$ is the pattern (*the kernel*) of the subdictionary $\mathcal{D}_s$ and where the atom given by the n^{th} column of subdictionary $\mathcal{D}_s$ is equal to $\frac{1}{\sqrt{N}}T_n g_s$, the translation operator defined in Equation 5.23 divided by the constant $\frac{1}{\sqrt{N}}$.
However, despite the polynomial constraint guarantees the localization of the atoms in the

vertex domain, it does not provide though any information about the spectral representation of the atoms. The two matrix inequalities already listed in Equation 5.215 take it into consideration:

$$0 \preceq \mathcal{D}_s \preceq cI, \quad \forall s \in \{1, 2, \dots, S\}, \tag{6.02}$$

and (6.03)

$$(c - \epsilon)I \preceq \sum_{s=1}^{S} D_s \preceq (c + \epsilon)I \tag{6.04}$$

Equation 6.02 accounts for the fact that our kernels are nonnegative and uniformly bounded by a given constant c (which we will further suppose equal to 1), while Equation 6.04 accounts for the fact that the signals we consider usually contain frequency components spread across the entire spectrum and so the kernels should cover it as well. These two spectral constraints together contribute to increase the stability of the dictionary. On the other hand, it might be noted that the constraint in Equation 6.04 should not be considered in the case of one single kernel, obviously, since imposing that the sum of all kernels has to stay around 1 would give the trivial solution $g(\lambda) = 1 \; \forall \lambda$

.

We finally underline the fact that for the design of the dictionary the eigenvectors used are the ones from the normalized graph Laplacian (and not the unnormalized one), since this configuration revealed to be more suitable for our bodywork.

6.1 The dictionary learning algorithm under smoothness constraints

With these basis we therefore implemented a dictionary learning algorithm based on the one presented in [13] in order to add the smoothness priors. As previously mentioned, we

can cast the general problem as the following optimization problem:

$$\begin{aligned} \underset{\alpha\in\mathbb{R}^{(K+1)*S},\, X\in\mathbb{R}^{SN\times N}}{\operatorname{argmin}} \quad & ||Y-\mathcal{D}X||_F^2+\mu||\alpha||_2^2 \\ \text{subject to} \quad & ||x_m||_0=T_0, \quad \forall m\in\{1,\dots,M\} \\ & \mathcal{D}_s=\sum_{k=0}^{K}\alpha_{sk}^k, \quad \forall s\in\{1,2,\dots,S\} \\ & 0\preceq\ \mathcal{D}_s\preceq cI, \quad \forall s\in\{1,2,\dots,S\}, \\ & (c-\epsilon)I\preceq\sum_{s=1}^{S}\mathcal{D}_s\preceq(c+\epsilon)I \end{aligned} \tag{6.15}$$

with:

- μ being a coefficient that should be enough guarantee the stability of the solution while preserving large values in the polynomial αs;
- x_m corresponding to the m^{th} column of the sparsity matrix X;
- T_0 being the sparsity level of the coefficient of each signal;
- l_0 being norm term, one time more standing for the sparsity need we already manifested.

Since the optimization problem is non convex, there is the need to approximate it by the alternation of two optimization subproblems that are convex, namely the sparse coding and the dictionary update step.

The sparse coding step aims at learning the sparsity matrix X solving

$$\underset{X}{\operatorname{argmin}}\,||Y-\mathcal{D}X||_F^2 \qquad \text{subject to} \quad ||x_m||_0\leq T_0, \tag{6.16}$$

and it is solved using the Orthogonal Matching Pursuit (OMP) method, an iterative greedy algorithm that at each step selects the element in the dictionary that is best correlated with the signal residual part; it subsequently produces a new estimation through the projection of the signal onto the dictionary elements that have already been selected. This method admits simple and fast implementation and it has been proven that his error on the approximation is only a small factor worse than the minimal error obtained through a non-sparse

representation. [14].

On the other side, the dictionary update step aims at finding the polynomial coefficients of the kernels by minimizing

$$\underset{\alpha \in \mathbb{R}^{(K+1)S}}{\operatorname{argmin}} ||Y - \mathcal{D}X||_F^2 + \mu||\alpha||_2^2 \tag{6.17}$$

subject to the remaining constraints in Equation 6.15

To this main problem we then added the smoothness prior through the two different implementations we already presented in subsection 5.2.2, the first one trying to insert the smoothness assumption inside the objective function of the dictionary update step, while the second one focusing on adding a smoothness prior in the constraints of the optimization function.

6.1.1 Smoothness prior in the objective function

We remember the equations 5.28, and we reformulate them as

$$g(\lambda) = \sum_{k=0}^{K} \alpha_k \lambda^k = \sum_{n=0}^{N} \gamma_n \lambda^n \times \sum_{m=0}^{M} \beta_m \lambda^m \tag{6.18}$$

With $h(\lambda) = \sum_{n=n}^{N} \gamma_n \lambda^n$ and $(\lambda - \lambda_M)(\lambda - \lambda_{M-1}) \cdot \ldots \cdot (\lambda - \lambda_{M-N+1})$ being the two sub-polynomials in which Equation 6.18 has been decomposed. Between these, $h(\lambda)$ represents the polynomial with the unknown coefficients to be found, while the polynomial in the unknown β is the one obtained from imposing the highest eigenvalues as roots of the general polynomial 6.18.

In particular, what we do in order to retrieve them is to generate V, a Vandermonde matrix with the root eigenvalues of the Laplacian raised to the powers of the kernel polynomial and impose that the vector of β_m coefficients we want is one of the vectors of the nucleus of V:

$$V = \begin{bmatrix} 1 & \lambda_{N-M+1} & \lambda_{N-M+1}^2 & \ldots & \lambda_{N-M+1}^K \\ 1 & \lambda_{N-M+2} & \lambda_{N-M+2}^2 & \ldots & \lambda_{N-M+2}^K \\ \vdots & \ddots & & \ldots & \vdots \\ 1 & \lambda_N & \lambda_N^2 & \ldots & \lambda_N^K \end{bmatrix}, \qquad B = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{bmatrix} \tag{6.19}$$

$$\text{and} \qquad \mathbf{V} \cdot \mathbf{B} = \mathbf{0} \tag{6.110}$$

With N being in this case the number of eigenvalues of $\mathcal{L}$.
In the case of the heat kernel, the function does not exactly go to 0 in the last eigenvalues considered, but it stays a little bit above it, that is why we relaxed this condition requesting the final λs to be root of a polynomial with a vertical indicative offset. The following results show how this procedure is still a good improvement in such a way that we could use this approach to better model general low frequency kernels in more precise ways than simply asking them to go directly to 0.

Since we deal with the optimization problem in Equation 6.15 through Linear Matrix Inequalities (LMI) tools, we try to model the vector containing the kernel coefficients in such a way that the structure itself already holds the information regarding the composition of the two sub polynomials. To do so, we multiply the two sub polynomials among each other without resolving the equation with known β, but leaving them as unknown, in such a way that the behavior of the overall polynomial emerges. We thus obtain the structure that every α coefficient should have in order to transmit the smoothness prior to the objective function. Namely:

$$\begin{aligned}\sum_{n=0}^{N}\gamma_n\lambda^n \times \sum_{m=0}^{M}\beta_m\lambda^m = (\beta_0\gamma_0)\lambda^0 + (\beta_0\gamma_1+\beta_1\gamma_0)\lambda + \cdots + \\ (\beta_0\gamma_N+\beta_1\gamma_{N-1}+\cdots+\beta_N\gamma_0)\lambda^N + \cdots + \\ (\beta_1\gamma_N+\beta_2\gamma_{N-1}+\cdots+\beta_{N+1}\gamma_0)\lambda^{N+1} + \cdots + \\ (\beta_N\gamma_N+\beta_{N+1}\gamma_{N-1}+\cdots+\beta_M\gamma_0)\lambda^M + \\ (\beta_{N+1}\gamma_N+\cdots+\beta_{2N+1}\gamma_1)\lambda_{M+1} + \cdots + \\ (\beta_M\gamma_N)\lambda^K\end{aligned} \tag{6.111}$$

Once we obtained it, we insert this new information about the kernels structure in the optimization objective function, where now the alpha vector has a reduced number of unknowns that have to be found. With the problem set in this way we can thus analyse the new results we obtain.

6.1.2 Smoothness prior in the problem constraints

Another approach to the problem is based on inserting this smoothness prior in the constraints of the optimization function. On one side this choice should be discarded in favour of the first approach, since adding other constraints to our already ill posed problem could

add more non-desired complexity; on another side this choice could reveal motivated by the fact that the control over the behavior of the optimization algorithm is in this way more verifiable than in the previous case, in which a black-box-stile coefficients vector detains all the smoothness priors. In this case the constraint added is the imposition that the Vandermonde matrix, obtained selecting only the last M high-frequency eigenvalues in the graph Laplacian, should give a null vector when multiplied by the vector of the alpha coefficients. If N is the total number of nodes in the considered graph, then we have:

$$\begin{bmatrix} 1 & \lambda_{N-M} & \lambda_{N-M}^2 & \dots & \lambda_{N-M}^K \\ 1 & \lambda_{N-M+1} & \lambda_{N-M+1}^2 & \dots & \lambda_{N-M+1}^K \\ \vdots & & & & \\ 1 & \lambda_P & \lambda_P^2 & \dots & \lambda_N^K \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_K \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.112}$$

6.1.3 The optimization algorithm

The final optimization algorithm is described by the pseudocode in the algorithm 1:

6.1.4 Results

Data Generation

The datasets used in the experiments are composed by a signal Y, an adjacency matrix W and the comparison kernels used after the algorithm execution to verify the quality of the results. The signal Y is obtained as the repetition of 2000 samples randomly generated by a uniform distribution, the graph manifold onto which the signal is structured has 30 nodes (for simplicity reasons) while the edges that are elements of the weight matrix W are generated from the gaussian function

$$w_{ij} = e^{-\frac{||x_i - x_j||_2^2}{2\sigma^2}} \tag{6.113}$$

following the approach in [7]. At the same time, the kernel coefficients we generated as ground truth values come respectively:

- form the Taylor expansion for the heat kernels

$$g(\hat{\lambda}) \approx e^{-2\lambda} \tag{6.114}$$

- from one of the synthetic kernels used in the dataset from [13]

The number of iterations we apply in this first part of the work is 50, as we found it is a good value for the algorithm to properly converge to a minimum, while the value of ϵ for the spectral boundaries in Equation 5.2.2 has been set to 0.2. Moreover, we assume the sparsity coefficient T_0 to be equal to 4 and the parameter μ to be 10^{-4}.

We use the *sdpt3* solver in the *yalmip* optimization toolbox to solve the quadratic problem and, in order to directly compare the methods here described, we always use the OMP optimization criteria for the sparse coding step.

Finally, the average normalized approximation error we examine in the results is estimated by:

$$\frac{1}{|Y_{test}|}\sum_{m=1}^{|Y_{test}|}\frac{||Y_m - \mathcal{D}X_m||_2^2}{||Y_m||^2} \tag{6.115}$$

with $|Y_{test}|$ being the cardinality of the testing set.

First approach: smoothness in the structure

Results are promising, especially from a computational time point of view. The experiments have been repeated several times, in order to be able to claim the stability of the algorithm and show a stable trend.

The two main metrics we analyse for the performance of the algorithm are the reproduction error and the computational time taken by the optimization step, in the algorithm named as $CPUTime$.

Iteration number	Original algorithm	Smoothness assumption
1	0.0251	0.0263
2	0.0252	0.0260
3	0.0250	0.0262
4	0.0252	0.0262
5	0.0252	0.0261
Average	0.02514	0.02616

Table 6.1: Reproduction error comparison for the Heat kernel dataset

Iteration number	Original algorithm	Smoothness assumption
1	0.0213	0.0220
2	0.0214	0.0220
3	0.0215	0.0220
4	0.0212	0.0221
5	0.0215	0.0220
Average	0.02138	0.0220

Table 6.2: Reproduction error comparison for the Thanou et al. kernel dataset

The two tables 6.1 and 6.2 list the reproduction errors for 5 sample trials using both datasets and comparing the effect of the smoothness over the learning quality. As can be seen, adding the smoothness prior does not bring automatically to a better reproduction; this is probably due to the fact that the definition of smooth kernel is not completely rigorous in describing "how much" the kernel goes to 0, thus we have to accept a margin of uncertainty in forcing the final smoothness part. The variation is anyway contained, such that it cannot be considered as properly worsening the dictionary learning.

On the other hand, this prior allows us to observe an improvement in the computational cost of the algorithm, here measured by $CPUTime$ and shown in the table 6.3. The results are promising since one of the next steps in this work can focus of the learning of high dimensional graphs, in which computational cost reduction is even more important. From this point of view we can thus state that adding the smoothness prior brings a significative advantage to the learning procedure.

	No smoothness assumption	Smoothness assumption
Heat Kernel dataset	0.9334 s	0.1001 s
Thanou et al. dataset	0.1661 s	0.0989 s

Table 6.3: Computational cost comparison for the Heat kernel and the Thanou et al. kernel dataset

The learning procedure behaves overall well, as depicted by the figures 6.1 and 6.2, where the learned kernels are shown, as also in Figure 6.3 and Figure 6.4, where the values of the

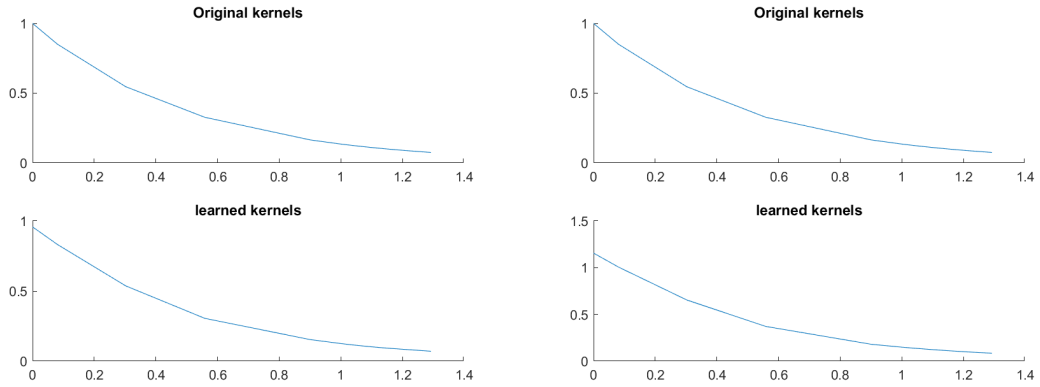


Figure 6.1: Comparison between kernels without and with smoothness prior. Heat kernel dataset

kernel coefficients can be seen, as well as how the smoothness prior guarantees that the coefficients follow the original ones in a more faithful way.

Second approach: smoothness in the constraints

In the same way, here results show a general improvement, both regarding the reproduction error and the behavior of the kernel coefficients. In this case we can notice a slight improvement in both datasets concerning the reproduction error, Table 6.4 and Table 6.5 compare the results in the case with and without the smoothness prior, and show better results on average. One interesting thing that can be noticed is the stability of the learning process: adding the smoothness prior to the kernels has the consequence that the output values are stable during several iterations, while in the original algorithm there could be even some discrete variations.

Moreover, making a comparison between the average computational cost we can see how the smoothness assumption improves the performance once again (Table 6.6). Here, though, the difference is lower than in the previous approach, but this is comprehensible since now we are focusing on the constraints again, without reducing the number of actual unknown that the objective function is holding.

The improvement can also be noticed in the behavior of the kernels' coefficients: Figure 6.5 represents another comparison between the values assumed by the original heat kernels' coefficients (blue lines), and the ones assumed by the learned coefficients in the

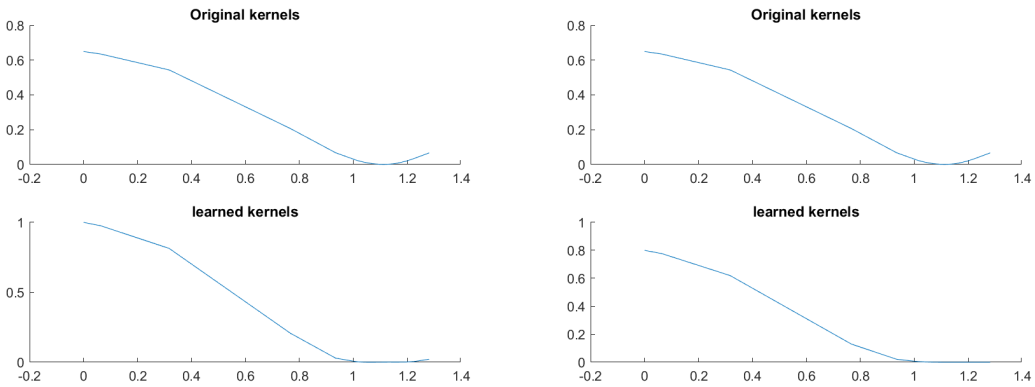


Figure 6.2: Comparison between kernels without and with smoothness prior. Thanou et al. kernel dataset

Iteration number	Original algorithm	Smoothness assumption
1	0.0250	0.0251
2	0.0251	0.0251
3	0.0250	0.0252
4	0.0251	0.0248
5	0.0251	0.0247
Average Error	0.02506	0.02498

Table 6.4: Reproduction error comparison for the Heat kernel dataset

Iteration number	Original algorithm	Smoothness assumption
1	0.0217	0.0216
2	0.0212	0.0215
3	0.0213	0.0216
4	0.0214	0.0216
5	0.0213	0.0215
Average Error	0.02138	0.02156

Table 6.5: Reproduction error comparison for the low frequency kernel from Thanou et al. dataset

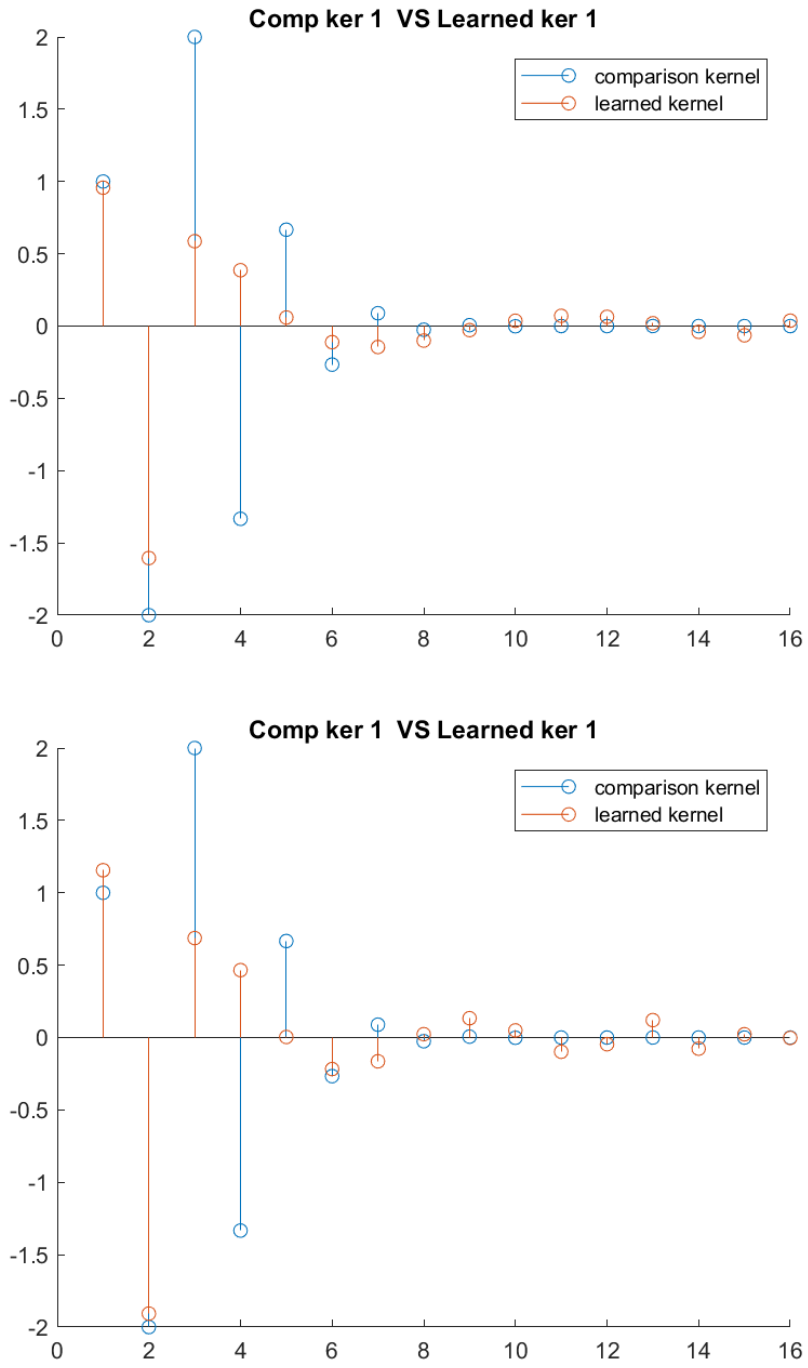


Figure 6.3: Comparison between kernels coefficients without and with smoothness prior. Heat kernel dataset

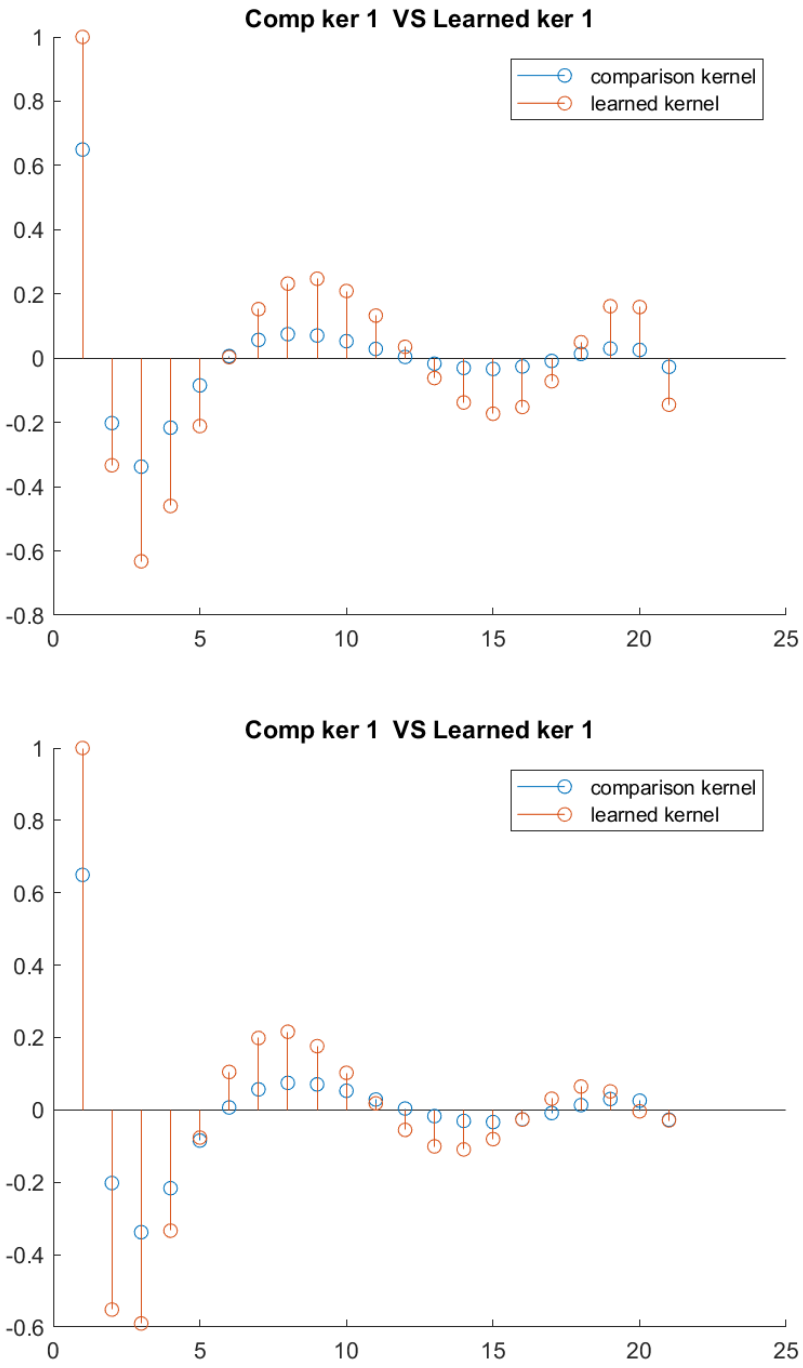


Figure 6.4: Comparison between kernels coefficients without and with smoothness prior. Thanou et al. dataset

	No smoothness assumption	Smoothness assumption
Heat Kernel dataset	0.9334 s	0.1015 s
Thanou et al. dataset	0.1661 s	0.1224 s

Table 6.6: Computational cost comparison for the Heat kernel and the Thanou et al. kernel dataset

case we are applying the smoothness prior or not. It can be seen how the coefficients in the case of no smoothness priors tend to have a less defined trend, while in the smoothness case they follow the original trend more faithfully. This aspect is shown even more clearly in the case of Thanou et al.'s dataset (Figure 6.6), in which the coefficients have almost the same behavior, apart from their absolute value. The reason for this is mainly the nature of our constraints: in fact, since in the problem we are working with LMI,we are not imposing the solution of the optimization problem to be constrained to certain specific values, but to a range of them. As a consequence, the solution of the optimization process represents a point localized in a certain range.

Finally, Figure 6.7 and Figure 6.8 show the comparison between the original kernel and the learned kernel without and with smoothness prior in the case of a Heat kernel (Figure 6.7) and the dataset from Thanou et al. (Figure 6.8). It can be seen how the use of smoothness improves the overall trend also for the kernel shape, which is the ultimate purpose of the algorithm.

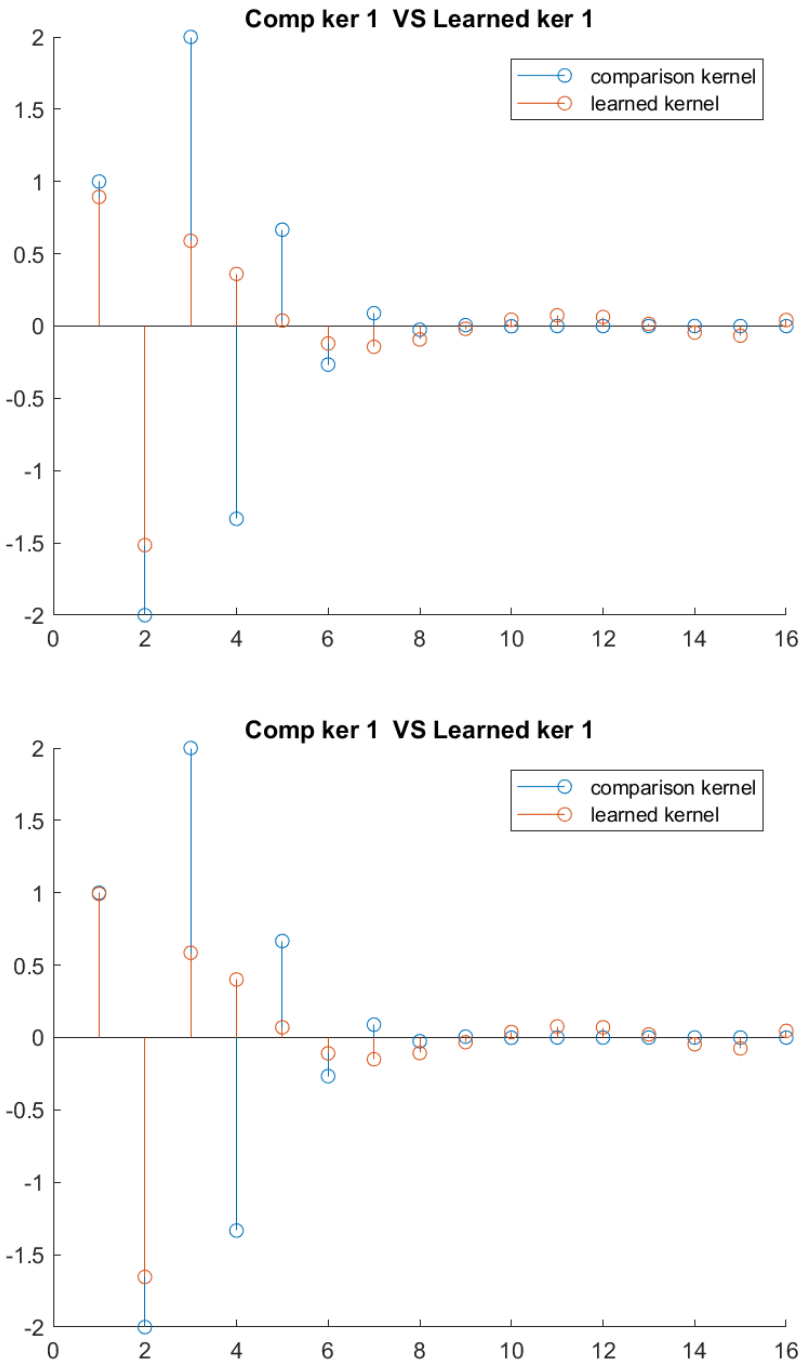


Figure 6.5: Comparison between kernels coefficients without and with smoothness prior. Heat kernel dataset

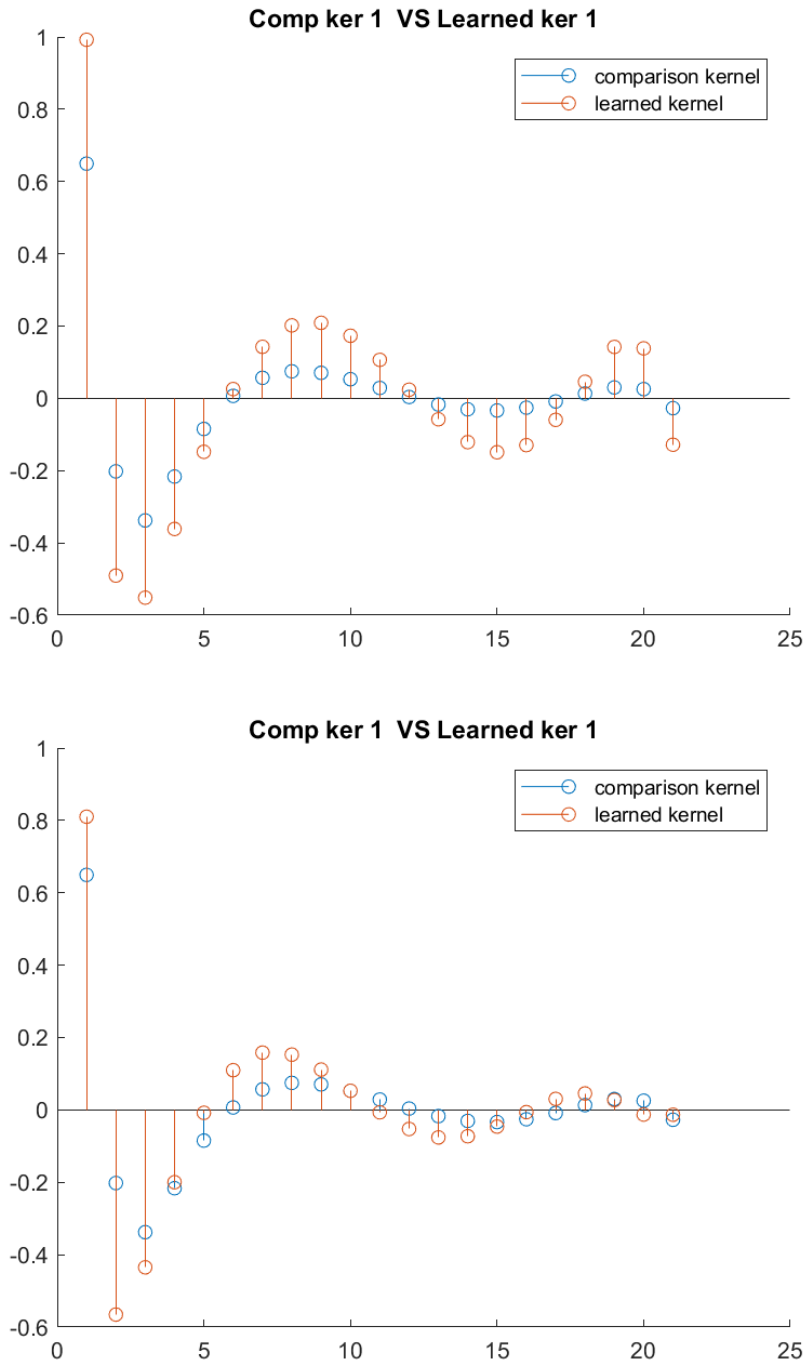


Figure 6.6: Comparison between kernels coefficients without and with smoothness prior. Thanou et al. dataset

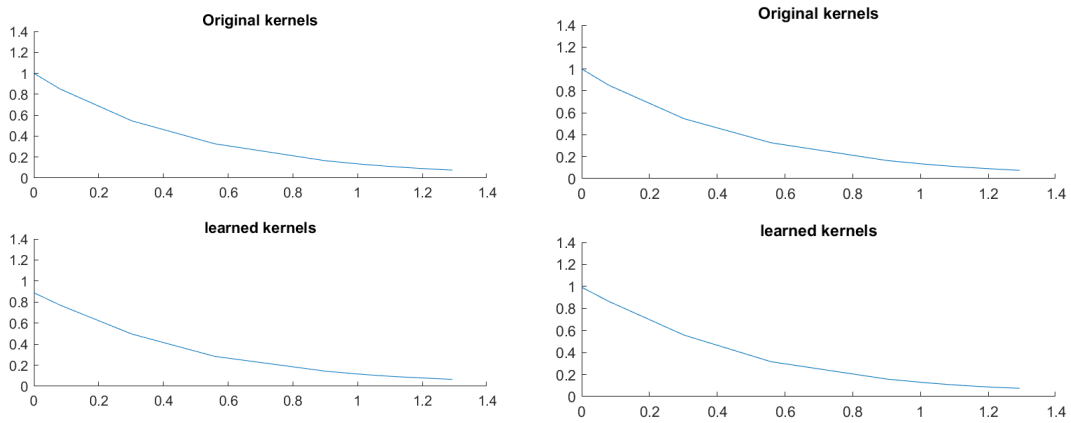


Figure 6.7: Comparison between kernels without and with smoothness prior. Heat kernel dataset

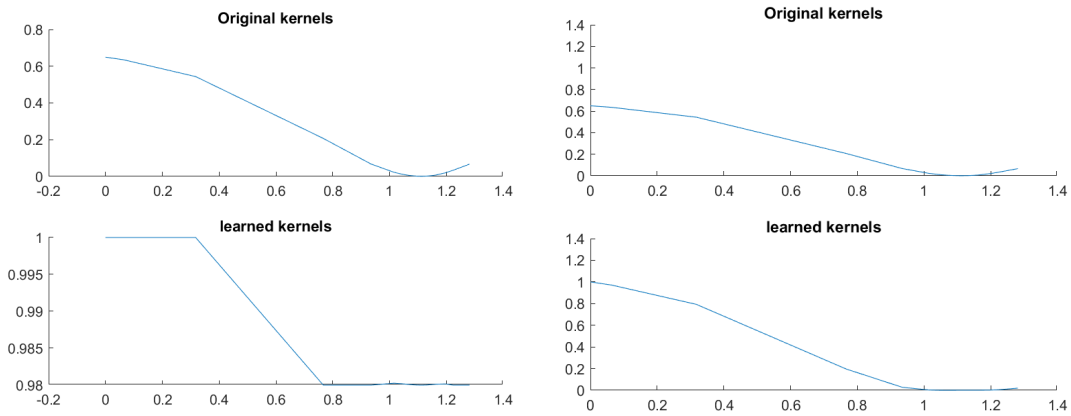


Figure 6.8: Comparison between kernels without and with smoothness prior. Thanou et al. kernel dataset

Algorithm 1 Parametric dictionary learning on graph with smoothness priors

procedure INITIALIZATION
    $Y \leftarrow$ Signal samples set
    $T_0 \leftarrow$ Target sparsity
    $K \leftarrow$ Polynomial degree
    $S \leftarrow$ Number of subdictionaries
    $iterNum \leftarrow$ Number of iterations
    $\mathcal{D} \leftarrow$ Initial dictionary computed thirugh random kernels
end procedure
for $i = 1, 2, \ldots, iter$ **do**
    procedure SPARSITY STEP
        Scale each atom in $\mathcal{D}$ to a unit norm
        $X \leftarrow$ Sparsity estimation with OMP
        Rescale X and $\mathcal{D}$ to recover the polynomial structure
    end procedure
    procedure DICTIONARY LEARNING STEP $\alpha \leftarrow$ kernels estimation through sdpt3
    end procedure
    procedure DICTIONARY UPDATE STEP
        $\mathcal{D} \leftarrow \sum_{k=0}^{K} \alpha_k \mathcal{L}^k$
    end procedure
end for

Chapter 7

The graph learning algorithm

As previously anticipated, our work involved also the need of learning the graph manifold structuring a signal. In order to accomplish it, we made use of the contribution of Maretic et al. to the problem, as it is presented in [4]. In the work, a generic model is considered and which is analogous to the one described in [13]. There the signals are represented by combinations of local overlapping patterns residing on graphs and the intrinsic graph structure is merged with the dictionary through the already well known graph Laplacian operator.
In the process, the optimization is performed over the weight matrix W instead of $\mathcal{L}$ due to the fact that the constraints defining a valid weight matrix W are less demanding than those defining a valid Laplacian.

7.1 The algorithm

Since also this optimization problem is non convex, Maretic et al. solved it through the alternation of the optimization steps: in the first step a weight matrix W is fixed and the objective function is optimized with respect to the sparsity matrix, using OMP, in an analogous way than in [13]; in the second step the focus is on the graph learning using the previously estimated X, and since the relative problem is still non convex, the proposed solution involved an approach based on gradient descent.

The optimization problem addressed as

$$\underset{W}{\text{argmin}} \quad ||Y - \mathcal{D}X||_F^2 + \beta_W ||W||_1 \tag{7.11}$$

$$\text{subject to} \quad \mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_S] \tag{7.12}$$

$$\mathcal{D} = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k, \qquad \forall s \in \{1, \dots, S\} \tag{7.13}$$

$$\mathcal{L} = I - D^{\frac{1}{2}} W D^{\frac{1}{2}} \tag{7.14}$$

$$W_{ij} = W_{ji} \geq 0, \qquad \forall i, j \quad i \neq j \tag{7.15}$$

$$W_{ij} = 0, \forall i \tag{7.16}$$

is then turned into the gradient equivalent form:

$$\begin{aligned} &\nabla_W ||Y - \sum_{s=1}^{S} \mathcal{D}_s X_s||_F^2 \\ &= \sum_{s=1}^{S} \sum_{k=1}^{K} \alpha_{sk} (- \sum_{r=0}^{k-1} 2 A_{k,r}^T + \mathbf{1}_{N \times N} (B_k \circ I)) \end{aligned} \tag{7.17}$$

Where the values the matrices $A_{k,r}$ and B_k hold are:

$$A_{k,r} = D^{-\frac{1}{2}} \mathcal{L}^{k-r-1} X_s (Y - DX)^T \mathcal{L}^r D^{-\frac{1}{2}} \tag{7.18}$$

$$and \tag{7.19}$$

$$B_k = \sum_{r=0} k - 1 D^{-\frac{1}{2}} W A_{k,r} D^{-\frac{1}{2}} + A_{k,r} W D^{-1} \tag{7.110}$$

moreover $\mathbf{1}_{N \times N}$ is a matrix of ones, I is an $N \times N$ identity matrix and the gradient of $\beta_W ||W||_1$ can be approximated with $\beta_W sign(W)$

Chapter 8

Merging the graph and the dictionary learning part

The next step of our work focuses on the attempt to learn at the same time the Dictionary and the graph structure, while in a further moment we will focus on adding the previous smoothness constraint. As it is formulated theoretically, the double optimization problem could reveal itself to be heavily ill posed, since now the number of parameters involved in the optimization would go up to three instead of the already challenging two. This could bring the problem to diverge, or on the contrary, to have infeasibility consequent issues. As a remedy we decided to create an algorithm which alternates the dictionary learning and the graph learning step, while the sparsity prior is checked at every step by the continuous optimization of the sparsity matrix X.

8.1 The initialization section

In this first part we initialize the weight matrix W with a set of random weights with a gaussian distribution, we obtain the related normalized Laplacian $\mathcal{L}$ following the procedure explained in [7] and compute the orthogonal basis connected to it on order to use the eigenvectors we find to have a first estimate on the dictionary.

8.2 The alternation between optimization steps

With this starting elements we are then able to pass to the alternating optimization steps: every iteration of the global cycle involves the sparsity matrix estimation X through the OMP method (the so-called *sparse coding step*); what occur in turn every time are the *dictionary learning* and the *graph learning* steps. After the learning steps there is then a re-estimation of the dictionary, from the new values of α and $\mathcal{L}$, which will then be used from the spars coding step in the next iteration.
The general algorithm thus articulates as described in **??**

We specify that, as shown in the pseudo code at line 14, in our algorithm there is no equal distribution of the iterations dedicated to graph and dictionary learning, but there is a rate $5 : 1$ in favour of the graph learning. This choice derived from two main facts:

- While the dictionary learning procedure converges to its minimums in every step, the graph learning part requires more steps for it, concluding that giving more space of action to it was a way to balance the optimization steps;

- The sparsity step revealed to be more sensible to alterations of its inputs (the dictionary in our case) when those were due to the kernel coefficients (and so the dictionary learning step) in such a way that limiting these learning steps in their number was advisable.

8.3 Results

For the tests we run regarding this part, we deployed the same datasets used in subsection 6.1.4: a graph signal with edge weights taken from a gaussian distribution, a Heat kernel and the kernel isolated from Thanou et al. dataset.

Moreover, the number of iterations we chose is $iterNum = 250$, the sparsity factor still equal to $T_0 = 4$ and the stability factor $\mu = 10^{-4}$.

Together with the representation error we added another metric to evaluate the quality of our approach, a function estimating the Precision and the Recall rate for the graph. In particular, the Precision metric expresses the percentage of estimated edges in the graph that are correct, while the Recall metric shows the rate of the original edges are correctly estimated.

Algorithm 2 Parametric dictionary and graph learning

procedure INITIALIZATION
    $Y \leftarrow$ Signal samples set
    $T_0 \leftarrow$ Target sparsity
    $K \leftarrow$ Polynomial degree
    $S \leftarrow$ Number of subdictionaries
    $iterNum \leftarrow$ Number of iterations
    $W \leftarrow$ Random thresholded adjacency matrix
end procedure
for $i = 1, 2, \ldots, iterNum$ **do**
    procedure SPARSITY STEP
        $X \leftarrow$ Sparsity estimation with OMP
    end procedure
    procedure LEARNING STEP
        if $mod(i, 5) = 0$ **then**
            $\alpha \leftarrow$ kernels estimation through sdpt3
        else
            $W \leftarrow$ graph estimation through gradient descent
            $\mathcal{L} \leftarrow D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$
        end if
    end procedure
    procedure DICTIONARY UPDATE STEP
        $\mathcal{D} \leftarrow \sum_{k=0}^{K} \alpha_k \mathcal{L}^k$
    end procedure
end for

The experiments showed promising results, not particularly degraded by the alternating process, and the results we present here are obtained imposing the smoothness prior over the structure of the objective function, since the other procedure was giving similar outcomes.

In Figure 8.1 and **??** the original and the learned kernels are compared: as it can be seen from the figures, the learning process is not particularly affected by the fact that at the beginning of our optimization procedure we have to suppose as unknowns both the graph structure and the dictionary coefficients. The only noticeable difference is in the absolute value of the kernels, as it is slightly reduced, and the spectral range of the eigenvalues: these ones are constantly updated at every graph learning step and the results indicate the tendency to "contract" the spectral range, bringing it further from the 0 value. The constraints over the dictionary, anyway, prevent the learned solution to be sensibly altered and assure the spectral distribution of the frequencies already assumed in the dictionary learning approach.

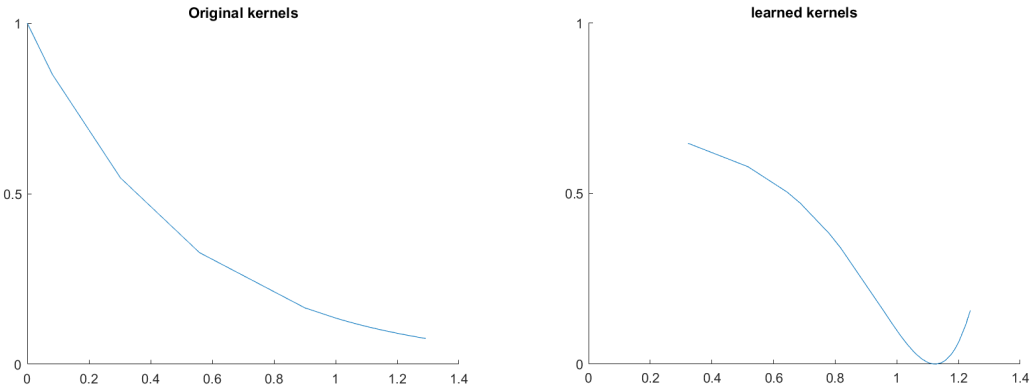


Figure 8.1: Comparison between the original and the learned kernels. Heat kernel dataset

In the Table 8.1 what is presented is the comparison between the Precision and Recall values obtained from the graph learning only approach and the approach combining graph and dictionary learning. The values listed show that alternating between the two learning phases affects the single learning process: in the case of the smooth kernel this difference is discrete, while in the second dataset can be clearly noted; at the same time, though, in the latter case we saw that the outcomes were not stables over the trials, having very good results in some of them and less good in others.

Finally, regarding the representation error the table 8.2 shows how the learned dictio-

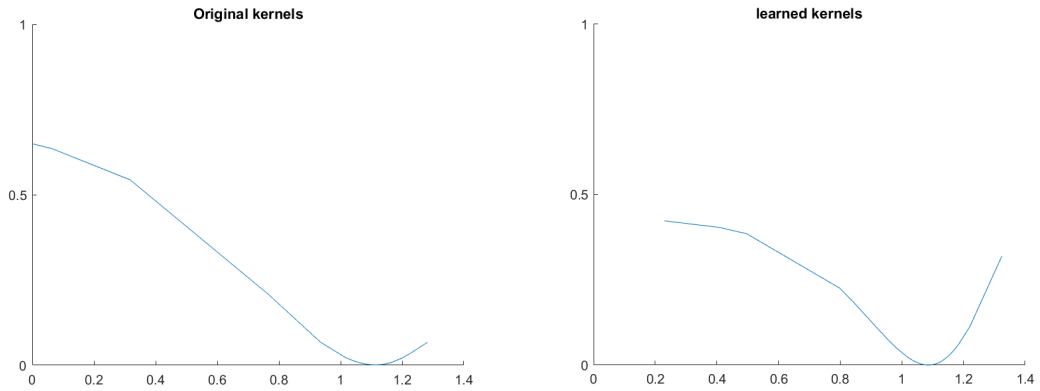


Figure 8.2: Comparison between the original and the learned kernels. Thanou et al. dataset

	Heat kernel		Thanou et al. kernel	
	Graph L.	Graph and D. L.	Graph L.	Graph and D. L.
Precision rate	99.56 %	91.67 %	95.6 %	69.277 %
Recall Rate	98.97 %	91.67 %	94.99 %	67.69 %

Table 8.1: Precision and Recall comparison between the graph-learning-only option and the one adding the dictionary learning

nary is not significantly degraded in quality with respect to the dictionary-only learning algorithm, but still remains acceptable:

	Heat kernel		**Thanou et al. kernel**	
	Dictionary L.	Graph and D. L.	Dictionary L.	Graph and D. L.
Avg. repr. Error	0.0250	0.0697	0.0217	0.0283

Table 8.2: Reproduction error comparison between the dictionary-only learning and the overall process

Chapter 9

The smoothness assumption in the global algorithm

The last part of our work focuses on the addition of the smoothness constraint to the overall algorithm described in chapter 8.

9.1 Results

Again, the starting conditions for the experiments are analogues to the ones supposed in chapters 8 and 6, the number of iterations is, again, 250 and there is still a rate $5:1$ for the number of iterations dedicated to the graph learning part, per each iteration dedicated to the dictionary learning one.

The figures 9.1 and 9.2 show the comparison between the kernels learned without and with the smoothness assumption. In both the figures we see how the smoothness assumption still improves the kernel learning process, at the same time respecting the overall spectral composition of the signal.

Tables **??** and 9.1 again show the comparison between the Precision and Recall values obtained from the approach without the smoothness prior and the approach accounting for it. The values listed clearly show how the algorithm benefits from the prior, having both the values increased. Moreover, in the case of the optimization without smoothness, the values of Precision and Recall tend to have a larger variance over the trials, while adding

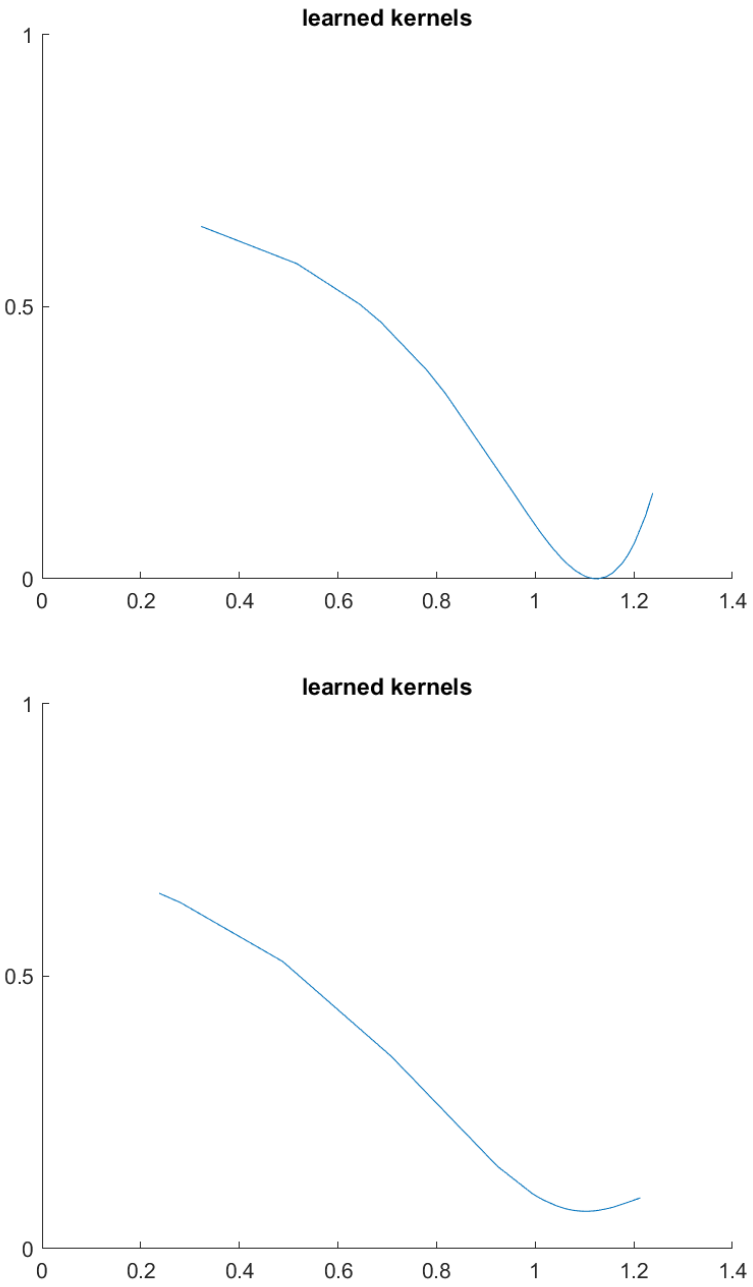


Figure 9.1: Comparison between kernels without and with smoothness prior. Heat kernel dataset

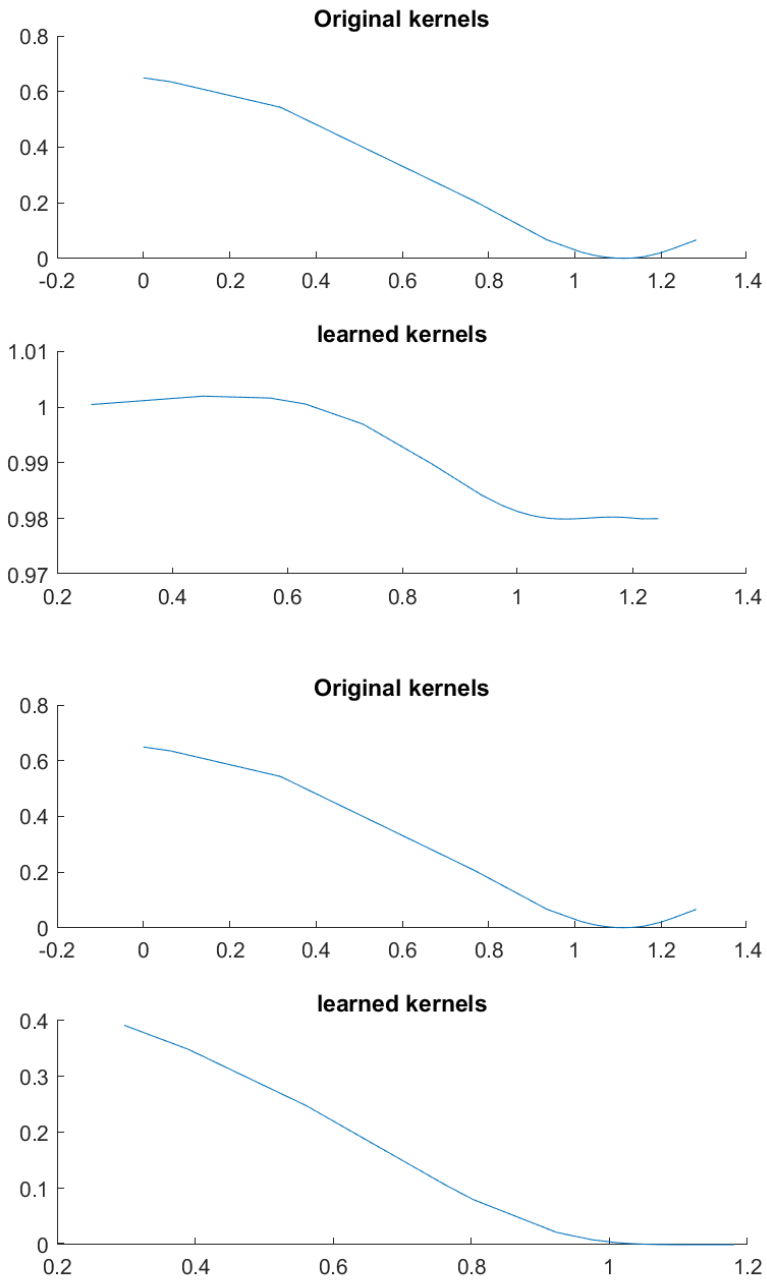


Figure 9.2: Comparison between kernels without and with smoothness prior. Thanou et al. dataset

the smoothness constraint the output tends to oscillate less.

	Heat kernel		Thanou et al. kernel	
	No Smoothness	Smoothness	No Smoothness	Smoothness
Precision rate	92.59 %	96.25 %	51.53 %	80.75 %
Recall Rate	92.59 %	95.06 %	53.50 %	79.75 %

Table 9.1: Precision and Recall outputs for the low frequency kernel from Thanou et al. dataset. Trial 1

	Heat kernel		Thanou et al. kernel	
	No Smoothness	Smoothness	No Smoothness	Smoothness
Precision rate	79.00 %	96.30 %	45.06 %	86.05 %
Recall Rate	77.70 %	96.89 %	44.79 %	84.66 %

Table 9.2: Precision and Recall outputs for the low frequency kernel from Thanou et al. dataset. Trial 2

Having a look in detail to the reproduction error, we can observe how in this general case the algorithm tends to behave better for the Thanou et al. dataset, while it's almost the same for the heat kernel dataset. Table 9.3 compares the reproduction error over 3 trials for both the datasets: it must be underlined how not only the smoothness prior improves the outcomes, but again the algorithm results in being more stable over the trials.

Moreover, for what concerns the kernels coefficients, figures 9.3 and 9.4 highlight one time more how the smoothness prior has a good repercussion over them, allowing to get more faithful behaviors.

Finally, in we examine the computational cost of the algorithm with and without the prior, we can see a general improvement of the average CPUTime per iteration, as shown in Table 9.4

Trial #	Heat kernel		Thanou et al. kernel	
	No Smoothness	Smoothness	No Smoothness	Smoothness
1	0.1114	0.1113	0.0225	0.0230
2	0.0310	0.1131	0.0384	0.0229
3	0.1024	0.1100	0.0239	0.0224
Average Error	0.0816	0.1115	0.0283	0.0227

Table 9.3: Reproduction error comparison for the low frequency kernel from both the datasets

Trial #	Heat kernel		Thanou et al. kernel	
	No Smoothness	Smoothness	No Smoothness	Smoothness
1	0.0219	0.0164	0.0267	0.0128
2	0.0163	0.0171	0.0253	0.0123
3	0.0328	0.0166	0.0269	0.0155
Average CPUTime	0.0191	0.0167	0.0263	0.0135

Table 9.4: Computational cost comparison for both the datasets with an without smoothness priors

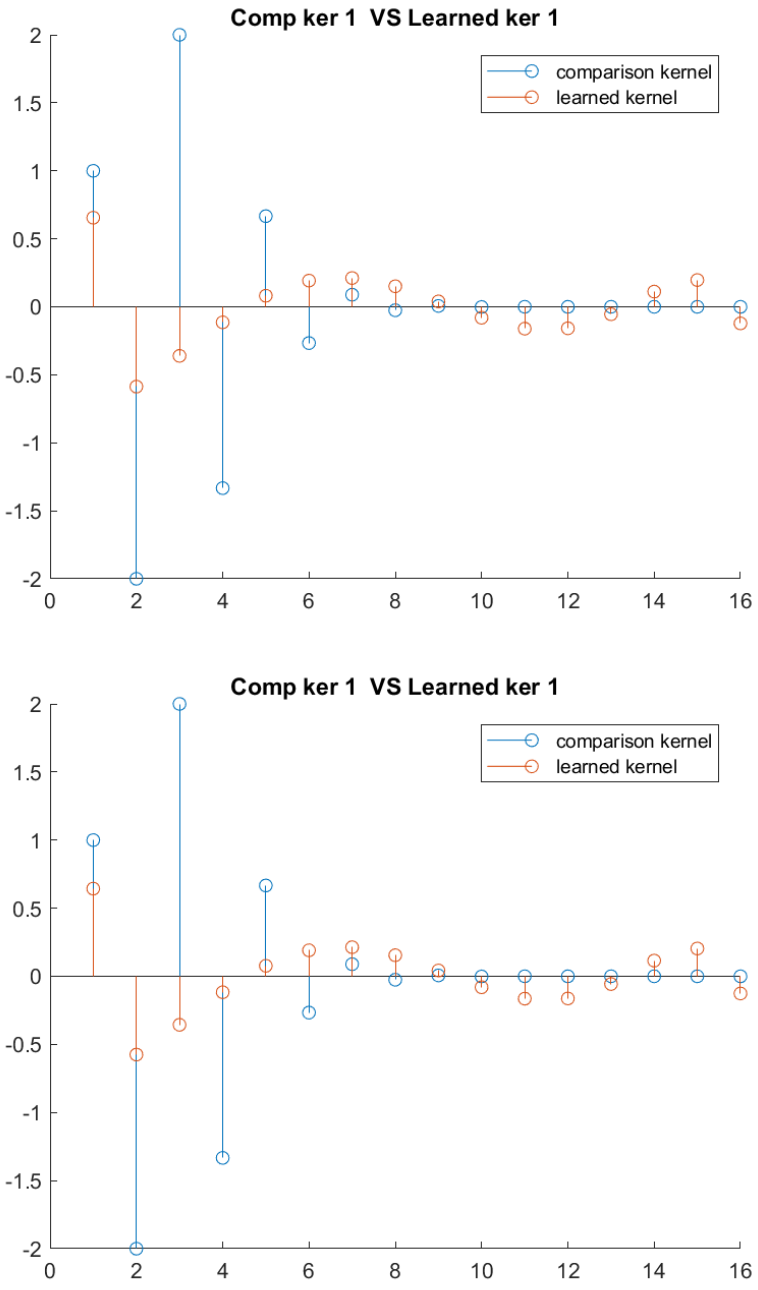


Figure 9.3: Comparison between kernels coefficients without and with smoothness prior. Heat kernel dataset

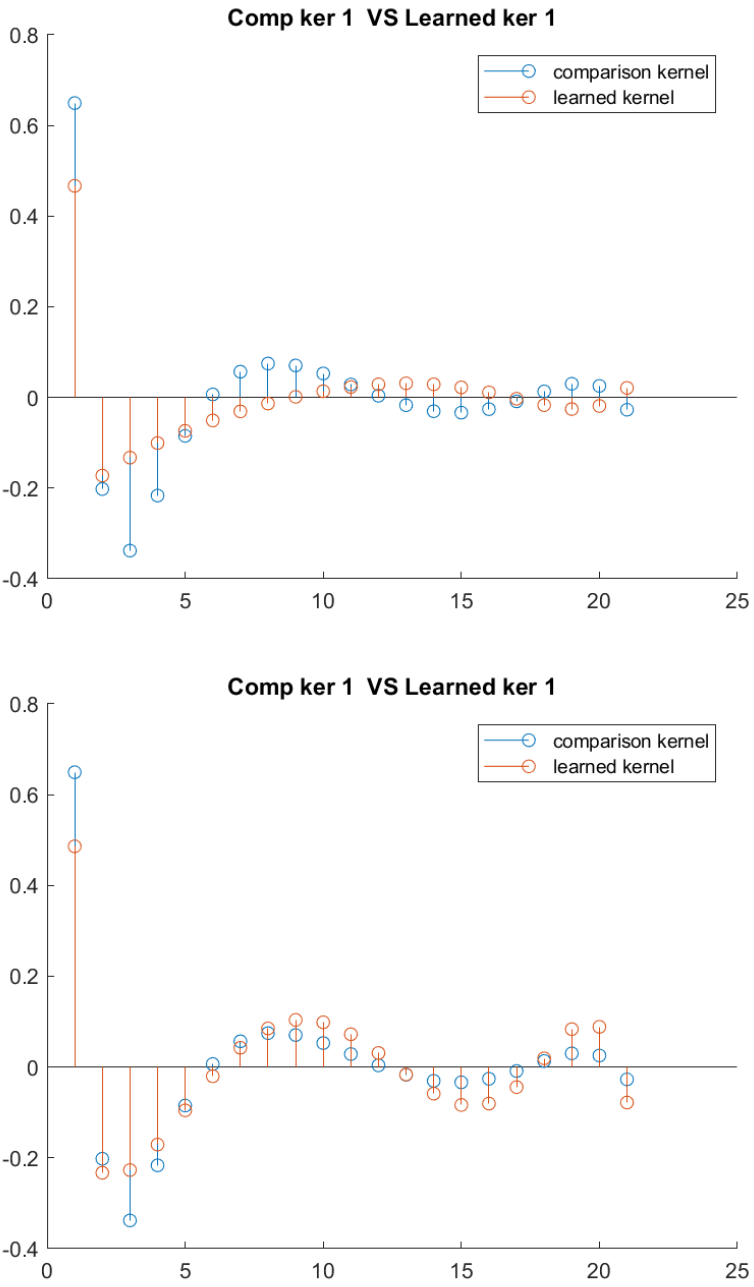


Figure 9.4: Comparison between kernels coefficients without and with smoothness prior. Thanou et al. dataset

Chapter 10

Main problems encountered during the work

During the work some problems emerged, underlying the weak points that this approach might have. In this section we will focus on some of them to explain the overall problem analysed in a more complete way.

10.1 A considerable amount of parameters

The parameters involved in the optimization problem are not a small quantity, since this has to constantly take into consideration:

- The number of eigenvalues being roots of the kernels polynomials;
- The value of σ in the constraints, indicating the deviation around the c value that the kernels could have;
- The number of non-root eigenvalues that has to be subject to the spectral constraints;
- The sparsity parameter;
- The number of iterations dedicated to the graph learning part and the number of them dedicated to the dictionary learning part;

This highlighted the need to spend some time on finding the right balance among them, in order for the linear programming solver to correctly go through the end and return a result without an unbounded objective function or infeasibility issues.

In this scenario, a particular attention was to be given to the σ value, the number of root eigenvalues and to the number of non-root ones involved in the spectral constraints. We noticed that imposing them in a non proper way brought almost directly to an infeasibility issue, and if this was not encountered clearly at first, then it was simply hidden under an unbounded objective function problem.
When the process had to deal with infeasibility, of course the learned kernels were not reflecting the priors of the problems, resulting in weird behaviors, like the ones shown in Figure 10.1 and for the α coefficients in Figure 10.2.

10.2 Working with the proper dataset

The choice of the dataset has not been an easy task: first of all not many datasets are already set in such a way that we could use them in a plug and play manner.
As an alternative to the ones we already presented we used other two datasets, the complete synthetic one from Thanou et al. and a dataset collecting the number of Uber rides occurring in an area of Manhattan, grouped by hour and in different times of the day. There was also an attempt to extrapolate a coherent dataset from a group of photos stored in the Flickr database [15], for example building a graph signal where the nodes were the locations of the pictures and the samples could have been the number of pictures taken from a single user in that location, repeated by the total number of users. This idea was later abandoned since the dataset obtained was not properly descriptive of the phenomena and the number of samples for the signal revealed to be insufficient for the training of the algorithm.

The problem with the other two datasets was the structure of the kernels: both Uber and the synthetic dataset were using not only low-frequency kernels, but also high frequency ones or even kernels having an ambiguous behavior, which can not be associated with either one (Figure 10.3). Therefore, in these cases we tried to modify the original signal in such a way that the learned dictionary could have returned only low frequency kernels. In particular, we tried to induce a well-know behavior either in the signal matrix or in the weight matrix, like for example through smoothing the values following a certain gradient, to see

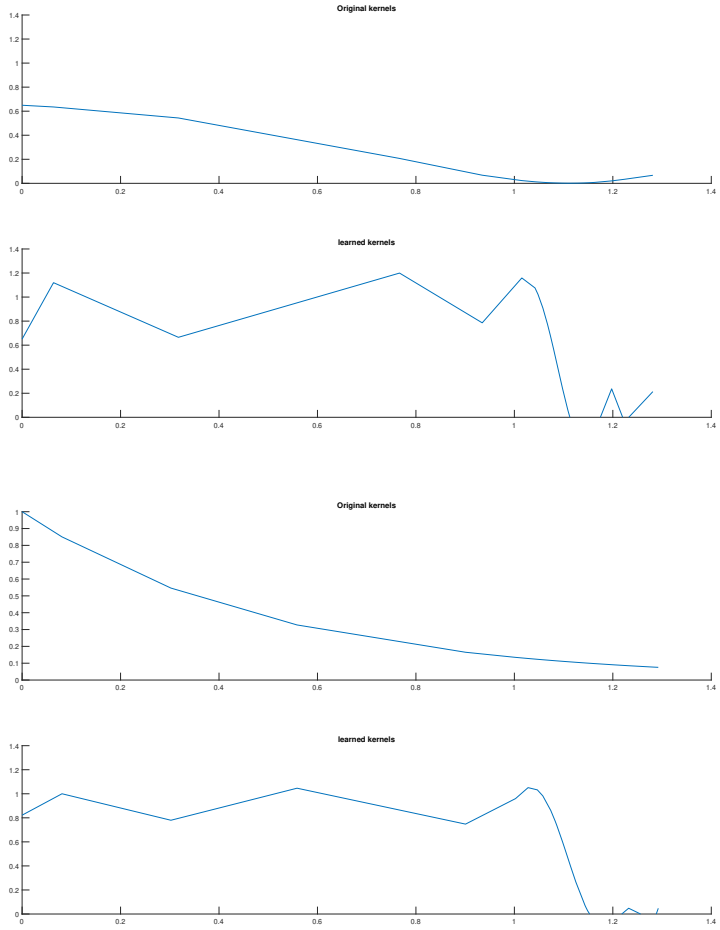


Figure 10.1: Comparison between the original kernel and the kernel learned with infeasibility problems

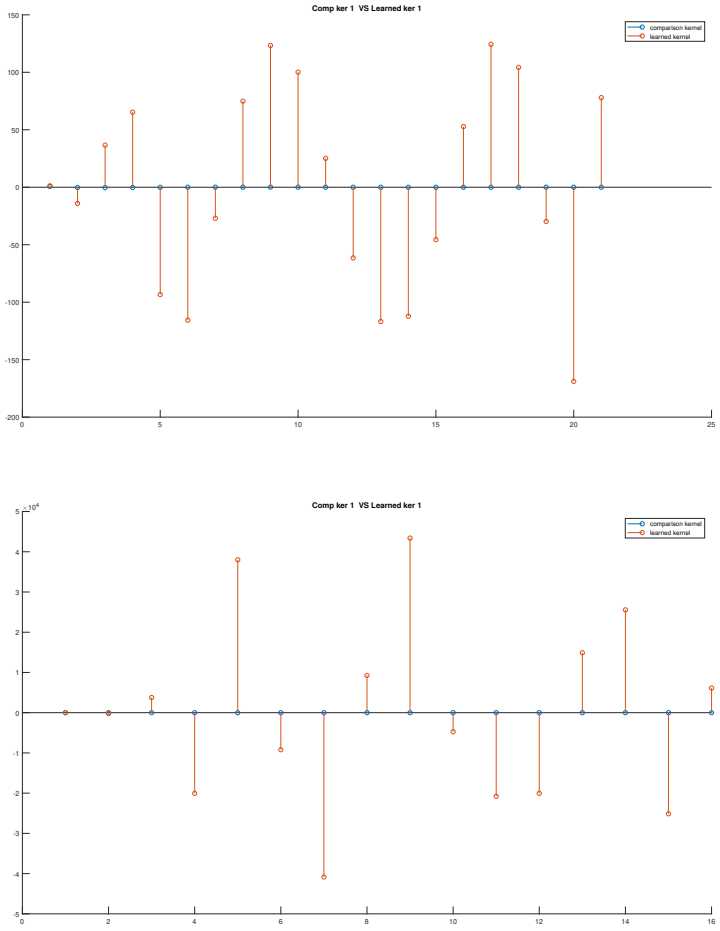


Figure 10.2: Comparison between the original α coefficients and the coefficients learned with infeasibility problems, for the Heat kernel and the synthetic dataset from Thanou et al.. It can be seen how now the learned αs are several times bigger than the magnitude of the original coefficients, these ones resulting compressed around 0 in the representation.

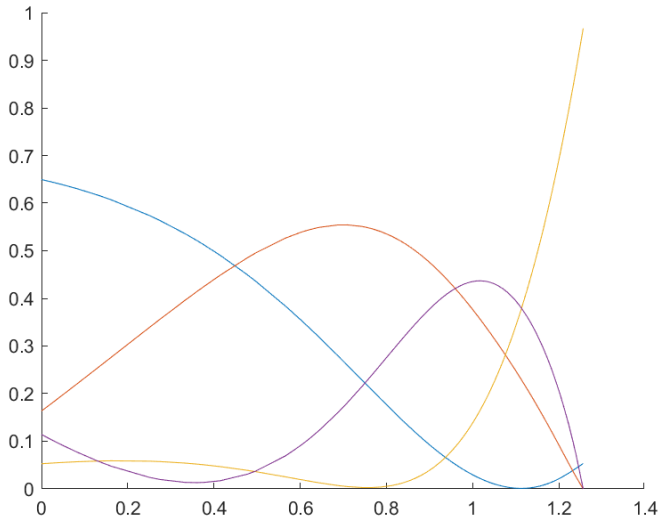


Figure 10.3: The original kernels learned from the synthetic dataset of Thanou et al.. It can be seen how the orange kernel does not have a clear behavior in terms of high or low frequency

if we would have detected a precise response in the learned kernels' structure. However the trials did not bring to meaningful achievements or significative changes in the kernels such that we could extrapolate a pattern.

10.3 The peculiar structure of the kernels

In order to deploy at least the real dataset from Uber rides, we tried to extend our hypothesis so to see if there could be a general improvement of the learning frame not only imposing the smoothness on the kernels, but also forcing in the algorithm a more general structure in such a way that it was able to follow all the kernel behaviors. In the case of Uber dataset, for example, the dictionary learned involves one low frequency and one high frequency kernel, as can be seen in figure 10.4, and we consequently modified the algorithm in order for it to "filter" both of them on the high and low frequencies respectively. The results obtained, however, show that the smoothness one is a much more effective prior than its counterpart working on the high frequencies and as a consequence the reproduction error was not reduced, but on the contrary was increasing.

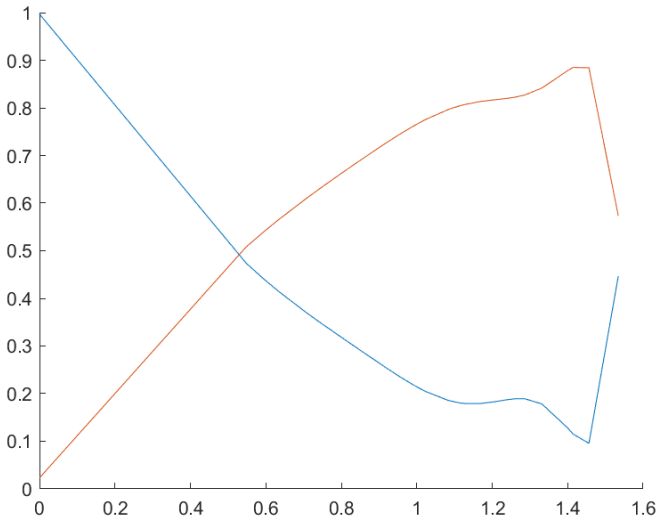


Figure 10.4: The original kernels learned from the Uber dataset.

10.4 Relaxing the spectral constraints

In order to make the optimization process more stable, we had to relax the constraints involved, in particular the spectral ones. At first we tried with relaxing and/or removing some of the bounds included in them in such a way:

$$\mathcal{D}_s \succeq 0 \qquad \text{and} \qquad \sum_{s=1}^{S} \mathcal{D}_s \succeq 0 \tag{10.41}$$

But through it we observed an increased imprecision in the results, as expected, such that the smoothness prior was not being beneficial as we desire. At the same time, when restoring this bounds, we had to be careful in order to make them consistent with the new smoothness priors we added: we could not simply impose the last values of the kernels to be 0 and in parallel require that all the other values were in a $(-\epsilon; +\epsilon)$ neighbourhood of c, since we would not have given the kernels the necessary margin to transition from that neighbourhood to 0. The result of this lack was, of course, infeasibility.

Chapter 11

Future improvements

11.1 The large scale approach

11.2 Incorporate the structure of an atom

11.3 graph signal clustering

11.4 varargin

Chapter 12

Conclusions

Acknoledgements

Bibliography

[1] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph Signal Processing," *IEEE Proceeedings*, pp. 1–18, 2017.

[2] A. Sandryhaila and J. M. Moura, "Big Data Analysis with Signal Processing on Graphs," *IEEE Signal Processing Magazine*, no. September, pp. 80–90, 2014.

[3] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[4] H. P. Maretic, D. Thanou, and P. Frossard, "Graph learning under sparsity priors," *CoRR*, vol. abs/1707.05587, 2017.

[5] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning Laplacian Matrix in Smooth Graph Signal Representations," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6160–6173, 2016.

[6] D. I. Shuman, B. Ricaud, and P. Vandergheynst, "Vertex-frequency analysis on graphs," *Applied and Computational Harmonic Analysis*, vol. 40, no. 2, pp. 260–291, 2016.

[7] V. Kalofolias, "How to learn a graph from smooth signals," *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, vol. 2, 2016.

[8] X. Zhu and M. Rabbat, ""Approximating signals supported on graphs"," *Signals, Department of Electrical and Computer Engineering, McGill University 3480 University St , Montreal QC , Canada H3A 2A7*, vol. 3, pp. 3921–3924, 2012.

[9] I. Tosic and P. Frossard, "Dictionary Learning, What is the right representation for my signal?," *Signal Processing Magazine, IEEE*, vol. 28, no. 2, pp. 27–38, 2011.

[10] B. R. Rubinstein, A. M. Bruckstein, and M. Elad, "Dictionaries for Sparse Representation Modeling.pdf," *IEEE Proceedings, VOL. X, NO. X*, vol. 98, no. 6, pp. 1045–1057, 2010.

[11] I. F. Gorodnitsky and B. D. Rao, "Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, pp. 600–616, 1997.

[12] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 40–44, 1993.

[13] D. Thanou, D. I. Shuman, and P. Frossard, "Learning parametric dictionaries for signals on graphs," *IEEE Transactions on Signal Processing*, vol. 62, no. 15, pp. 3849–3862, 2014.

[14] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[15] J. McAuley and J. Leskovec, "Image labeling on a network: Using social-network metadata for image classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7575 LNCS, no. PART 4, pp. 828–841, 2012.