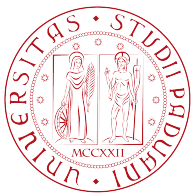



**Università degli Studi di Padova**

Department of Information Engineering

*Master Thesis in* Telecommunication Engineering


# Graph Signal Processing: Reconstruction Algorithms


*Supervisor*
Tomaso Erseghe
Università di Padova

*Master Candidate*
Marco Ceccon



28 February 2017

Academic Year 2016/2017

# Abstract

In the last years we have been experiencing an explosion of information generated by large networks of sensors and other data sources. Much of this data is intrinsically structured, such as traffic evolution in a transportation network, temperature values in different geographical locations, information diffusion in social networks, functional activities in the brain, or 3D meshes in computer graphics. The representation and analysis of such data is a challenging task and requires the development of new tools that can identify and properly exploit the data structure.

In this thesis, we formulate the processing and analysis of structured data using the emerging framework of graph signal processing. Graphs are generic data representation forms, suitable for modeling the geometric structure of signals that resides on topologically structured domains. The vertices of the graph represent the discrete data domain, and the edge weights capture the pairwise relationships between the vertices. A graph signal is then defined as a function that assigns a real value to each vertex. Graph signal processing is a useful framework for handling efficiently such data as it takes into consideration both the signal and the graph structure.

In this work, we study the common features and properties of signals defined on graphs and we focus on a specific application related to the reconstruction of graph signals in both centralized and distributed settings.

# Sommario

Negli ultimi anni abbiamo sperimentato una esplosione di informazioni generate da grandi reti di sensori e da altre fonti di dati. Gran parte di questi dati hanno una struttura intrinseca, come l'evoluzione del traffico in una rete di trasporto, i valori di temperatura in diverse località geografiche, la diffusione di informazioni nelle reti sociali, le attività cerebrali, o superfici tridimensionali in computer grafica. La rappresentazione e l'analisi di tali dati è un compito impegnativo e richiede lo sviluppo di nuovi strumenti in grado di identificare e sfruttare correttamente la struttura dei dati.

In questa tesi, impostiamo l'elaborazione e l'analisi di dati strutturati che utilizzano il contesto emergente dell'elaborazione di segnali definiti su grafi. I grafi sono forme di rappresentazione di dati generiche, adatte per modellare la struttura geometrica di segnali che risiedono in domini topologicamente strutturati. I vertici del grafo rappresentano i dati in un dominio discreto, e i pesi dei lati del grafo esprimono le relazioni tra vertici connessi. Un segnale su un grafo viene quindi definito come una funzione che assegna un valore reale a ciascun vertice. L'elaborazione dei segnali definiti su grafi è un contesto utile per la gestione efficiente di tali dati, che tiene in considerazione sia il segnale e la struttura del grafo.

In questo lavoro, studiamo le principali caratteristiche e proprietà dei segnali definiti su grafi e ci concentriamo su una specifica applicazione relativa alla ricostruzione dei segnali definiti su grafi, sia in contesti centralizzati che distribuiti.

# Contents


Abstract **ii**

List of figures **ix**

List of tables **xi**

Listing of acronyms **xiii**

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Thesis Outline . . . 4

2 Graph Signal Processing Overview **5**
- 2.1 Introduction . . . 5
- 2.2 Graphs And Signals On Graphs . . . 6
- 2.3 Graph Spectral Domain . . . 11
- 2.4 Applications Of Graph-Based Signal Processing . . . 19
  - 2.4.1 Processing with graph-based priors . . . 19
  - 2.4.2 Distributed Processing Of Graph Signals . . . 20
  - 2.4.3 Graph-Based Multimedia Processing . . . 23

3 Graph Signal Reconstruction **27**
- 3.1 Reconstruction Problem . . . 27
- 3.2 Random Sampling and Frequency Ordering . . . 34
- 3.3 Performance of the LS Reconstruction Algorithm . . . 36
- 3.4 $\ell_1$ Regularization Sparsity . . . 42

4 Distributed Algorithms **51**
- 4.1 Average Consensus . . . 51
- 4.2 ADMM: Alternating Direction Method of Multipliers . . . 55
  - 4.2.1 Distributed ADMM - LS Solution . . . 58
  - 4.2.2 Distributed ADMM - $\ell_1$ Regularization Solution . . . 66

5 Conclusions And Future Work **79**
- 5.1 Future Work . . . 80

References **82**

# Listing of figures


2.1 A graph defined on 50 nodes . . . . . 6
2.2 Gershgorin circles for Metropolis weights . . . . . 8
2.3 Gershgorin circles for unweighted adjacency matrix . . . . . 9
2.4 Random signal defined on 50 nodes . . . . . 10
2.5 Eigenvectors $u_0$, $u_1$, $u_2$ and $u_{49}$ . . . . . 11
2.6 Number of zero-crossings . . . . . 12
2.7 Exponential kernel and its IGFT . . . . . 14
2.8 Translated versions of kernel signal . . . . . 16

3.1 Temperature for 2 different months . . . . . 31
3.2 An example of signal sampling . . . . . 34
3.3 Frequency ordering . . . . . 35
3.4 MSE for different weighting methods . . . . . 39
3.5 MSE for different weighting methods . . . . . 40
3.6 LS signal reconstruction for month March, Metropolis weights . . 41
3.7 LASSO and ridge constraint comparison . . . . . 44
3.8 $\ell_1$-norm signal reconstruction for month March, Metropolis weights 46
3.9 MSE for $\ell_1$-norm problem, different $\lambda$ . . . . . 46
3.10 MSE for $\ell_1$-norm problem, different number of samples . . . . . 47
3.11 MSE comparison, 99 frequencies, $\lambda = 1$ . . . . . 48
3.12 MSE comparison, 10 frequencies, $\lambda = 1$ . . . . . 49

4.1 Convergence of average consensus algorithm . . . . . 54
4.2 Convergence of the ADMM solution - LS . . . . . 63
4.3 Convergence of the ADMM variables - LS . . . . . 64
4.4 Average time to run ADMM - LS . . . . . 65
4.5 Convergence of the ADMM solution - $\ell_1$-norm regularization (10 freq.) . . . . . 69
4.6 Convergence of the ADMM solution - $\ell_1$-norm regularization (99 freq.) . . . . . 70
4.7 Convergence of the ADMM solution - comparison . . . . . 71
4.8 Average time to run ADMM - $\ell_1$-norm regularization . . . . . 72
4.9 Comparison of distributed algorithms - 1 . . . . . 73
4.10 Convergence of ADMM, LS solution, updated penalty parameter ($\epsilon_0 = 0.001$) . . . . . 74

4.11 Convergence of ADMM, LS solution, updated penalty parameter ($\epsilon_0 = 0.01$) . . . . . 75
4.12 Convergence of ADMM, LASSO solution, updated penalty parameter ($\epsilon_0 = 0.001$) . . . . . 76
4.13 Convergence of ADMM, LASSO solution, updated penalty parameter ($\epsilon_0 = 0.01$) . . . . . 77
4.14 Comparison of distributed algorithms - 2 . . . . . 78

# Listing of tables

3.1 Temperature dataset, index 1:33 . . . . . 28
3.2 Temperature dataset, index 34:66 . . . . . 29
3.3 Temperature dataset, index 67:99 . . . . . 30

# Listing of acronyms

**ADMM** Alternating Direction Method of Multipliers

**ATC** Adapt To Combine

**DCT** Discrete Cosine Transform

**DFT** Discrete Fourier Transform

**GFT** Graph Fourier Transform

**IGFT** Inverse Graph Fourier Transform

**LASSO** Least Absolute Shrinkage and Selection Operator

**LS** Least Square

**MSE** Mean Squared Error

**SVM** Support Vector Machine

**WSN** Wireless Sensor Network

# 1 Introduction

Modern information processing inevitably involves an extremely large volume of increasingly complex data. The complexity comes, in particular, from the intrinsic structure of the framework on which these data resides. Data observed by different sensors could be intrinsically related by some structures, where the data could represent different kind of information. For instance, temperatures observed at different regions are related to their geographical proximities, traffic volumes at different locations in a transportation network depends on the topology of the network, and behaviour of a group of persons may be influenced by the friendships among them. To handle such complex data efficiently, we need to understand the interactions between different sources of information as well as the relationships and structures among them.

Graphs are powerful mathematical tools to model relationships and structures of the data. In a graph representation, the vertices represent the entities and the edges represent the pairwise relationships between these entities. Moreover, graph-based data are flexible and adaptable to incorporate multiple information with relationships and structures among them, yet remaining sufficiently simple for efficient processing: we can think to the temperature of different sensors, taken at different time instants.

Signal processing on graphs is an emerging research field which has recently attracted growing interests in the signal processing community. In this setting, the

vertices of the graph represent entities and the edge weights reflect the pairwise relationships between them, while a graph signal assigns a scalar value to each vertex based on some observation associated with the entities. Graph signals capture the relationships between the observations, thus reflect the structures in the data, and they can represent a various sources of information. Numerical examples of graph signals can be found in geographical, transportation, biomedical and social networks, such as temperatures within a geographical area, traffic capacities at hubs in a transportation network, or human behaviour in a social network.

## 1.1 Motivation

Over the past few years, we have attended so many information generated by numerous data sources, in a large variety of applications. For example, sensor networks have been widely deployed to measure a plethora of physical entities, like temperature and solar radiation, traffic volumes in transportation networks, brain activities in biological networks. Online social networks have turned into a significant means of communication and contain a lot of information. 3D depth cameras are yet becoming more powerful and widely used to capture dynamic 3D scenes in emerging applications such as gaming, immersive communication and virtual reality. Such data are usually very complex since they are high-dimensional and occupy a large amount of storage space. Furthermore, data are intrinsically and possibly irregularly structured. For instance, wireless sensor networks are irregularly deployed in space and their measurements depend on their geographical positions. Also, data and structure may be generated by different sources of information. For example, the information spread in social networks may be influenced by the relationships between the entities, as well as the type of data itself. The representation, analysis, and compression of such data is a challenging task that requires the development of new tools that can identify and properly exploit data structures.

In this thesis, we study the representation and analysis of structured data in the context of the emerging graph signal processing framework. Graphs are generic data representation forms that are suitable for modeling the geometric structure of signals that live on topologically complicated domains, including social networks, electricity networks, transportation networks, and sensor networks, where data

naturally reside on the vertices of weighted graphs. These signals are either intrinsically discrete (e.g., attributes of entities in social networks) or sampled from a continuous process. Typically, the vertices of the graph represent the discrete data domain and carry the data values. The edge weights of the graph capture the pairwise relationships between the vertices, like geographical distance or biological connections, for example. A graph signal is then defined as a function that assigns a real value to each vertex.

The weight associated with each edge in the graph often represents the similarity between the two vertices it connects. The connectivities and edge weights are either dictated by the physics of the problem or inferred from the data. For instance, the edge weight may be related to the physical distance between nodes in the network, or it may be related to the degrees of the connected vertices (that is, the number of edges connected to the same vertex). The data on these graphs can be visualized as a finite collection of samples, with one sample at each vertex of the graph.

Graph representations lead to rich data description on irregular domains and, if properly exploited, permit to efficiently capture the evolution of signals in a priori complex high-dimensional data sets. Signals and graphs are usually defined using different types of information which, if combined properly, can be quite helpful in analyzing or inferring information in the datasets. Moreover, graph signal representations provide a natural way to handle signals that cannot be easily processed with classical tools due to their irregular structure. The price to pay for this flexibility is the fact that one has to develop new tools and algorithms that handle efficiently the graph structure, possibly by leveraging intuition from classical signal processing in Euclidean spaces. Adapting classical signal processing tools to signals defined on graphs has however raised significant interest in the last few years. It requires the combination of different fields such as algebraic and spectral graph theory, harmonic analysis, and application domain expertise. Even if this research area looks highly promising because it provides a framework for modeling complex and irregularly structured discrete datasets, the challenges are many and the field is still in expansion.

## 1.2 Thesis Outline

The goal of this thesis is to present solutions as well as analysis of a few of the most important issues that arise in the emerging field of graph signal processing.

The thesis is organized as follows:

- Initially, we review in chapter 2 the current state-of-the-art methods for graph signal representations and their applications in both centralized and distributed settings. First, we give the basic definitions and notation used in this thesis for graphs and signals on graphs, and we review the generalization of classical transforms, such as Fourier, to the irregular graph domain. Then, the chapter concludes with applications of graph signal processing in visual data representation, processing and compression.

- Chapter 3 introduces and studies a common application of graph signal processing, that is the reconstruction problem: given a sampled graph signal, defined only on a subset of the vertices of the graph, the main challenge consists in extract the missing part of the signal. To solve this problem, we propose two different algorithms, based on two different function that need to be minimized.

- In chapter 4 we propose distributed versions of the reconstruction algorithms. First, we present a simple interpretation of the solution, based on the average consensus algorithm. Then, we apply a more sophisticated method that involves the introduction of some auxiliary variables, that are minimized in an alternating fashion. Finally, we expose some numerical results of the distributed algorithms.

- Finally, Chapter 5 draws some conclusions and suggests possible future research topics that will continue the work of this thesis.

# 2

# Graph Signal Processing Overview

## 2.1 Introduction

In order to efficiently represent graph signals, it is necessary to take into account for the intrinsic geometric structure of the underlying graph. Signal characteristics, such as smoothness, depend on the irregular topology of the graph on which the signal is defined. Classical signal processing tools designed for regular signal structures are therefore inappropriate for the irregular structure in the graph setting. In the last years a lot of work has been dedicated to design new tools and algorithms that can handle efficiently the new challenges arising from the irregular structure of networks or other graph supports. These tools are based on a combination of computational harmonic analysis with algebraic and spectral graph theoretical concepts [1].

In this chapter, we review principal graph signal processing methods from the literature, which are related to the problems studied in this thesis. First, we give some basic definitions and notation for graphs and signals on graphs, that will be used in the rest of the thesis. Next, we review the generalization of classical transforms such as Fourier to the irregular graph domain. In the sequel, we focus on the use of graph-based signal processing tools in different applications. In particular, we focus on graph signals reconstruction and distributed processing. Finally,

we quick review the use of graph-based signal processing tools for image and 3D data, which represent a popular application area for this emerging framework.

## 2.2 Graphs And Signals On Graphs

In this section, we briefly recall a few basic definitions for signals on graphs. We generally consider a connected, weighted and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{A})$ where $\mathcal{V}$ and $\mathcal{E}$ represent the vertex and edge sets of the graph respectively, and $\boldsymbol{A}$ represents the weighted adjacency matrix, with $A_{ij} = A_{ji}$ (since the graph is undirected) denoting the weight of the edge connecting vertices $i$ and $j$. If there is not an edge between node $i$ and node $j$, we assume $A_{ij} = 0$. The degree of a node $i$ is defined as the sum of the weight of the edges incident on that node, that can be computed as the sum of the weight values in the $i$-th row of the weighted adjacency matrix $A$. We assume that the graph is connected and that it consists of $N$ nodes. The $n$-hop neighborhood $\mathcal{N}(i, n) = \{v \in \mathcal{V} : d(v, i) \leq n\}$ of node $i$ is the set of all nodes that are at most $n$-hop away from node $i$.

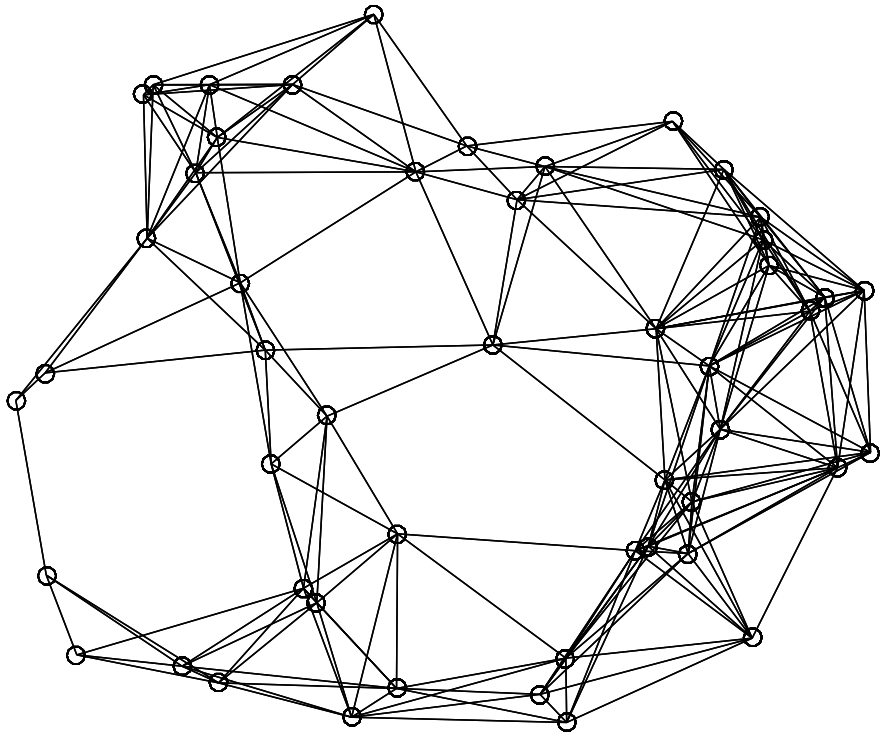


**Figure 2.1:** A graph defined on 50 nodes

The combinatorial graph Laplacian operator, also called the non-normalized graph Laplacian, is defined as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \tag{2.1}$$

where $\boldsymbol{D}$ is the diagonal degree matrix whose $i$-th diagonal element is equal to the degree of node $i$ (the sum of the weights of all the edges incident to vertex $i$ [2]) and $\boldsymbol{A}$ is the weighted adjacency matrix. It is a positive semi-definite matrix that has a complete set of real orthonormal eigenvectors with corresponding non-negative eigenvalues. We denote its eigenvectors by $\{u_\ell\}_{\ell=0,1,\ldots,N-1}$, and the associated real, non-negative sorted spectrum of eigenvalues by

$$\sigma(\boldsymbol{L}) = \{0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{N-1}\}$$

satisfying $\boldsymbol{L}u_\ell = \lambda_\ell u_\ell$ for $\ell = 0, 1, \ldots, N-1$. Then we can write the Laplacian eigendecomposition $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U}$ collects all the eigenvectors of $\boldsymbol{L}$ in its columns, whereas $\boldsymbol{\Lambda}$ contains the eigenvalues of $\boldsymbol{L}$. The spectral properties of matrix $\boldsymbol{L}$, that is its eigenvalues and eigenvectors structure, are of particular importance to study the behaviour in the spectral domain. The following useful results are taken from non-negative matrix theory, [3].

According to the Gershgorin circle theorem, the eigenvalues of the Laplacian $\boldsymbol{L}$ of a graph $\mathcal{G}$ are located inside the discs in the complex plane with centers in $L_{ii}$ and radius given by the row-sums $\sum_{j=1, j\neq i}^{N} |L_{ij}|$ for each $i$, where $|\cdot|$ denotes absolute value. Since by definition the diagonal entries of $\boldsymbol{L}$ are non-negative and all row-sums are equal to zero, the Gershgorin circles are tangent to the imaginary axis at zero. Fig. 2.2 visualizes an example of Gershgorin circles for the Laplacian in the complex plane, if we consider the weighted adjacency matrix by taking the Metropolis weights as defined in (3.10).

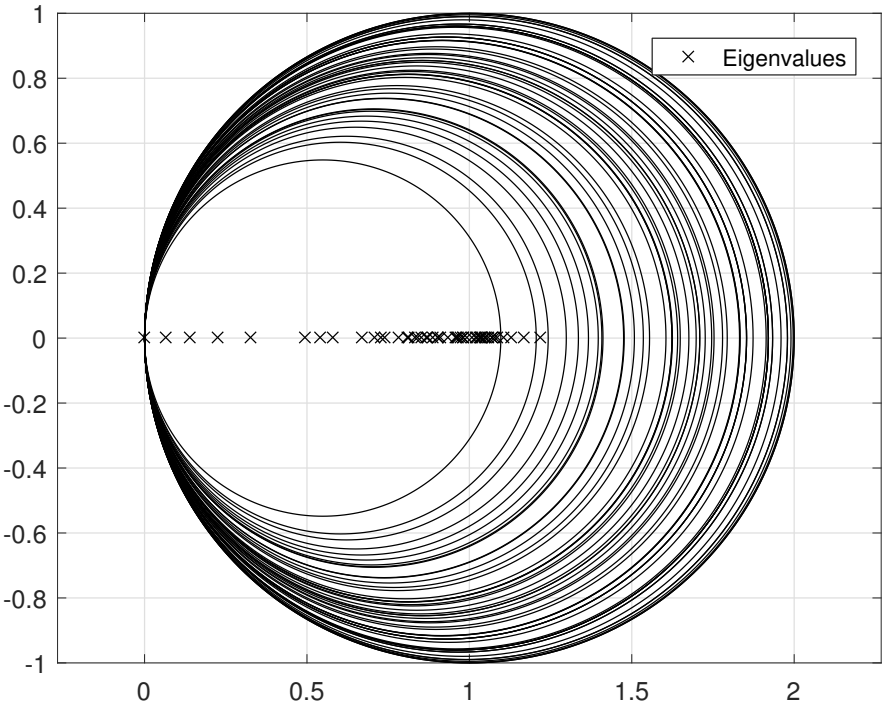



**Figure 2.2:** Gershgorin circles for Metropolis weights

Therefore, the eigenvalues of $\boldsymbol{L}$ have non-negative real parts and are all inside a circle of radius $2d_{max}$ where $d_{max}$ is the maximum degree over all vertices. In this case we can see that the eigenvalues, in the case of Metropolis weight adjacency matrix, are limited within a circle of radius 1 centered in $[1, 0]$. In the case of unweighted adjacency matrix, defined in (3.15), we can see from Fig.2.3 that the eigenvalues are restricted inside a circle of bigger radius, because the adjacency matrix is not normalized as in the Metropolis case.

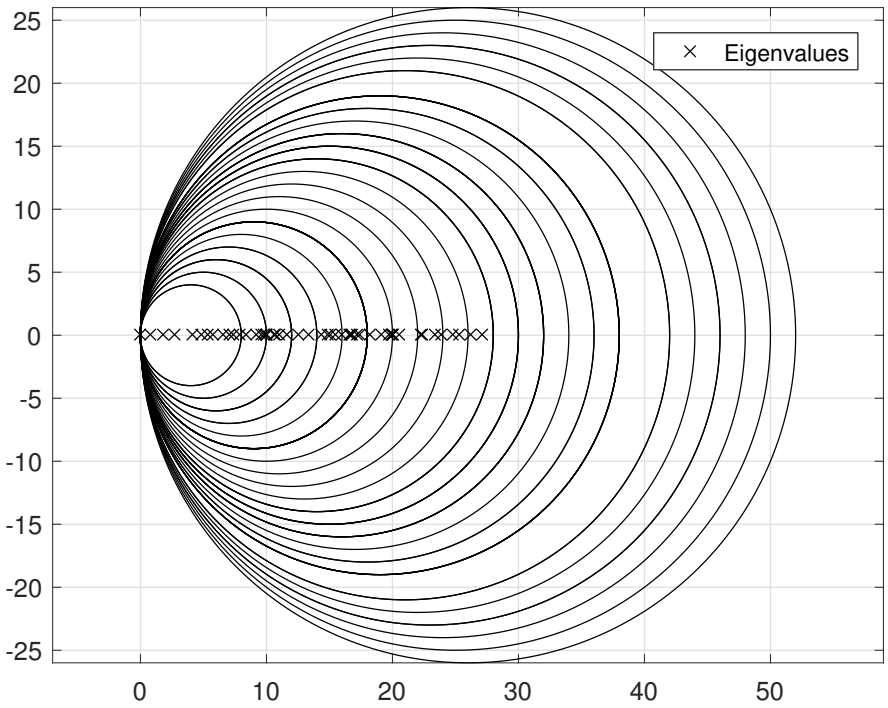



**Figure 2.3:** Gershgorin circles for unweighted adjacency matrix

Since $\boldsymbol{L} \cdot \mathbf{1} = \mathbf{0}$, where $\mathbf{0}$ is a zero vector of length $N$, the smallest eigenvalue of the non-normalized Laplacian is always zero and its multiplicity is equal to the number of connected components of the graph, and the corresponding eigenvector is a constant vector. The largest eigenvalue depends on the maximum degree of the graph. Moreover, the combinatorial Laplacian is associated with the incidence matrix, as shown in [2].

For connected graphs, the normalized graph Laplacian is closely related to the combinatorial Laplacian and is defined as

$$\boldsymbol{\mathcal{L}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}} \tag{2.2}$$

where $\boldsymbol{I}$ is the identity matrix. As in the case of the non-normalized Laplacian, the eigenvalues are non-negative, with the smallest one equal to zero. A nice property of the eigenvalues of the normalized Laplacian is that they are contained between the interval $[0, 2]$, which makes it easier to compare the distribution of the eigenvalues between different graphs, especially if there is a large difference in the number of vertices; the maximum value $\lambda_{max} = 2$ is reached if and only

if $\mathcal{G}$ is bipartite, i.e. the set of vertices $\mathcal{V}$ can be partitioned into two subsets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that every edge $e \in \mathcal{E}$ connects one vertex in $\mathcal{V}_1$ and one vertex in $\mathcal{V}_2$. Furthermore, the normalized Laplacian eigenvalues are consistent with the eigenvalues in the spectral geometry and in stochastic processes, such as random walks [2].

The combinatorial and the normalized graph Laplacians are both examples of generalized graph Laplacians [4] and they are both popular in many graph related frameworks. In general, when the graph is almost regular, the combinatorial and the normalized Laplacian have similar spectra. In these thesis we mainly use the combinatorial graph Laplacian and we focus only on undirected graph. For the sake of completeness, we note that the definition of the Laplacian can be easily extended to directed graphs [5].

A graph signal $y$ in the vertex domain is a real-valued function defined on the vertices of the graph $\mathcal{G}$, such that $y(n)$ is the value of the function at vertex $n \in \mathcal{V}$. An example of a graph and a signal on the graph is given in Fig. 2.4. This signal is generated randomly from a Gaussian distribution, with zero mean and standard deviation of 5.

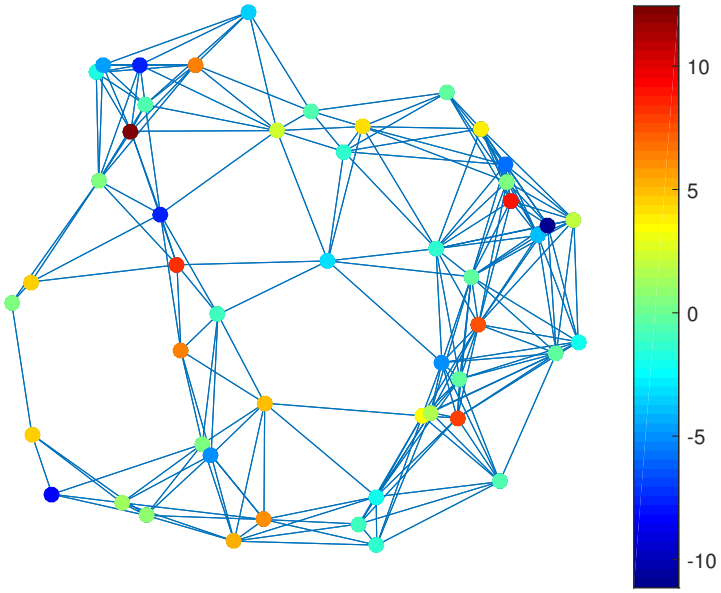



**Figure 2.4:** Random signal defined on 50 nodes

## 2.3 Graph Spectral Domain

The fundamental analogy between traditional signal processing and graph signal processing is established through the spectral graph theory [2]. In particular, the generalization of the classical Fourier transform to graph settings has been established through the eigenvectors and the eigenvalues of the graph Laplacian matrix [6], which carry a notion of frequency for graph signals. In particular, the graph Laplacian eigenvectors associated with small eigenvalues correspond to signals that vary slowly across the graph, hence they can be associated with the notion of low frequency. For connected graphs, the Laplacian eigenvector $u_0$ associated with the eigenvalue 0 is constant and equal to $\frac{1}{\sqrt{N}}$ at each vertex.

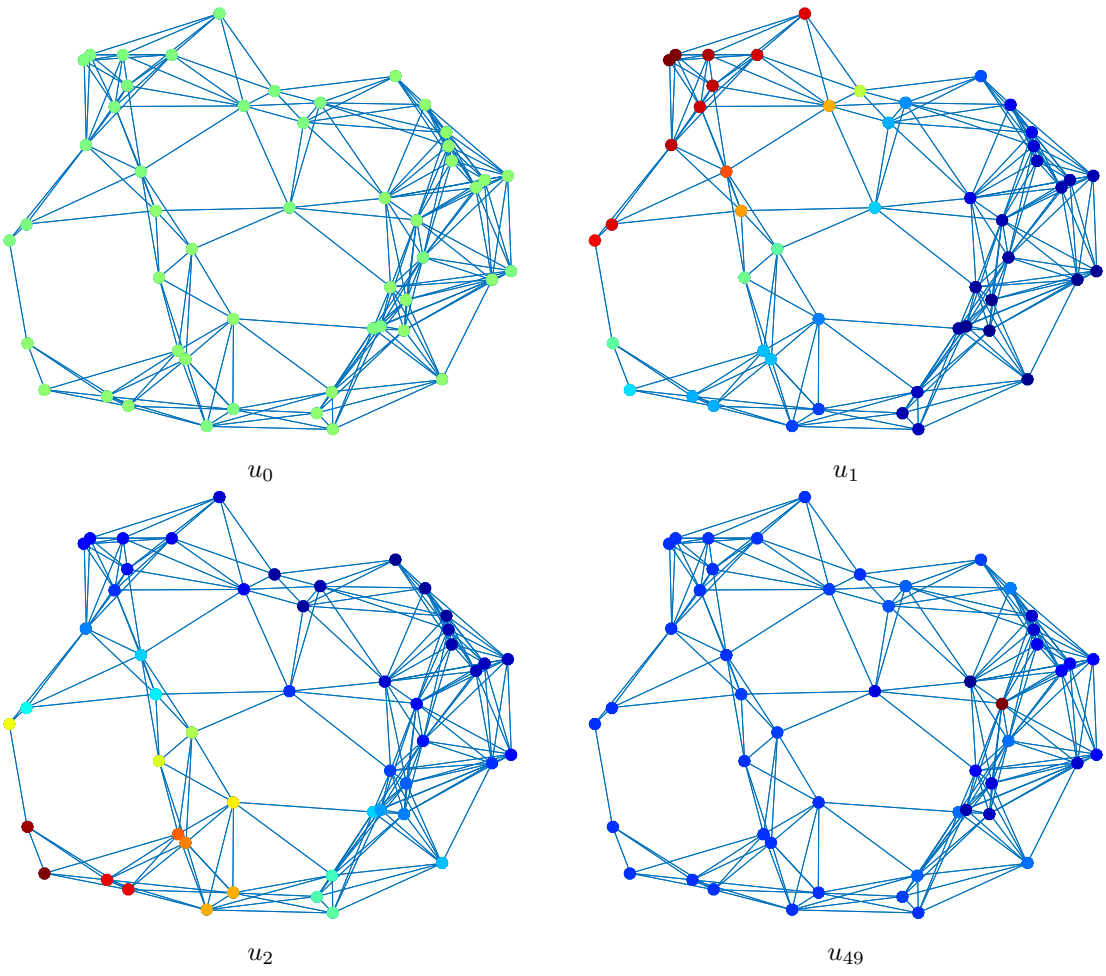



**Figure 2.5:** Eigenvectors $u_0$, $u_1$, $u_2$ and $u_{49}$

In other words, if two vertices are connected by an edge with a large weight, the values of the low frequency eigenvectors at those locations are likely to be similar. The eigenvectors associated with larger eigenvalues take values that change more rapidly on the graph: they are more likely to have dissimilar values on vertices connected by an edge with high weight. This is demonstrated in both Fig. 2.5, which shows different graph Laplacian eigenvectors for a random sensor network graph, and in Fig. 2.6, which shows the number of zero crossings of each graph Laplacian eigenvector. The set of zero crossings of a signal $y$ on a graph $\mathcal{G}$ is defined as

$$\mathcal{Z}_{\mathcal{G}}(y) = \{e = (i,j) \in \mathcal{E} : y(i)\, y(j) < 0\};$$

that is, the set of edges connecting a vertex with a positive signal to a vertex with a negative signal.

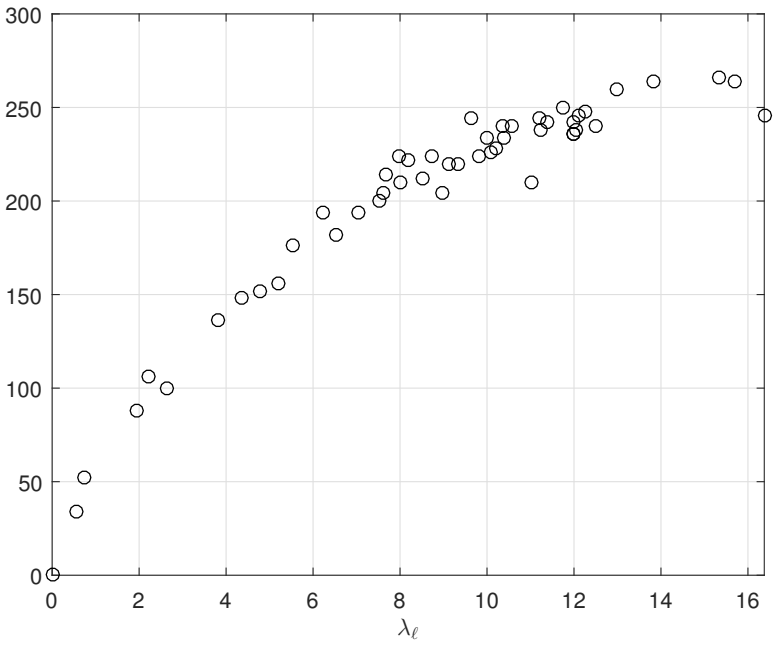



**Figure 2.6:** Number of zero-crossings

The eigenvectors of the graph Laplacian are therefore considered to represent a Fourier basis for graph signals. For any function $y$ defined on the vertices of the graph, the Graph Fourier Transform (GFT) $\hat{y}(\lambda_\ell)$ at frequency $\lambda_\ell$ is thus defined

as the inner product with the corresponding eigenvector $u_\ell$ [6]

$$\hat{y}(\lambda_\ell) = \langle y, u_\ell \rangle = \sum_{n=1}^{N} y(n)\, u_\ell^*(n) \tag{2.3}$$

where the inner product is conjugate-linear in the first argument, and $u_\ell^*(n)$ is the conjugate value of the eigenvector $u_\ell$ at node $n$. Therefore we can say that the GFT of a signal $y$ is $\hat{y} = \boldsymbol{U}^* y$.

The Inverse Graph Fourier Transform (IGFT) is

$$y(n) = \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\, u_\ell(n), \quad \forall n \in \mathcal{V}. \tag{2.4}$$

The Fourier basis can be chosen as the eigenvectors of either the combinatorial or the normalized graph Laplacian matrices, since both spectrums have a frequency-like interpretation [1]. We notice that, as in the classical Euclidean settings, the spectral domain representation provides important information about the graph signals. For example, analogously to the classical case, the graph Fourier coefficients of a smooth signal decay rapidly. Such signals are compressible as they can be closely approximated by just a sparse set of Fourier coefficients [7]. This property is used in many applications such as compression or regularization of graph signals.

The graph Fourier transform and its inverse give us a way to equivalently represent a signal in two different domains: the vertex domain and the graph spectral domain. While we often start with a signal in the vertex domain, it may also be useful to define a signal directly in the graph spectral domain. We refer to such a signals as kernels. In Fig. 2.7 one such signal, a heat kernel, is shown in both domains.

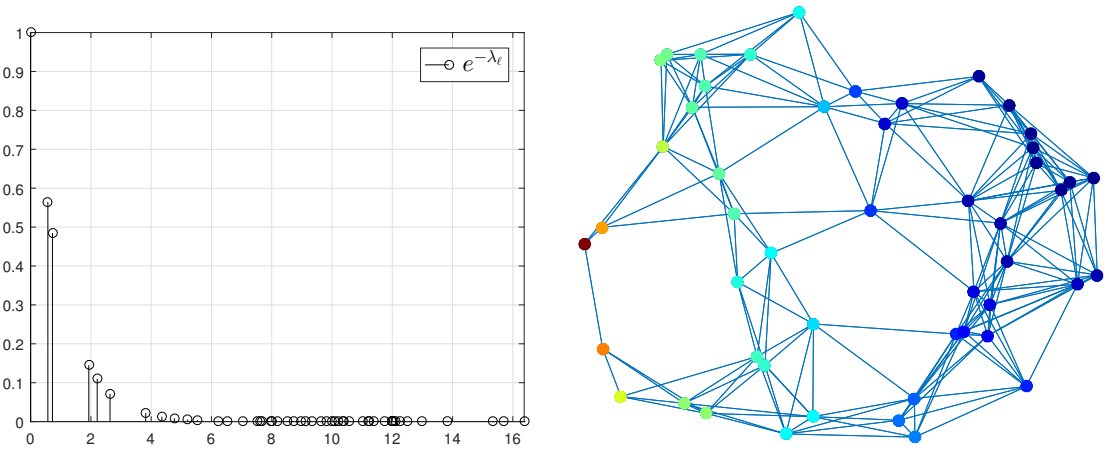



**Figure 2.7:** Exponential kernel and its IGFT

Analogously to the classical analog case, the graph Fourier coefficients of a smooth signal such as the one shown in Fig. 2.7 decay rapidly. Such signals are compressible as they can be closely approximated by just a few Fourier coefficients.

Besides its use in spectral analysis, the graph Fourier transform is also useful in generalizing traditional signal processing concepts such as convolution, translation, or modulation to graph settings. In particular, the relation between the vertex and the spectral graph domain has been used to define the convolution on the graph. Given two signals $y$ and $h$, the result of the convolution of these two signals on vertex $n$ is defined as [8, 6]

$$(y * h)(n) = \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\, \hat{h}(\lambda_\ell)\, u_\ell(n), \tag{2.5}$$

which imposes the property that the convolution in the vertex domain is equivalent to a multiplication in the graph spectral domain, as in the classical Euclidean settings.

The generalized convolution product defined in Eq.(2.5) satisfies the following properties, as discussed in [8]:

1. Generalized convolution in the vertex domain is multiplication in the graph spectral domain:

$$\widehat{f * h} = \hat{f}\hat{h}. \tag{2.6}$$

2. Let $\alpha \in \mathbb{R}$ be arbitrary. Then

$$\alpha (f * h) = (\alpha f) * h = f * (\alpha h) . \tag{2.7}$$

3. Commutativity:

$$f * h = h * f. \tag{2.8}$$

4. Distributivity:

$$f * (g + h) = f * g + f * h. \tag{2.9}$$

5. Associativity:

$$(f * g) * h = f * (g * h) \tag{2.10}$$

6. Define a function $h_0 \, in \mathbb{R}^N$ by $h_0 (i) := \sum_{\ell=0}^{N-1} u_\ell (i)$. Then $h_0$ is an identity for the generalized convolution product:

$$f * h_0 = f. \tag{2.11}$$

7. An invariance property with respect to the graph Laplacian (a difference operator):

$$\boldsymbol{L} (f * h) = (\boldsymbol{L} f) * h = f * (\boldsymbol{L} h) . \tag{2.12}$$

8. The sum of the generalized convolution product of two signals is a constant times the product of the sums of the two signals:

$$\sum_{i=1}^{N} (f * h) (i) = \sqrt{N} \hat{f} (0) \hat{h} (0) = \frac{1}{\sqrt{N}} \left[ \sum_{n=1}^{N} f (i) \right] \left[ \sum_{n=1}^{N} h (i) \right] . \tag{2.13}$$

The classical translation operator is defined through the change of variable $(T_v y) (t) = y (t - v)$, which cannot be generalized to graph settings. However, it is possible to define a generalized translation operator $T_v$ of a graph signal as a convolution with a Kronecker $\delta$ centered at vertex $v$ [9, 8, 6]:

$$T_v y (n) = \sqrt{N} (y * \delta_v) (n) = \sqrt{N} \sum_{\ell=0}^{N-1} \hat{y} (\lambda_\ell) u_\ell^* (v) u_\ell (n) \tag{2.14}$$

where

$$\delta_v(n) = \begin{cases} 1 & \text{if } n = v \\ 0 & \text{otherwise} \end{cases}$$

where the normalizing constant $\sqrt{N}$ ensures that the translation operator preserves the mean of the signal. The Kronecker function $\delta_v$ is an $N$-dimensional signal that is zero everywhere on the graph except from node $v$, where it takes the value of one. This is a kernelization operation, acting on a signal $\hat{y}$ defined in the graph spectral domain rather than translating a signal $y$ defined in the vertex domain. An example of the translation of a signal $y$ in different nodes of the graph is illustrated in Fig. 2.8. We can observe that the classical shift in the classical definition of the translation does not apply on graphs.

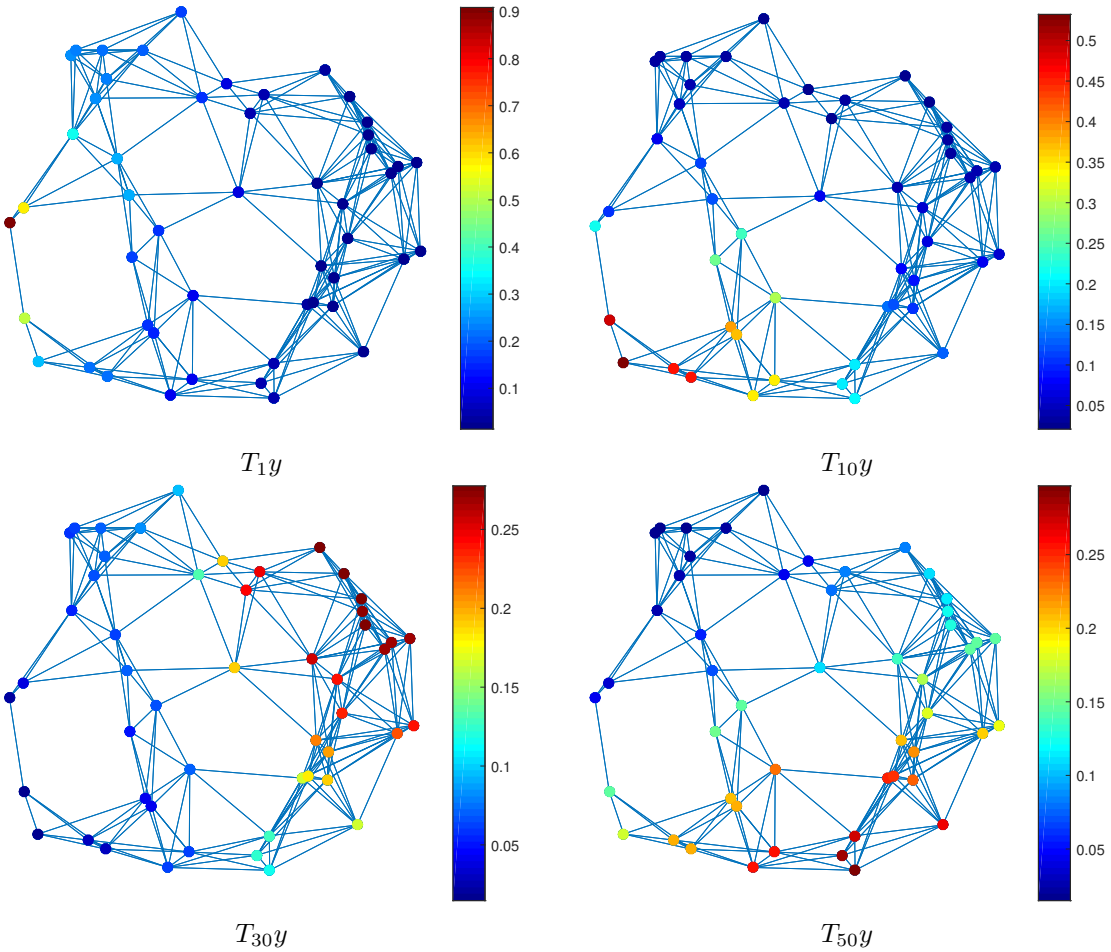



**Figure 2.8:** Translated versions of kernel signal

Some expected properties of the generalized translation operator follow immediately from the generalized convolution properties:

1. $T_i\left(f * h\right) = \left(T_i f\right) * h = f * \left(T_i h\right).$
2. $T_i T_j f = T_j T_i f.$
3. $\sum_{i=1}^{N} \left(T_i f\right)(i) = \sqrt{N}\,\hat{f}\,(0) = \sum_{i=1}^{N} f\,(i).$

Unlike the classical case, the set of translation operators $\{T_i\}_{i\in\{1,2,\ldots,N\}}$ does not form a mathematical group; i.e., $T_i T_j \neq T_{i+j}$. In the very special case of shift-invariant graphs [10], which are graphs for which the Discrete Fourier Transform (DFT) basis vectors are graph Laplacian eigenvectors, we have

$$T_i T_j = T_{[((i-1)+(j-1)) \mod N]+1}, \quad \forall i, j \in \{1, 2, \ldots, N\}. \tag{2.15}$$

However, Eq.(2.15) is not true in general for arbitrary graphs. Moreover, while the idea of successive translations $T_i T_j$ carries a clear meaning in the classical case, it is not a particularly meaningful concept in the graph setting due to our definition of generalized translation as a kernelized operator.

Filtering is another fundamental operation in graph signal processing. Similarly to classical signal processing, the outcome $y_{out}$ of the filtering of a graph signal $y$ with a graph filter $h$ is defined in the spectral domain as the multiplication of the graph Fourier coefficient $\hat{y}\left(\lambda_\ell\right)$ with the transfer function $\hat{h}\left(\lambda_\ell\right)$ such that

$$\hat{y}_{out}\left(\lambda_\ell\right) = \hat{y}\left(\lambda_\ell\right)\hat{h}\left(\lambda_\ell\right), \quad \forall \lambda_\ell \in \sigma\left(\boldsymbol{L}\right). \tag{2.16}$$

The filtered signal $y_{out}$ at node $n$ is given by taking the IGFT of $\hat{y}_{out}$ in (2.16):

$$y_{out}\left(n\right) = \sum_{\ell=0}^{N-1} \hat{y}\left(\lambda_\ell\right)\hat{h}\left(\lambda_\ell\right) u_\ell\left(n\right). \tag{2.17}$$

Eq.(2.17) can be expressed in matrix notation [11] as

$$y_{out} = \hat{h}\left(\boldsymbol{L}\right) y, \tag{2.18}$$

where

$$\hat{h}\left(\boldsymbol{L}\right)=\boldsymbol{U}\begin{bmatrix}\hat{h}\left(\lambda_0\right) & & 0\\ & \ddots & \\ 0 & & \hat{h}\left(\lambda_{N-1}\right)\end{bmatrix}\boldsymbol{U}^*$$

is a graph filter or kernel defined in the spectral domain of the graph.

Interestingly, when the graph filter is a polynomial of order $K$ with coefficients $\{\alpha_k\}_{k=0}^{K}$ such that

$$\hat{h}\left(\lambda_\ell\right)=\sum_{k=0}^{K}\alpha_k\lambda_\ell^k, \tag{2.19}$$

filtering in the spectral domain of the input signal $y\left(n\right)$ at node $n$ can be interpreted as a linear combination of the components of the input signal at vertices that are within a $K$-hop neighborhood of $n$. Combining Eqs. (2.18) and (2.19) we obtain

$$\begin{aligned}y_{out}\left(n\right)&=\sum_{\ell=0}^{N-1}\hat{y}\left(\lambda_\ell\right)\hat{h}\left(\lambda_\ell\right)u_\ell\left(n\right)\\&=\sum_{m=1}^{N}y\left(n\right)\sum_{k=0}^{K}\alpha_k\sum_{\ell=0}^{N-1}\lambda_\ell^k u_\ell^*\left(m\right)u_\ell\left(n\right)\\&=\sum_{m=1}^{N}y\left(m\right)\sum_{k=0}^{K}\alpha_k\left(\boldsymbol{L}^k\right)_{n,m}\end{aligned} \tag{2.20}$$

where $\left(\boldsymbol{L}^k\right)_{n,m}=0$ if the shortest-path distance between vertices $n$ and $m$ id greater than $k$ [6]. This property can be quite useful for designing signals that are localized in the vertex domain of the graph. A detailed overview of these basic operations can be found in [1].

## 2.4 Applications Of Graph-Based Signal Processing

In this section, we present some graph-based signal processing applications. We review some of the works from the literature that use graph-based tools to process graph signals in both centralized and distributed settings.

### 2.4.1 Processing with graph-based priors

Many of the representation methods of the previous section have been applied to different signal processing tasks such as denoising, semi-supervised learning, and classification. Similar to the traditional Euclidean domain, notions such as smoothness and sparsity have been used as regularizers for solving many inverse graph-based problems in both centralized and distributed settings.

The smoothness of the signal on the graph has been one of the core assumptions in semi-supervised learning with applications in classification, link prediction, and ranking problems. A signal is considered to be smooth on the graph if it exhibits little variations between strongly connected vertices. Typically, the notion of global smoothness $S_p(y)$ of a signal $y$ is defined through the discrete $p$-Dirichlet norm of $y$ as

$$S_p(y) = \frac{1}{p}\sum_{v\in\mathcal{V}} \|\nabla_v y\|_2^p = \frac{1}{p}\sum_{v\in\mathcal{V}} \left[\sum_{u\in\mathcal{N}_v} A_{vu}\left[y(v) - y(u)\right]^2\right]^{\frac{p}{2}}, \tag{2.21}$$

where $\mathcal{N}_v$ denotes the one-hop neighborhood of node $v$, and $A_{vu}$ is the edge weight between nodes $v$ and $u$. When $p = 1$, Eq.(2.21) defines the total variation of the signal $y$ on the graph. When $p = 2$, we have a widely-used Laplacian based form of smoothness defined as

$$\begin{aligned} S_2(y) &= \sum_{u,v\in\mathcal{E}} A_{vu}\left[y(v) - y(u)\right]^2 \\ &= y^T L y = \sum_{\ell=0}^{N-1} \lambda_\ell \hat{y}(\lambda_\ell)^2 . \end{aligned} \tag{2.22}$$

Eq.(2.22) implies that the signal $y$ is smooth, i.e., $S(y)$ is small, only if the graph Fourier coefficients corresponding to big eigenvalues are small. This definition of smoothness or similar notions have been imposed as regularizers in the graph-based semi-supervised learning literature, where the goal is to compute the unknown signal entries by exploiting the assumption that the signal values vary slowly between nodes that are connected by strong edges. The extension to more sophisticated regularization techniques has been developed through the definition of kernels on graphs that are typically of the form of the power series of the graph Laplacian. Recently, a framework for active semi-supervised learning based on sampling theory for graph signals has been introduced and is based on the above notion of smoothness of signals on the graph.

While smoothness priors have been widely employed, the use of sparse prior for graph signals has been mostly overlooked so far. The reason is that the link between sparsity and signal structure is not well understood in graph settings. However, there are still some works that try to exploit sparsity in learning applications. For example, the sparsity of the Fourier coefficients has been exploited for the reconstruction of bandlimited graph signals.

### 2.4.2 Distributed Processing Of Graph Signals

The processing of graph signals in centralized settings has received considerable attention, but less work has been devoted to solving similar tasks in distributed settings like sensor networks. Many distributed processing tasks consider the graph signal to be the result of the application of a linear graph-based operator to an initial input signal. When the signal can be represented as a filtering operation in the vertex domain of the graph, distributed processing of the signal is significantly simplified. More formally, given an initial signal $y$, every signal $y_{out}$ that can be expressed as filtering of $y$ in the graph vertex domain with a graph operator $P \in \mathbb{R}^{N \times N}$, such that

$$y_{out}(i) = \sum_{j \in \mathcal{N}_i} P_{i,j} y(j) \tag{2.23}$$

can be computed by local exchange of information only within the neighborhood of node $i$. $P_{i,j}$ is the filtering weight corresponding to the edge between nodes $i$

and $j$. The operator or graph filter $P$ is then defined according to the model of the signal.

Most of the existing works in such settings focus on reaching distributively an agreement between sensors, using only local communication. In that case, the operator $P$ is a doubly stochastic weight matrix that leads to an output $y_{out}$ that is the average value of components of the initial signal $y$. Examples of such operators are the Metropolis and the Laplacian weight matrices [12] defined respectively as:

- Metropolis weights

$$P_{ij} = \begin{cases} 1/\left(1 + \max\{d_i, d_j\}\right) & j \in \mathcal{N}_i, i \neq j \\ 1 - \sum\limits_{k \in \mathcal{N}_i} A_{ik} & i = j \\ 0 & \text{otherwise,} \end{cases} \tag{2.24}$$

  where $d\left(i\right), d\left(j\right)$ denotes the degree of the $i$-th and the $j$-th sensors respectively.

- Laplacian weights

$$P = I - \alpha L \tag{2.25}$$

  where $L$ denotes the Laplacian matrix of the graph $\mathcal{G}$ and the scalar $\alpha$ must satisfy $0 < \alpha < 1/d_{max}$, where $d_{max}$ consists of the maximum degree of the graph.

Among the most common applications, distributed consensus algorithms in both synchronous (average consensus algorithms) [13] and asynchronous versions (gossip algorithms) [14] have been widely used for performing various aggregations tasks in ad-hoc sensor networks. In particular, the authors in [15] solve the problem of distributed classification of multiple observations exploiting average consensus while consensus-based distributed algorithms for Support Vector Machine (SVM) training for binary classification have been proposed in [16]. In addition, [17] solves a distributed field estimation problem from compressed measurements while [18] introduces an algorithm for distributed subspace estimation based on average consensus. Gossip algorithms find also numerous applications in problems such as distributed parameter estimation, source localization, distributed compression [14], and decentralized sparse approximation [19].

Distributed average consensus operators are however only a specific case of the general family of graph-based operators. More in general, distributed processing of graph signals requires the definition of more sophisticated graph operators $P$. To that end, the authors in [20] have introduced a special category of linear graph operators called graph Fourier multipliers, which has been eventually extended to generalized graph multiplier operators in [21]. Such operators are defined with respect to a real symmetric positive semi-definite matrix $\Phi = UVU^T$, where $U$ and $V$ are the eigenvectors and the eigenvalues of $\Phi$, and are expressed as

$$P = \sum_{\ell=0}^{N-1} g\left(V_\ell\right) U_\ell U_\ell^*, \tag{2.26}$$

where $g\left(\cdot\right) : \left[0 : V_{max}\left(\Phi\right)\right] \rightarrow \mathbb{R}$ is a positive function defined in the spectral domain of the graph. When the matrix $\Phi$ is the graph Laplacian matrix then

$$P = \sum_{\ell=0}^{N-1} g\left(\lambda_\ell\right) u_\ell u_\ell^*, \tag{2.27}$$

which corresponds to a graph Fourier operator. The union of such operators $P = \left[u_{g_1}\left(\Lambda\right), u_{g_2}\left(\Lambda\right), \ldots, u_{g_S}\left(\Lambda\right)\right]$ represents the graph Fourier multipliers. From Eq.(2.23), a graph signal $y_{out}$ is then the result of filtering a set of initial signals $y = [y_1; y_2; \ldots; y_S]$ in the spectral domain with each of the graph Fourier multipliers, such that

$$y_{out} = \sum_{s=1}^{S} u_{g_s}\left(\Lambda\right) u^T y_s. \tag{2.28}$$

An example of a union of graph Fourier multipliers is the spectral graph wavelet transform [6], where each of the multipliers corresponds to a particular scale. An efficient way to apply graph Fourier multipliers in distributed settings is by approximating them with Chebyshev polynomials [6], [20]. In that case, the output signal $y_{out}$ is the linear combination of a set of graph filtering operations (in the vertex domain) of some initial signals on the graph. Such an approximation permits the distributed approximation of $y_{out}$ from the set of initial signals as well as the implementation of the forward and adjoint operators, which can be useful in tasks such as distributed denoising and distributed smoothing, as shown in [20].

A few more distributed processing algorithms of graph signals are based on

the above mentioned ideas of graph filtering in the vertex domain. Recently, a distributed least square reconstruction algorithm of bandlimited graph signals has been proposed in [22]. The initial observations are sampled only on a subset of nodes and the algorithm is shown to be efficient in tracking the unobserved data of time-varying graph signals. The distributed graph signal inpainting algorithm of [23] uses a regularizer that minimizes a metric term related to the variation of the signal on the graph. The underlying assumption is that the signal is smooth on the graph. The problem of interpolation of bandlimited graph signals from a few samples is also studied in . The reconstruction is achieved using iterative graph filtering, which can be approximated by polynomials of the graph Laplacian matrix and implemented in distributed settings. Graph filters have also been used to accelerate the convergence of the average consensus algorithm on a sensor graph [24, 25]. Finally, matrix polynomials of a graph-shift operator have been proposed in [26] to design graph filters for distributed linear network operators such as finite-time consensus or analog network coding. Most of all the above mentioned works show the potentials of graph signal processing techniques for distributed tasks, but do not explicitly consider practical aspects such as quantization, which is of significant importance in real word applications.

### 2.4.3 Graph-Based Multimedia Processing

Apart from processing signals that live on networks, graphs have been used for modeling structured signals that live on other irregular domains. In particular, graph signal processing algorithms have been successfully applied in numerous multimedia applications in order to capture the geometrical structure of complex high-dimensional signals such as images, videos, and 3D data. This type of data provides a promising application domain for the emerging field of graph signal processing.

First, we note that graphs and features based on graphs have recently started to gain attention in the computer vision and shape analysis community mainly due to the fact that the graph Laplacian has been shown to approximate successfully the Laplace-Beltrami operator on a manifold [27], [28], [29]. Spectral features defined on the graph have been successfully applied in a wide variety of shape analysis tasks. The heat kernel signatures [30], their scale-invariant version [31],

the wave kernel signatures [32], the optimized spectral descriptors of [33], have already been used in 3D shape processing with applications in graph matching [34] or in mesh segmentation and surface alignment problems [35]. These features have been shown to be stable under small perturbations of the edge nodes of the graph. In all these works however, the descriptors are defined based only on the graph structure, and the information about the attributes of the nodes such as color and 3D positions, if any, is assumed to be introduced in the weights of the graph. Thus, the performance of these descriptors largely depends on the quality of the defined graph.

Signal compression is a second application domain where graph signal processing tools have been applied successfully. Analogously to the classical analog case, the graph Fourier coefficients of a smooth signal decay rapidly , making the graph Fourier transform a good candidate for compression. In particular, the graph Fourier transform has been widely used to compress efficiently smooth images. For example, the graph-based Fourier transform has been used in [36] for the compression of image and video signals, as an alternative to the classical Discrete Cosine Transform (DCT). The authors in [37] adapted the graph for maximally smooth signals and optimized the graph Fourier transform for better compression of 3D smooth images. A set of edge-adaptive transforms was presented as an alternative to the standard DCT and used in depth-map coding in [38]. A few steps towards the theoretical analysis of the analogy between the graph Fourier transform and the classical DCT have been taken in [39]. Under a Gaussian Markov Random Field image model, the graph Fourier transform has been shown to be optimal in decorrelating the signal and used for predictive transform coding. Graphs have also been used for compressing multiview images, where the graph is designed by connecting corresponding pixels in different views [40]. In [41] graph-based transforms have been used to code luminance values in RGB. The problem of multiview images of asymmetric quality has been studied in [42], where the construction of a graph from high quality images has led to the enhancement of low quality images. In the same line of works, a graph regularizer that imposes smoothness has been proposed in [43] to enhance the quality of quantized depth images. Thus, graph representations are an interesting tool for compression of image and video signals.

Finally, graph-based transforms have recently been used in computer graph-

ics where the structural organization of 3D objects is captured by a graph. In particular, the authors in [44] represent a moving human body by a sequence of 3D meshes with a fixed and known connectivity represented by a graph. The geometry and the color information have then been considered as time-varying signals on a graph, which are compressed using the graph wavelet filter banks . Graph representations have been also used in to model the structure of 3D point clouds and connect nearby points. The graph Fourier transform, which is equivalent to Karhunen-Love transform on such graphs, is adopted to decorrelate and eventually compress the point cloud attributes that are treated as signals on the graph.

# 3

# Graph Signal Reconstruction

## 3.1 Reconstruction Problem

We consider the problem of reconstruct a graph signal from observations taken from a subset of vertices of the graph [45]. The problem fits well, e.g., to a Wireless Sensor Network (WSN) scenario, where the nodes are observing a spatial field related to some physical parameter of interest. Let us assume that the nodes' topology is fixed and that the corresponding graph is symmetric and connected. Suppose now that the WSN is equipped with nodes that, at every time instant, can take observations of the underlying signal or not, depending on, e.g., energy constraints, failures, limited memory and/or processing capabilities, etc. Our purpose is to build a technique that allows the recovery of the field values at each node. In this way, the information is processed on the fly by all nodes and the data diffuse across the network by means of a real-time sharing mechanism.

The signal corresponds to the minimum temperatures of 99 Italian cities, assigned to each month of the year. The temperatures data set were taken from the "Ministero delle politiche agricole, alimentari e forestali" site [46], where each value of the temperature is specified for the last 12 months. Furthermore, the coordinates of latitude and longitude were taken from the same source.

| city | ind | lat | long | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Torino | 1 | 45.1 | 7.7 | -4.8 | -4.3 | -3 | 1.4 | 3.9 | 8.7 | 11.3 | 11.1 | 9.4 | 3 | 0.6 | -1.9 |
| Alessandria | 2 | 44.9 | 8.6 | 6.4 | 4.6 | 1.3 | 3.5 | 4.9 | 9.8 | 11.6 | 16.3 | 18.6 | 18 | 15.6 | 9.1 |
| Asti | 3 | 44.9 | 8.2 | 6.3 | 3.4 | 0.8 | 2.7 | 4.6 | 9.5 | 11.6 | 16.3 | 18.5 | 18.1 | 15.9 | 9.1 |
| Biella | 4 | 45.6 | 8.1 | -6.5 | -4.6 | -3.4 | 1.8 | 4.6 | 9.7 | 12 | 11.3 | 9.1 | 3.1 | -0.8 | 2.9 |
| Cuneo | 5 | 44.4 | 7.5 | -0.8 | -0.1 | 1.7 | 6.2 | 8.2 | 12.8 | 15.4 | 15.1 | 13.4 | 6.9 | 4.7 | 1.9 |
| Novara | 6 | 45.1 | 8.6 | -3.4 | -0.5 | 1.2 | 6.6 | 9.2 | 14.2 | 16.4 | 15.5 | 13.1 | 6.9 | 1.8 | -0.7 |
| Verbania | 7 | 45.9 | 8.5 | -8.5 | -7 | -6.1 | -1.6 | 1.4 | 6.5 | 9 | 8.6 | 6.8 | 1.1 | -2.8 | -4.1 |
| Vercelli | 8 | 45.3 | 8.3 | -5.7 | -3.8 | -2.6 | 2.5 | 5.4 | 10.3 | 12.6 | 12 | 9.7 | 3.6 | -0.2 | -2.1 |
| Aosta | 9 | 45.7 | 7.3 | -9.6 | -9.7 | -9.4 | -5.3 | -2.5 | 2.2 | 4.9 | 5 | 3.4 | -2.2 | -4.6 | -5.4 |
| Milano | 10 | 45.5 | 9.2 | -1.3 | 2.1 | 3.6 | 8.8 | 11 | 15.9 | 18.5 | 17.4 | 15.3 | 8.8 | 3.8 | 1 |
| Bergamo | 11 | 45.7 | 9.7 | -2.7 | 0 | 1.2 | 5.8 | 7.9 | 12.7 | 15.7 | 14.6 | 13.2 | 6.5 | 2.9 | -0.3 |
| Brescia | 12 | 45.5 | 10.2 | -2.6 | -0.1 | 0.8 | 5.2 | 7.6 | 12.4 | 15.4 | 14 | 12.6 | 6 | 2.9 | 0.1 |
| Como | 13 | 45.8 | 9.1 | -3.8 | -1.2 | 0.3 | 5.2 | 7.4 | 12 | 14.9 | 14.1 | 12.6 | 6.3 | 1.9 | -0.9 |
| Cremona | 14 | 45.1 | 10 | -0.6 | 2.9 | 3.7 | 8.8 | 11.8 | 16.3 | 19.4 | 17.6 | 15.5 | 8.9 | 4.5 | 1.8 |
| Lecco | 15 | 45.8 | 9.4 | -2.3 | 0.6 | 2.1 | 7.2 | 9.1 | 13.8 | 16.6 | 15.7 | 14.1 | 7.6 | 3.2 | 0.1 |
| Lodi | 16 | 45.3 | 9.5 | -0.4 | 3 | 3.8 | 8.8 | 11.7 | 16.5 | 19.2 | 17.7 | 15.6 | 8.8 | 4.8 | 2 |
| Mantova | 17 | 45.1 | 10.8 | -0.7 | 3.1 | 3.8 | 8.6 | 11.9 | 15.9 | 19.6 | 17.4 | 15.2 | 9.1 | 4.5 | 1.8 |
| Pavia | 18 | 45.2 | 9.2 | -0.5 | 2.4 | 3.5 | 8.5 | 10.7 | 15.5 | 18 | 16.8 | 14.4 | 8.1 | 4.6 | 2.3 |
| Sondrio | 19 | 46.2 | 9.9 | -6.4 | -4.6 | -3.6 | 0.2 | 2.4 | 6.8 | 10.2 | 9.5 | 8.5 | 1.9 | 0 | -2.2 |
| Varese | 20 | 45.8 | 8.8 | -3.6 | -0.7 | 0.9 | 6.2 | 8.7 | 13.5 | 16 | 15.2 | 13.3 | 7.2 | 1.8 | -0.9 |
| Trento | 21 | 46.1 | 11.1 | -4.9 | -3.2 | -2.5 | 1.2 | 3.8 | 8.6 | 11.3 | 10.5 | 9 | 3 | 1.3 | -1.1 |
| Bolzano | 22 | 46.5 | 11.3 | -6.2 | -4.1 | -3.6 | 0.5 | 2.9 | 7.5 | 10 | 9.1 | 7.8 | 1.6 | 0 | -1.9 |
| Venezia | 23 | 45.4 | 12.3 | 0.6 | 4.9 | 5.8 | 9.8 | 12.6 | 17.4 | 20.2 | 18 | 16.6 | 10.2 | 5.4 | 2.3 |
| Belluno | 24 | 46.2 | 12.2 | -4.2 | -1.3 | -0.7 | 3.4 | 5.9 | 10.8 | 13 | 11.7 | 10.5 | 4.2 | 1.3 | -1.2 |
| Padova | 25 | 45.4 | 11.9 | -0.8 | 3.9 | 4.9 | 8.6 | 11.9 | 16.5 | 19.3 | 17.2 | 15.4 | 9.4 | 4.4 | 0.9 |
| Rovigo | 26 | 45.1 | 11.8 | 0.1 | 4.3 | 5.2 | 9 | 12.4 | 16.8 | 19.7 | 17.4 | 15.4 | 9.6 | 5.2 | 2.2 |
| Treviso | 27 | 45.7 | 12.2 | -1.1 | 2.8 | 4 | 7.9 | 10.7 | 15.4 | 18.1 | 16.3 | 14.9 | 8.6 | 4.1 | 0.9 |
| Verona | 28 | 45.4 | 11 | -1 | 2.6 | 3.5 | 7.7 | 10.8 | 15.1 | 18.3 | 16.5 | 14.6 | 8.5 | 4.2 | 0.9 |
| Vicenza | 29 | 45.5 | 11.5 | -1.8 | 1.8 | 2.8 | 6.2 | 9.4 | 14.1 | 16.6 | 15.3 | 13.4 | 7.7 | 3.5 | 0 |
| Trieste | 30 | 45.6 | 13.8 | 2.1 | 6.1 | 6.7 | 10 | 12.4 | 17.6 | 20.1 | 18.9 | 17.4 | 10.3 | 6.3 | 4.1 |
| Udine | 31 | 46 | 13.2 | -2.5 | 1.6 | 2.1 | 6 | 8.5 | 13.6 | 15.4 | 13.7 | 12.6 | 6.2 | 2.3 | -0.3 |
| Gorizia | 32 | 45.9 | 13.6 | 0.5 | 4.9 | 5.2 | 8.4 | 11.2 | 16.4 | 18.2 | 17.1 | 15.6 | 9.2 | 5 | 2.6 |
| Pordenone | 33 | 46 | 12.6 | -2 | 2.5 | 3.5 | 7.7 | 10.2 | 15.1 | 17.5 | 15.5 | 14.4 | 7.6 | 3 | 0.2 |

**Table 3.1:** Temperature dataset, index 1:33

| city | ind | lat | long | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Genova | 34 | 44.4 | 12.6 | 8.7 | 7.6 | 4.1 | 5.2 | 5.9 | 10.1 | 11.5 | 15.6 | 18.2 | 18 | 16.3 | 10.6 |
| Imperia | 35 | 43.9 | 8.9 | 3.2 | 3.4 | 5.2 | 9 | 10.9 | 15 | 17.9 | 18.2 | 16.7 | 10.7 | 7.9 | 5.9 |
| La Spezia | 36 | 44.1 | 8 | 5.1 | 5.8 | 6 | 9.6 | 11.3 | 15 | 17.7 | 17.6 | 16.1 | 11.1 | 9.4 | 8.3 |
| Savona | 37 | 44.3 | 9.8 | 3.2 | 4.5 | 5.9 | 10.1 | 11.6 | 16 | 18.5 | 18.7 | 16.8 | 10.7 | 8.2 | 6.8 |
| Bologna | 38 | 44.5 | 8.5 | -0.7 | 2.7 | 3.1 | 7.1 | 10 | 14.1 | 17.8 | 15.7 | 13.6 | 7.9 | 4.3 | 1.7 |
| Ferrara | 39 | 44.8 | 11.3 | -0.1 | 4 | 5 | 8.8 | 12.4 | 16.6 | 19.7 | 17.1 | 15.1 | 9.3 | 5.4 | 2.3 |
| Forl | 40 | 44.2 | 11.6 | 0.7 | 3.8 | 4.1 | 7.5 | 10.4 | 14.4 | 17.5 | 15.7 | 13.8 | 8.7 | 5.2 | 1.6 |
| Modena | 41 | 44.6 | 12 | -1.4 | 1.4 | 1.3 | 5.6 | 8.1 | 12.2 | 16.2 | 14.3 | 11.8 | 6.5 | 3.5 | 1.8 |
| Parma | 42 | 44.8 | 10.9 | -0.2 | 2.1 | 2.1 | 6.7 | 9.2 | 13.4 | 16.7 | 15 | 13.1 | 7.3 | 4.6 | 2.9 |
| Piacenza | 43 | 45 | 10.3 | -0.3 | 2.2 | 2.5 | 7.2 | 9.7 | 14.4 | 17.1 | 15.6 | 13.6 | 7.4 | 4.6 | 2.5 |
| Ravenna | 44 | 44.4 | 9.7 | 0.8 | 4.1 | 4.7 | 8.3 | 11.3 | 15.4 | 18.4 | 16.3 | 14.5 | 9 | 5.7 | 2.4 |
| Reggio Emilia | 45 | 44.7 | 12.2 | -1.3 | 1.6 | 1.4 | 6.1 | 8.8 | 12.7 | 16.7 | 14.5 | 12.3 | 6.8 | 3.7 | 2.1 |
| Rimini | 46 | 44.1 | 10.6 | 1.8 | 4.7 | 5.2 | 8.9 | 11.5 | 16 | 19.4 | 17.6 | 15.2 | 10.2 | 7.1 | 3.1 |
| Firenze | 47 | 43.8 | 12.6 | 1.6 | 4.1 | 3.9 | 7.1 | 9.9 | 13.9 | 16.9 | 16.4 | 14.1 | 9 | 5.6 | 2.8 |
| Arezzo | 48 | 43.5 | 11.9 | 1.6 | 4 | 3.3 | 6.6 | 9.4 | 13.4 | 16.4 | 15.4 | 13.1 | 8.8 | 4.7 | 1.9 |
| Grosseto | 49 | 42.8 | 11.9 | 4.7 | 5.7 | 5.4 | 9.3 | 11.4 | 15.9 | 18.9 | 18.4 | 16.1 | 11.7 | 8.4 | 6.2 |
| Livorno | 50 | 43.6 | 11.1 | 6.5 | 7.2 | 7 | 10.3 | 12.4 | 16.7 | 19.6 | 19.5 | 17.5 | 13.1 | 10.2 | 8.6 |
| Lucca | 51 | 43.8 | 10.3 | 1.6 | 2.7 | 2.2 | 6.1 | 7.9 | 12 | 15.1 | 14.3 | 12.1 | 7.6 | 5.9 | 4.8 |
| Massa | 52 | 44 | 10.5 | 2.2 | 3.3 | 3 | 7 | 8.8 | 12.7 | 15.6 | 14.7 | 13.1 | 8.1 | 6.6 | 5.6 |
| Pisa | 53 | 43.7 | 10.1 | 4.8 | 6.1 | 6 | 9.1 | 11.6 | 15.8 | 18.3 | 18.2 | 16.1 | 11.4 | 8.7 | 6.6 |
| Pistoia | 54 | 43.9 | 10.4 | 0.9 | 2.6 | 2.3 | 5.8 | 8 | 12.2 | 15.2 | 14.9 | 12.5 | 7.6 | 5.2 | 3.4 |
| Prato | 55 | 43.9 | 10.9 | 0.7 | 3 | 2.8 | 6.1 | 8.6 | 12.8 | 16 | 15.7 | 13.3 | 7.9 | 5 | 2.7 |
| Siena | 56 | 43.3 | 11.1 | 3.4 | 4.8 | 4.4 | 8.2 | 10.4 | 14.9 | 17.8 | 17.3 | 14.9 | 10.4 | 6.9 | 4.6 |
| Perugia | 57 | 43.1 | 11.3 | 1.4 | 3.8 | 2.7 | 7 | 8.6 | 13.1 | 16.7 | 15.2 | 12.7 | 8.4 | 5.1 | 2.6 |
| Terni | 58 | 42.6 | 12.4 | 1.6 | 4 | 2.8 | 7.4 | 8.7 | 13.4 | 17.2 | 15.9 | 13.3 | 9.1 | 5.7 | 3 |
| Ancora | 59 | 43.6 | 12.7 | 4.6 | 6.5 | 6.3 | 10.3 | 12.2 | 16.8 | 20.6 | 18.7 | 16.2 | 11.2 | 8.5 | 5.5 |
| Ascoli Piceno | 60 | 42.8 | 13.6 | 1.1 | 3.2 | 2.8 | 6.6 | 8.3 | 13.3 | 17.2 | 15.2 | 12.7 | 8.4 | 5.3 | 2.6 |
| Macerata | 61 | 43.3 | 13.4 | 3.2 | 5.3 | 4.7 | 8.8 | 10.4 | 15.1 | 19.1 | 17 | 14.6 | 9.9 | 7.2 | 4.4 |
| Pesaro | 62 | 43.9 | 12.9 | 3 | 5.3 | 5.3 | 8.9 | 11.5 | 15.9 | 19.2 | 17.5 | 15.3 | 10.3 | 7.2 | 3.8 |
| Roma | 63 | 41.9 | 12.4 | 3.7 | 6.3 | 5.2 | 9 | 11 | 15.1 | 18.5 | 18 | 15.4 | 11.5 | 7 | 3.2 |
| Frosinone | 64 | 41.7 | 13.4 | 0.2 | 3.3 | 2 | 6.4 | 8.2 | 12.7 | 16.2 | 15.1 | 12.6 | 8.8 | 3.7 | -0.9 |
| Latina | 65 | 41.5 | 12.9 | 5.8 | 7.9 | 6.8 | 10.4 | 12.5 | 16.6 | 19.6 | 19.3 | 16.9 | 13.3 | 9.2 | 5.4 |
| Rieti | 66 | 42.4 | 12.9 | -0.9 | 2.1 | 0.5 | 4.8 | 6.3 | 10.9 | 14.9 | 13.4 | 10.7 | 6.8 | 3.1 | 0.3 |

**Table 3.2:** Temperature dataset, index 34:66

| city | ind | lat | long | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Viterbo | 67 | 42.4 | 12.1 | 4.1 | 6.1 | 5.4 | 9.6 | 11.1 | 15.6 | 19 | 18.3 | 15.7 | 11.6 | 7.9 | 4.9 |
| L'Aquila | 68 | 42.2 | 13.2 | -1.9 | 0.9 | -0.4 | 4 | 5.7 | 10.3 | 14.2 | 12.6 | 10 | 6.2 | 1.9 | -1.4 |
| Chieti | 69 | 42.2 | 14.1 | 3.2 | 4.8 | 4.6 | 9 | 11 | 15.8 | 19.5 | 18.1 | 15.4 | 11.5 | 7.1 | 3.9 |
| Pescara | 70 | 42.5 | 14.2 | 1 | 2.6 | 2.2 | 6.2 | 8.3 | 13.2 | 17 | 15.2 | 12.7 | 8.7 | 4.8 | 1.9 |
| Teramo | 71 | 42.7 | 13.7 | 1.3 | 2.9 | 2.7 | 6.5 | 8.5 | 13.5 | 17.4 | 15.5 | 13 | 8.8 | 5.3 | 2.8 |
| Campobasso | 72 | 41.6 | 14.7 | 3.5 | 6.1 | 5.5 | 10.2 | 11.6 | 16.3 | 19.8 | 18.6 | 15.8 | 12.1 | 7.3 | 4.1 |
| Isernia | 73 | 41.6 | 14.3 | 0.5 | 3.5 | 2.7 | 7.7 | 8.9 | 13.8 | 17.3 | 16 | 13.3 | 9.8 | 4.1 | -0.2 |
| Napoli | 74 | 40.9 | 14.2 | 7.7 | 9 | 8.2 | 12 | 13.7 | 18.4 | 21 | 20.8 | 18 | 14.7 | 10.8 | 8 |
| Avellino | 75 | 40.9 | 14.8 | 3.4 | 5.4 | 4.6 | 8.8 | 10.2 | 14.8 | 18 | 16.9 | 14.1 | 10.9 | 7.1 | 3.7 |
| Benevento | 76 | 41.1 | 14.8 | 2.5 | 5.1 | 4.3 | 8.9 | 10.1 | 14.9 | 18.1 | 17.1 | 14.3 | 10.8 | 5.9 | 2.5 |
| Caserta | 77 | 41.1 | 14.3 | 2.5 | 5.3 | 4.3 | 8.8 | 10.4 | 15.4 | 18.4 | 17.5 | 14.9 | 11.3 | 5.8 | 1.2 |
| Salerno | 78 | 40.7 | 14.8 | 5 | 6.9 | 5.9 | 10.1 | 11.4 | 16 | 19.1 | 18.3 | 15.6 | 12.6 | 8.8 | 5.5 |
| Bari | 79 | 41.2 | 16.9 | 4.1 | 6.5 | 6.1 | 9.6 | 11.5 | 16.4 | 19.6 | 18.4 | 15.4 | 12.2 | 8 | 4.4 |
| Brindisi | 80 | 40.6 | 17.8 | 5.6 | 7.8 | 7.7 | 11 | 13 | 18.2 | 21 | 20.2 | 17 | 14.1 | 9.5 | 6 |
| Foggia | 81 | 41.5 | 15.5 | 4.7 | 6.8 | 6.1 | 10.4 | 11.9 | 16.4 | 20.3 | 18.9 | 15.9 | 12.4 | 9.1 | 6.3 |
| Lecce | 82 | 40.4 | 18.2 | 6.6 | 9.2 | 8.4 | 11.9 | 13.9 | 19 | 22.1 | 21.4 | 18.3 | 15.4 | 10.9 | 7.3 |
| Taranto | 83 | 40.5 | 17.2 | 5.6 | 8 | 7.8 | 10.9 | 12.7 | 17.9 | 20.5 | 19.7 | 16.7 | 14.1 | 9.1 | 5.5 |
| Potenza | 84 | 40.6 | 15.8 | 3.5 | 5.6 | 4.7 | 8.8 | 10.4 | 15.1 | 18.5 | 17.1 | 14.4 | 11.5 | 7.8 | 4.2 |
| Matera | 85 | 40.7 | 16.6 | 4.1 | 6.2 | 5.6 | 9 | 11.1 | 16 | 19.2 | 18.1 | 15.3 | 12.3 | 7.8 | 3.5 |
| Catanzaro | 86 | 38.9 | 16.6 | 7.1 | 8.8 | 7 | 11.1 | 13.2 | 18.1 | 21.2 | 20.3 | 17.3 | 15.6 | 12 | 7.6 |
| Cosenza | 87 | 39.9 | 16.3 | 4.8 | 6.8 | 5.5 | 9.9 | 11.3 | 16.2 | 19.4 | 18.5 | 15.5 | 13.4 | 9.8 | 6 |
| Crotone | 88 | 39.1 | 17.2 | 6.8 | 8.2 | 7 | 10.8 | 12.5 | 17.9 | 21.2 | 20.4 | 17.2 | 15 | 11.5 | 7.4 |
| Reggio Calabria | 89 | 38.1 | 15.7 | 10.2 | 11.7 | 10.1 | 13.8 | 15.9 | 20.6 | 23.8 | 23.4 | 21.1 | 19.3 | 14.8 | 11.1 |
| Vibo Valentia | 90 | 38.7 | 16.1 | 10.4 | 12.1 | 10.4 | 14 | 16.2 | 20.8 | 23.8 | 23.2 | 20.7 | 18.9 | 14.9 | 11.3 |
| Palermo | 91 | 38.1 | 13.4 | 7.6 | 8.4 | 7.6 | 11.8 | 13.1 | 17.3 | 20.8 | 20.3 | 18.1 | 16.4 | 11.5 | 8.3 |
| Agrigento | 92 | 37.3 | 13.6 | 7.8 | 9 | 7.8 | 12.4 | 13.7 | 17.8 | 20.3 | 20.2 | 18.4 | 17.1 | 12.1 | 8.3 |
| Caltanisetta | 93 | 37.5 | 14 | 6.2 | 7.9 | 6.5 | 11.4 | 12.8 | 17.2 | 20.2 | 19.5 | 17.4 | 16 | 11.1 | 7.5 |
| Catania | 94 | 37.5 | 15.1 | 6.3 | 8.2 | 6.7 | 11 | 13.1 | 17.6 | 20.8 | 20.5 | 18.6 | 16.9 | 11.6 | 7.7 |
| Enna | 95 | 37.5 | 14.3 | 4.9 | 6.7 | 5.2 | 9.9 | 11.6 | 16.2 | 19.7 | 18.9 | 16.5 | 14.8 | 10 | 6.4 |
| Messina | 96 | 38.2 | 15.5 | 10.4 | 11.7 | 10.5 | 14.1 | 15.7 | 20 | 23.4 | 23.1 | 21.3 | 19.3 | 14.9 | 12.1 |
| Ragusa | 97 | 36.9 | 14.8 | 8 | 9.7 | 8.4 | 12.6 | 14.6 | 18.9 | 21.6 | 21.4 | 19.8 | 18.3 | 13.2 | 9.6 |
| Siracusa | 98 | 37.1 | 15.3 | 8.4 | 9.9 | 8.8 | 12.6 | 14.9 | 19.2 | 22.2 | 22.2 | 20.5 | 19 | 13.6 | 9.7 |
| Trapani | 99 | 38 | 12.5 | 9.9 | 10.1 | 9.5 | 12.7 | 14.4 | 18.4 | 21.2 | 21.5 | 19.9 | 17.8 | 13.3 | 9.7 |

**Table 3.3:** Temperature dataset, index 67:99

We also assume that the sampling procedure in the vertex domain is random, therefore we are able to detect the signal only for a subset of nodes: for the non-sampled nodes we have to reconstruct their own value of the signal.

Assume that we have the time-sampled signal $\boldsymbol{r}_t, t \in \tau$ where $\tau$ denotes the set of sampled nodes. The signal $\boldsymbol{r}_t$ is known and we want to reconstruct the entire signal $\boldsymbol{s}_t$ for $t \in \tau \cup \bar{\tau}$, where $\bar{\tau}$ is the set of non-sampled nodes, starting from an estimate in the frequency domain limited to the component indexed by $\mathcal{F}$. We assume an optimal frequency ordering for the set $\mathcal{F}$, which means that the frequencies are indexed in such a way that the first provide higher signal energy and the last does not represent a significant contribution: therefore the frequency set $\mathcal{F}$ is ordered according to each nodes energy contribution. This assumption provides stability for the estimation process and ensures that the reconstruction error is lower than the one obtained assuming random ordering (or some other indexing strategy). Then we can say that $\mathcal{F}$ acts as a low-pass filter, which passes the lowest frequencies guaranteeing higher signal energy.

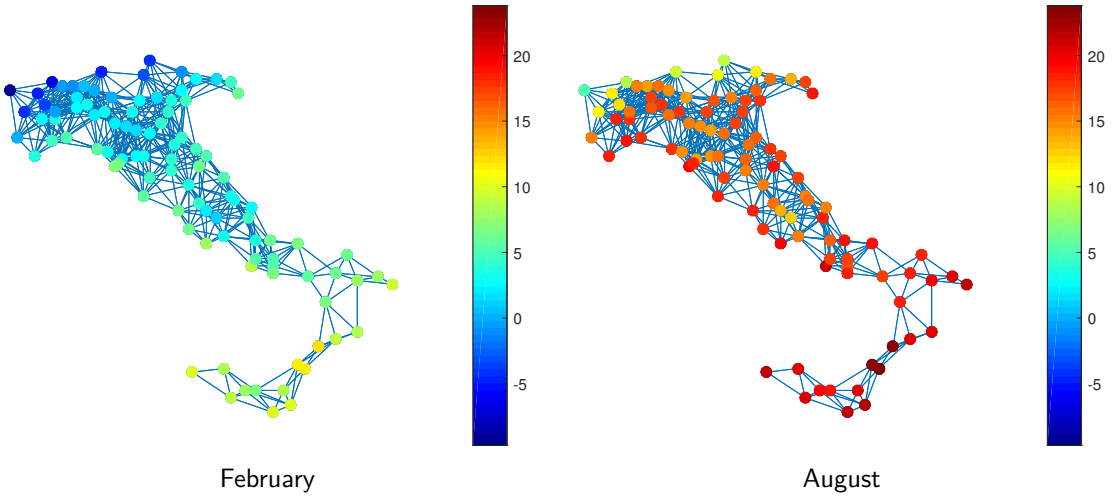



**Figure 3.1:** Temperature for 2 different months

The problem corresponds to reconstruct the non-sampled component $\boldsymbol{s}_t, t \in \bar{\tau}$ of the signal

$$\begin{cases} \hat{\boldsymbol{s}}_t = \boldsymbol{r}_t & \text{for } t \in \tau \\ \hat{\boldsymbol{s}}_t = \boldsymbol{U}_{t,f}\hat{\boldsymbol{S}}_f & \text{for } t \in \bar{\tau} \end{cases} \tag{3.1}$$

because for $t \in \tau$ the signal is known.

$\boldsymbol{U}_{t,f}$ is the eigenvector matrix associated to rows $t \in \bar{\tau}$ and columns $f \in \mathcal{F}$, then we need to know the component sampled in the vertex domain only for the subset of eigenvector chosen by frequency index. $\hat{\boldsymbol{S}}_f, f \in \mathcal{F}$ is the estimated signal in the frequency domain.

The Least Square (LS) solution to the problem (3.1) is

1. $\hat{\boldsymbol{S}}_f = \underset{\boldsymbol{S}_f \in \mathbb{R}^{|\mathcal{F}|}}{\arg\min} \|\boldsymbol{r}_t - \boldsymbol{U}_{t,f}\boldsymbol{S}_f\|_2^2$, for $t \in \tau$ and $f \in \mathcal{F}$

2. $\hat{\boldsymbol{s}}_t = \boldsymbol{U}_{t,f}\hat{\boldsymbol{S}}_f$, for $t \in \bar{\tau}$ and $f \in \mathcal{F}$

Step 2 is easy, because $\boldsymbol{U}_{t,f}$ is known and $\hat{\boldsymbol{S}}_f$ is what we have to estimate (the GFT of the observed signal).

Assuming for clearness $\boldsymbol{U}_{t,f} = \boldsymbol{U}_t^T$, step 1 can be solved as follow:

$$\begin{aligned}
\hat{\boldsymbol{S}}_f &= \underset{\boldsymbol{S}_f}{\arg\min} \sum_{t\in\tau} \left(\boldsymbol{r}_t - \boldsymbol{U}_{t,f}\boldsymbol{S}_f\right)^2 \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^{|\mathcal{F}|}}{\arg\min} \sum_{t\in\tau} \left(\boldsymbol{r}_t - \boldsymbol{U}_t^T\boldsymbol{x}\right)^2 \\
&= \underset{\boldsymbol{x}}{\arg\min} \sum_{t\in\tau} \boldsymbol{r}_t^2 + \boldsymbol{x}^\tau \left(\boldsymbol{U}_t\boldsymbol{U}_t^\tau\right)\boldsymbol{x} - 2\boldsymbol{r}_t\boldsymbol{U}_t^T\boldsymbol{x} \\
&= \underset{\boldsymbol{x}}{\arg\min} \sum_{t\in\tau} \boldsymbol{r}_t^2 + \boldsymbol{x}^\tau \left(\sum_{t\in\tau} \boldsymbol{U}_t\boldsymbol{U}_t^T\right)\boldsymbol{x} - 2\left(\sum_{t\in\tau} \boldsymbol{r}_t\boldsymbol{U}_t^T\right)\boldsymbol{x}
\end{aligned} \tag{3.2}$$

and taking the derivative of Eq.(3.2) with respect to $\boldsymbol{x}$ and setting it equal to 0 we have

$$\left(\sum_{t\in\tau} \boldsymbol{U}_t\boldsymbol{U}_t^T\right)\boldsymbol{x} = \sum_{t\in\tau} \boldsymbol{U}_t\boldsymbol{r}_t \tag{3.3}$$

which leads to

$$\hat{\boldsymbol{x}} = \left(\sum_{t\in\tau} \boldsymbol{U}_t\boldsymbol{U}_t^T\right)^{-1} \cdot \left(\sum_{t\in\tau} \boldsymbol{U}_t\boldsymbol{r}_t\right) \tag{3.4}$$

and finally

$$\hat{\boldsymbol{S}}_f = \left(\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T\boldsymbol{U}_{t,f}\right)^{-1} \cdot \left(\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T\boldsymbol{r}_t\right). \tag{3.5}$$

Now we discuss a necessary and sufficient condition that guarantees exactly signal reconstruction from its samples, as discussed in [45]. It is clear from Eq.(3.5)

that reconstruction of the original signal is possible only if the matrix

$$\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T \boldsymbol{U}_{t,f} \tag{3.6}$$

is invertible. From (3.6), a necessary condition enabling reconstruction is

$$|\tau| \geq |\mathcal{F}| \tag{3.7}$$

i.e., the number of nodes in the sampling set must be greater than equal to the signal bandwidth. However, this condition is not sufficient, because matrix $\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T \boldsymbol{U}_{t,f}$ in (4.33) may loose rank, or easily become ill-conditioned, depending on the graph topology and sampling strategy. It is invertible if $\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T \boldsymbol{U}_{t,f} = \boldsymbol{U}_f^T \boldsymbol{R}_\tau \boldsymbol{U}_f$ has full rank, where $\boldsymbol{R}_\tau$ is the vertex-limiting operator that projects into the sampling set $\tau$. Introducing the operator

$$\boldsymbol{R}_{\bar{\tau}} = \boldsymbol{I} - \boldsymbol{R}_\tau, \tag{3.8}$$

which projects into the complement of the sampling set. Then, exploiting (4.34) in $\boldsymbol{U}_f^T \boldsymbol{R}_\tau \boldsymbol{U}_f$, signal reconstruction is possible if $\boldsymbol{I} - \boldsymbol{U}_f^T \boldsymbol{R}_{\bar{\tau}} \boldsymbol{U}_f$ is invertible, i.e., if condition

$$\left\| \boldsymbol{R}_{\bar{\tau}} \boldsymbol{U}_f \right\|_2 < 1 \tag{3.9}$$

is satisfied. Condition (4.35) is related to the localization properties of graph signals: it implies that there are no $\mathcal{F}$-bandlimited signals that are perfectly localized over the set $\bar{\tau}$. As explained in [45], [47], [48], it is easy to show that condition (4.35) is necessary and sufficient for signal reconstruction.

Eq.(3.5) can be written as $\hat{\boldsymbol{S}}_f = \boldsymbol{M}\boldsymbol{r}_t$, where $\boldsymbol{M} = \left(\boldsymbol{U}_{t,f}^T \boldsymbol{U}_{t,f}\right)^{-1} \boldsymbol{U}_{t,f}^\tau$ is the matrix projecting the known signal defined in the vertex domain into the estimated spectral domain signal. This procedure need to collect all the values of the term $\sum_{t\in\tau} \boldsymbol{U}_{t,f}^T \boldsymbol{r}_t$ for each $t \in \tau$ in a central node and to sum them. Therefore this solution can be seen as a centralized implementation; in chapter 4 "Distributed Algorithms" we will see a distributed solution to the reconstruction problem.

## 3.2 Random Sampling and Frequency Ordering

Now we want to briefly describe how the choice of sampling strategy and frequency ordering affects the reconstruction algorithm.

First of all, we decided to sample the nodes in the vertex domain according to a random procedure: given the number of sampled nodes, we select the samples over all the nodes according to a uniform distribution, where each node has the same probability of being chosen. This is justified because we can not decide under which conditions a node is selected or not: in some cases we are aware of a node function value, in some other we are not, depending on, e.g., energy constraints, failures, limited memory and/or processing capabilities, etc. Therefore we randomly pick up a sufficient number of nodes over the entire network, and then we build the sampling set. In Fig. 3.2 is reported an example of random sampling of graph signal.

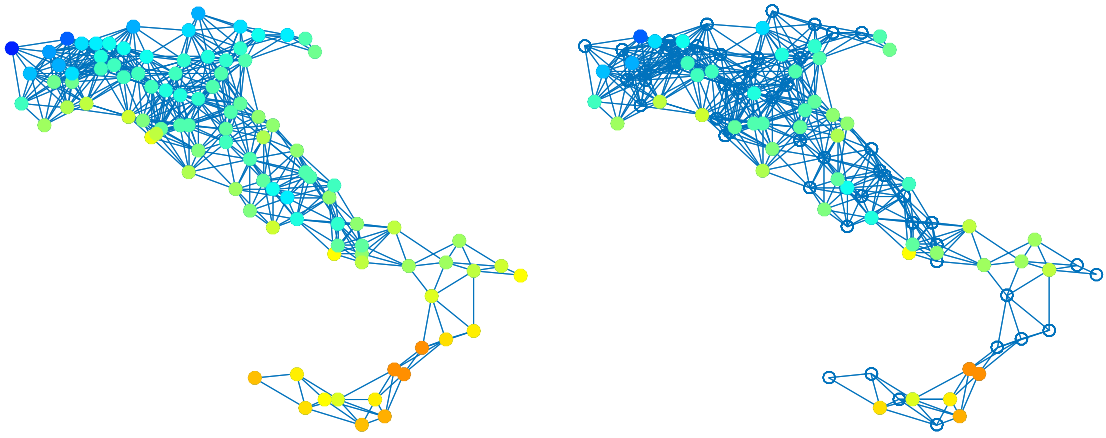
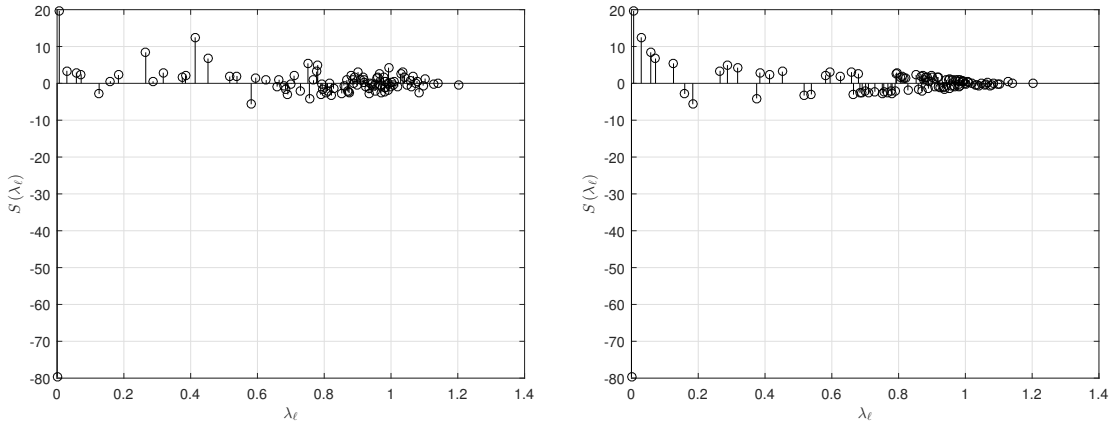



**Figure 3.2:** An example of signal sampling

Now we consider the choice of the frequency indexing in the graph spectral domain. We assumed that they are sorted according to the energy contribution which gives to the signal: we sorted the frequency indices in the spectral domain according to the mean of the squared absolute value of the same component, for a collection of signals, where first (the lowest) frequency gives an higher contribution

to signal energy in vertex domain, and last (the biggest) gives a small contribution to signal energy. Therefore the set of frequencies is ordered in such a way as to have the energy of the signal decreasing as the frequencies increases. An example is reported in Fig 3.3.




Original frequency ordering Induced frequency ordering

**Figure 3.3:** Frequency ordering

## 3.3 Performance of the LS Reconstruction Algorithm

Now we present some results obtained through the implementation of the algorithm described in Section 3.1. For this simulation we took into account some different definitions of the weighted adjacency matrix $\boldsymbol{A}$ from [12], [49]. The weighting methods we considered for our simulations are Metropolis-Hasting, Laplacian, maximum degree, exponential of the distance, unweighted and normalized unweighted. We briefly describe how they are defined.

- Metropolis weights: these are a form of local-degree weights, which are defined as

$$\boldsymbol{A}_{ij} = \begin{cases} 1/\left(1+\max\{d_i, d_j\}\right) & j \in \mathcal{N}_i, i \neq j \\ 1 - \sum\limits_{k \in \mathcal{N}_i} A_{ik} & i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

  In other words, the Metropolis weight on each edge is one over one plus the larger degree at its two indicent nodes, and the self-weights $\boldsymbol{A}_{ii}$ are chosen in such a way that the sum of the weights at each node is 1. The Metropolis weights are very simple to compute and are well suited for distributed implementation using local information. In particular, each node only needs to know the degrees of its neighbors to determine the weights on its adjacent edges. The nodes do not need any global knowledge of the communication graph, or even the number of nodes $N$. Furthermore, the weighted adjacency matrix $\boldsymbol{A}$, in this case, is a doubly stochastic matrix: $\boldsymbol{A} \cdot \mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T \cdot \boldsymbol{A} = \mathbf{1}^T$.

- Laplacian weights: the weight matrix has entries given by

$$\boldsymbol{A}_{ij} = \begin{cases} \alpha & j \in \mathcal{N}_i, i \neq j \\ 1 - \alpha|\mathcal{N}_i| & i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

where $|\cdot|$ denotes cardinality, or expressed in matrix form

$$\boldsymbol{A} = \boldsymbol{I} - \alpha \boldsymbol{L} \tag{3.12}$$

where $\boldsymbol{L}$ is the Laplacian matrix of the associated underlying graph. The parameter $\alpha$ must satisfy $\alpha \leq 1/\max\{d_k\}$. Even in this case, $\boldsymbol{A}$ is a doubly stochastic matrix.

- Maximum-degree weights: are defined as

$$\boldsymbol{A}_{ij} = \begin{cases} 1/\max_k\{d_k\} & j \in \mathcal{N}_i, i \neq j \\ 1 - d_i/\max_k\{d_k\} & i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

- Exponential of the distance: the weights are function of the exponential of the distance, that is the weight is large when two neighboring nodes are closer and is low when they are distant:

$$\boldsymbol{A}_{ij} = \begin{cases} e^{-dist(i,j)} & j \in \mathcal{N}_i, i \neq j \\ 1 & i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

  where the distance between two nodes is defined as $dist\,(i,j) = \|coord_i - coord_j\|_2$, where $coord_i$ corresponds to latitude and longitude coordinates of node $i$.

- Unweighted: the weights matrix is simple and it contains a 1 only for neighboring nodes:

$$\boldsymbol{A}_{ij} = \begin{cases} 1 & j \in \mathcal{N}_i, i \neq j \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

- Normalized unweighted: weights are defined in such a way that $\boldsymbol{A} \cdot \mathbf{1} = \mathbf{1}$, where we first take the unweighted adjacency matrix (3.15) and then we normalize each row by dividing each element by its corresponding row-sum. In this case the adjacency matrix is row-stochastic.

The performance measure taken is the Mean Squared Error (MSE) between the non-sampled original signal and the reconstructed one, summed for each month:

$$MSE = \sqrt{\sum_{t \in \bar{\tau}} |\hat{\boldsymbol{s}}_t - \boldsymbol{s}_t|^2} \tag{3.16}$$

Considering the sampling strategies, we averaged the simulation over 1000 different random sampling layout, in order to smooth the reconstruction error and to reduce the effect of the randomness of the sampling. The frequency set is ordered as described in Section 3.1.

In Fig. 3.4 we show the behaviour of the MSE for different numbers of samples and frequencies, changing the weighting strategy. We can clearly see that the best results are obtained for the doubly-stochastic Metropolis weights.

The results of the simulation can be better interpreted in Fig. 3.5, which shows the behaviour of the error by fixing the number of frequencies: we can note that the MSE for Metropolis weights is more smoothed and has a more regular trend.

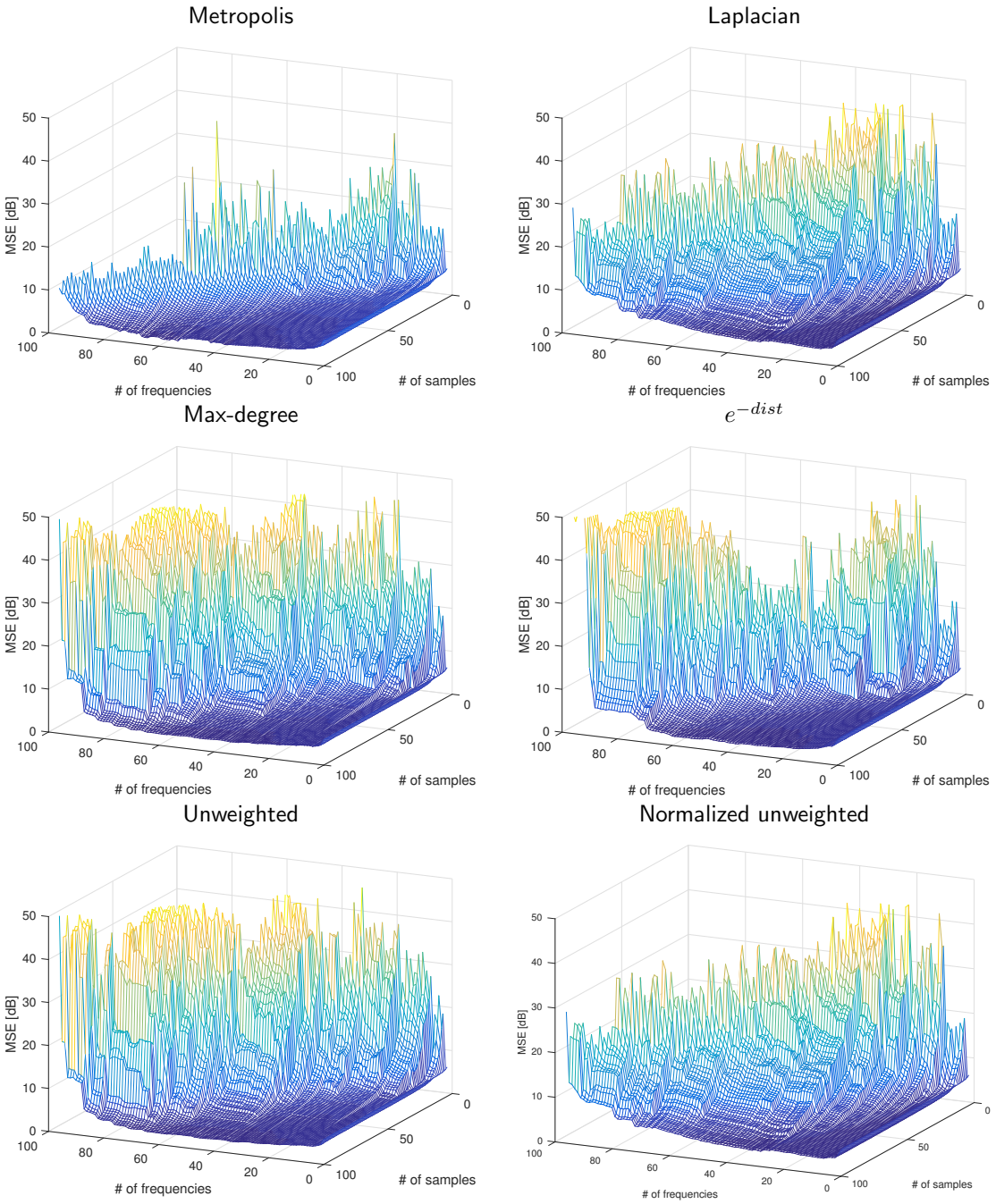



**Figure 3.4:** MSE for different weighting methods

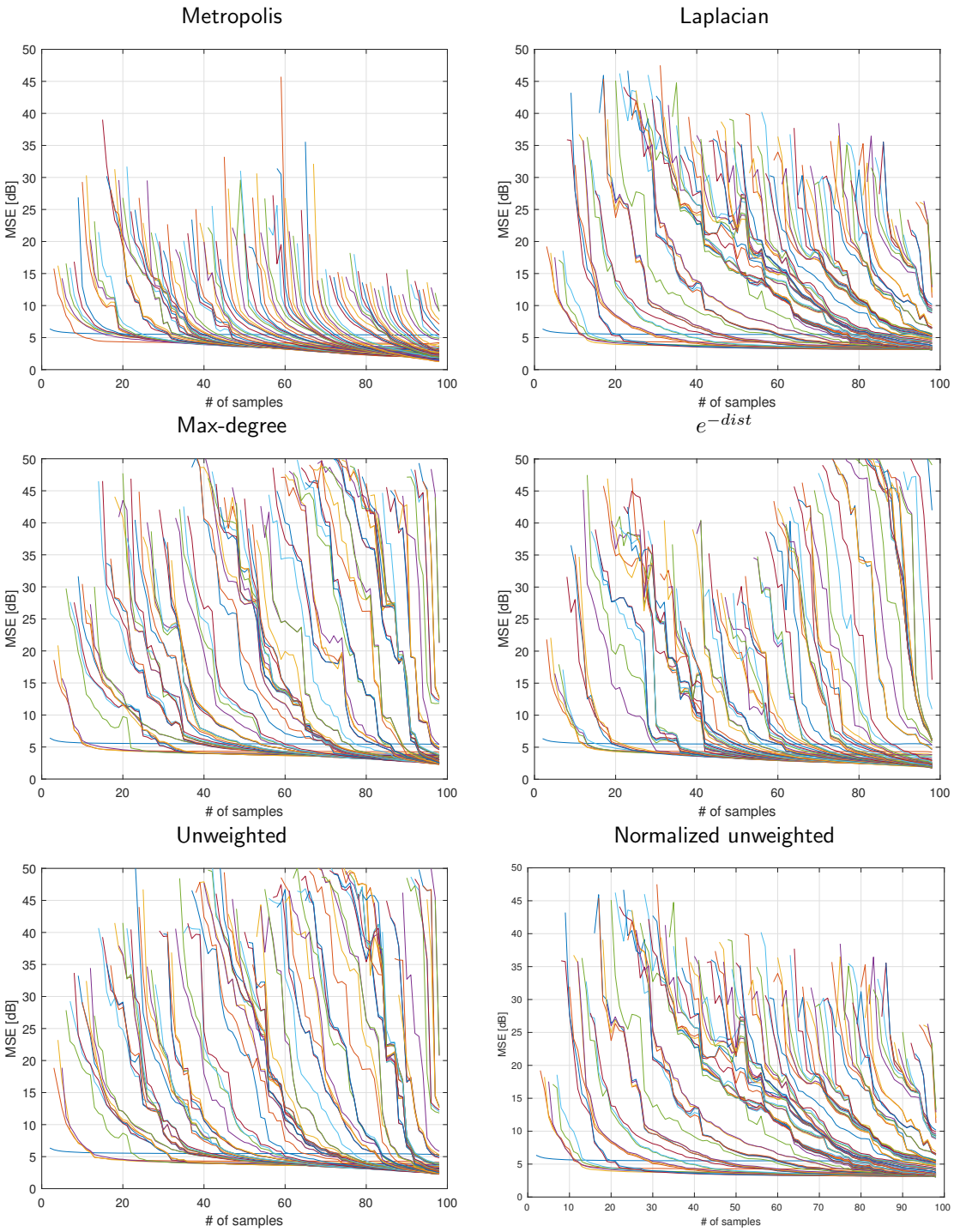



**Figure 3.5:** MSE for different weighting methods

In Fig 3.6 is reported an example of signal reconstruction, using 50 original sampled values and 10 frequencies: it is possible to observe that the reconstructed signal is quite similar to the original one, except for some outlier values that can not be exactly estimated since their neighboring nodes does not bring sufficient information to precisely evaluate an outlying value.

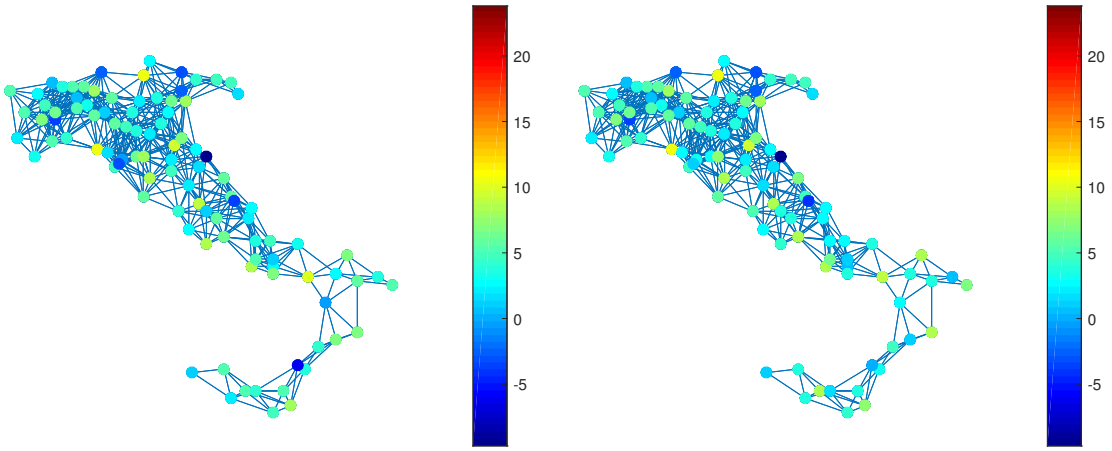



**Figure 3.6:** LS signal reconstruction for month March, Metropolis weights

## 3.4 $\ell_1$ Regularization Sparsity

In this section we want to deal with a possible problem: what if the number of sampled nodes is smaller than the number of active frequencies? Typically, the estimation model can be represented using matrix notation as

$$\boldsymbol{r} = \boldsymbol{U}\boldsymbol{S}, \tag{3.17}$$

where $\boldsymbol{r}$ is the observed vector in vertex domain, $\boldsymbol{U}$ is the $T \times F$ matrix of eigenvectors (where $F$ denotes the number of frequencies and $T$ the number of sampled nodes) and $\boldsymbol{S}$ is the vector of unknown parameters to be estimated. The estimation problem, as we have seen, is usually solved through LS where the parameters are estimated by the values minimizing the residual sum of squares $\|\boldsymbol{r} - \boldsymbol{U}\boldsymbol{S}\|^2$. Provided $\boldsymbol{U}$ is full rank, such that $\boldsymbol{U}^T\boldsymbol{U}$ is nonsingular and can be inverted, this gives $\hat{\boldsymbol{S}} = \left(\boldsymbol{U}^T\boldsymbol{U}\right)^{-1}\boldsymbol{U}^T\boldsymbol{r}$.

From a statistician's point of view, high-dimensional problems, that is when $F \gg T$, are interesting because they cannot be solved by classical estimation procedures like LS. The standard procedures rely on the assumption that $\boldsymbol{U}^T\boldsymbol{U}$ is nonsingular, otherwise $\boldsymbol{U}^T\boldsymbol{U}$ cannot be inverted and the parameters cannot be uniquely estimated. This obviously does not hold when $F > T$, as the covariate matrix does not have full column rank. There are no other differences in the model than the fact that $F > T$, but this highly influences the estimation problem. Thus to cope with regression when $F \gg T$, some kind of preselection or regularization is needed.

The Least Absolute Shrinkage and Selection Operator (LASSO) was proposed by Tibshirani in 1996 [50] as a new method for estimation in linear models. Inspired by the work of Breiman [51] on the non-negative garotte and wishing to improve upon unsatisfactory properties of the ordinary LS estimates, he introduced regression with a $\ell_1$-norm penalty. The $\ell_1$ penalty appeared to have desirable properties that could be exploited with great benefit in high-dimensional regression problems, and it is in the $F \gg T$ problems that the LASSO-type methods have really proven their superiority compared to other existing methods. Today, the methods of the LASSO-type are by far the most popular group of methods solving regression problems when $F \gg T$. In this section, we describe the LASSO

pointing especially to why it has become such an appreciated tool for regression.

Assuming the linear model (3.1), the LASSO estimator is defined by

$$\hat{\boldsymbol{S}} = \arg\min_{\boldsymbol{S}} \|\boldsymbol{r} - \boldsymbol{U}\boldsymbol{S}\|_2^2 + \lambda \|\boldsymbol{S}\|_1 \tag{3.18}$$

where $\lambda$ is a tuning parameter controlling the amount of shrinkage. This formulation of the problem is called Lagrangian form. We call the penalty of this form a $\ell_1$ penalty. In addition to shrinking the coefficients toward zero, the $\ell_1$ penalty has the advantageous property of doing variable selection. In this way the LASSO performs a kind of continuous subset selection [50].

To understand in more detail how the LASSO leads some regression coefficients to be exactly equal to zero, note first that problem (3.18) is equivalent to minimizing the residual sum of squares with a size constraint of the form $\|\boldsymbol{S}\|_1 \leq t$ on the parameters. Here $t$ is a tuning parameter that, by Lagrangian duality, has a one-to-one correspondence with the penalty parameter $\lambda$.

For all penalized regression methods having similar size constraints, like also for ridge regression [52] (where the size constraint minimizes the residual sum of squares as $\|\boldsymbol{S}\|_1^2 \leq t$), $t$ controls the amount of shrinkage imposed on the estimates. By the form of the size constraint $\|\boldsymbol{S}\|_1^r \leq t$, larger values of $\lambda$ correspond to more shrinkage, forcing the estimates toward zero. For the LASSO, large values of $\lambda$ will shrink all coefficients, but in addition put some of them exactly equal to zero. This is a direct consequence of using the $\ell_1$-norm in the constraint. Since the LASSO constraint is not differentiable at zero, the LASSO has the ability of producing estimates that are exactly equal to zero. The ridge constraint, on the other hand, does not share this property as having $r > 1$ gives constraints that are differentiable at zero [53], [54]. That is, the difference really lies in the shape of the constraint region. To illustrate this, we consider the simple situation with only two parameters in Fig.3.7. It shows the estimation picture for the LASSO and ridge regression. The elliptical contour lines represent the residual sum of squares centered at the LS estimate, while the shaded regions represent the constraint region for the lasso and ridge regression respectively.

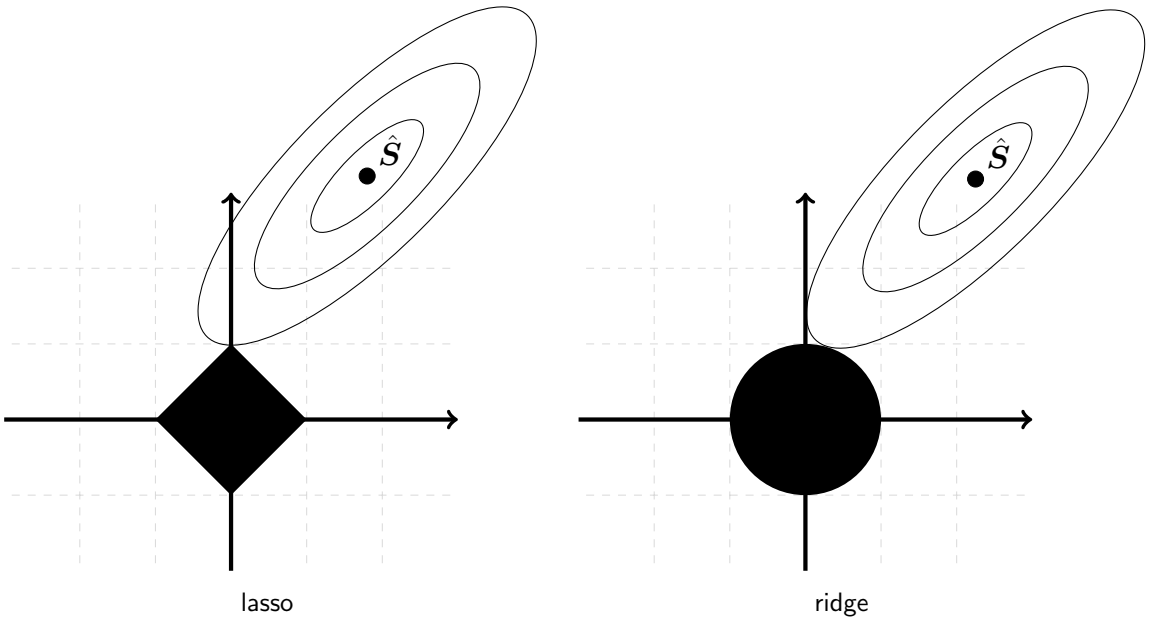



**Figure 3.7:** LASSO and ridge constraint comparison

In both cases, the solution is at the first point where the elliptical contour lines of the residual sum of squares hit the constraint region, which gives a minimum value for cost function. The important advantage of the LASSO is that, because of the diamond shape, it is more likely that the first time the elliptical contour lines hit the constraint region is in the corner, hence one of the parameters is estimated to be exactly zero. In higher dimensions the constraint region will have many corners and flat edges causing even more estimates to be zero [53]. Since the size of the constraint region is controlled by $t$, taking $t$ small enough, that correspond to a large $\lambda$, will force coefficients to be exactly zero. For ridge regression there are no sharp edges making it less likely for the contour lines to hit a corner. Hence estimated regression coefficients exactly equal to zero will rarely occur.

When the LASSO puts coefficients to zero, we say that it is producing a sparse solution. That is, only a few of the regression coefficients are estimated to be nonzero. This means that using the LASSO there is an underlying assumption about sparsity: we assume that there are only a few of the frequencies that are actually explaining the response. It is exactly this sparsity assumption that makes the LASSO such a successful tool in high-dimensional regression analysis. Not only sparsity is a consequence of using the $\ell_1$-norm constraint and an important theoretical aspect to reduce the complexity and the number of effective parameters

in the model, there are also intuitive as well as practical and computational reasons to assume sparsity in regression. The intention of producing more interpretable models is especially fruitful in the high-dimensional context. It is easier and more convenient to interpret results from a LASSO fit rather than a result involving estimated coefficients.

In standard regression models, the set of covariates is typically composed by a few variables that are well chosen and believed to be relevant and contributing to the model. The difference between the traditional setting and the high-dimensional problems is that the number of potential frequencies is much larger, but more importantly, we do not know which of them might be relevant. In this sense, the fact that the LASSO does variable selection makes it extremely attractive in determining the relevant frequencies exhibiting the strongest effects. In fact, all constraints of the form $\|\boldsymbol{S}\|_1^r$ with $r \leq 1$ perform variable selection, but the LASSO is the only constraint that has the advantage of producing a sparse solution while at the same time being convex. This also makes it an attractive method for computational reasons as non-convex constraints make the optimization much more difficult [53], [54], [55].

There is no closed form expression for the estimates in the LASSO solution. The optimization problem becomes that of a convex problem with inequality constraints that are typically solved through quadratic programming [56]. We solved the estimation problem through the MATLAB `fmincon` function, which automatically solves the problem using an interior-point algorithm. The only changeable parameter is the $\ell_1$-norm weight $\lambda$, which through we can adjust the sparsity of the solution: with a small value of $\lambda$ the $\ell_1$-norm has a lower weight in the minimizing function, so the solution, and therefore the error, will be similar to the LS ones; instead for higher values of $\lambda$ the solution will be more sparse, and then the error would be more affected by the regularization term.

In Fig 3.8 we show a comparison between the original temperature of month March and the reconstructed version through the LASSO regularization, imposing the number of frequencies to 99. As for the example in Fig 3.6, we use 50 samples; furthermore, the $\ell_1$-norm is weighted with a value $\lambda = 1$. We can notice that the reconstructed values are similar to the correct ones, but the outlier quantities can not correctly estimated because of their intrinsic structure, which is different from their neighbors.

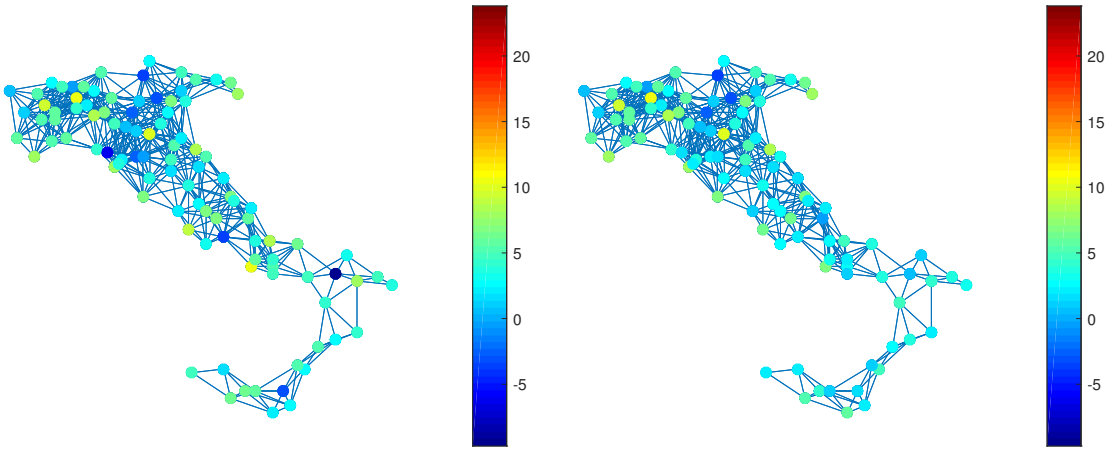



**Figure 3.8:** $\ell_1$-norm signal reconstruction for month March, Metropolis weights

For a more general purpose, we need to estimate the correct value of the parameter $\lambda$: therefore we report the behaviour of the MSE for different values of the parameter $\lambda$ and for different values of the number of samples, respectively in Fig 3.9 and in Fig 3.10.

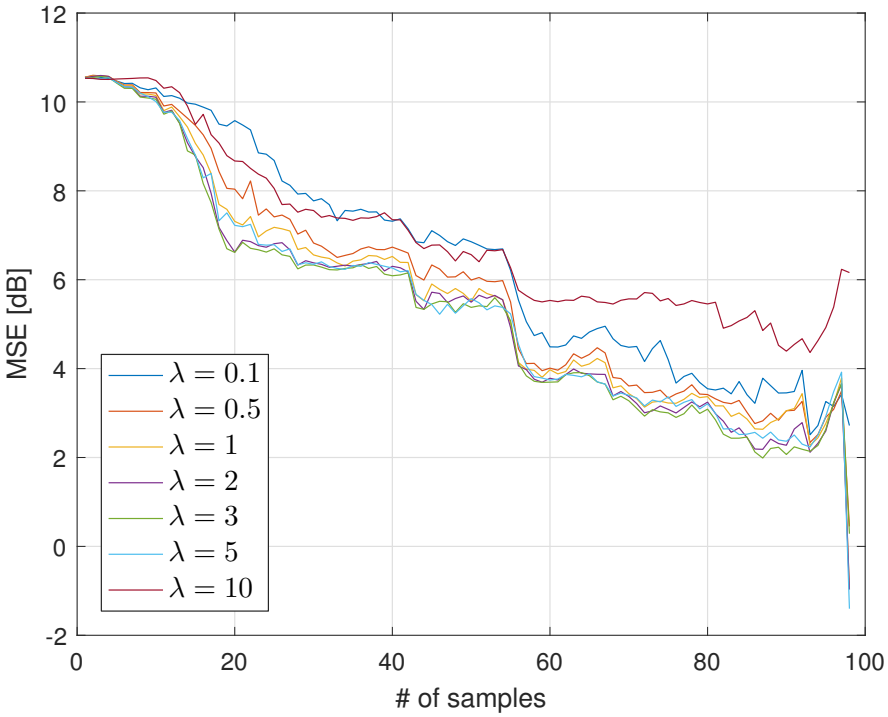



**Figure 3.9:** MSE for $\ell_1$-norm problem, different $\lambda$

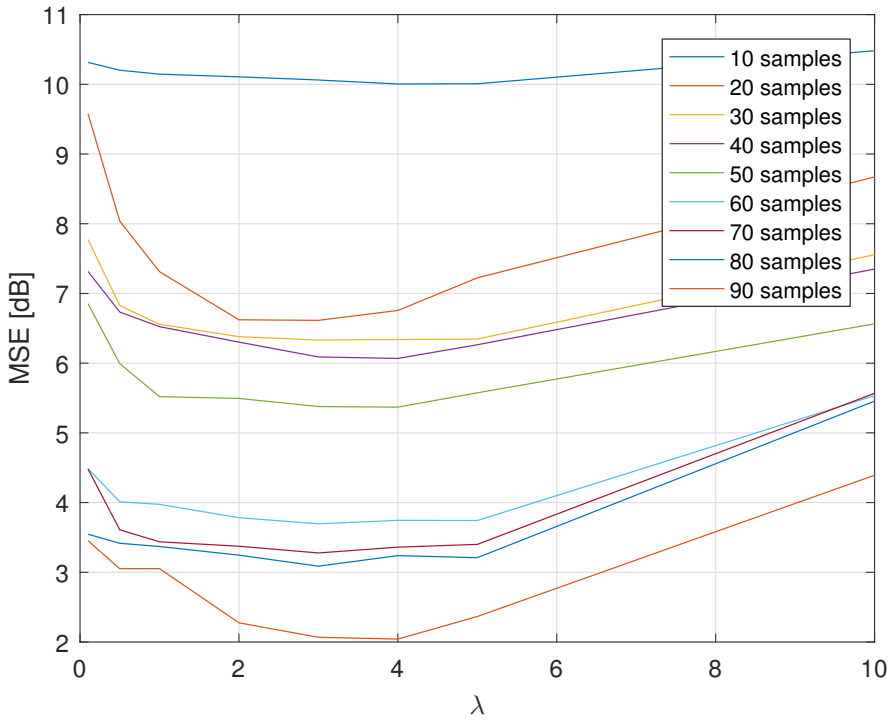



**Figure 3.10:** MSE for $\ell_1$-norm problem, different number of samples

In Fig 3.11 is reported a comparison of the MSEs for the LS and the LASSO problems, for each weighted adjacency matrix described in Section 3.3, imposing the number of active frequency to its maximum value. We can see that the performance of the LS solution are worse than the ones of the $\ell_1$-norm regularization, which improves significantly the precision of the reconstructed signal, specially in the case when the number of sampled nodes are less than the number of active frequencies.

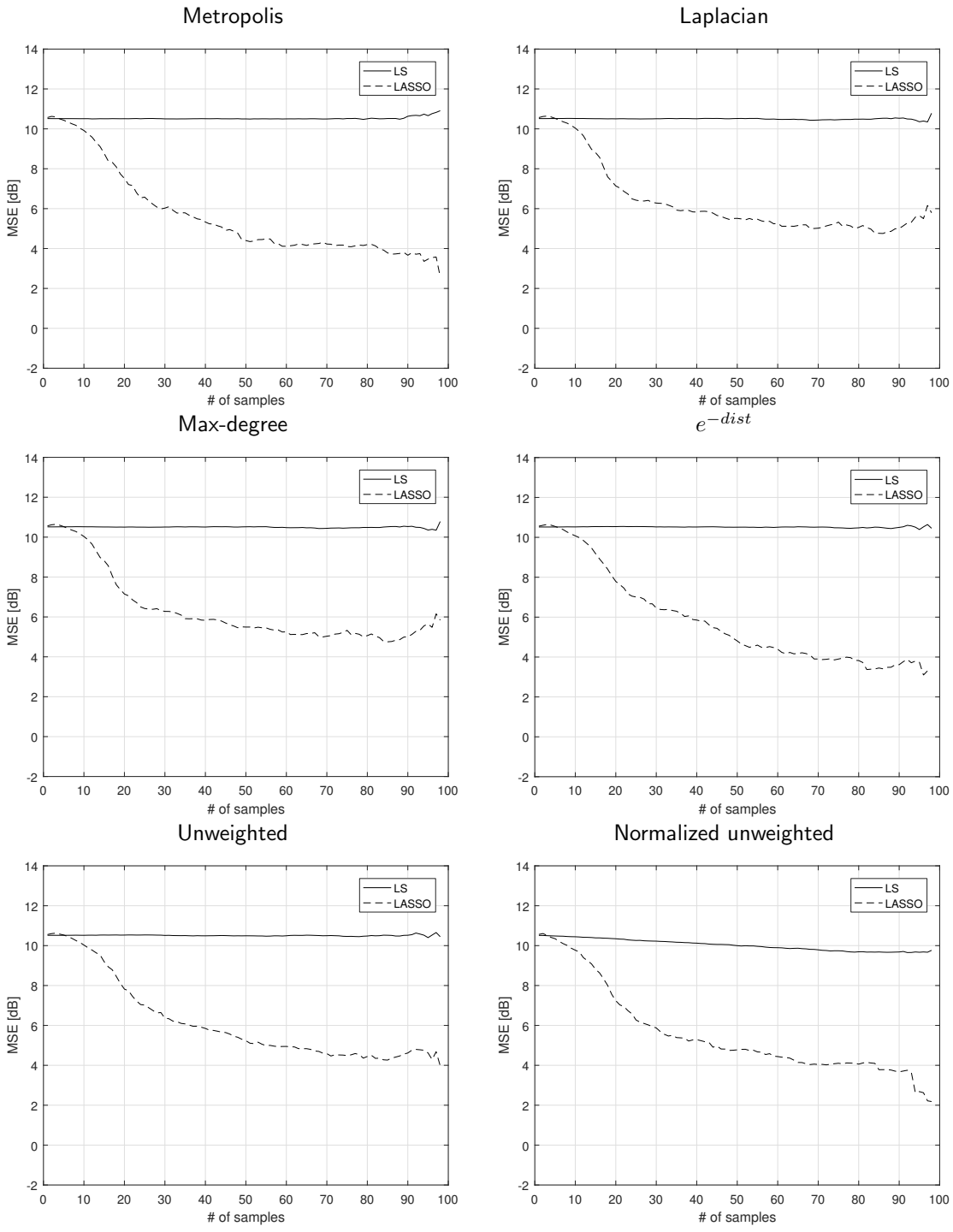



**Figure 3.11:** MSE comparison, 99 frequencies, $\lambda = 1$

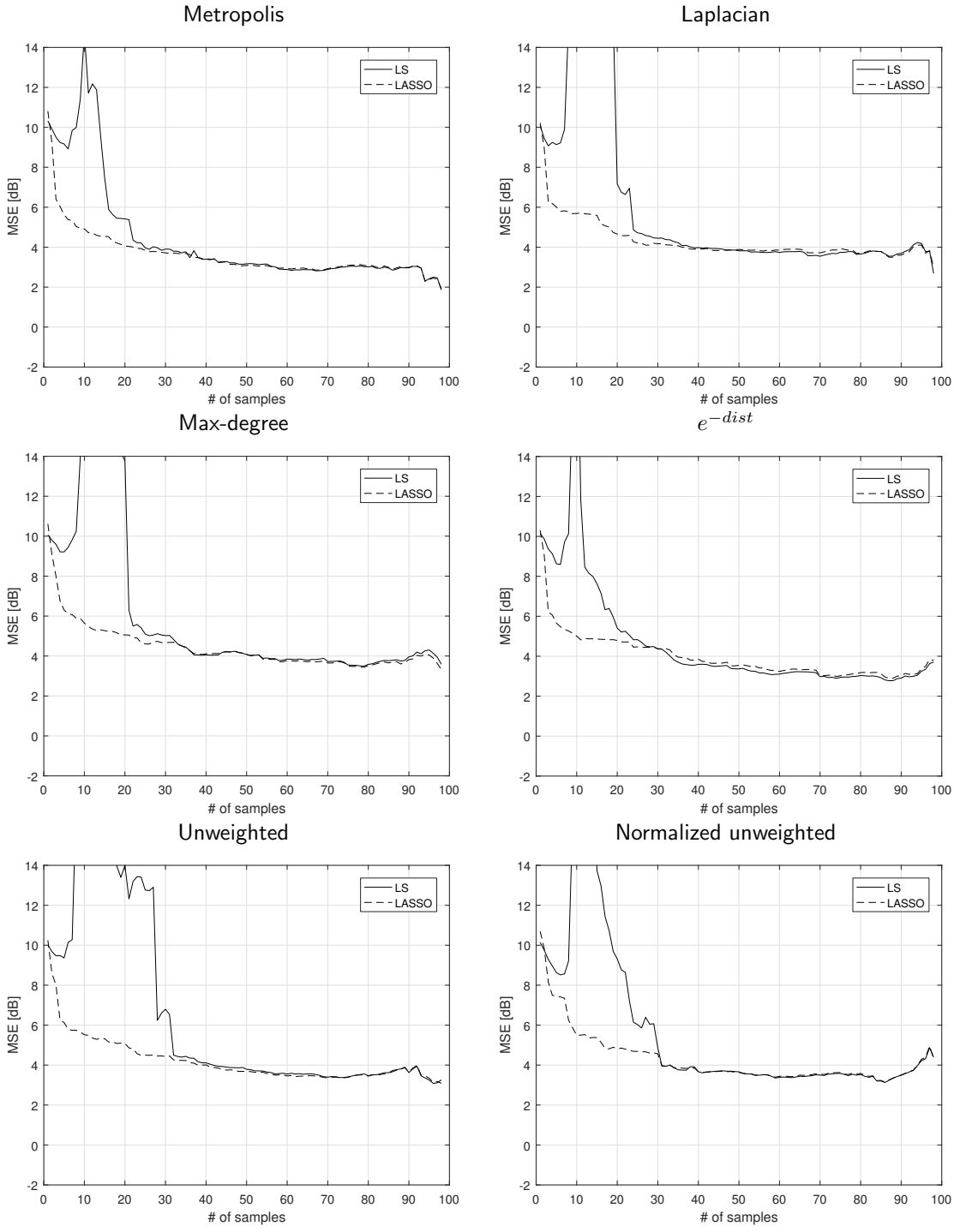



**Figure 3.12:** MSE comparison, 10 frequencies, $\lambda = 1$

In Fig 3.12 we show the results for a different number of active frequencies, which in this case is 10: the LS solution behaves bad when the number of sampled nodes is smaller than the size of the estimated frequency content, i.e., $T < F$, especially when it is equal to the number of estimated frequencies, which exploit the uninvertibility of the eigenvectors matrix, and the $\ell_1$-norm regularization is useful, because it deviates the solution of problem (3.18) to sparse vector; on the other hand, when the number of sampled nodes increases, the performance of the LS achieves the ones of the LASSO. In both the simulations we set the $\ell_1$-norm weight $\lambda = 1$.

We can conclude that the $\ell_1$-norm regularization is useful when the frequency set cardinality is greater than the number of sampled nodes, because it induces sparsity in the solution and therefore the reconstructed signal is more similar to the original one (because of the intrinsic structure of the GFT of the signal), and that its solution corresponds to the LS one when we have a sufficiently high number of sampled nodes, even if the complexity of the computation is larger.

# 4

# Distributed Algorithms

In this chapter we want to formalize and solve the reconstruction problem in a distributed way.

A first simple idea of distributed algorithm is the average consensus, in which the nodes of the network try to reach the same common value which is the average between their initial quantities.

Another distributed algorithm that can be useful is the Alternating Direction Method of Multipliers (ADMM), which is more complex than the average consensus. It is a powerful algorithm that solves optimization problems decomposing them into smaller local sub-problems, which are easier to handle. The solution of there local subproblems are coordinated to find the solution to a global problem. This algorithm is well suited for distributed optimization and in the latest years has found several applications in different areas.

## 4.1 Average Consensus

Consensus is a commonly adopted term to denote the efficient exchange of information between nodes in a network, with the final aim to converge to a common and agreed value.

A typical application of consensus finds a place in the field of WSN, where

distributed sensor measurements need to be averaged to reduce the uncertainty on the measure, in which case we talk of average consensus. This is so far the most studied problem, carrying the simplicity of the target (an average) together with the complexity involved in the identification of efficient distributed methods to reach this target.

In distributed consensus problem a group of nodes (agents, sensors) have to reach a common decision in a distributed fashion. Some of its applications include distributed agreement, synchronization problem [57], multi-vehicle control and navigation [58] and load balancing in parallel processors [59].

Distributed consensus algorithm in its most simple form reduces to average consensus algorithm, where the nodes have to compute the average of their initial states [58], [60], [61], [62], [63]. Average consensus problem is an inevitable part of the solution for more complex problems in several applications. Some of these applications are multiagent distributed coordination and flocking [62], [64], distributed data fusion in sensor networks [65], gossip algorithms [66], sensor localization [67] and distributed estimation and detection for decentralized sensor networks [68], [69].

In average consensus algorithm each node updates its state by a weighted average of its own and neighbors' states. Convergence rate of the algorithm depends on the choice of weights.

Average consensus algorithm intends to compute the average of initial states of node, $\bar{\boldsymbol{x}} = \left(\mathbf{1}\mathbf{1}^T/N\right)\boldsymbol{x}\left(0\right)$, by using local communication between neighboring nodes. $\boldsymbol{x}\left(0\right)$ is the vector of initial states of nodes and $\mathbf{1}$ denotes the column vector with all coefficients one. At each iteration the statte of each node updates according to

$$\boldsymbol{x}\left(t+1\right) = \boldsymbol{A}\boldsymbol{x}\left(t\right). \tag{4.1}$$

$\boldsymbol{A}$ is a $N \times N$ real valued doubly stochastic matrix modeling the exchange of information, which in this case corresponds to the weighted adjacency matrix of the graph, $t = 0, 1, 2, \ldots$ is the discrete time index and $N$ is the number of nodes in the network.

We have seen that considering a centralized approach the implementation of the solution needs to collect all the values $\boldsymbol{U}_{t,f}^{\tau}\boldsymbol{r}_t$ in a "central node" and sum them for each $t \in \tau$.

Now we want to consider a distributed approach, where each node knows only local information about its neighborhood: by means of the average consensus algorithm, each vertex can exchange information between itself and its directly-connected neighbors about the quantity that needs to be estimated, and after a sufficiently high number of messages exchange the network is able to reach the same result of the centralized approach.

The application of the average consensus in our case is the following: for the estimated signal in frequency, instead of computing the product $\hat{\boldsymbol{S}}_f = \boldsymbol{M}\boldsymbol{r}_t$, where $\boldsymbol{M} = \left(\boldsymbol{U}_{t,f}^T \boldsymbol{U}_{t,f}\right)^{-1} \boldsymbol{U}_{t,f}^\tau$, we can write for each component of $\hat{\boldsymbol{S}}_f$:

$$\hat{\boldsymbol{S}}_{f,i} = \sum_{j=1}^{N_s} \boldsymbol{M}_{i,j}\boldsymbol{r}_j = \sum_{j=1}^{N_s} \boldsymbol{C}_j, \quad \forall i = 1, \ldots, N_f. \tag{4.2}$$

where $N_s$ is the number of samples of the signal in the vertex domain and $N_f$ is the length of the estimated frequency content.

Therefore for each frequency index $i = 1, \ldots, N_f$ we can separate the contribution of each sampled node, while for the centralized approach we did not because the computation of the matrix product involves summations between different indices, and we can collect them in a matrix $\boldsymbol{C}$ where $\boldsymbol{C}_{i,j}$ is the contribution of vertex $i$ for the frequency index $j$. The values for the non-sampled nodes in $\boldsymbol{C}$ is initialized to 0.

At each iteration of the consensus algorithm the values of $\boldsymbol{C}$ are updated according to

$$\boldsymbol{C}\,(t+1) = \boldsymbol{A}\boldsymbol{C}\,(t) \tag{4.3}$$

that is, each vertex shares its value with its neighbors and updates it according to the values received from the neighborhood. After a sufficient number of iteration, that depends on the values of the adjacency matrix, the contribution of each node will converge to the same value.

The fundamental property of the adjacency matrix $\boldsymbol{A}$ is that it is doubly-stochastic: in this way the update rule (4.3) consists in a weighted average between the values of the node itself and its neighborhood. Finally, in order to obtain the summation of each contribution, it is necessary to multiply the estimated signal by the number of nodes $N$: first we report the estimated signal (for each node) in the vertex domain by taking its IGFT, then we multiply the mean value by

$N$, obtaining the same result as the centralized approach as we show in Fig. 4.1, where dashed lines represent the centralized reconstruction error while solid lines depict the consensus behaviour, for different number of sampled nodes.

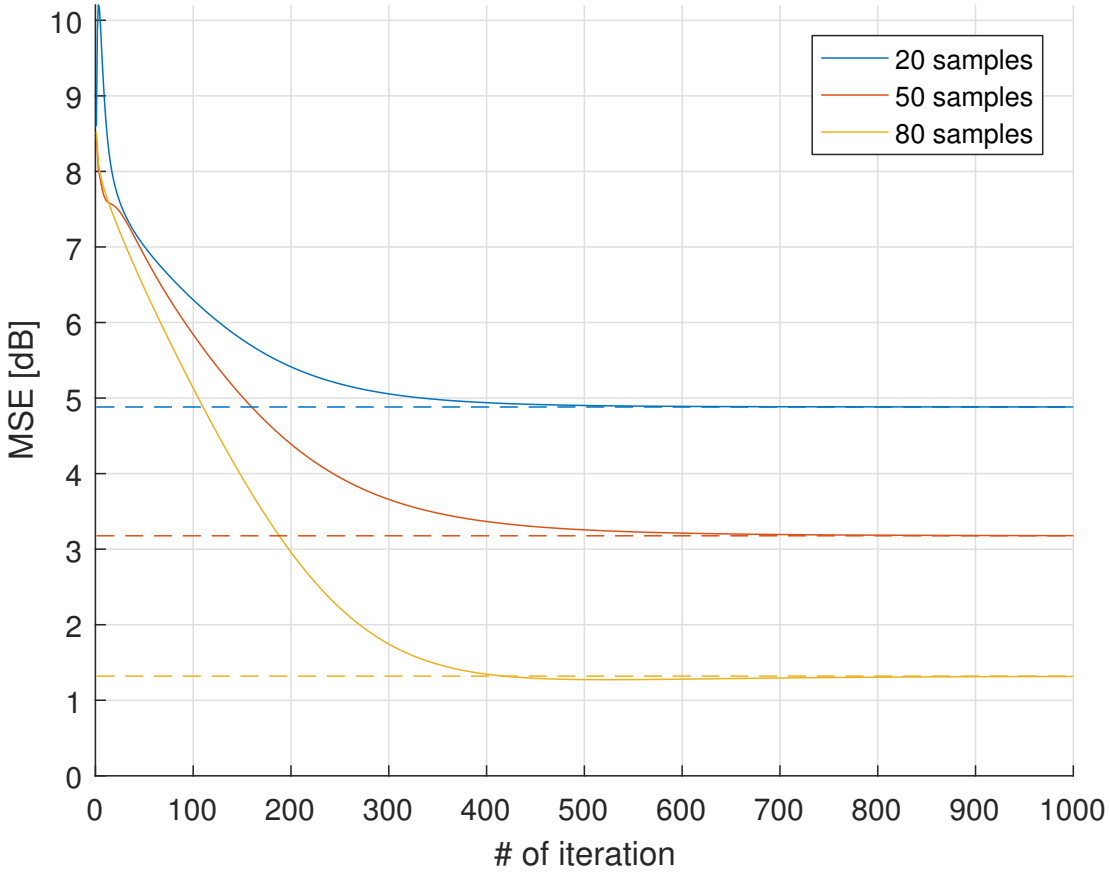



**Figure 4.1:** Convergence of average consensus algorithm

The convergence of the consensus solution is obviously related to the intrinsic structure of the adjacency matrix, for which in this case we defined the weights as the Metropolis ones (3.10), and to the structure of the sampling pattern: both these parameters affects the number of iterations required for convergence.

## 4.2 ADMM: Alternating Direction Method of Multipliers

In this section we want to exploit the ADMM to improve the performance of the reconstruction algorithm. ADMM is a powerful method introduced in [70], [71] in the 1970s which has the robustness of method of multipliers and can support decomposition. Today it finds applications thanks to the presence of large-scale distributed computing systems and the needs to solve massive optimization problems. In this section, we refer to the formulation of [72].

In order to introduce the algorithm in a generic way, which will be useful for further applications with different optimization functions, we start from a generic problem in the form

$$\underset{\boldsymbol{y}}{\text{minimize}} \quad F(\boldsymbol{y}) = \sum_{i=1}^{N} F_i(\boldsymbol{y}_i) \tag{4.4}$$

assuming that the objective function $F(\boldsymbol{y}) = \sum_{i=1}^{N} F_i(\boldsymbol{y}_i)$ is a separable function which corresponds to the summation of local objective function $F_i$ over all the nodes of the network. In our reconstruction problem, the function that needs to be minimized is

$$F_i(\boldsymbol{y}_i) = \begin{cases} \frac{1}{2}(\boldsymbol{r}_i - \boldsymbol{U}_i \boldsymbol{y}_i)^2 & i \in \tau \\ 0 & i \in \bar{\tau} \end{cases} \tag{4.5}$$

where $\boldsymbol{r}_i$ is the $i$-th component of the sampled signal, $\boldsymbol{U}_i$ is the $i$-th row of the eigenvector matrix $\boldsymbol{U}_{t,f}$ and the factor $\frac{1}{2}$ is considered to simplify the derivation.

We need to put this problem in a form which is suitable for the application of the ADMM. To do this, we duplicate the $\boldsymbol{y}$ variable in many variables $\boldsymbol{y}_i$ such that the previous problem can be rewritten into a new form, taking in consideration these duplicated variables

$$\begin{aligned} &\underset{\boldsymbol{y}_i}{\text{minimize}} \quad \sum_{i=1}^{N} F_i(\boldsymbol{y}_i) \\ &\text{subject to} \quad \boldsymbol{y}_i = \boldsymbol{y}_j, \forall j \in \mathcal{N}_i, \forall i = 1, \ldots, N, \end{aligned} \tag{4.6}$$

and imposing the equality between neighboring nodes frequency vectors: the local copy of each node variable $\boldsymbol{y}_i$ must be equal to the one of its neighbors $\boldsymbol{y}_j$. With this constraint each node will have the same values of the variable and, instead of having only one vector, we will have many vectors (as many as the number of nodes) but every one equal to each other.

The constraint on the duplicated variables can be rewritten as $\boldsymbol{y}_i = \boldsymbol{z}$, such that the minimization problem becomes

$$\begin{aligned} \underset{\boldsymbol{y}_i}{\text{minimize}} \quad & \sum_{i=1}^{N} F_i\left(\boldsymbol{y}_i\right) \\ \text{subject to} \quad & \boldsymbol{y}_i = \boldsymbol{z}, \forall i = 1, \ldots, N. \end{aligned} \tag{4.7}$$

In this way, we need a global knowledge of some variables: specifically, each local variable $\boldsymbol{y}_i$ depends on the same $\boldsymbol{z}$. We need to enable each node to update its variables in an autonomous way using information gathered from its neighbors.

As described in [73], in order to obtain a distributed version of the problem, we can write

$$\begin{aligned} \underset{\boldsymbol{y},\boldsymbol{z}}{\text{minimize}} \quad & \sum_{i=1}^{N} F_i\left(\boldsymbol{y}_i\right) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{z} \\ & \boldsymbol{z} \in \mathcal{Z} = \{\boldsymbol{z}_{i,j} = \boldsymbol{z}_{j,i}\} \end{aligned} \tag{4.8}$$

where $\boldsymbol{y} = [\boldsymbol{y}_i]_{i=1,\ldots,N}$, $\boldsymbol{z} = [\boldsymbol{z}_{i,j}]_{\forall j \in \mathcal{N}_i, i=1,\ldots,N}$, imposing the constraints $\boldsymbol{y}_i = \boldsymbol{z}_{i,j},\ \forall j \in \mathcal{N}_i,\ i = 1, \ldots, N$ through the block diagonal matrix $\boldsymbol{A}$, which duplicates the information of each node to its neighbors. The set $\mathcal{Z}$ is useful for coordinate the values of the different $\boldsymbol{z}_{i,j}$, obtaining a distributed version of the problem. This approach is equivalent to the one in [74]. In its distributed version, the ADMM algorithm is the most used method for distributed coordination of agents [72], [75]. In our specific reconstruction algorithm the objective function takes values as stated in (4.5), therefore our problem can be finally written as

$$\begin{aligned} \underset{\boldsymbol{y},\boldsymbol{z}}{\text{minimize}} \quad & \sum_{i \in \tau} \frac{1}{2} \left(\boldsymbol{r}_i - \boldsymbol{U}_i \boldsymbol{y}_i\right)^2 \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{z} \\ & \boldsymbol{z} \in \mathcal{Z} = \{\boldsymbol{z}_{i,j} = \boldsymbol{z}_{j,i}\}. \end{aligned} \tag{4.9}$$

To find the solution of problem (4.9), and in general for every problem in the form (4.8), we look for the stationary points of the augmented Lagrangian function, which is defined as

$$L(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\lambda}) = F(\boldsymbol{y}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{y} - \boldsymbol{z} \rangle + \frac{\epsilon}{2} \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{z}\|^2 \tag{4.10}$$

where the vector $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_i]_{i=1,\dots,N}$ collects the Lagrangian multipliers of the nodes and $\epsilon$ is the penalty parameter, which weights the penalty term and must be positive.

To look for stationary points of the augmented Lagrangian function (4.10) we perform an alternating search, which, at each iteration, performs an update of vectors $\boldsymbol{y}$, $\boldsymbol{z}$ and $\boldsymbol{\lambda}$. We define the new values of these vectors at each iteration by using the notation $\boldsymbol{y}^*$, $\boldsymbol{z}^*$ and $\boldsymbol{\lambda}^*$.

We start setting null initial conditions, imposing $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ variables to be equal to $\mathbf{0}$ before the first iteration: this initialization will also be successively useful to obtain a more compact formulation.

At each iteration, the ADMM update consists of

$$\begin{aligned} \boldsymbol{y}^* &= \arg\min_{\boldsymbol{y}} \; L(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\lambda}) \\ \boldsymbol{z}^* &= \arg\min_{\boldsymbol{z} \in \mathcal{Z}} \; L(\boldsymbol{y}^*, \boldsymbol{z}, \boldsymbol{\lambda}) \\ \boldsymbol{\lambda}^* &= \boldsymbol{\lambda} + \epsilon (\boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{z}^*) \end{aligned} \tag{4.11}$$

which can be rewritten, exploiting the Lagrangian function, as

$$\begin{aligned} \boldsymbol{y}^* &= \arg\min_{\boldsymbol{y}} \; F(\boldsymbol{y}) + \boldsymbol{\lambda}^T \boldsymbol{A}\boldsymbol{y} + \frac{\epsilon}{2} \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{z}\|^2 \\ \boldsymbol{z}^* &= \arg\min_{\boldsymbol{z} \in \mathcal{Z}} \; -\boldsymbol{\lambda}^T \boldsymbol{z} + \frac{\epsilon}{2} \|\boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{z}\|^2 \\ \boldsymbol{\lambda}^* &= \boldsymbol{\lambda} + \epsilon (\boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{z}^*) . \end{aligned} \tag{4.12}$$

A special consideration must be done on the importance of the penalty parameter $\epsilon$, that must be set to a proper value in order to reach the convergence [76].

### 4.2.1 Distributed ADMM - LS Solution

Now we want to compute the distributed solution of the ADMM algorithm: starting from (4.12), we look for a closed form solution for each variable update rules.

- $\boldsymbol{y}$ update: the update can assume two different forms, depending on whether the node $i$ is sampled or not. Exploiting the objective function (4.5), for each node $i \in \tau$ we have

$$\boldsymbol{y}_i^* = \arg\min_{\boldsymbol{y}_i} \quad \frac{1}{2}\left(\boldsymbol{r}_i - \boldsymbol{U}_i\boldsymbol{y}_i\right)^2 + \sum_{j\in\mathcal{N}_i} \boldsymbol{\lambda}_{i,j}^T\boldsymbol{y}_i + \frac{\epsilon}{2}\sum_{j\in\mathcal{N}_i}\left\|\boldsymbol{y}_i - \boldsymbol{z}_{i,j}\right\|^2. \tag{4.13}$$

  Deriving (4.13) with respect to $\boldsymbol{y}_i$ and imposing the equality with 0:

$$\boldsymbol{U}_i^T\left(\boldsymbol{U}_i\boldsymbol{y}_i - \boldsymbol{r}_i\right) + \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\left(\boldsymbol{y}_i - \boldsymbol{z}_{i,j}\right) = 0 \tag{4.14}$$

$$\left(\boldsymbol{U}_i^T\boldsymbol{U}_i + \epsilon|\mathcal{N}_i|\boldsymbol{I}\right)\boldsymbol{y}_i = \boldsymbol{U}_i^T\boldsymbol{r}_i - \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j} \tag{4.15}$$

$$\boldsymbol{y}_i^* = \left(\epsilon|\mathcal{N}_i|\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i - \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right). \tag{4.16}$$

  Instead for $i \in \bar{\tau}$:

$$\boldsymbol{y}_i^* = \frac{1}{\epsilon|\mathcal{N}_i|}\left(-\sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right) \tag{4.17}$$

  Therefore the update for the local variable $\boldsymbol{y}$ at each vertex $i$ can be written as

$$\boldsymbol{y}_i^* = \begin{cases} \left(\epsilon|\mathcal{N}_i|\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i - \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right) & \text{for } i \in \tau \\ \frac{1}{\epsilon|\mathcal{N}_i|}\left(-\sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right) & \text{for } i \in \bar{\tau} \end{cases} \tag{4.18}$$

- $\boldsymbol{z}$ update: expanding the second of (4.12) we obtain

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}\in\mathcal{Z}} - \boldsymbol{z}^T\boldsymbol{\lambda} - \epsilon\left(\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{y}^*\right) + \frac{\epsilon}{2}\left\|\boldsymbol{z}\right\|^2. \tag{4.19}$$

By dividing each element by $\epsilon$ and collecting $-\boldsymbol{z}^T$:

$$\begin{aligned}\boldsymbol{z}^* &= \arg\min_{\boldsymbol{z}\in\mathcal{Z}} \frac{1}{2}\left\|\boldsymbol{z}\right\|^2 - \boldsymbol{z}^T\left(\boldsymbol{A}\boldsymbol{y}^* + \frac{\boldsymbol{\lambda}}{\epsilon}\right)\\ &= \arg\min_{\boldsymbol{z}\in\mathcal{Z}} \frac{1}{2}\left\|\boldsymbol{z} - \left(\boldsymbol{A}\boldsymbol{y}^* + \frac{\boldsymbol{\lambda}}{\epsilon}\right)\right\|^2\\ &= \arg\min_{\boldsymbol{z}\in\mathcal{Z}} \sum_{i,j\in\mathcal{N}_i} \frac{1}{2}\left(\boldsymbol{z}_{i,j} - \boldsymbol{m}_{i,j}\right)^2\end{aligned} \tag{4.20}$$

where $\boldsymbol{m} = \boldsymbol{A}\boldsymbol{y}^* + \frac{\boldsymbol{\lambda}}{\epsilon}$ is a vector built from neighbors information: $\boldsymbol{m}_{i,j} = \boldsymbol{y}_i^* + \boldsymbol{\lambda}_{i,j}/\epsilon$.

Imposing the constraint $\boldsymbol{z} \in \mathcal{Z}$, which corresponds to balance the $\boldsymbol{z}$ variables as $\boldsymbol{z}_{i,j} = \boldsymbol{z}_{j,i}$, we can separate each $\boldsymbol{z}$, obtaining

$$\boldsymbol{z}_{i,j}^* = \boldsymbol{z}_{j,i}^* = \arg\min_{\boldsymbol{z}} \frac{1}{2}\left(\boldsymbol{z} - \boldsymbol{m}_{i,j}\right)^2 + \frac{1}{2}\left(\boldsymbol{z} - \boldsymbol{m}_{j,i}\right)^2 \tag{4.21}$$

whose derivative imposed equal to 0 gives

$$\boldsymbol{z}^* - \boldsymbol{m}_{i,j} + \boldsymbol{z}^* - \boldsymbol{m}_{j,i} = 0 \tag{4.22}$$

and finally we obtain

$$\boldsymbol{z}_{i,j}^* = \boldsymbol{z}_{j,i}^* = \frac{\boldsymbol{m}_{i,j} + \boldsymbol{m}_{j,i}}{2} = \frac{\boldsymbol{y}_i^* + \boldsymbol{y}_j^*}{2} + \frac{\boldsymbol{\lambda}_{i,j} + \boldsymbol{\lambda}_{j,i}}{2\epsilon} \tag{4.23}$$

that is the result of the communication between node $i$ and its neighbor $j$.

- $\boldsymbol{\lambda}$ update: for each $i = 1, \ldots, N$ and $\forall j \in \mathcal{N}_i$ the values of the corresponding Lagrangian multiplier is updated as

$$\boldsymbol{\lambda}_{i,j}^* = \boldsymbol{\lambda}_{i,j} + \epsilon\left(\boldsymbol{y}_i^* - \boldsymbol{z}_{i,j}^*\right). \tag{4.24}$$

Therefore the update rules of the ADMM in its distributed version, for our

specific reconstruction problem, can be written as

$$\begin{aligned}
\boldsymbol{y}_i^* &= \begin{cases} \left(\epsilon|\mathcal{N}_i|\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i - \sum\limits_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum\limits_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right) & \text{for } i \in \tau \\ \frac{1}{\epsilon|\mathcal{N}_i|}\left(-\sum\limits_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum\limits_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}\right) & \text{for } i \in \bar{\tau} \end{cases} \\
\boldsymbol{z}_{i,j}^* &= \frac{\boldsymbol{y}_i^* + \boldsymbol{y}_j^*}{2} + \frac{\boldsymbol{\lambda}_{i,j} + \boldsymbol{\lambda}_{j,i}}{2\epsilon} \\
\boldsymbol{\lambda}_{i,j}^* &= \boldsymbol{\lambda}_{i,j} + \epsilon\left(\boldsymbol{y}_i^* - \boldsymbol{z}_{i,j}^*\right).
\end{aligned} \tag{4.25}$$

We can conclude that, locally (at node $i$) we have the variables $\boldsymbol{z}_{i,j}$ and $\boldsymbol{\lambda}_{i,j}$ for each neighbor $j \in \mathcal{N}_i$, which comes from the previous iteration update. They are used to update the variable $\boldsymbol{y}_i$ and then, exchanging with neighbors the information about the updated $\boldsymbol{y}_i^*$, each node updates its own $\boldsymbol{z}_{i,j}$ and its $\boldsymbol{\lambda}_{i,j}$, using the information of itself and of its neighbors, until all nodes reaches an agreement on each of the values of $\boldsymbol{y}$ and $\boldsymbol{z}$.

In order to simplify the expression of the updates, it is useful to assume $\tilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}/\epsilon$, such that

$$\begin{aligned}
\boldsymbol{y}_i^* &= \begin{cases} \left(\epsilon|\mathcal{N}_i|\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i + \epsilon\left(\sum\limits_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j} - \tilde{\boldsymbol{\lambda}}_{i,j}\right)\right) & \text{for } i \in \tau \\ \frac{1}{|\mathcal{N}_i|}\left(\sum\limits_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j} - \tilde{\boldsymbol{\lambda}}_{i,j}\right) & \text{for } i \in \bar{\tau} \end{cases} \\
\boldsymbol{z}_{i,j}^* &= \frac{\boldsymbol{y}_i^* + \boldsymbol{y}_j^*}{2} + \frac{\tilde{\boldsymbol{\lambda}}_{i,j} + \tilde{\boldsymbol{\lambda}}_{j,i}}{2} \\
\tilde{\boldsymbol{\lambda}}_{i,j}^* &= \tilde{\boldsymbol{\lambda}}_{i,j} + \boldsymbol{y}_i^* - \boldsymbol{z}_{i,j}^*.
\end{aligned} \tag{4.26}$$

Now we can exploit a nice property of the set $\mathcal{Z}$: assume that we start from $\boldsymbol{z} = \boldsymbol{0}$ and $\tilde{\boldsymbol{\lambda}} = \boldsymbol{0}$. By construction, we have that $\boldsymbol{z} \in \mathcal{Z}$ and $\tilde{\boldsymbol{\lambda}} \in \mathcal{Z}^\perp$. Let us explain why.

At the first iteration the update of $\tilde{\boldsymbol{\lambda}}$ is computed as

$$\begin{aligned}
\tilde{\boldsymbol{\lambda}}_{i,j}^* &= \tilde{\boldsymbol{\lambda}}_{i,j} + \boldsymbol{y}_i^* - \boldsymbol{z}_{i,j}^* \\
\tilde{\boldsymbol{\lambda}}_{j,i}^* &= \tilde{\boldsymbol{\lambda}}_{j,i} + \boldsymbol{y}_j^* - \boldsymbol{z}_{j,i}^*
\end{aligned} \tag{4.27}$$

and, since $\tilde{\boldsymbol{\lambda}} = \mathbf{0}$ and $\boldsymbol{z}_{i,j} = \boldsymbol{z}_{j,i}$ by construction, we are summing $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ to the same value of $\boldsymbol{z}_{i,j} = \boldsymbol{z}_{j,i}$, respectively in the first and in the second of (4.27). We can write $\boldsymbol{z}^* = \boldsymbol{L}_{\mathcal{Z}} \left( \boldsymbol{A}\boldsymbol{y}^* + \tilde{\boldsymbol{\lambda}} \right)$, where $\boldsymbol{L}$ is called projector: it is a matrix that extract from the vector $\left( \boldsymbol{A}\boldsymbol{y}^* + \tilde{\boldsymbol{\lambda}} \right)$ the component that belongs to the linear space $\mathcal{Z}$; it has the nice property that $\boldsymbol{L}^2 = \boldsymbol{L}$ and therefore has eigenvalues 1, in $\boldsymbol{z}$, and 0 elsewhere. Therefore the update of $\tilde{\boldsymbol{\lambda}}$ can be written as $\tilde{\boldsymbol{\lambda}}^* = \tilde{\boldsymbol{\lambda}} + \boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{z}^* = (\boldsymbol{I} - \boldsymbol{L}_{\mathcal{Z}}) \left( \boldsymbol{A}\boldsymbol{y}^* + \tilde{\boldsymbol{\lambda}} \right)$ and we can conclude that $\tilde{\boldsymbol{\lambda}}$ and $\boldsymbol{z}$ are orthogonal to each other, since they are the same vector multiplied by $\boldsymbol{I} - \boldsymbol{L}_{\mathcal{Z}}$ and $\boldsymbol{L}_{\mathcal{Z}}$.

Formally, if we sum the left and the right of (4.27), we obtain:

$$\begin{aligned}
\tilde{\boldsymbol{\lambda}}^*_{i,j} + \tilde{\boldsymbol{\lambda}}^*_{j,i} &= \tilde{\boldsymbol{\lambda}}_{i,j} + \boldsymbol{y}^*_i - \boldsymbol{z}^*_{i,j} + \tilde{\boldsymbol{\lambda}}_{j,i} + \boldsymbol{y}^*_j - \boldsymbol{z}^*_{j,i} \\
&= \tilde{\boldsymbol{\lambda}}_{i,j} + \tilde{\boldsymbol{\lambda}}_{j,i} + \boldsymbol{y}^*_i + \boldsymbol{y}^*_j - \frac{\boldsymbol{y}^*_i + \boldsymbol{y}^*_j}{2} - \frac{\tilde{\boldsymbol{\lambda}}_{i,j} + \tilde{\boldsymbol{\lambda}}_{j,i}}{2} - \frac{\boldsymbol{y}^*_j + \boldsymbol{y}^*_i}{2} - \frac{\tilde{\boldsymbol{\lambda}}_{j,i} + \tilde{\boldsymbol{\lambda}}_{i,j}}{2} \\
&= \tilde{\boldsymbol{\lambda}}_{i,j} + \tilde{\boldsymbol{\lambda}}_{j,i} + \boldsymbol{y}^*_i + \boldsymbol{y}^*_j - \boldsymbol{y}^*_i - \boldsymbol{y}^*_j - \tilde{\boldsymbol{\lambda}}_{i,j} - \tilde{\boldsymbol{\lambda}}_{j,i} \\
&= 0
\end{aligned} \tag{4.28}$$

that ensures the orthogonality between $\boldsymbol{z}$ and $\tilde{\boldsymbol{\lambda}}$.

Thus $\boldsymbol{z} \in \mathcal{Z}$ and $\tilde{\boldsymbol{\lambda}} \in \mathcal{Z}^\perp$ by construction and, for the Lagrangian multipliers, it must be satisfied $\tilde{\boldsymbol{\lambda}}_{i,j} = -\tilde{\boldsymbol{\lambda}}_{j,i}$ and so the update of $\boldsymbol{z}$ is simplified to

$$\boldsymbol{z}^*_{i,j} = \frac{\boldsymbol{y}^*_i + \boldsymbol{y}^*_j}{2} \tag{4.29}$$

With this simplification, we can exploit compact updates and simplify them: if we define $\boldsymbol{z}_i = \sum_{j \in \mathcal{N}_i} \boldsymbol{z}_{i,j}$ and $\tilde{\boldsymbol{\lambda}}_i = \sum_{j \in \mathcal{N}_i} \tilde{\boldsymbol{\lambda}}_{i,j}$, the updating rules of $\boldsymbol{z}$ and $\tilde{\boldsymbol{\lambda}}$ becomes

$$\boldsymbol{z}^*_i = \sum_{j \in \mathcal{N}_i} \boldsymbol{z}^*_{i,j} = \frac{1}{2} \left( \boldsymbol{y}^*_i |\mathcal{N}_i| + \sum_{j \in \mathcal{N}_i} \boldsymbol{y}_j \right) \tag{4.30}$$

$$\tilde{\boldsymbol{\lambda}}^*_i = \sum_{j \in \mathcal{N}_i} \tilde{\boldsymbol{\lambda}}^*_{i,j} = \tilde{\boldsymbol{\lambda}}_i + \boldsymbol{y}^*_i |\mathcal{N}_i| - \boldsymbol{z}^*_i \tag{4.31}$$

that, instead of collecting many replicas, it is sufficient to gather just one variable.

Now we can write the final expressions for the update rules of the ADMM

algorithm for the reconstruction problem:

$$\begin{aligned}
\boldsymbol{y}_i^* &= \begin{cases} \left(\epsilon|\mathcal{N}_i|\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i + \epsilon\left(\boldsymbol{z}_i - \tilde{\boldsymbol{\lambda}}_i\right)\right) & i \in \tau \\ \frac{1}{|\mathcal{N}_i|}\left(\boldsymbol{z}_i - \tilde{\boldsymbol{\lambda}}_i\right) & i \in \bar{\tau} \end{cases} \\
\boldsymbol{z}_i^* &= \frac{1}{2}\left(\boldsymbol{y}_i^*|\mathcal{N}_i| + \sum_{j\in\mathcal{N}_i}\boldsymbol{y}_j^*\right) \\
\tilde{\boldsymbol{\lambda}}_i^* &= \tilde{\boldsymbol{\lambda}}_i + \boldsymbol{y}_i^*|\mathcal{N}_i| - \boldsymbol{z}_i^*
\end{aligned} \tag{4.32}$$

where $\boldsymbol{z}_i$ and $\tilde{\boldsymbol{\lambda}}_i$ are defined as (4.30) and (4.31), respectively.

This version of the ADMM is still expensive from the computational point of view, since it requires the inversion of a matrix in the update of the $\boldsymbol{y}$ variables for the sampled nodes, whose dimension is $|\mathcal{F}| \times |\mathcal{F}|$: when the number of active frequencies is high, the inversion requires a considerable computational time. Since the matrix that needs to be inverted is an identity, multiplied by a constant factor, plus something, in the form $\left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right)$, the inverse has the form $\left(\beta\boldsymbol{I} + \gamma\boldsymbol{U}^T\boldsymbol{U}\right)$. Therefore, since $\left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right)^{-1}\left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right) = \boldsymbol{I}$, we can write the inverse matrix in a closed form:

$$\begin{aligned}
\boldsymbol{I} &= \left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right)\left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right)^{-1} \\
&= \left(\alpha\boldsymbol{I} + \boldsymbol{U}^T\boldsymbol{U}\right)\left(\beta\boldsymbol{I} + \gamma\boldsymbol{U}^T\boldsymbol{U}\right) \\
&= \alpha\beta\boldsymbol{I} + \alpha\gamma\boldsymbol{U}^T\boldsymbol{U} + \beta\boldsymbol{U}^T\boldsymbol{U} + \gamma\left\|\boldsymbol{U}\right\|^2\boldsymbol{U}^T\boldsymbol{U}
\end{aligned} \tag{4.33}$$

and we obtain the system

$$\begin{cases} \alpha\beta = 1 \\ \alpha\gamma + \beta + \gamma\left\|\boldsymbol{U}\right\|^2 = 0 \end{cases} \rightarrow \begin{cases} \beta = 1/\alpha \\ \gamma = -\beta/\left(\alpha + \left\|\boldsymbol{U}\right\|^2\right) \end{cases} \tag{4.34}$$

which exploits the values of $\beta$ and $\gamma$.

Finally, since in the first of (4.32) the inverse matrix is multiplied by the vector $\boldsymbol{x} = \left(\boldsymbol{U}_i^T\boldsymbol{r}_i + \epsilon\left(\boldsymbol{z}_i - \tilde{\boldsymbol{\lambda}}_i\right)\right)$, it is possible to compact the expression as

$$\begin{aligned}
\left(\beta\boldsymbol{I} + \gamma\boldsymbol{U}^T\boldsymbol{U}\right)\boldsymbol{x} &= \beta\boldsymbol{x} + \underbrace{\left(\gamma\left(\boldsymbol{U}^T\boldsymbol{x}\right)\right)}_{\delta\,=\,\gamma\,\cdot\,\langle\boldsymbol{U},\,\boldsymbol{x}\rangle}\boldsymbol{U} \\
&= \beta\boldsymbol{x} + \delta\boldsymbol{U}
\end{aligned} \tag{4.35}$$

where $\delta = \gamma \cdot \langle \boldsymbol{U}, \boldsymbol{x} \rangle$ is a constant term (for each vertex of the network), which ensures linear complexity, instead of a quadratic one, for the $\boldsymbol{y}$ update.

Now we want to exhibit some numerical results obtained by simulating the distributed algorithm on the network presented is Section 3.1. The signal here is represented by the temperatures of month March and the random sampling structure and the frequencies ordering are the same of the centralized algorithm, as described in Section 3.2.

In Fig. 4.2 we show the behaviour of the MSE, between the estimated signal and the original one in vertex domain, at each iteration of the distributed ADMM algorithm, for different values of the penalty parameter $\epsilon$. In this case we fixed the number of sampled nodes to 50 and the number of frequencies to 10, and the weighted adjacency matrix is constructed with Metropolis weights (3.10).

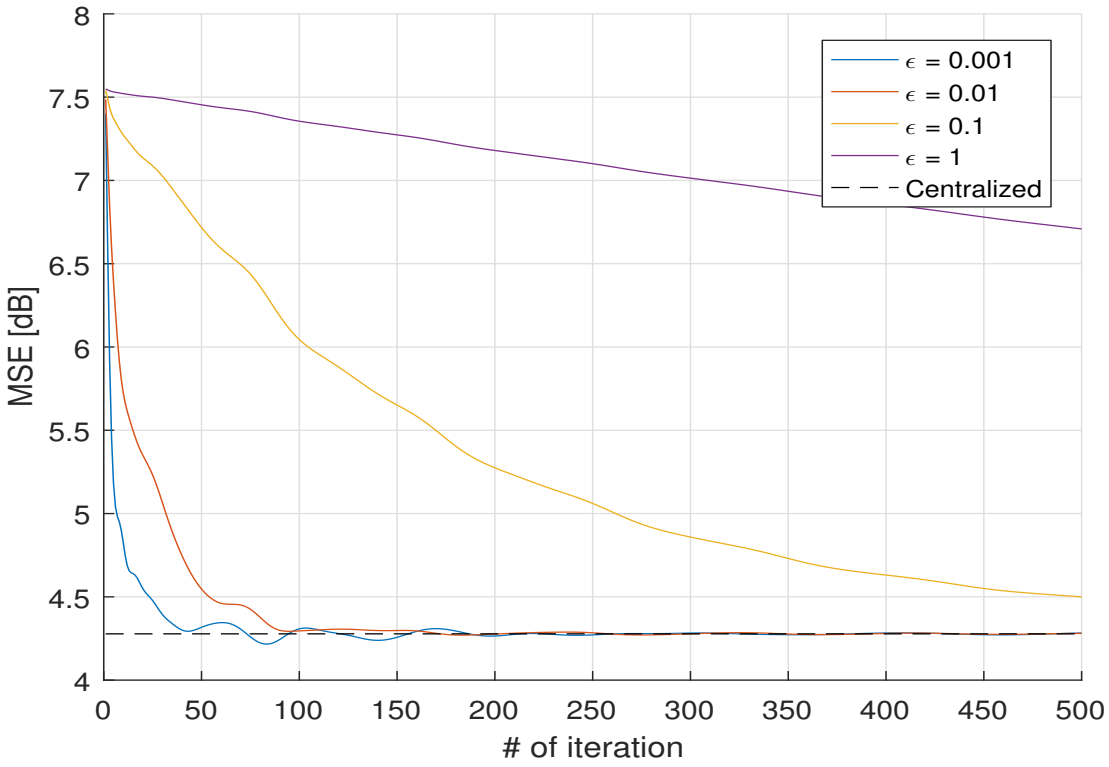



**Figure 4.2:** Convergence of the ADMM solution - LS

It is fundamental to understand how the choice of the penalty parameter impacts on the MSE performance: we can observe that a small value of $\epsilon$ ensures faster convergence to the same centralized MSE with respect to an higher one.

Instead, if we fix the penalty parameter to a bigger value, the error does not converges within a few number of iterations of the algorithm and so its estimated signal is not reliable.

Furthermore, the penalty parameter affects the convergence of the two ADMM variables $\boldsymbol{y}$ and $\boldsymbol{z}$, as stated in (4.32): a larger value of $\epsilon$ ensures quick convergence, as shown in Fig. 4.3. However, this is a measure of the similarity of the two variables and the value of the difference in itself does not mean anything, except that they are converging. What is more meaningful is the value of the cost function: we are interested in evaluating how the MSE saturates. From Fig. 4.2 we can conclude that a small value of the penalty parameter $\epsilon$ ensures a correct estimation of the signal even for few number of iterations of the algorithm. Thus the local estimates, even if the convergence of $\boldsymbol{y}$ and $\boldsymbol{z}$ is not perfect, after a small number of iterations reach the desired level, which is the centralized one.

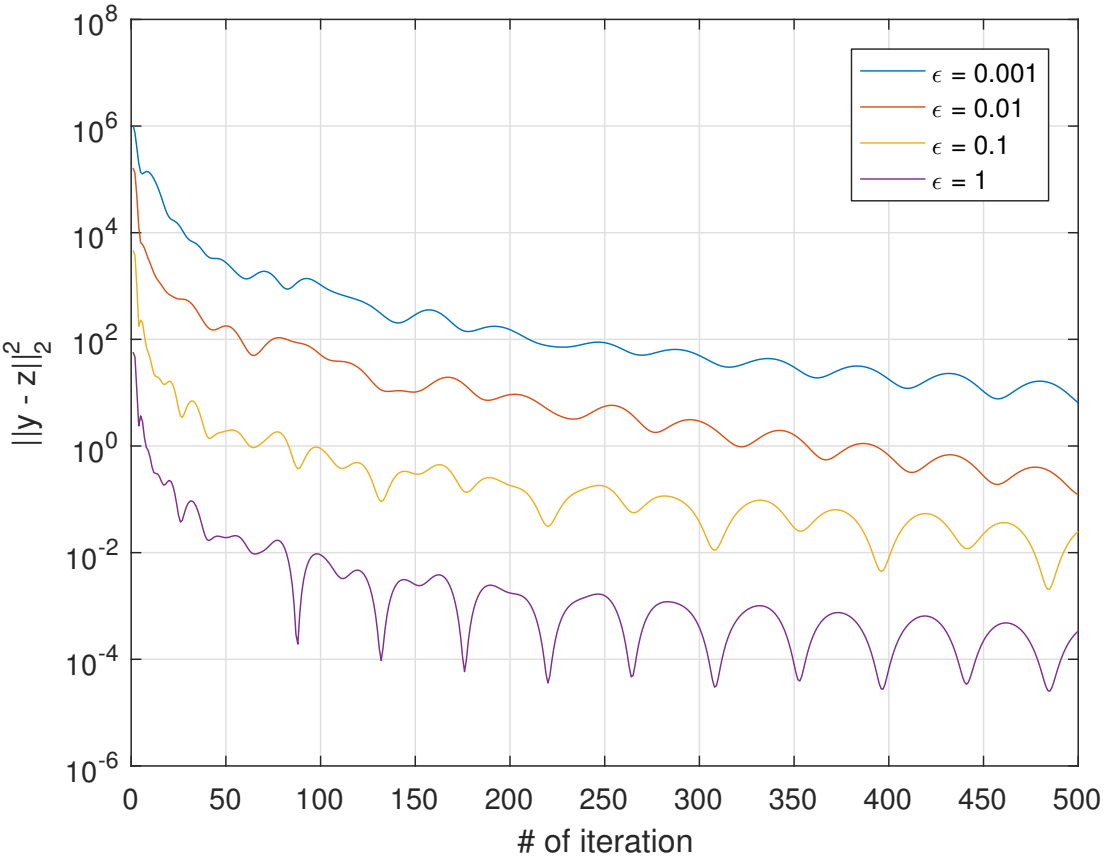



**Figure 4.3:** Convergence of the ADMM variables - LS

Furthermore, we report in Fig. 4.4 the average time taken to run a simulation of the ADMM algorithm, for different numbers of iteration: the compact update rules (4.32), but especially the formulation of the inverse matrix in (4.33), (4.34) and (4.35), guarantee a substantial improvement in the computation time, considering that it is a distributed algorithm involving a hundred of nodes.

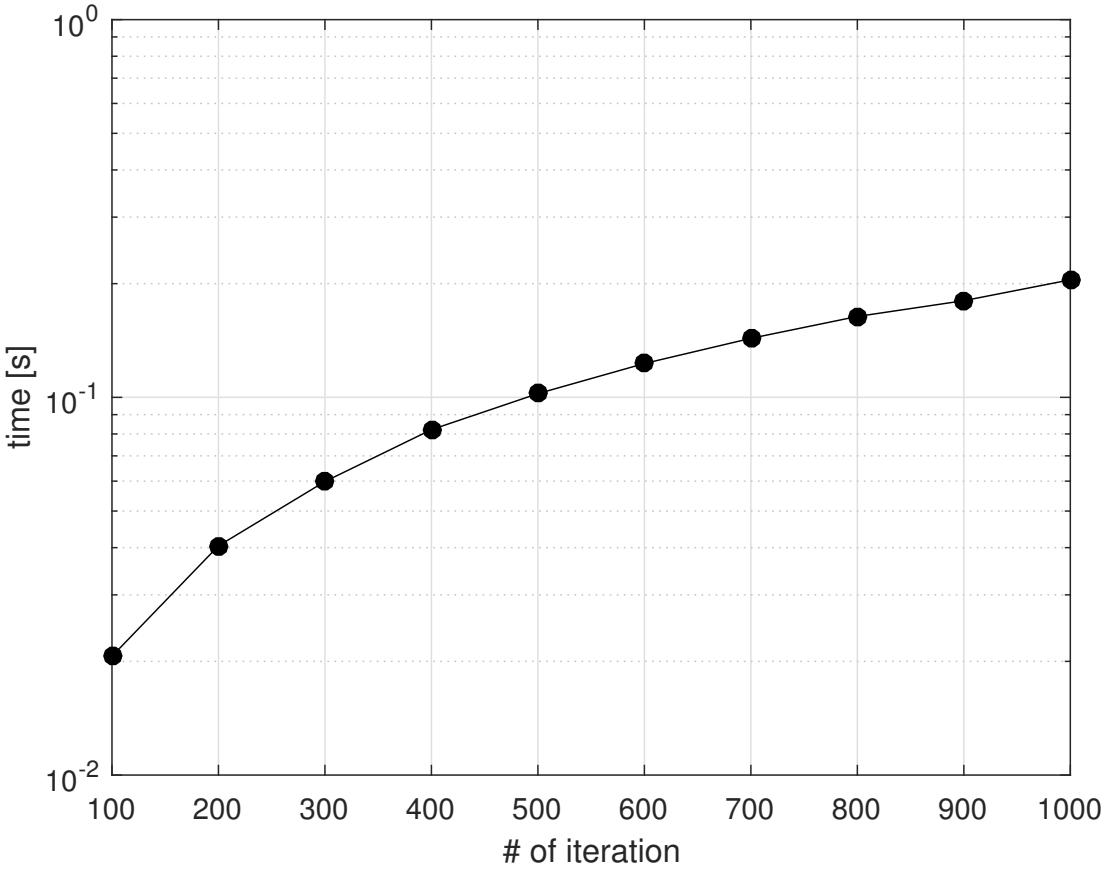



**Figure 4.4:** Average time to run ADMM - LS

Finally, we can conclude that if the penalty parameter $\epsilon$ is set to a proper value, the algorithm runs efficiently, even for a small number of iterations, reducing therefore the computational time.

### 4.2.2 Distributed ADMM - $\ell_1$ Regularization Solution

In this section we want to exploit the solution of the ADMM distributed algorithm for problem (3.18), where we take into consideration the $\ell_1$-norm penalty term. As for the LS solution, we can write the problem in a compact form, introducing another variable, $\boldsymbol{x}$ that considers only the $\ell_1$-norm penalty term and that need to be equal to the $\boldsymbol{y}$, since they represent the same estimated frequencies vector. Therefore the minimization problem becomes

$$\begin{aligned} \underset{\boldsymbol{y},\boldsymbol{z}}{\text{minimize}} \quad & \sum_{i\in\tau} \frac{1}{2} \left(\boldsymbol{r}_i - \boldsymbol{U}_i\boldsymbol{y}_i\right)^2 + \lambda_{\ell_1} \left\|\boldsymbol{x}\right\|_1 \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{z} \\ & \boldsymbol{y} = \boldsymbol{x} \\ & \boldsymbol{z} \in \mathcal{Z} \end{aligned} \tag{4.36}$$

where $\lambda_{\ell_1}$ is the weight of the $\ell_1$-norm penalty term.

Again, to find the solution of (4.36) we search the stationary points of the augmented Lagrangian, which has the form

$$L\left(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{\mu},\boldsymbol{\lambda}\right) = F\left(\boldsymbol{y}\right) + \lambda_{\ell_1}\left\|\boldsymbol{x}\right\|_1 + \left\langle\boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{y} - \boldsymbol{z}\right\rangle + \frac{\epsilon}{2}\left\|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{z}\right\|^2 \tag{4.37}$$

$$+ \left\langle\boldsymbol{\mu}, \boldsymbol{y} - \boldsymbol{x}\right\rangle + \frac{\epsilon}{2}\left\|\boldsymbol{y} - \boldsymbol{x}\right\|^2. \tag{4.38}$$

The derivation of the closed form updates of the variables is computed as follow:

- $\boldsymbol{y}$ update: the update can assume two different forms, depending on whether the node $i$ is sampled or not. For each node $i \in \tau$ we obtain

$$\begin{aligned} \boldsymbol{y}_i^* = \arg\min_{\boldsymbol{y}_i} \quad & \frac{1}{2}\left(\boldsymbol{r}_i - \boldsymbol{U}_i\boldsymbol{y}_i\right)^2 + \sum_{j\in\mathcal{N}_i} \boldsymbol{\lambda}_{i,j}^T\boldsymbol{y}_i + \frac{\epsilon}{2}\sum_{j\in\mathcal{N}_i}\left\|\boldsymbol{y}_i - \boldsymbol{z}_{i,j}\right\|^2 \\ & + \boldsymbol{\mu}_i^T\boldsymbol{y}_i + \frac{\epsilon}{2}\left\|\boldsymbol{y}_i - \boldsymbol{x}_i\right\|^2. \end{aligned} \tag{4.39}$$

  Deriving (4.39) with respect to $\boldsymbol{y}_i$ and imposing the equality with 0:

$$\boldsymbol{U}_i^T\left(\boldsymbol{U}_i\boldsymbol{y}_i - \boldsymbol{r}_i\right) + \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\left(\boldsymbol{y}_i - \boldsymbol{z}_{i,j}\right) + \boldsymbol{\mu}_i + \epsilon\left(\boldsymbol{y}_i - \boldsymbol{x}_i\right) = 0 \tag{4.40}$$

$$\left(\boldsymbol{U}_i^T\boldsymbol{U}_i + \epsilon\left(|\mathcal{N}_i|+1\right)\boldsymbol{I}\right)\boldsymbol{y}_i = \boldsymbol{U}_i^T\boldsymbol{r}_i - \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} + \epsilon\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j} - \boldsymbol{\mu}_i + \epsilon\boldsymbol{x}_i \quad (4.41)$$

$$\begin{aligned}\boldsymbol{y}_i^* &= \left(\epsilon\left(|\mathcal{N}_i|+1\right)\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i - \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j} - \boldsymbol{\mu}_i + \epsilon\left(\sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j} + \boldsymbol{x}_i\right)\right)\\ &= \left(\epsilon\left(|\mathcal{N}_i|+1\right)\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i + \epsilon\left(\boldsymbol{z}_i + \boldsymbol{x}_i - \tilde{\boldsymbol{\lambda}}_i - \tilde{\boldsymbol{\mu}}_i\right)\right) \quad (4.42)\end{aligned}$$

where $\boldsymbol{z}_i = \sum_{j\in\mathcal{N}_i}\boldsymbol{z}_{i,j}$, $\boldsymbol{\lambda}_i = \sum_{j\in\mathcal{N}_i}\boldsymbol{\lambda}_{i,j}$, $\tilde{\boldsymbol{\lambda}}_i = \frac{\boldsymbol{\lambda}_i}{\epsilon}$ and $\tilde{\boldsymbol{\mu}}_i = \frac{\boldsymbol{\mu}_i}{\epsilon}$.

Instead for $i \in \bar{\tau}$:

$$\boldsymbol{y}_i^* = \frac{1}{\left(|\mathcal{N}_i|+1\right)}\left(\boldsymbol{z}_i + \boldsymbol{x}_i - \tilde{\boldsymbol{\lambda}}_i - \tilde{\boldsymbol{\mu}}_i\right) \quad (4.43)$$

- $\boldsymbol{z}$ update: does not change, since it does not depend on $\boldsymbol{x}$

$$\boldsymbol{z}_i^* = \frac{1}{2}\left(\boldsymbol{y}_i^*|\mathcal{N}_i| + \sum_{j\in\mathcal{N}_i}\boldsymbol{y}_j^*\right) \quad (4.44)$$

- $\boldsymbol{x}$ update: we need to take care of the $\ell_1$-norm penalty parameter $\lambda_{\ell_1}$. Since it affects all the variables $\boldsymbol{x}$, for each frequency vector $\boldsymbol{x}_i$ the multiplicative constant becomes $\lambda_{\ell_1}/N$, because we need to take into account the contribution for each node by separating the penalty term. Therefore the update rule for each $\boldsymbol{x}_i$ is derived as follow:

$$\begin{aligned}\boldsymbol{x}_i &= \arg\min_{\boldsymbol{x}_i}\frac{\lambda_{\ell_1}}{N}\left\|\boldsymbol{x}_i\right\|_1 - \boldsymbol{\mu}_i^T\boldsymbol{x}_i + \frac{\epsilon}{2}\left\|\boldsymbol{y}_i - \boldsymbol{x}_i\right\|^2\\ &= \arg\min_{\boldsymbol{x}_i}\frac{\lambda_{\ell_1}}{N}\left\|\boldsymbol{x}_i\right\|_1 + \frac{\epsilon}{2}\left\|\boldsymbol{x}_i\right\|^2 - \epsilon\boldsymbol{x}_i^T\left(\boldsymbol{y}_i^* + \frac{\boldsymbol{\mu}_i}{\epsilon}\right)\\ &= \arg\min_{\boldsymbol{x}_i}\frac{\lambda_{\ell_1}}{\epsilon N}\left\|\boldsymbol{x}_i\right\|_1 + \frac{1}{2}\left\|\boldsymbol{x}_i - (\boldsymbol{y}_i^* + \tilde{\boldsymbol{\mu}}_i)\right\|^2\\ &= \arg\min_{\boldsymbol{x}_i}\frac{1}{2}\left\|\boldsymbol{x}_i - \boldsymbol{m}_i\right\|^2 + \frac{\lambda_{\ell_1}}{\epsilon N}\left\|\boldsymbol{x}_i\right\|_1 \quad (4.45)\end{aligned}$$

assuming $\boldsymbol{m}_i = \boldsymbol{y}_i^* + \tilde{\boldsymbol{\mu}}_i$ is a local built vector. The solution of (4.45) is

$$\boldsymbol{x}_i^* = \text{sign}\left(\boldsymbol{m}_i\right)\left[|\boldsymbol{m}_i| - \frac{\lambda_{\ell_1}}{\epsilon N}\right]^+ \tag{4.46}$$

where the plus function is defined as

$$[x]^+ = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.47}$$

- $\boldsymbol{\lambda}$ update: the Lagrangian multiplier is updated as previous

$$\tilde{\boldsymbol{\lambda}}_i^* = \tilde{\boldsymbol{\lambda}}_i + \boldsymbol{y}_i^*|\mathcal{N}_i| - \boldsymbol{z}_i^* \tag{4.48}$$

- $\boldsymbol{\mu}$ update: the values of the other Lagrangian multiplier is updated as

$$\tilde{\boldsymbol{\mu}}_i^* = \tilde{\boldsymbol{\mu}}_i + \boldsymbol{y}_i^* - \boldsymbol{x}_i^* \tag{4.49}$$

Now we are able to express all the ADMM variables update for the $\ell_1$-norm regularization problem in a compact form:

$$\begin{aligned}
\boldsymbol{y}_i^* &= \begin{cases} \left(\epsilon\left(|\mathcal{N}_i|+1\right)\boldsymbol{I} + \boldsymbol{U}_i^T\boldsymbol{U}_i\right)^{-1}\left(\boldsymbol{U}_i^T\boldsymbol{r}_i + \epsilon\left(\boldsymbol{z}_i + \boldsymbol{x}_i - \tilde{\boldsymbol{\lambda}}_i - \tilde{\boldsymbol{\mu}}_i\right)\right) & i \in \tau \\ \frac{1}{(|\mathcal{N}_i|+1)}\left(\boldsymbol{z}_i + \boldsymbol{x}_i - \tilde{\boldsymbol{\lambda}}_i - \tilde{\boldsymbol{\mu}}_i\right) & i \in \bar{\tau} \end{cases} \\
\boldsymbol{z}_i^* &= \frac{1}{2}\left(\boldsymbol{y}_i^*|\mathcal{N}_i| + \sum_{j\in\mathcal{N}_i}\boldsymbol{y}_j^*\right) \\
\boldsymbol{x}_i^* &= \text{sign}\left(\boldsymbol{m}_i\right)\left[|\boldsymbol{m}_i| - \frac{\lambda_{\ell_1}}{\epsilon N}\right]^+, \quad \boldsymbol{m}_i = \boldsymbol{y}_i^* + \tilde{\boldsymbol{\mu}}_i \\
\tilde{\boldsymbol{\lambda}}_i^* &= \tilde{\boldsymbol{\lambda}}_i + \boldsymbol{y}_i^*|\mathcal{N}_i| - \boldsymbol{z}_i^* \\
\tilde{\boldsymbol{\mu}}_i^* &= \tilde{\boldsymbol{\mu}}_i + \boldsymbol{y}_i^* - \boldsymbol{x}_i^*.
\end{aligned} \tag{4.50}$$

As in the previous problem formulation, we can write the inverse matrix of the $\boldsymbol{y}$ update in a closed form, where in this case $\alpha = \epsilon\left(|\mathcal{N}_i|+1\right)$.

Now we want to show the numerical behaviour of the convergence of the ADMM algorithm for the $\ell_1$-norm regularization problem. As stated in section 4.2.1, the

penalty parameter $\epsilon$ affects the slope convergence, for both the MSE saturation and for the similarity between local variable $\boldsymbol{y}$ and $\boldsymbol{z}$ and $\boldsymbol{y}$ and $\boldsymbol{x}$. In Fig. 4.5 we can see the behaviour of the MSE for each iteration of the algorithm, for different values of $\epsilon$. We imposed Metropolis weights and we fixed the number of sampled nodes to 50 and the number of estimated frequencies to 10, while the $\ell_1$-norm weight $\lambda_\ell$ is set to 1.

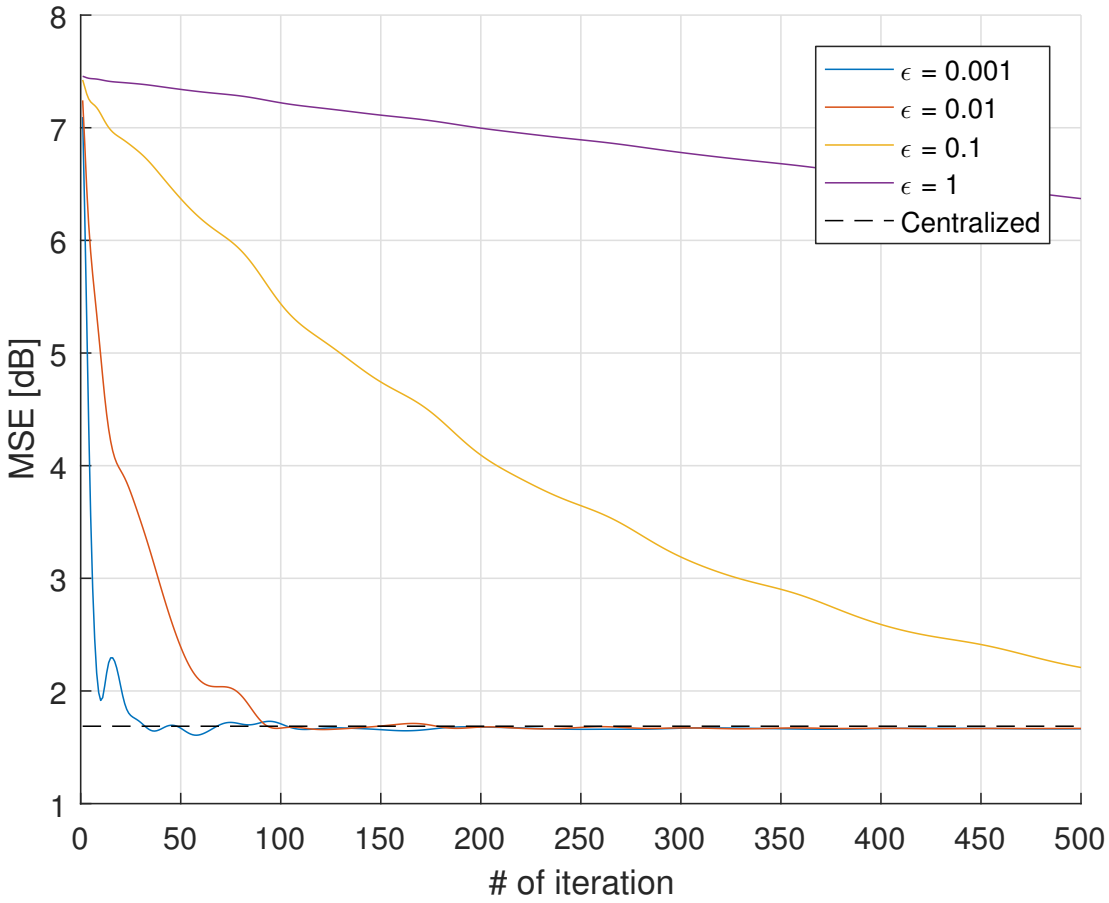



**Figure 4.5:** Convergence of the ADMM solution - $\ell_1$-norm regularization (10 freq.)

We can see that the trend of the curves is comparable to the one of the LS solution and that the penalty parameter $\epsilon$ must be set to a proper value.

What is interesting to observe is the behaviour of the convergence when the number of frequencies is greater than the number of sampled nodes, for which the LS solution error diverges as discussed in section 3.4: we show in Fig. 4.6 the evolution of the mean square error when the number of frequencies is 99. The

parameter $\lambda_\ell$ is again set to 1.

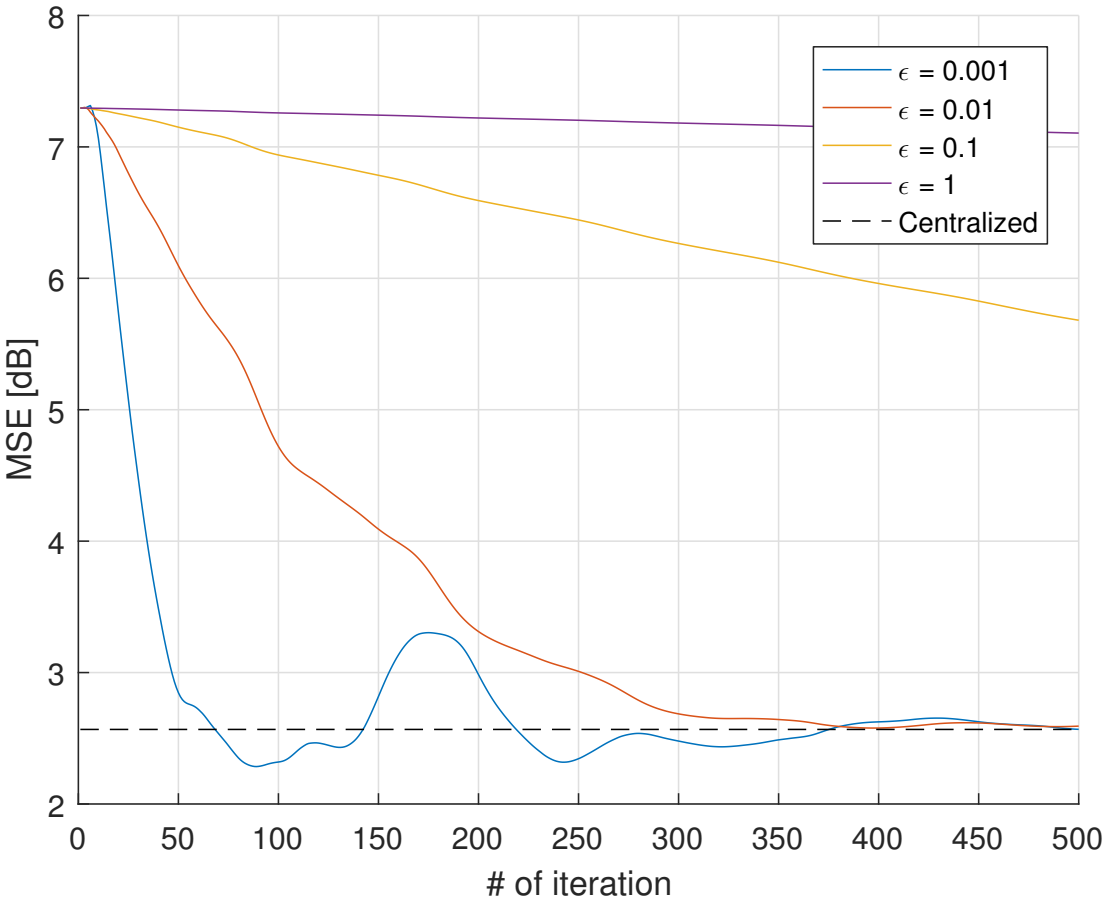



**Figure 4.6:** Convergence of the ADMM solution - $\ell_1$-norm regularization (99 freq.)

We can observe that the convergence is slightly slower with respect to the previous favorable case, but the result is significantly better adding the regularization parameter, which impose the solution vector to be sparse depending on the value of $\lambda_\ell$.

In order to compare the performance of the two algorithms, we report in Fig. 4.7 a comparison between the convergence of the ADMM solution for both problems, when the estimated frequencies are 10: the LS behaviour is represented by dashed lines, while solid lines depict $\ell_1$-norm trend. The results are comparable.

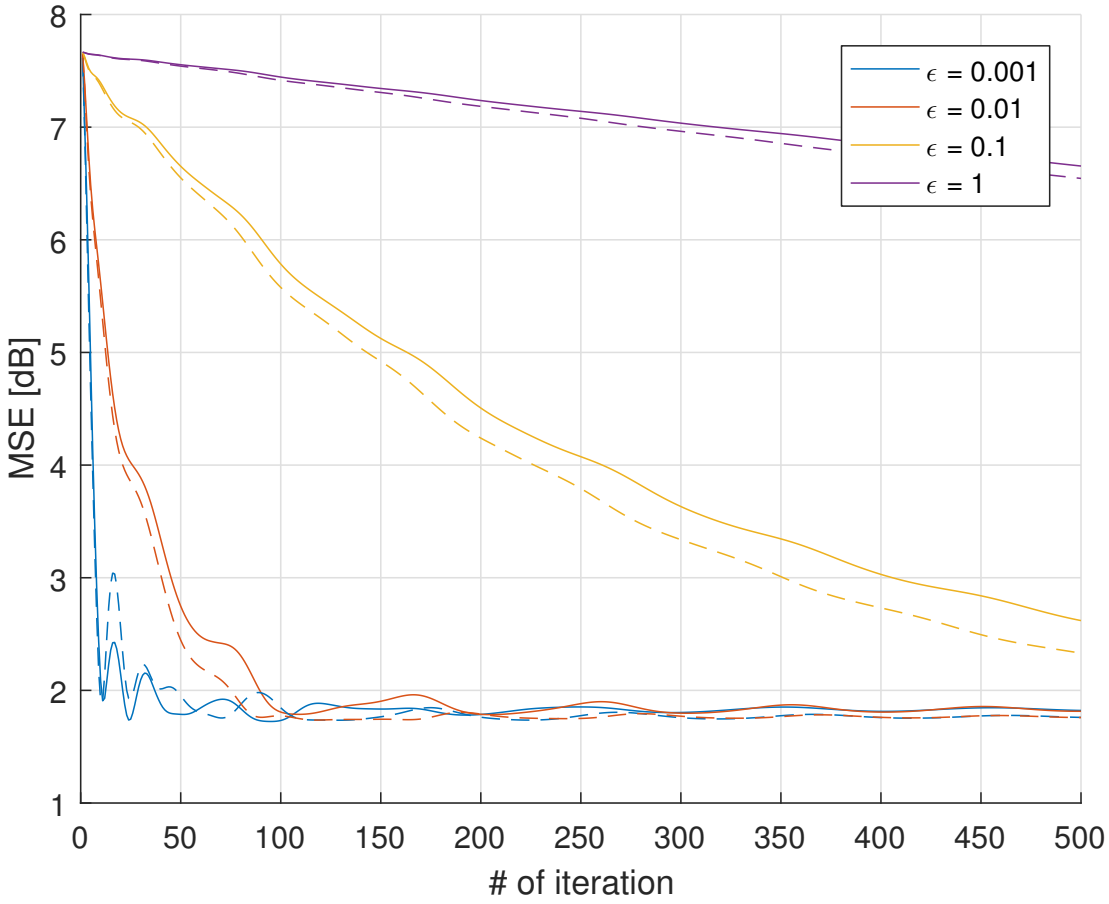



**Figure 4.7:** Convergence of the ADMM solution - comparison

For the sake of completeness, in Fig. 4.8 we show the average time spent to run a simulation of this version of the ADMM algorithm: we can see that it is similar to the previous one reported in Fig. 4.4, since in addition now we have to update two local variables.

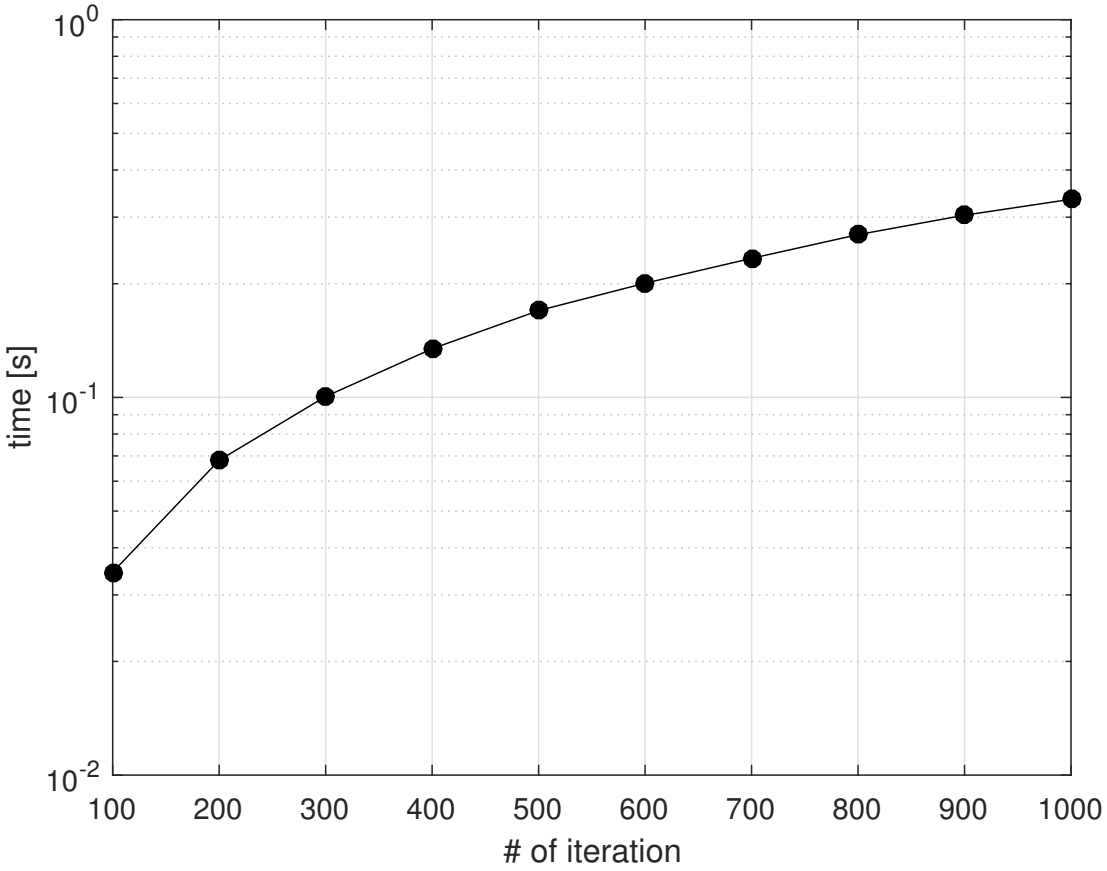



**Figure 4.8:** Average time to run ADMM - $\ell_1$-norm regularization

Now, our aim is to compare the proposed distributed algorithms with the one illustrated in [45] by Barbarossa, Di Lorenzo, Banelli and Sardellitti. Their solution proceed by minimizing the Lagrangian function of the LS problem by means of a steepest descent procedure. The adaptive implementation is termed Adapt To Combine (ATC) diffusion strategy, and is based on two steps: first, in the adaptation step, the intermediate estimate is updated adopting the observation at each node; then, in the diffusion step, where intermediate estimates are combined through neighboring nodes. Finally, given the estimated signal in the graph spectral domain, the graph signal can be computed locally by taking the IGFT.

We show in Fig. 4.9 a comparison between average consensus, ADMM for LS problem, ADMM for LASSO problem and the ATC solutions. We can observe that, as previous in Fig. 4.7, LS and LASSO behaves very similar, while the average consensus algorithm converges slower than the ADMM solutions. Our

implementation of the steepest descent procedure proposed by Barbarossa et al. is outperformed by our distributed algorithms implementations, but obviously it converges. We set the value of the $\ell_1$-norm penalty term to $\lambda_{\ell_1} = 1$ and the value of the Lagrangian penalty parameter to $\epsilon = 0.01$.

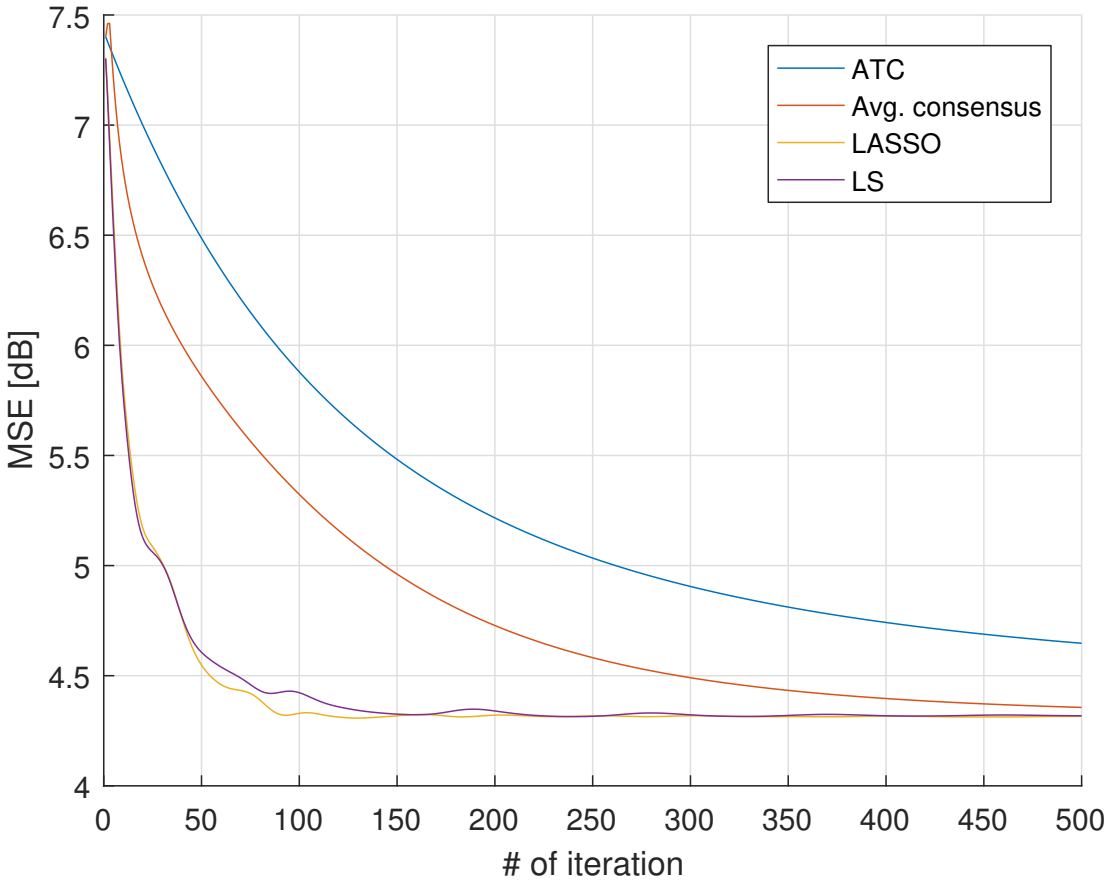



**Figure 4.9:** Comparison of distributed algorithms - 1

Finally, we introduce an update rule for the penalty parameter $\epsilon$: at each iteration of the ADMM algorithm, we can update (increase) the weighting parameter as $\epsilon^* = \delta\epsilon$, with $\delta > 1$, in order to improve convergence speed and stability, as suggested by several works [76], [77], [78]. A good idea is to decide to update the local penalty values at each iteration: since the problem is convex, the solution is moving towards the optimum and so we can increase the values of $\epsilon$, since higher penalty parameter implies slower shift from current point. Therefore we set an initial small value of penalty parameter and we increase it by a constant factor at

each iteration, in order to improve convergence speed.

We implemented this new version of the ADMM algorithm exploiting the update rules as function of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, instead of $\tilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}/\epsilon$ and $\tilde{\boldsymbol{\mu}} = \boldsymbol{\mu}/\epsilon$, because we are updating the penalty parameter but not the Lagrangian multiplier: we need to take into account of their values instead of their fraction with respect to $\epsilon$, otherwise the solution will not converge.

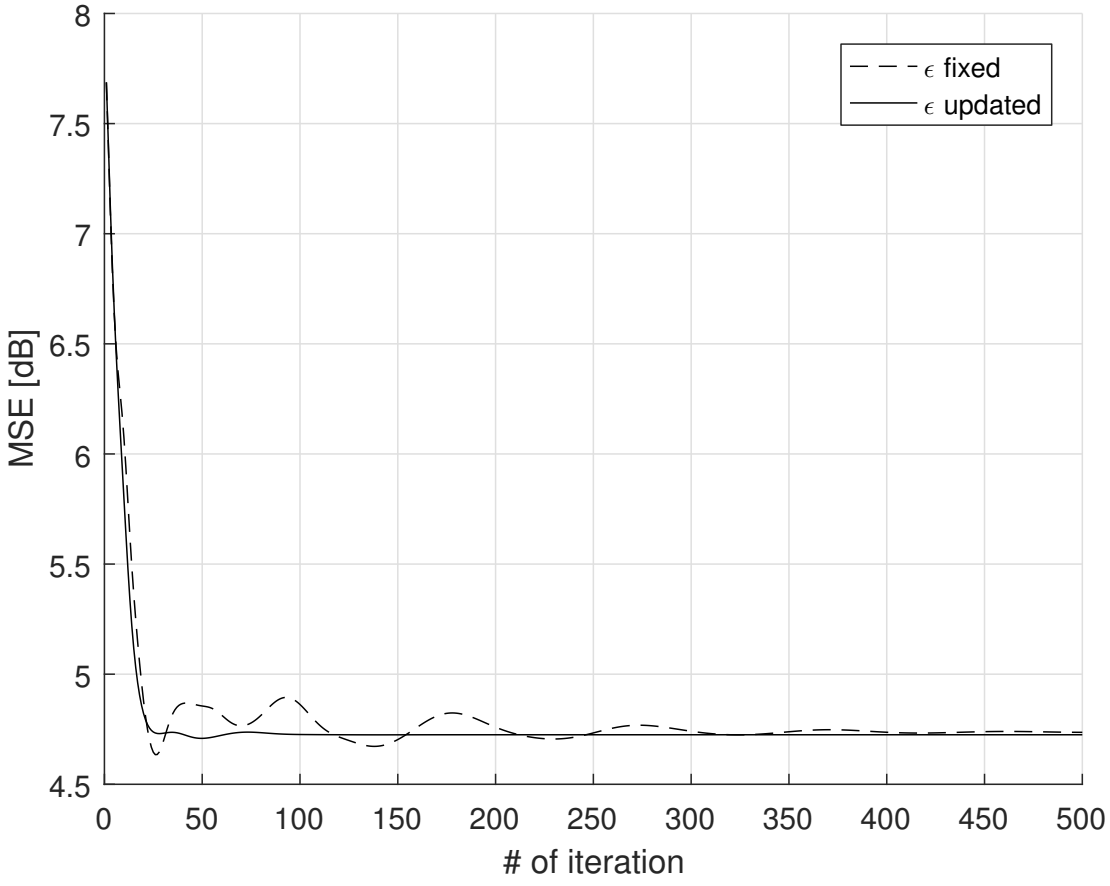



**Figure 4.10:** Convergence of ADMM, LS solution, updated penalty parameter ($\epsilon_0 = 0.001$)

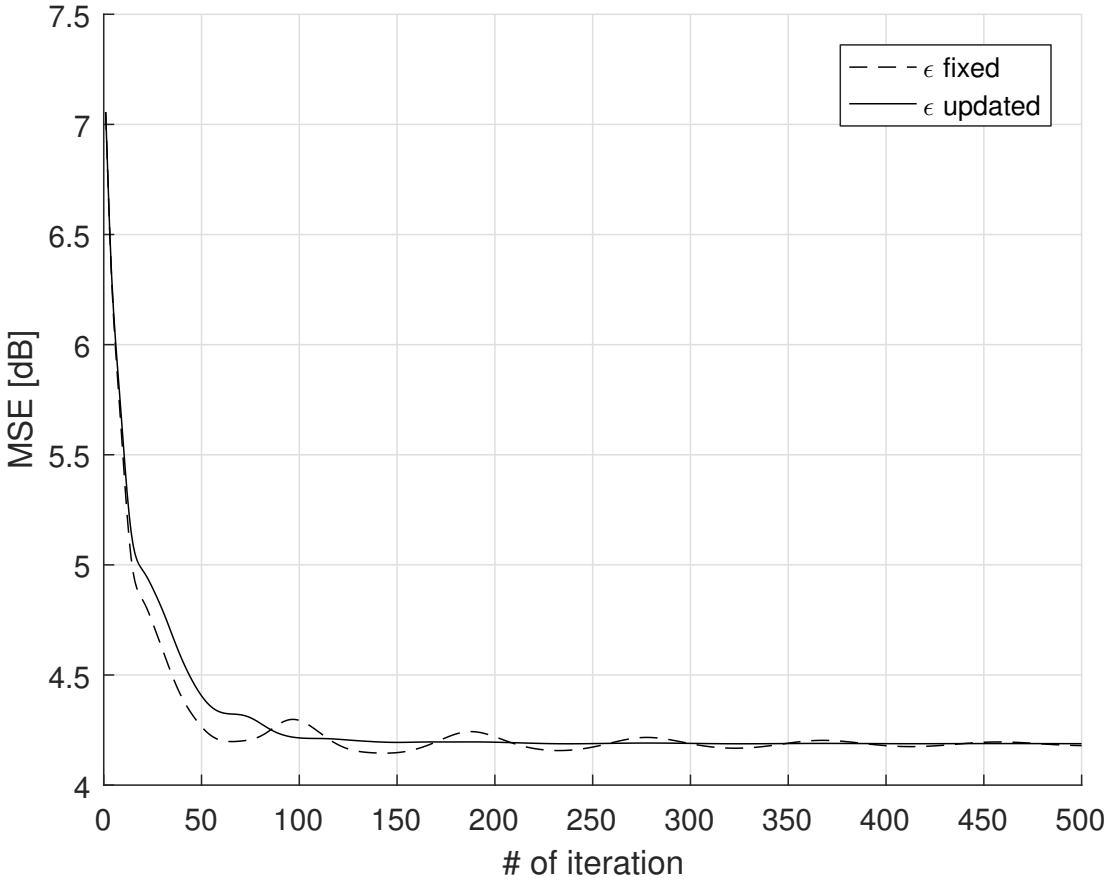



**Figure 4.11:** Convergence of ADMM, LS solution, updated penalty parameter ($\epsilon_0 = 0.01$)

We report in Fig. 4.10 and in Fig. 4.11 a comparison of the behaviour of the MSEs between static and dynamic penalty parameters, for LS problem and for two different initial values of $\epsilon_0 = 0.001$ and $\epsilon_0 = 0.01$, respectively. We can see, especially in Fig. 4.10 where the initial value of $\epsilon$ is very small, ensuring an initial fast convergence, that updating the penalty parameter ensures a much stable and quicker convergence for the MSE, with respect to the one of the static version.

In Fig. 4.12 we show the behaviour of the MSE for the $\ell_1$-norm regularization problem, for both fixed and updated penalty parameter, starting from an initial value of $\epsilon_0 = 0.001$. As well as for the LS minimization, there are no fluctuations in the $\epsilon$ updated version. In Fig. 4.13 it is shown the same behaviour considering $\epsilon_0 = 0.01$. The results are the same as in the previous cases.

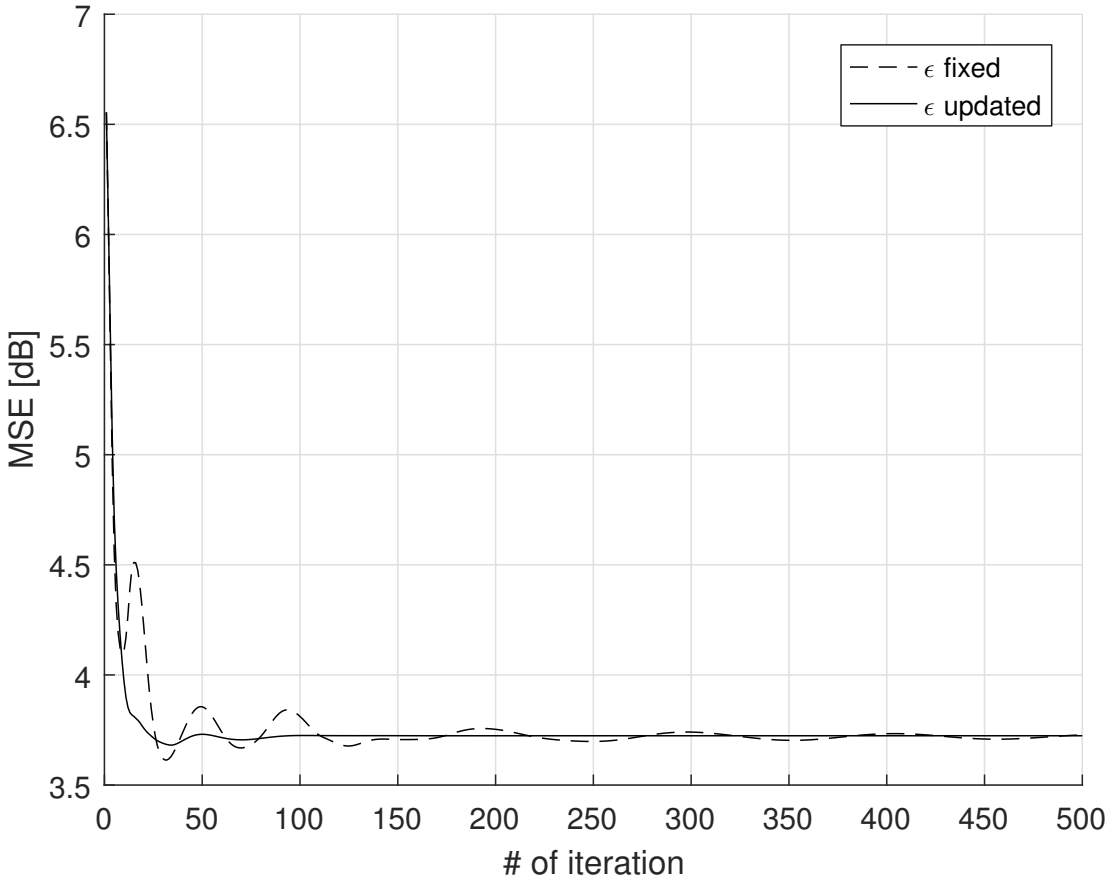



**Figure 4.12:** Convergence of ADMM, LASSO solution, updated penalty parameter ($\epsilon_0 = 0.001$)

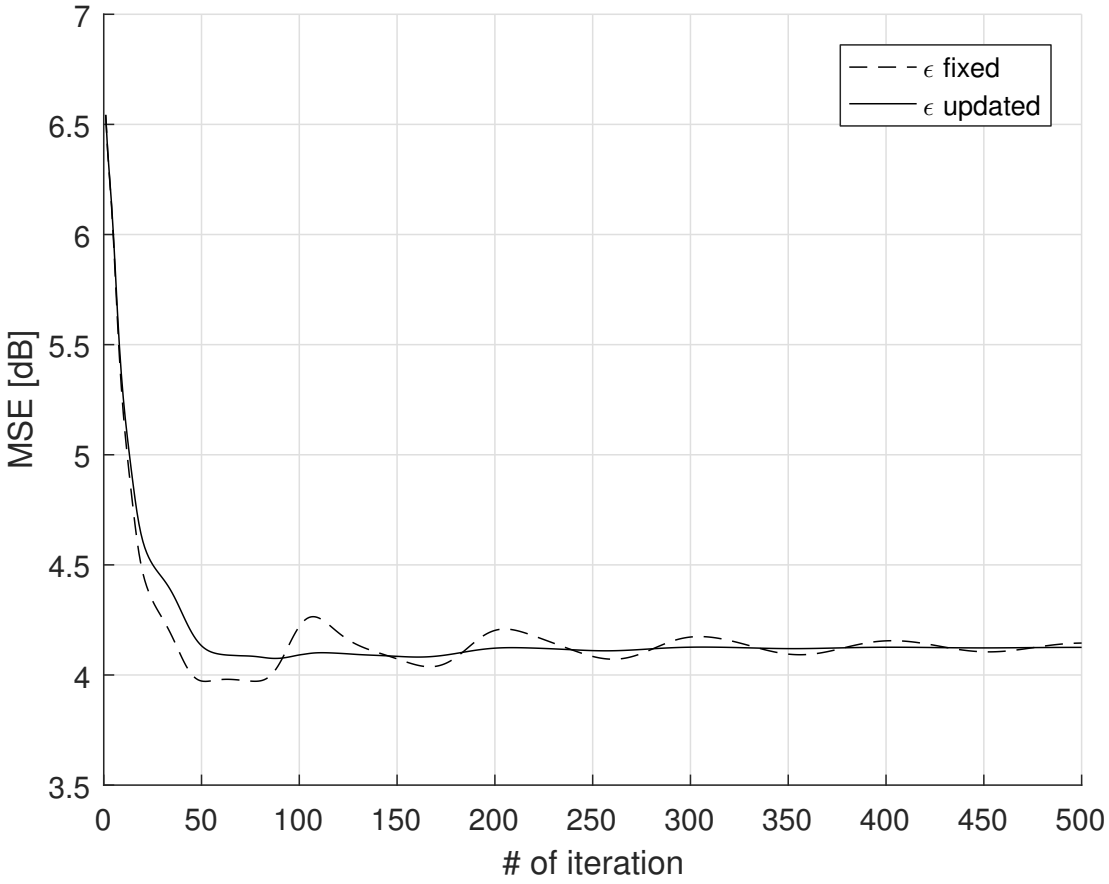



**Figure 4.13:** Convergence of ADMM, LASSO solution, updated penalty parameter ($\epsilon_0 = 0.01$)

We show in Fig. 4.14 a comparison of the best performance of the distributed algorithms: we consider ADMM algorithm applied to LS and LASSO problems, average consensus and ATC approach. The number of sampled nodes is 50 and the number of active frequencies is 10. The other parameters are set to $\lambda_{\ell_1} = 1$, $\epsilon_0 = 0.001$.

We can see that the convergence of LS and LASSO are approximately equal and reaches the minimum after few iterations, while consensus and the solution proposed in [45], confirming that ADMM algorithm is the best solution.

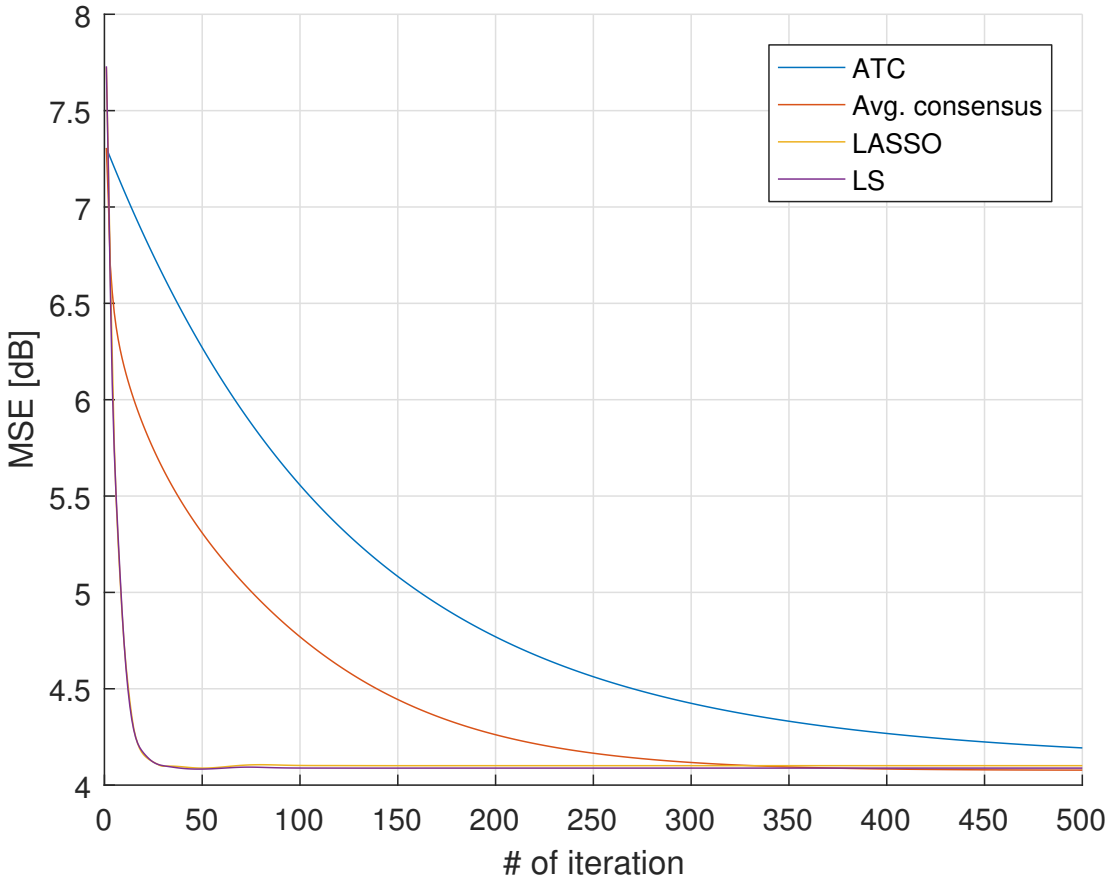



**Figure 4.14:** Comparison of distributed algorithms - 2

# 5

# Conclusions And Future Work

This thesis introduced graph signal processing framework as a developing tool which has grown in recent years. We addressed several problems related to the representation and processing of structured signals defined on weighted and undirected graphs. In particular, we introduced some concepts and properties related to the graph-based transforms, which are commonly known in the graph signal processing framework. Next, we addressed the problem of reconstruction of a graph signal by using some well known tools, starting from an estimation of the signal in the graph spectral domain. Finally, we proposed a distributed processing algorithm for reconstructing graph signals, focusing on the convergence of the solution with respect to the centralized one.

After having illustrated the main common assumptions and definition in the graph signal processing field, we focused on a specific problem which corresponds to reconstruct a graph signal, starting from an estimation in the spectral domain, by observing only a subset of values of the original signal. We explained and justified some assumptions for the choice of the samples and the frequency index ordering, as we proved necessary and sufficient conditions for reconstructing graph signals. Besides, the signal estimation was derived, comparing different weighted adjacency matrices and two different estimation models: the LS and the LASSO. We found that both solution behaves similar in relaxed setting, instead when we have weak reconstruction condition the LASSO performs better.

Then we focused on the distributed version of the algorithms and on their convergence to the centralized solution: first we introduced average consensus, which is a simple form of distributed processing, and then the ADMM algorithm were explained. Starting from the common formulation of the problem, we have derived simple variables update rules in a closed form, which enable the network to derive an estimation of the spectral signal in a distributed way. Simulations results proved the convergence of the algorithm.

To summarize, we have studied in this thesis several important problems related to the emerging field of signal processing on graphs. We have provided solutions for processing and analyzing graph signals in both centralized and distributed settings. We believe that the contribution of this thesis can be useful for understanding the interplay between signals and graphs.

## 5.1 Future Work

Signal processing on graphs is a relatively new research field that is still in its infancy. Parts of this field are old as there exists a lot of research mainly in the machine learning and the computer science community on analyzing and understanding the graph structure. However, the concept of a signal on a graph is new and very interesting from a signal processing perspective. While this thesis brings contributions in the theory of distributed graph signal reconstruction and its applications, it provides answers to only some of the open questions that are related to the interdependence between the graph structure and the signal on the graph proper data analysis. There are therefore many more exciting directions that graph signal processing research can pursue.

Graphs are powerful and promising discrete tool for analyzing complex high-dimensional data sets. However, in order to fully exploit their power, we should further learn how to use them properly. The challenges are many: we need to understand the theoretical and empirical role of the graph structure, and define meaningful criteria for constructing the graph; also, understanding the role of the graph structure in graph signals is a necessary step for designing more efficient graph-based signal processing algorithms that can be used for analysis and inference tasks on complex high-dimensional data sets; finally, an important parameter that should be considered is the computational complexity of the graph-based

algorithms: these should be designed in a scalable manner in order to handle large-size data.

# References


[1] 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.

[2] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, vol. 92.

[3] F. R. Gantmacher, *Theory of Matrices. 2V.* Chelsea publishing company, 1960.

[4] T. Bıyıkoglu, J. Leydold, and P. F. Stadler, "Laplacian eigenvectors of graphs. perron-frobenius and faber-krahn type theorems, volume 1915 of lecture notes in mathematics," 2007.

[5] F. Chung, "Laplacians and the cheeger inequality for directed graphs," *Annals of Combinatorics*, vol. 9, no. 1, pp. 1–19, 2005.

[6] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[7] X. Zhu and M. Rabbat, "Approximating signals supported on graphs." in *ICASSP*. Citeseer, 2012, pp. 3921–3924.

[8] 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.

[9] ——, "A windowed graph fourier transform," in *2012 IEEE Statistical Signal Processing Workshop (SSP)*. Ieee, 2012, pp. 133–136.

[10] L. J. Grady and J. R. Polimeni, "Discrete calculus: History and future," in *Discrete Calculus*. Springer, 2010, pp. 1–9.

[11] N. J. Higham, *Functions of matrices: theory and computation*. Siam, 2008.

[12] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[13] V. Blondel, J. M. Hendrickx, A. Olshevsky, J. Tsitsiklis *et al.*, "Convergence in multiagent coordination, consensus, and flocking," in *IEEE Conference on Decision and Control*, vol. 44, no. 3. IEEE; 1998, 2005, p. 2996.

[14] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[15] E. Kokiopoulou and P. Frossard, "Distributed classification of multiple observation sets by consensus," *IEEE Transactions on Signal Processing*, vol. 59, no. 1, pp. 104–114, 2011.

[16] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Distributed consensus algorithms for svm training in wireless sensor networks," in *Signal Processing Conference, 2008 16th European*. IEEE, 2008, pp. 1–5.

[17] A. Schmidt and J. M. Moura, "A distributed sensor fusion algorithm for the inversion of sparse fields," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*. IEEE, 2009, pp. 1332–1336.

[18] L. Li, A. Scaglione, and J. H. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 725–738, 2011.

[19] D. Ustebay, R. Castro, and M. Rabbat, "Efficient decentralized approximation via selective gossip," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 805–816, 2011.

[20] D. I. Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. IEEE, 2011, pp. 1–8.

[21] ——, "Distributed signal processing via chebyshev polynomial approximation," *arXiv preprint arXiv:1111.5239*, 2011.

[22] X. Wang, M. Wang, and Y. Gu, "A distributed tracking algorithm for reconstruction of graph signals," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 728–740, 2015.

[23] S. Chen, A. Sandryhaila, and J. Kovačević, "Distributed algorithm for graph signal inpainting," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 3731–3735.

[24] E. Kokiopoulou and P. Frossard, "Polynomial filtering for fast convergence in distributed consensus," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 342–354, 2009.

[25] A. Sandryhaila, S. Kar, and J. M. Moura, "Finite-time distributed consensus through graph filters," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 1080–1084.

[26] S. Segarra, A. G. Marques, and A. Ribeiro, "Distributed linear network operators using graph filters," *arXiv preprint arXiv:1510.03947*, 2015.

[27] M. Hein, J.-Y. Audibert, and U. Von Luxburg, "From graphs to manifolds–weak and strong pointwise consistency of graph laplacians," in *International Conference on Computational Learning Theory*. Springer, 2005, pp. 470–485.

[28] A. Singer, "From graph to manifold laplacian: The convergence rate," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 128–134, 2006.

[29] M. Belkin and P. Niyogi, "Towards a theoretical foundation for laplacian-based manifold methods," in *International Conference on Computational Learning Theory*. Springer, 2005, pp. 486–500.

[30] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Computer graphics forum*, vol. 28, no. 5. Wiley Online Library, 2009, pp. 1383–1392.

[31] M. M. Bronstein and I. Kokkinos, "Scale-invariant heat kernel signatures for non-rigid shape recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1704–1711.

[32] M. Aubry, U. Schlickewei, and D. Cremers, "The wave kernel signature: A quantum mechanical approach to shape analysis," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1626–1633.

[33] R. Litman and A. M. Bronstein, "Learning spectral descriptors for deformable shape correspondence," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 1, pp. 171–180, 2014.

[34] N. Hu, R. M. Rustamov, and L. Guibas, "Stable and informative spectral signatures for graph matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2305–2312.

[35] W. Hwa Kim, M. K. Chung, and V. Singh, "Multi-resolution shape analysis via non-euclidean wavelets: Applications to mesh segmentation and surface alignment problems," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2139–2146.

[36] W.-S. Kim, S. K. Narang, and A. Ortega, "Graph based transforms for depth video coding," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 813–816.

[37] W. Hu, G. Cheung, A. Ortega, and O. C. Au, "Multiresolution graph fourier transform for compression of piecewise smooth images," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 419–433, 2015.

[38] G. Shen, W.-S. Kim, S. K. Narang, A. Ortega, J. Lee, and H. Wey, "Edge-adaptive transforms for efficient depth map coding," in *Picture Coding Symposium (PCS), 2010*. IEEE, 2010, pp. 566–569.

[39] C. Zhang and D. Florêncio, "Analyzing the optimality of predictive transform coding using graph-based models," *IEEE Signal Processing Letters*, vol. 20, no. 1, pp. 106–109, 2013.

[40] T. Maugey, A. Ortega, and P. Frossard, "Graph-based representation for multiview image geometry," *IEEE Transactions on Image Processing*, vol. 24, no. 5, pp. 1573–1586, 2015.

[41] T. Maugey, Y.-H. Chao, A. Gadde, A. Ortega, and P. Frossard, "Luminance coding in graph-based representation of multiview images," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 130–134.

[42] D. Tian, H. Mansour, A. Knyazev, and A. Vetro, "Chebyshev and conjugate gradient filters for graph image denoising," in *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1–6.

[43] P. Wan, G. Cheung, D. Florencio, C. Zhang, and O. C. Au, "Image bit-depth enhancement via maximum-a-posteriori estimation of graph ac component," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 4052–4056.

[44] H. Q. Nguyen, P. A. Chou, and Y. Chen, "Compression of human body sequences using graph wavelet filter banks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6152–6156.

[45] P. Di Lorenzo, P. Banelli, S. Barbarossa, and S. Sardellitti, "Distributed adaptive learning of graph signals," *arXiv preprint arXiv:1609.06100*, 2016.

[46] a. e. f. Ministero delle politiche agricole. (2017) Statistiche meteoclimatiche. [Online]. Available: https://www.politicheagricole.it/flex/FixedPages/Common/miepfy700_riferimentiAgro.php/L/IT

[47] M. Tsitsvero, S. Barbarossa, and P. Di Lorenzo, "Signals on graphs: Uncertainty principle and sampling," 2015.

[48] P. Di Lorenzo, S. Barbarossa, P. Banelli, and S. Sardellitti, "Adaptive least mean squares estimation of graph signals," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 4, pp. 555–568, 2016.

[49] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 33–46, 2007.

[50] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[51] L. Breiman, "Better subset regression using the nonnegative garrote," *Technometrics*, vol. 37, no. 4, pp. 373–384, 1995.

[52] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[53] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics Springer, Berlin, 2001, vol. 1.

[54] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical learning with sparsity: the lasso and generalizations*. CRC Press, 2015.

[55] P. Bühlmann and S. Van De Geer, *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.

[56] J. Friedman, T. Hastie, H. Höfling, R. Tibshirani *et al.*, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 302–332, 2007.

[57] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1996.

[58] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[59] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *Journal of parallel and distributed computing*, vol. 7, no. 2, pp. 279–301, 1989.

[60] G. Scutari, S. Barbarossa, and L. Pescosolido, "Distributed decision through self-synchronizing sensor networks in the presence of propagation delays and asymmetric channels," *IEEE Transactions on Signal Processing*, vol. 56, no. 4, pp. 1667–1684, 2008.

[61] M. Cao, A. S. Morse, and B. D. Anderson, "Agreeing asynchronously," *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1826–1838, 2008.

[62] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[63] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on automatic control*, vol. 50, no. 2, pp. 169–182, 2005.

[64] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic control*, vol. 52, no. 5, pp. 863–868, 2007.

[65] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 33–55, 2009.

[66] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[67] U. A. Khan, S. Kar, and J. M. Moura, "Distributed sensor localization in random environments using minimal number of anchor nodes," *IEEE Transactions on Signal Processing*, vol. 57, no. 5, pp. 2000–2016, 2009.

[68] J. N. Tsitsiklis *et al.*, "Decentralized detection," *Advances in Statistical Signal Processing*, vol. 2, no. 2, pp. 297–344, 1993.

[69] Z.-Q. Luo, "An isotropic universal decentralized estimation scheme for a bandwidth constrained ad hoc sensor network," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 735–744, 2005.

[70] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[71] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires," *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, vol. 9, no. 2, pp. 41–76, 1975.

[72] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

[73] T. Erseghe, D. Zennaro, E. Dall'Anese, and L. Vangelista, "Fast consensus by the alternating direction multipliers method," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5523–5537, 2011.

[74] H. Zhu, G. B. Giannakis, and A. Cano, "Distributed in-network channel decoding," *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 3970–3983, 2009.

[75] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[76] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[77] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt, "On augmented lagrangian methods with general lower-level constraints," *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1286–1309, 2007.

[78] E. G. Birgin and J. M. Martínez, *Practical augmented Lagrangian methods for constrained optimization.* SIAM, 2014.