



A DIDACTIC INTRODUCTION TO GRAPH SIGNAL PROCESSING TECHNIQUES
AND APPLICATIONS

Pedro Angelo Medeiros Fonini

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

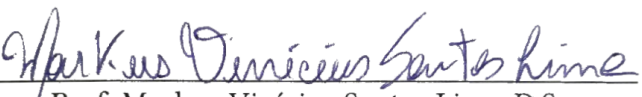
Orientadores: Paulo Sergio Ramirez Diniz
Markus Vinícius Santos Lima

Rio de Janeiro
Março de 2019

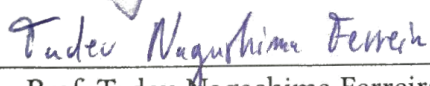
A DIDACTIC INTRODUCTION TO GRAPH SIGNAL PROCESSING TECHNIQUES
AND APPLICATIONS

Pedro Angelo Medeiros Fonini

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ
COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM
ENGENHARIA ELÉTRICA.


Prof. Markus Vinícius Santos Lima, D.Sc.


Prof. Marcelo Luiz Rodrigues de Campos, Ph.D.


Prof. Tadeu Nagashima Ferreira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2019

Fonini, Pedro Angelo Medeiros

A didactic introduction to Graph Signal Processing techniques and applications/Pedro Angelo Medeiros Fonini.

– Rio de Janeiro: UFRJ/COPPE, 2019.

XII, 50 p.: il.; 29, 7cm.

Orientadores: Paulo Sergio Ramirez Diniz

Markus Vinícius Santos Lima

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 48 – 50.

1. graphs. 2. graph signal processing. 3. adaptive filtering. I. Diniz, Paulo Sergio Ramirez *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Qualquer agradecimento que eu escreva aqui será pouco face à importância que teve para este trabalho o suporte e a paciência do Markus e do Diniz. Se eu aprendi nem que seja 10% do que eles tentaram me ensinar, o mérito é deles, e o lucro é meu.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA INTRODUÇÃO DIDÁTICA ÀS TÉCNICAS E APLICAÇÕES DO
PROCESSAMENTO DE SINAIS EM GRAFOS

Pedro Angelo Medeiros Fonini

Março/2019

Orientadores: Paulo Sergio Ramirez Diniz
Markus Vinícius Santos Lima

Programa: Engenharia Elétrica

Processamento de sinais é uma área tradicionalmente bem-sucedida em diversas aplicações em que o domínio de interesse é uniforme. Embora a maioria das ferramentas desta área sejam baseadas no fato de que é esperado que tais domínios sejam uniformes—por exemplo o contínuo de instantes de tempo ou um domínio discreto com amostras igualmente espaçadas—, muitos problemas interessantes são colocados sobre estruturas mais irregulares. Ao lidar com aplicações como redes sociais, *arrays* de sensores arbitrariamente distribuídos, redes neuronais ou redes de distribuição de energia, uma das estratégias matemáticas para respeitar as irregularidades nessas estruturas é interpretar os dados como sendo definidos em vértices de um grafo ponderado.

Uma vez que tais estruturas subjacentes costumam carregar informações valiosas o campo do processamento de sinais em grafos (GSP) tem estado ativo na última década. Neste trabalho, revisamos alguns dos avanços obtidos por este novo campo de pesquisa. Damos foco especial para sinais variantes no tempo—nos quais cada vértice do grafo está associado a uma série temporal—e para os algoritmos adaptativos projetados para esta nova forma de processamento de sinais.

Quando técnicas de filtragem adaptativa são aplicadas ao GSP, é possível projetar algoritmos que aprendem a partir das estatísticas de um sinal em um grafo. Ao levar em consideração a estrutura subjacente do domínio irregular no qual os dados residem, os resultados são um maior poder de inferência, e decisões melhor-informadas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A DIDACTIC INTRODUCTION TO GRAPH SIGNAL PROCESSING TECHNIQUES
AND APPLICATIONS

Pedro Angelo Medeiros Fonini

March/2019

Advisors: Paulo Sergio Ramirez Diniz
Markus Vinícius Santos Lima

Department: Electrical Engineering

Traditional signal processing thrives when applied to uniform, euclidean domains. Even though most DSP tools are built around the fact that signal domains are expected to be uniform—e.g. continuous time or an equally spaced discrete domain—, many interesting problems are defined on top of more irregular structures. When dealing with applications such as social networks, arbitrarily distributed arrays of sensors, neuronal networks, and power grids, one of the mathematical strategies to acknowledge the irregular structure is to interpret the data as being defined on the vertices of a weighted graph.

Since these subjacent structures usually carry valueable information, the field of signal processing on graphs, or graph signal processing (GSP), has been active in the recent decade. In this work, we review some of the advances made possible by this new field. We give special focus to time-varying signals—in which each graph vertex represents a time-series—and the adaptive algorithms designed for this new kind of signal processing.

By applying to GSP techniques borrowed from adaptive filtering, one is able to derive processing algorithms that learn from the statistics of a graph signal. When the subjacent structure of the irregular domain on which the data reside is accounted for, the results are higher inference power, and better-informed decisions.

Contents

List of Figures	ix
List of Symbols	xi
1 Introduction	1
1.1 Contributions	2
1.1.1 The graphdsp computer library	2
1.2 Structure of this text	3
1.3 Notation and conventions	3
2 Graph signal processing fundamentals	4
2.1 Introduction	4
2.1.1 Unweighted and weighted graphs	4
2.1.2 Edge weights and kernels	6
2.1.3 Graph signals	7
2.2 The graph Laplacian	9
2.3 The graph Fourier transform	13
2.3.1 Frequency interpretation and analogy to classical Fourier analysis	14
2.3.2 Frequency-domain processing	17
2.3.3 Wavelets	21
3 The Least Mean Squares algorithm	27
3.1 The algorithm	28
3.2 Probabilistic sampling	33
4 Further exploratory results	35
4.1 Sparse wavelet representation	35
4.1.1 Proposed algorithm	36
4.1.2 Results and stability considerations	39
4.2 Optimal sampling for Least Mean Squares	40
5 Conclusions	43

A List of selected routines from the graphdsp package	45
Bibliography	48

List of Figures

2.1	A visualization of a graph with $N = 5$ nodes. Each dot represents a node, and is labeled by an integer from 0 to 4. We draw a line segment between each pair of connected nodes; i.e., each line represents an edge of the graph. If the graph were directed, arrows would be drawn instead of lines; one for each (directed) connection.	5
2.2	Visualization of a classical discrete-time signal with positive real values.	8
2.3	Visualization of a graph signal.	9
2.4	Zero-crossing edges for Laplacian eigenvectors on a randomly-generated graph with two different kernels for edge weights.	15
2.4	[cont.] Zero-crossing edges for Laplacian eigenvectors on a randomly-generated graph with two different kernels for edge weights.	16
2.5	Using the same graph as in Figure 2.4, with the inverse-square kernel, we attempt to mimic the Euclidean-domain phenomenon of constructing an impulse signal out of in-phase complex exponentials.	19
2.6	An example of frequency-domain filtering.	21
2.7	Wavelet examples.	22
2.8	Mother functions that generate an SGWT.	25
2.9	An example of a set of generating functions for a Spectral Graph Wavelet Transform. The scales are uniformly spaced in the log-domain, hence the log-scale in the horizontal axis. In order to handle graphs with edges normalized for $\lambda_{\max} = 1$, the smallest (finest) scale, s_J , shown in yellow (rightmost curve), has been chosen so that $g(s_J \lambda_{\max}) = 1$; that is, $s_J = 1$	26
3.1	Results of an LMS example.	32
4.1	Example of application of Algorithm 1 in a graph with $N = 64$ vertices, using $J = 4$ scales.	38
4.2	Number of coefficients of the sampled SGWT transform, for $J = 4$. Critical sampling ($\alpha = 0.5$) generates a transform with exactly $1 \cdot N = N$ coefficients, while the full transform ($\alpha = 1.0$) needs $(J + 1) N = 5N$ coefficients.	39

4.3	Condition number of the problem of inverting the sampled SGWT transform.	40
4.4	Less greedy sampling for the bandlimited LMS algorithm.	42

List of Symbols

$\mathbb{C}^{\mathcal{G}}$	Set of functions from the vertices of the graph \mathcal{G} to the complex field \mathbb{C} . Such functions are to be interpreted as (complex) signals on the graph \mathcal{G} , p. 7
$\mathbb{C}^{\mathcal{V}}$	Set of functions from \mathcal{V} to the complex field \mathbb{C} . Such functions are to be interpreted as (complex) signals on the vertices $v_i \in \mathcal{V}$ of a graph, p. 7
D_i	Degree of a vertex in a graph, p. 9
D	Degree matrix of a graph, p. 9
\mathcal{G}	Graph structure, defined as $(\mathcal{V}, \mathbf{W})$, p. 5
K_2	Gaussian kernel for assigning edge weights, p. 7
K_1	Inverse-square kernel for assigning edge weights, p. 7
K	Kernel function used in the construction of graphs to assign edge weights based on similarity values, p. 7
Λ	Diagonal matrix of eigenvalues of L , p. 13
L	Laplacian matrix of a graph, p. 9
M_{ki}	Entry of the incidence matrix of a graph, corresponding to edge e_k and node e_i , p. 12
M	Incidence matrix of a graph., p. 12
N	Number of nodes in a graph, p. 4
U	Orthogonal matrix of eigenvectors of L , p. 13
\mathcal{V}	Set of graph vertices or nodes; graph domain, p. 4
W_{ij}	Weight of the edge between vertices v_i and v_j of a graph, or zero if there is no such edge, p. 6

W	Weights matrix, or adjacency matrix, p. 5
d_{ij}	Similarity measure between vertices v_i and v_j of a graph. Usually, euclidean distance between vectors represented by the nodes of the graph, or some other distance metric in a feature space. The smaller the value d_{ij} , the more similar the nodes are, p. 7
λ_l	l -th eigenvalue of the Laplacian L , when listed by increasing value., p. 12
τ	Threshold used for forcing the sparsity of the adjacency matrix of a graph, p. 7
\mathbf{u}_l	Element of the orthonormal eigenvector basis of matrix L correspondent to eigenvalue λ_l , p. 12
v_i	A node, or vertex, of a graph, p. 4
\mathbf{x}	Signal on a graph, interpreted as a vector $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$, p. 7
x	Signal on a graph, interpreted as a function $x : \mathcal{G} \rightarrow \mathbb{C}$, p. 7

Chapter 1

Introduction

Traditional signal processing thrives when applied to uniform, euclidean domains. The uniformity in the time intervals between each sample of a time series, even though being an essential reason for the simplicity and power of the Fourier transform and most other classical signal processing techniques [1], is most often unacknowledged and taken for granted. Indeed, it is quite easy to see the reason why we usually speak of *time series* when we think about classical discrete-domain signals—they are usually even called *time-domain* signals! The reason is that time is a rare example of a resource with the uniformity properties mentioned above. Of course, non-time discrete domains also make up for important applications of traditional signal processing theory [2]—after all, the very first applications of Fourier series were solving partial differential equations (PDEs) in space, or space and time jointly [3], [4]. Such applications, however, still depend on the uniformity of their domains (or, in the case of differential equations, the uniformity of the discretization of the domains).

Not all interesting domains have such a desirable property, unfortunately. When positioning sensors along a large area, geography and topography will usually not let us draw a perfect, uniform square grid out of our sensors. Likewise, when studying social networks, one has to account for the fact that not everyone has the same number of friends! Energy and transportation networks are also examples of networks in which the nodes, distribution centers or road connections, are spatially structured in a non-uniform fashion.

Graph signal processing (GSP), or signal processing on graphs [5], [6], is a set of techniques for analysing, filtering, and manipulating signals defined on non-uniformly structured domains. GSP is usually concerned with *weighted graphs*; a weighted graph is any domain of elements related to each other through some property which enables us to assign a strength measure for such a relation. When the elements are points in a geometric setting, for instance, this property could be taken to be the distance between points.

Graphs, being highly flexible constructions, can be the stage to a wide range of phe-

nomena. When an epidemic of some contagious disease outbreaks, graphs can represent the transportation networks that will help identify patterns in the spread of the disease. Even when there is no outbreak at all, migration patterns of different species in an ecosystem can be encoded in a graph. Further applications that benefit from constructing graph structures from arbitrary datasets, and using the spectral properties of the graphs that arise, include clustering (since clustered data naturally generates disconnected, or almost-disconnected graphs) [7], design of processor interconnections, and quantum physics (in which graph edges can represent couplings between particles) [8].

1.1 Contributions

In this text, we introduce the reader, in a didactic way, to the field of graph signal processing. We analyze some of the most important tools used, making explicit their connections and their differences to analogous tools traditionally used in signal processing applications, and we give special focus to a Least Mean Square algorithm for estimating band-limited signals on a graph.

We also contribute two experiments that build on the themes that will be developed in the text, which we hope that will help the reader develop an intuition about the possibilities and limitations of graph signal processing.

1.1.1 The `graphdsp` computer library

We have developed a computer library, using the SciPy ecosystem [9], for performing basic graph signal processing calculations and plotting their results. The library also implements a functionality for quickly generating random graphs for the purpose of prototyping and testing.

All of the illustrations in this text were generated by this library. Attached to some of the illustrations are code snippets that illustrate how the library can be used to generate such figures. All of these snippets assume that the library has been downloaded and installed by issuing the following commands

```
# Download package
wget "http://www02.smt.ufrj.br/~pedro.fonini/graphdsp.tar.xz"
# Extract archive
tar xf graphdsp.tar.xz
# Install python package
cd graphdsp; pip install .
```

and that the library has been loaded in Python with `import graphdsp as gsp`.

The documentation of the functionalities distributed is available by using Python's built-in documentation system. For instance, to learn more about `gsp.GraphSignal.gft`,

a method in the class `gsp.GraphSignal` for computing graph Fourier transforms, just type `help(gsp.GraphSignal.gft)` at a Python shell. In Appendix A we list some of the most important classes and routines implemented in the package.

1.2 Structure of this text

In Chapter 2 we lay down the Graph Signal Processing framework. Most of the tools that will be used during the rest of the text are defined and examined in this chapter. In Chapter 3, we describe the Least Mean Square algorithm for graphs, which aims to estimate a band-limited, time-varying signal on a graph. We show the results of some complementary experiments in Chapter 4, and summarize our conclusions in Chapter 5.

1.3 Notation and conventions

In this text, vectors, denoted in boldface (e.g. \mathbf{x}), usually represent *graph signals*, and are construed as column vectors. If \mathbf{x} is graph signal, each component of the vector is the value of the signal on some node of the graph. If such values are complex, we write $\mathbf{x} \in \mathbb{C}^N$ and index \mathbf{x} using the integers $i = 0, 1, \dots, N - 1$. Each x_i is thus an *entry* of the signal \mathbf{x} . When we want to emphasize that \mathbf{x} is a signal defined on a specified graph \mathcal{G} , we will also write $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$ or $\mathbf{x} \in \mathbb{C}^{\mathcal{V}}$, where \mathcal{V} is a list of graph vertices. Most of the time, $\mathbb{C}^{\mathcal{G}}$, $\mathbb{C}^{\mathcal{V}}$, and \mathbb{C}^N will be used interchangeably if $N = |\mathcal{V}|$ is the number of vertices in the graph. Generally, if X is a finite set, $|X|$ is the number of elements of X . Graphs and signals on graphs will be described in more detail in Section 2.1.

Upper-case bold letters (e.g. \mathbf{A}) are used for matrices. The transpose of a matrix is written as \mathbf{A}^T ; for instance, if \mathbf{x}, \mathbf{y} are vectors, then their inner product is $\mathbf{x}^T \mathbf{y}$. The symbol \mathbf{I} is reserved for the identity matrix. The dimensionality of \mathbf{I} will always be clear from context.

The *norm* of a vector $\mathbf{x} \in \mathbb{C}^N$ is denoted by $\|\mathbf{x}\|$, and is always defined as the Euclidean (or ℓ^2) norm:

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}.$$

Finally, in probabilistic contexts, the expected value of a random variable is denoted by the operator $E[\cdot]$.

Chapter 2

Graph signal processing fundamentals

This chapter is dedicated to make an overview of graph signal processing techniques. We will review how the Laplacian operator in \mathbb{R}^n gives rise to the Fourier transform, and describe how this can be mimicked to define a graph Fourier transform (GFT). We will also discuss the assumptions made by the GSP framework that we describe, and mention other possibilities.

In section 2.1 we present the formal definitions that will be used throughout the text, and discuss how these definitions can be used to represent irregular structures on networks. In section 2.2 we introduce the graph Laplacian, a linear operator that measures smoothness of graph signals and enables the translation of tools from continuous domains to discrete domains described as graphs. Section 2.3 uses the spectral decomposition of the Laplace operator to define the GFT on arbitrary weighted graphs.

2.1 Introduction

2.1.1 Unweighted and weighted graphs

A *graph* is an ordered pair $(\mathcal{V}, \mathbf{A})$, where \mathcal{V} is a finite list of N elements, and $\mathbf{A} \in \{0, 1\}^{N \times N}$ is a matrix with entries 0 and 1. The elements of \mathcal{V} are called *nodes* or *vertices* of the graph. Though strictly not necessary for the formal definition, it is important, for the definiteness of the algorithms and for the computational perspective that surrounds graph theory, that the vertices be labeled by integers, as in:

$$\mathcal{V} = \{v_0, v_1, \dots, v_{N-1}\},$$

which is why we called \mathcal{V} a list instead of a set, above. The actual order of the labeling does not affect the results of the principal theorems of graph theory (and, indeed, any result that does depend on the order of the labeling is actually a result *about the labeling*, instead of about *the graph*). \mathcal{V} is also called the graph domain.

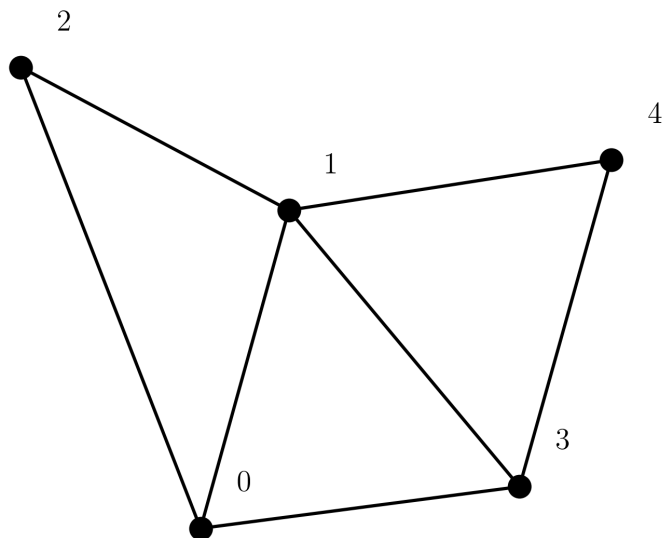


Figure 2.1: A visualization of a graph with $N = 5$ nodes. Each dot represents a node, and is labeled by an integer from 0 to 4. We draw a line segment between each pair of connected nodes; i.e., each line represents an edge of the graph. If the graph were directed, arrows would be drawn instead of lines; one for each (directed) connection.

The matrix $\mathbf{A} = [A_{ij}]$ is called the *adjacency matrix* of the graph, and for each pair (i, j) such that $A_{ij} = 1$, we say that node v_i is *connected to* node v_j . Here arises the first categorical bifurcation we need to be specific about. When the adjacency matrix of a graph is symmetric, a node v_i will be connected to another v_j if and only if v_j is connected to v_i . In this case, we say the the graph is *undirected*. Otherwise, the graph is said to be *directed*. Typically, when one uses the word *graph* without specifying whether directed or not, they mean an undirected graph¹. In this work, we will focus on undirected graphs, and will refer to such as simply *graphs*. The consequences of restricting our attention away from directed graphs are discussed in [6].

Whenever nodes v_i and v_j are connected—no need to specify if v_i is connected to v_j or the other way around when the graphs are undirected—we say that they are *adjacent* or *neighbors*, and that there is an *edge* between them, this edge being *incident* on each of them. It is usual to visualize the relationship between nodes and edges in a graph as in Figure 2.1.

A *weighted graph* is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathbf{W})$, where \mathcal{V} is again a finite list of N elements, and $\mathbf{W} \in \mathbb{R}^{N \times N}$ is a real, symmetric², square matrix with nonnegative entries.

¹For instance, Godsil and Royle [10] bypass the adjacency matrix, and define: “A graph X consists of a vertex set $V(X)$ and an edge set $E(X)$, where an edge is an *unordered* pair of distinct vertices of X .” The word “unordered” (not emphasized in the original text) implies that connections between nodes are not directed.

²We are again focusing on undirected graphs. For the definition of a *directed* weighted graph we would remove the requirement of \mathbf{W} being a symmetric matrix.

```

graph = gsp.GeometricUWG.make_random(N=5)
zeros = len(graph) * [0]
signal = gsp.GraphSignal(zeros, graph=graph) # zero signal
signal.plot(colorbar=False, show_indexes=True, dotcolor='k') # plot

```

Listing 1: Instructions to generate Figure 2.1.

We will still call \mathbf{W} the *adjacency matrix* of the graph \mathcal{G} , but its semantics are richer. Given any two nodes v_i and v_j , if $W_{ij} = 0$ then there is no connection between them. However, if $W_{ij} > 0$, then not only we say that v_i and v_j are connected by an edge, but also the *weight* of this connection is given by the value of W_{ij} . The interpretation is that high edge weights represent strong connections, whereas low edge weights represent weak connections, or occasionally almost the same as no connection at all.

As mentioned above, it is important for GSP applications to be able to specify the intensity of connections between nodes—they might represent geographical closeness, or similarity in some feature space. In this work, all graphs are weighted.

A last restriction is in order. In a graph, an edge from a node to itself is called a *loop*. That happens whenever $W_{ii} \neq 0$ for some vertex v_i . Since GSP applications are concerned about *relations* between elements, and loops are not useful to represent them, it is usual to ban loops. For the rest of this text, all graphs will obey the following definition:

Definition 2.1. A *graph* is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathbf{W})$ where \mathcal{V} is a finite list of N elements, and \mathbf{W} is a real, symmetric, square, $N \times N$ matrix with zeros on the diagonal and nonnegative entries elsewhere. In other words, all graphs are undirected, weighted with real, nonnegative weights, and without loops.

Although graph nodes and edges could represent anything, visualizations such as the one in Figure 2.1, in which the nodes are thought of as points on the (two-dimensional) plane, will be useful throughout this text. It should be noted, however, that graphs are useful to represent non-uniform data in any number of dimensions.

2.1.2 Edge weights and kernels

Unlike classical signal processing, in which the topology of the domain is always given, one of the key steps of applying GSP tools to solve a problem is constructing the graph itself. Depending on what data is available and what kind of restrictions are imposed by the problem at hand, the implementer can be responsible for designing any number of the three graph features: nodes, connections between nodes, and weights of these connections. Often, an ingenious way of assigning graph weights can single-handedly solve a problem [5].

When weights are to be chosen to represent similarity, one of the most straightforward ways to assign weights is to map similarity values d_{ij} between vertices v_i and v_j (e.g., euclidean distances between vectors represented by the nodes of the graph) to edge weights via a kernel function K :

$$W_{ij} = K(d_{ij}).$$

Since having less edges makes the visualizations cleaner, and also makes the adjacency matrix sparse, which helps in the computational complexity of the algorithms that will process these matrices, it is useful to assign edges only to pairs of vertices whose similarities do not exceed some threshold τ :

$$W_{ij} = \begin{cases} K(d_{ij}), & \text{if } d_{ij} \leq \tau \\ 0, & \text{otherwise.} \end{cases}$$

Another way of forcing the sparsity of \mathbf{W} is to connect each node only to the k nearest (most similar) nodes.

Although the construction of the graph affects most GSP techniques used, it is not clear exactly how each design choice (e.g. which kernel function, similarity measure, or sparsification strategy to use) influences the results [5]. For this reason, in this work we will sometimes repeat some experiments with both of the following kernel functions:

$$K_1(d) = \frac{1}{d^2}$$

$$K_2(d) = \exp\left\{-\frac{1}{2}d^2\right\}$$

2.1.3 Graph signals

In traditional signal processing, we call a *signal* any kind of linearly-indexed data. For example, a mapping from continuous time instants to signal values could be called an analog signal; here, time is the linear index mentioned. In GSP, the index domain is the graph itself. A signal is, therefore, a mapping from the graph nodes to signal values.

We write

$$x : \mathcal{V} \rightarrow \mathbb{C} \quad \text{or} \quad \mathbf{x} \in \mathbb{C}^{\mathcal{V}}$$

to mean that x is a signal on the graph whose vertices are \mathcal{V} . The difference in typeface from x to \mathbf{x} is to emphasize that the signal can be seen as a function from the domain of vertices to the field of complex numbers; or as a vector of complex sample values. In the first case, the value of the signal on the i -th vertex is denoted $x(v_i)$, and in the second, the i -th entry of the vector is denoted x_i . Both interpretations are equivalent, although the second is more amenable to linear algebra techniques.

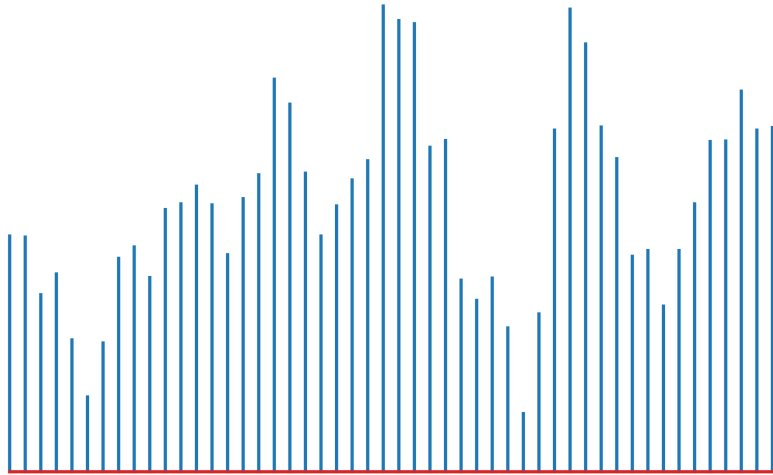


Figure 2.2: Visualization of a classical discrete-time signal with positive real values.

```
noise = np.random.standard_normal((len(graph),))
signal = gsp.GraphSignal(noise, graph=graph)
signal.plot()
```

Listing 2: Instructions to generate Figure 2.3, using the same graph object as in Listing 1.

It is sometimes convenient to write

$$x : \mathcal{G} \rightarrow \mathbb{C} \quad \text{or} \quad \mathbf{x} \in \mathbb{C}^{\mathcal{G}}$$

using \mathcal{G} in the place of \mathcal{V} , when we want to emphasize that the domain of the signal has the structure of a graph, instead of being just a list of elements with no connectivity information.

Visualizing real signals on one-dimensional domains is usually done via a cartesian plot: we plot the independent variable (e.g. time) in the axis of abscissae, and the signal values in the axis of ordinates. Figure 2.2 shows an example.

Graph domains usually represent, at best, two-dimensional spaces, which makes the cartesian plots mentioned above more difficult to render on two dimensional media and to interpret—visualizing 3d plots is always hard. In this work, we will usually represent real signals on graphs as colored plots. Diagrams like the one in Figure 2.1 will feature color-coded dots, and the color of each dot will serve as visual indicator of the strength of the signal value at that node, as in Figure 2.3.

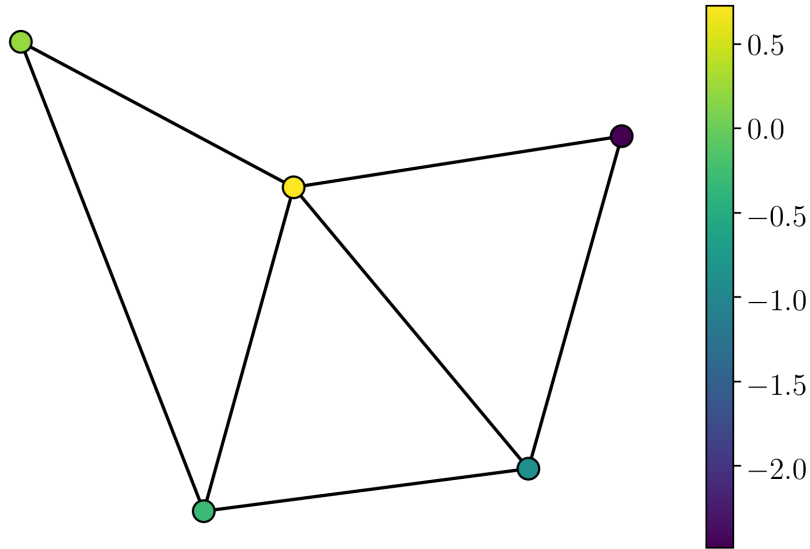


Figure 2.3: Visualization of a graph signal.

2.2 The graph Laplacian

The *degree* of a node in a graph is the sum of the weights of the edges that incide on it. In other words, the degree D_i on vertex v_i is:

$$D_i = \sum_j W_{ij}.$$

(It doesn't matter if, in the above summation, j is restricted to indexes of neighboring vertices of v_i or not, since W_{ij} is zero whenever v_i and v_j are not adjacent.) The degree matrix of a graph is the diagonal matrix formed by the degrees of each node:

$$\mathbf{D} = \begin{bmatrix} D_0 & & \\ & \ddots & \\ & & D_{N-1} \end{bmatrix}$$

and the Laplacian matrix is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}.$$

Note that the Laplacian matrix ignores loops in the graph, if they are present, since adding or removing a loop at node v_i would just increase or decrease the loop weight to both matrices \mathbf{D} and \mathbf{W} in the same entry, maintaining the value of $D_i - W_{ii}$ unchanged.

The Laplacian matrix, when seen as a linear transformation on the space of graph signals, acts according to the following formula. If $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$ is a signal on the graph \mathcal{G} ,

and \mathbf{L} is the Laplacian matrix of \mathcal{G} , then \mathbf{Lx} is another signal on the same graph, with components given by:

$$(\mathbf{Lx})_i = ((\mathbf{D} - \mathbf{W}) \mathbf{x})_i = D_i x_i - \sum_j W_{ij} x_j = D_i \left[x_i - \frac{\sum_j W_{ij} x_j}{\sum_j W_{ij}} \right]$$

This is proportional to the degree D_i , but the most interesting factor (the one that determines the polarity of $(\mathbf{Lx})_i$, and also the only one that actually depends on the signal x itself) is the subtraction in brackets. This second factor is the difference between the value of the signal x at vertex v_i and the mean value of the signal at the neighboring vertices; this mean is pondered by the weights of the edges that connect v_i to each of its respective neighbors.

In other words, each sample of the signal \mathbf{Lx} is proportional to the difference between the original signal value at that vertex, and the mean of the signal values at neighboring vertices (pondered by how close they are to the central vertex, as measured by the connection strength). This is similar to the behaviour of the Laplacian operator in the continuous domain \mathbb{R}^n . The continuous Laplacian operator in \mathbb{R}^n , usually called Δ , ∇^2 , or $\nabla \cdot \nabla$, is given by:

$$\begin{aligned} & \text{if } f : \mathbb{R}^n \rightarrow \mathbb{C} \text{ is twice-continuously differentiable,} \\ & \text{then } \Delta f(\mathbf{t}) = \sum_i \frac{\partial^2 f}{\partial t_i^2}(\mathbf{t}). \end{aligned}$$

The Laplacian of a continuous function f also yields, at each point \mathbf{t} of the Euclidean space, a measure of how much $f(\mathbf{t})$ differs from the mean value of f at the neighborhood of \mathbf{t} , as made precise by the following theorem: [11]

Theorem 2.2. *Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be twice-continuously differentiable. For each $r > 0$, let $S_r(\mathbf{t})$ be the surface of the sphere of radius r centered at \mathbf{t} , and let $m(S_r(\mathbf{t}))$ be its area. For a fixed \mathbf{t} , let $\mu(r)$ be the mean value of f on this surface, as given by the surface integral:*

$$\mu(r) = \frac{1}{m(S_r(\mathbf{t}))} \int_{S_r(\mathbf{t})} f(\mathbf{s}) \, d\mathbf{S}(\mathbf{s}).$$

Then, as r approaches 0 from above:

$$\mu(r) = f(\mathbf{t}) + \frac{1}{2} \frac{\Delta f(\mathbf{t})}{n} r^2 + o(r^2),$$

where n is the dimensionality of the domain \mathbb{R}^n , and $o(r^2)$ represents a function of r that converges to 0 faster than r^2 when $r \rightarrow 0$.

Alternatively, stated without the little- o notation:

$$\lim_{r \searrow 0} \frac{f(\mathbf{t}) - \mu(r)}{r^2} = -\frac{1}{2} \frac{\Delta f(\mathbf{t})}{n}.$$

The last formula in the theorem above states that, for small r , the difference between the value of f at some point $\mathbf{t} \in \mathbb{R}^n$ and the average value of f in a neighborhood of \mathbf{t} given by the surface of the sphere of radius r is proportional to minus the Laplacian of f at \mathbf{t} :

$$f(\mathbf{t}) - \mu(r) \approx -\frac{r^2}{2n} \Delta f(\mathbf{t}).$$

Using this formula, we see that the continuous Laplacian operator for the Euclidean space is actually analogous to *minus* the Laplacian operator for graphs. If, for some graph signal $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$, the Laplacian $(\mathbf{L}\mathbf{x})_i$ at vertex v_i is *positive*, this means that the value x_i of the signal at v_i is *higher*, on average, than the values at x at neighboring vertices. Meanwhile, for a twice-continuously differentiable function in \mathbb{R}^n , if its Laplacian $\Delta f(\mathbf{t})$ is positive, then its value $f(\mathbf{t})$ is *lower*, on average, than the values of f in the neighborhood of \mathbf{t} .

The difference in the polarities of the conventions is, of course, just a matter of terminology. Still, it serves to show that \mathbf{L} is a discrete version of the continuous Laplacian operator, in the sense that both of them yield, at each point in the domain, a measure of how much the signal value at that point differs from the average signal value in the point's neighborhood.

One consequence of this property of \mathbf{L} is that, since $\mathbf{L}\mathbf{x}$ depends, at each vertex, on a value of x minus an average of values of x , the global average of x is irrelevant. In other words, if x is a constant signal, then $\mathbf{L}\mathbf{x} = \mathbf{0}$. Furthermore, since the averages mentioned above are all *local*, meaning that all differences are differences between values of x at connected vertices, if x is constant at each connected component of the graph, then $\mathbf{L}\mathbf{x} = \mathbf{0}$.

This is a recipe for finding eigenvectors of the Laplacian with eigenvalue zero. Suppose that \mathcal{G} has M connected components; i.e., the set of vertices \mathcal{V} can be partitioned in M subsets:

$$\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_{M-1}$$

in such a way that there are no edges from \mathcal{V}_k to \mathcal{V}_l (unless $k = l$, of course). For each $k = 0, 1, \dots, M - 1$, we define the k -th indicator signal $\mathbf{x}^{(k)}$ as $x_i^{(k)} = 1$ whenever $v_i \in \mathcal{V}_k$, and $x_i^{(k)} = 0$ otherwise. Then all M indicator signals are orthogonal to one another, and all of them satisfy $\mathbf{L}\mathbf{x}^{(k)} = \mathbf{0}$. Conversely, if \mathbf{x} is such that $\mathbf{L}\mathbf{x} = \mathbf{0}$, then it is possible to show, using mathematical induction on the size of the graph \mathcal{G} , that \mathbf{x} is constant at each connected component of the graph.³ Therefore, the kernel of \mathbf{L} has

³This is a manifestation of the finite character of graphs. In infinite (and connected) domains, a signal

dimension M , and $\{\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(M-1)}\}$ is a basis for this kernel. We will usually assume that a graph has only one connected component, in which case its Laplacian matrix has exactly one eigenvalue equal to zero, and its respective eigenvector is the constant signal.

As for the other eigenvalues of \mathbf{L} , we can show that they are all positive. We define the incidence matrix \mathbf{M} of a graph as follows. \mathbf{M} is an $E \times N$ real matrix, where E is the number of edges in the graph. We assume that the edges are labeled e_0 to e_{E-1} , and for each edge e_k and each vertex v_i we let M_{ki} be given by:

$$M_{ki} = \begin{cases} 0, & \text{if } e_k \text{ does not incide on } v_i \text{ or } v_j; \\ +\sqrt{W_{ij}}, & \text{if } e_k \text{ incides on } v_i \text{ and } i < j; \\ -\sqrt{W_{ij}}, & \text{if } e_k \text{ incides on } v_j \text{ and } i < j. \end{cases}$$

In words, for each k , if we let v_i and v_j be the vertices connected by edge e_k , then the k -th line of \mathbf{M} has all zero entries, except that M_{ki} and M_{kj} are $\pm\sqrt{W_{ij}}$. If the graph were directed, the choice of which of M_{ki} and M_{kj} would be assigned which sign (plus or minus) could be informed by the direction of the edge e_k . However when the graph is undirected, the signs can really be chosen at random. If, for definiteness, we choose as we did above, and assign $+\sqrt{W_{ij}}$ to the entry corresponding to the node with lower index and $-\sqrt{W_{ij}}$ to the other one, then the resulting signs of the entries will be directly dependent on the order of the labeling of the nodes in the graph. Still, since the signs are not relevant, the labeling order is also not relevant.

Making the calculations, one can show that $\mathbf{L} = \mathbf{M}^T \mathbf{M}$. Because of this formula, \mathbf{L} is a semi-definite positive matrix, and therefore all non-zero eigenvalues must be positive, as promised above.

Since \mathbf{L} is symmetric, it has a complete set of eigenvectors and eigenvalues. If we order the eigenvalues increasingly:

$$\lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{N-1},$$

then λ_0 will always be 0, as mentioned above (since all graphs have at least one connected component). For a connected graph (which has exactly one connected component), we have:

$$0 = \lambda_0 < \dots \leq \lambda_{N-1}.$$

Let \mathbf{u}_l be the eigenvectors of \mathbf{L} corresponding to λ_l . Since \mathbf{L} is a real symmetric matrix, we will always choose the vectors \mathbf{u}_l to be real and to form an orthonormal basis

can have null Laplacian, but still be non-constant. In the realm of continuous functions on the Euclidean space, for instance, such signals are called harmonic functions, and finding all of them given boundary conditions amounts to solving a partial differential equation called the Laplace equation.

of $\mathbb{C}^{\mathcal{G}}$. If the eigenvectors are arranged, in order, as columns of an orthogonal matrix \mathbf{U} , and the eigenvalues are also arranged in order as diagonal elements of a diagonal matrix $\mathbf{\Lambda}$, then:

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T.$$

2.3 The graph Fourier transform

In classical signal processing, the Fourier transform is a fundamental tool. It helps analyse signals, explain phenomena, design filters, and solve problems. As mentioned above, one of the first problems that Fourier analysis was applied to was solving partial differential equations, the most simple of which was the steady-state heat equation, also called the Laplace equation:

$$\Delta f(\mathbf{t}) = 0.$$

As in any linear equation, it is instructive to search for eigenvalues and eigenvectors of the linear operators that arise in the problem. In the case for the one-dimensional Laplacian operator, for example, the eigenfunctions are the so-called complex exponentials:

$$\Delta \{e^{2\pi j\xi t}\} = -(2\pi\xi)^2 e^{2\pi j\xi t}.$$

Therefore, the eigenvalues, $-(2\pi\xi)^2$, of the Laplacian operator are related to the *frequencies* ξ . The Fourier transform on \mathbb{R} is exactly the change of basis that diagonalizes the Laplacian operator Δ , which yields, for each frequency ξ , the inner product between a signal f and the complex exponential $e_{\xi}(t) = e^{2\pi j\xi t}$:

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

Although it is not obvious at once how to define the analog of complex exponentials for signals on a graph, the Laplacian operator can be exploited to this end. The analogy between \mathbf{L} and $-\Delta$ mentioned in the last section will be main connector used to transform traditional signal processing tools into graph signal processing tools.

The first thing to do in order to make this connection is to define a Fourier transform for signals on graphs. If $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$ is a graph signal, we mimic the definition for continuous time, and define the Fourier transform of \mathbf{x} to be the vector $\hat{\mathbf{x}} \in \mathbb{C}^N$ with components:

$$\hat{x}(\lambda_l) = \hat{x}_l = \langle \mathbf{x}, \mathbf{u}_l \rangle = \mathbf{u}_l^T \mathbf{x}.$$

Using this definition, the vector $\hat{\mathbf{x}}$ can be directly calculated as:

$$\hat{\mathbf{x}} = \mathbf{U}^T \mathbf{x}.$$

Thus, \mathbf{U} is called the matrix of the graph Fourier transform, or GFT. Since \mathbf{U} is orthogonal, the inverse GFT is given by:

$$\mathbf{x} = \sum_l \hat{x}_l \mathbf{u}_l = \mathbf{U}\hat{\mathbf{x}}.$$

2.3.1 Frequency interpretation and analogy to classical Fourier analysis

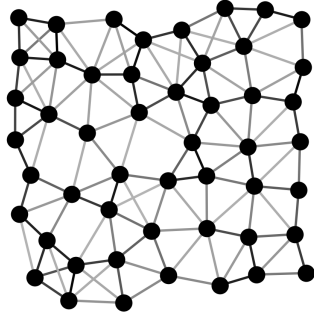
In this construction of the graph Fourier transform, the graph Laplacian eigenvalues λ_l are analog to frequencies in traditional signal processing—or, at least, to positive squared frequencies, since the eigenvalues of $-\Delta$ are $(2\pi\xi)^2$. Also, the eigenvectors \mathbf{u}_l are analog to the complex exponentials, and this analogy isn't just algebraic; intuitively, the higher the corresponding frequency λ_l , the “faster” the eigenvector \mathbf{u}_l changes values. As already mentioned, the eigenvectors corresponding to zero eigenvalues are constant at each connected component of the graph—if the graph is itself connected, then the zero eigenvalue has multiplicity one and its only eigenvector is the constant signal.

As for nonzero eigenvalues, in Euclidean domains the concept of a *fast-changing* complex exponential is closely linked to periodicity, which does not happen in graph Laplacian eigenvectors. Still, as pointed out in [5], another way of measuring the frequency of classical sinusoids, which can be interpreted in a graph domain, is to measure how often the signal changes polarity. That is, to measure how often the signal changes from positive samples to negative samples and vice-versa. In a graph setting, we can simply count the number of edges connecting samples with different polarity.

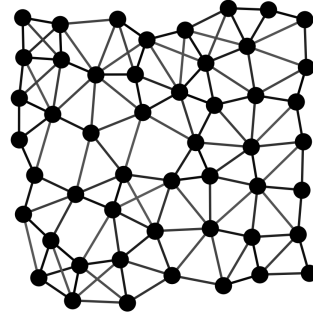
Figure 2.4 shows the result of an experiment in which we measure the rate of change of polarity of various eigenvectors of the Laplacian operator for a randomly generated graph. To generate the graph, we sample $N = 50$ points from a uniform distribution over a square on the plane repeatedly until no two chosen points are too close to one another (that is, forcing the distance between pairs of points to be higher than a chosen threshold). The chosen points are the nodes of the graph. Edge weights are assigned using a kernel function, and \mathbf{W} is sparsified using another threshold for the distance between points—i.e., if the distance d_{ij} is higher than the threshold, then W_{ij} is set to zero instead of $K(d_{ij})$. The experiment is repeated for inverse-square and Gaussian kernels K . Also, the edge weights of both graphs are scaled to force $\lambda_{N-1} = \lambda_{\max} = 1$.

The first two plots, Figures 2.4a and 2.4b, show the generated graph. Edge colors are drawn in grayscale to represent the weights. It can be seen that edge weight values are less diverse when using a gaussian kernel, which happens because the inverse-square kernel, $K_1(d)$, tends to infinity when d goes to zero, while the Gaussian is capped at a maximum peak value.

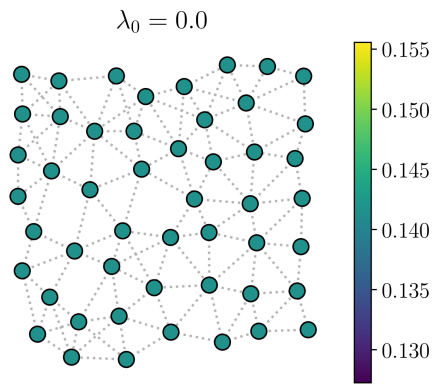
The rest of the plots show color-coded plots of the Laplacian eigenvector signals—analogue to complex exponentials for classical signal processing, as mentioned above—in the style of Figure 2.3. Below the first two plots, Figures 2.4c and 2.4d show the



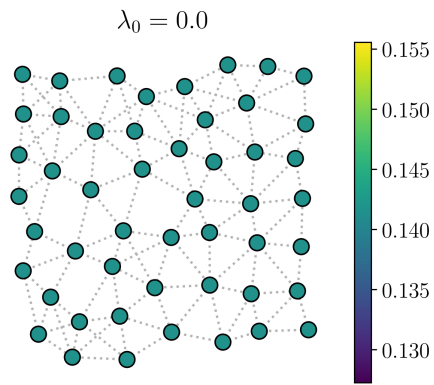
(a) Inverse-square kernel graph.



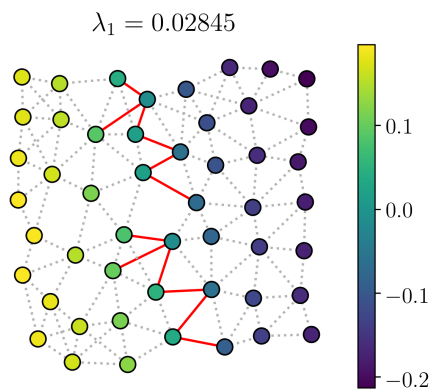
(b) Gaussian kernel graph.



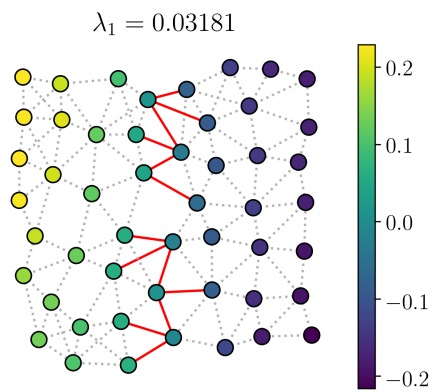
(c) Inverse-square kernel graph.



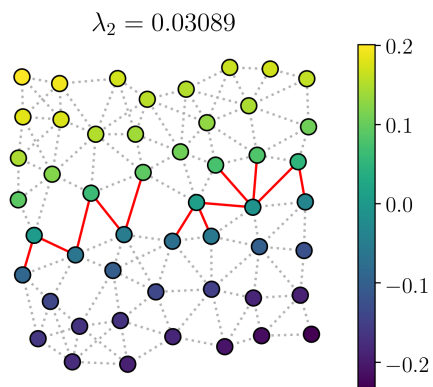
(d) Gaussian kernel graph.



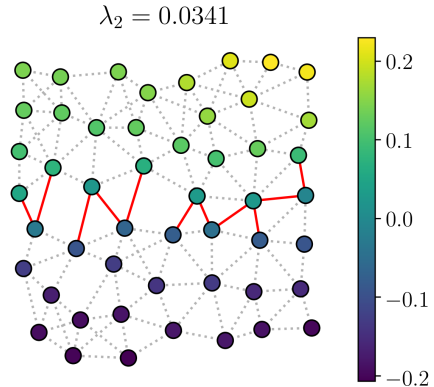
(e) Inverse-square kernel graph.



(f) Gaussian kernel graph.



(g) Inverse-square kernel graph.



(h) Gaussian kernel graph.

Figure 2.4: Zero-crossing edges for Laplacian eigenvectors on a randomly-generated graph with two different kernels for edge weights.

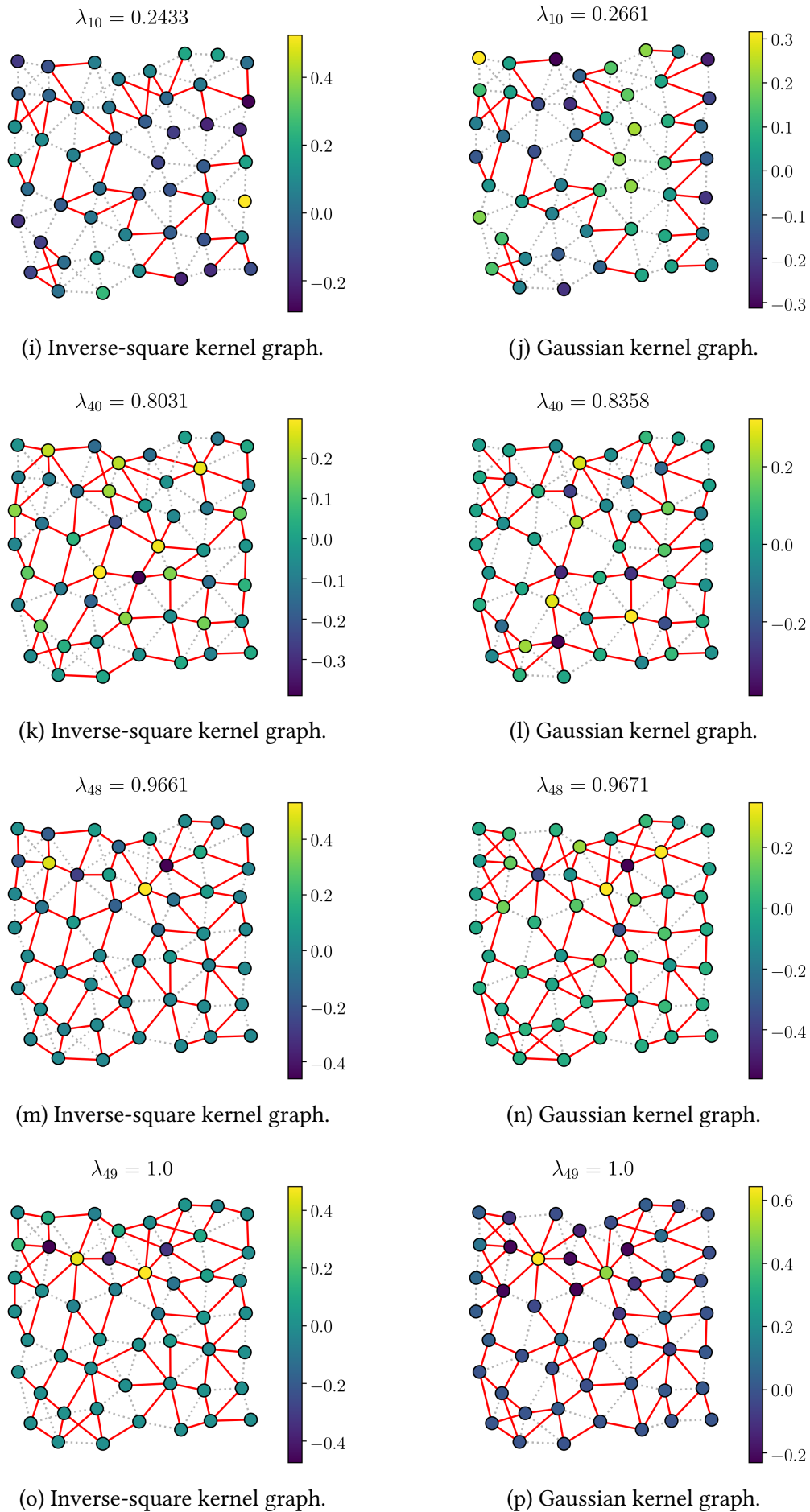


Figure 2.4: [cont.] Zero-crossing edges for Laplacian eigenvectors on a randomly-generated graph with two different kernels for edge weights.

```

graph = gsp.GeometricUWG.make_random()
zeros = len(graph) * [0]
signal = gsp.GraphSignal(zeros, graph=graph)
signal.plot(colorbar=False, dotcolor='k', edge_color='grayscale')
lambd, U = graph.spectral_decomposition()
for k in range(len(graph)):
    u_k = gsp.GraphSignal(U[:,k], graph)
    u_k.plot(title=f'$\\lambda_{{k}}={lambd[k]:.4}$',
             edge_color='zero-crossing')

```

Listing 3: Instructions to generate Figure 2.4.

first eigenvector, corresponding to $\lambda_0 = 0$. As mentioned above, the first eigenvector is always constant. All other eigenvectors, being orthogonal to the first, show both positive and negative entries, since the entries' sum must be zero. We will say that an edge is a zero-crossing edge if it connects a positive sample to a negative one. In the eigenvector signal plots, we have drawn zero-crossing edges in red, solid lines, and non-zero-crossing edges in faint-gray dotted lines.

It can be seen that the higher the eigenvalue, the more zero-crossing edges there are, implying that eigenvectors corresponding to high eigenvalues do indeed change values more often, which is a confirmation of the interpretation of the eigenvalues λ_l as frequencies.

2.3.2 Frequency-domain processing

The irregular nature of graphs can make it difficult to translate traditional signal processing tools into the GSP framework. However, graph frequencies λ_l are always just real numbers, and therefore more amenable to the algebraic manipulations that form the basis for classical GSP. This section is dedicated to an overview of such processing techniques.

The first thing to notice is that, although there are analogies, there are also important differences between traditional DSP and GSP. One important feature of euclidean domains is that the algebraic structure of the real field \mathbb{R} gives rise to *algebraic asymmetries*—special numbers with unique properties around which the algebraic structure is formed. The number $0 \in \mathbb{R}$ is one such number, sometimes called the *origin* in the real number line. It serves special purpose in classical Fourier analysis also: whenever a time-domain signal exhibits some kind of symmetry around zero, e.g. an even or odd signal, it will also exhibit special properties in the frequency domain. If this signal is shifted away from zero in the time domain, the frequency-domain representation be-

comes modulated.

When we define a signal on the frequency domain and apply the inverse Fourier transform to derive the time-domain representation, zero also plays a special role. For instance, if we define a signal with frequency component equal to 1 for all frequencies—clearly a construction that’s agnostic with respect to domain elements—the resulting time-domain signal is an impulse signal supported exactly on zero.

The very definition of the Fourier transform uses the origin of the time-domain as a special element. The orthonormal basis of eigenvectors of the Laplacian is not unique—each eigenvector e_ξ could be multiplied by some constant $c_\xi \in \mathbb{C}$ on the unit circle ($|c_\xi| = 1$), and the resulting basis $\{c_\xi e_\xi\}_{\xi \in \mathbb{R}}$ would still be an orthonormal eigenvector basis for the Laplacian operator; given any such basis, the Fourier transform is defined by choosing these constants in such a way that the value of any of the basis signals at the origin is exactly 1.

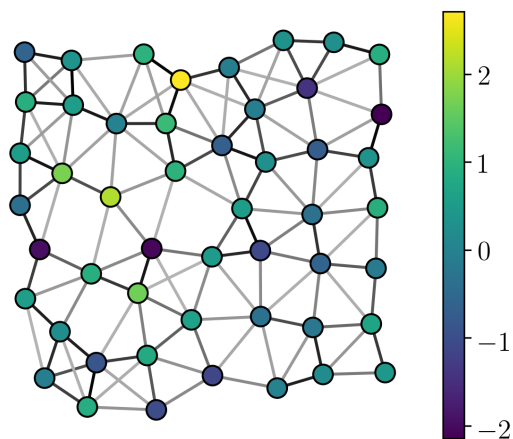
When the domain is a graph, however, there’s no algebraic structure giving rise to special domain elements. For an arbitrary graph \mathcal{G} , there’s simply no systematic way to pick one “central” vertex. This is a fundamental limit to the analogies between GSP and DSP tools. For example, the problem of choosing one of the many orthonormal basis of eigenvectors of the graph Laplacian based on the values of such signals on one specific domain element is, in a way, simpler than the same problem for the classical Fourier transform. Instead of having to choose, for each eigenvector \mathbf{u}_l , a complex multiplier c_l on the unit circle, we need only to choose between $c_l = +1$ and $c_l = -1$, since graph Laplacian eigenvectors are required to be real. However, in another way this problem is harder, because there is no central vertex we can use to inform our decision on the direction of each eigenvector. The best we could do is choose one vertex v_i —arbitrarily, or perhaps based on the application—and choose the direction of each eigenvector \mathbf{u}_l in such a way that $U_{il} \geq 0$. Figure 2.5 show the result of an experiment that explores this issue.

We will now explore a few more operations on graph signals that can be borrowed from the classical signal processing toolset.

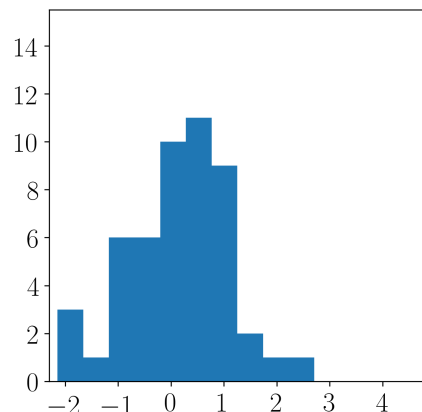
2.3.2.1 Frequency-domain filtering

Traditionally, frequency-domain filtering consists in decomposing a signal into its Fourier components, and then choosing which components to keep, which to enhance, and which to discard. In a more concrete description, we take a signal $f(t)$, multiply its Fourier transform $\hat{f}(t)$ by a filter function $\hat{h}(t)$, and then take the inverse Fourier transform of the resulting product. This is a simple enough specification of an operation that it can be translated into the GSP framework without much effort.

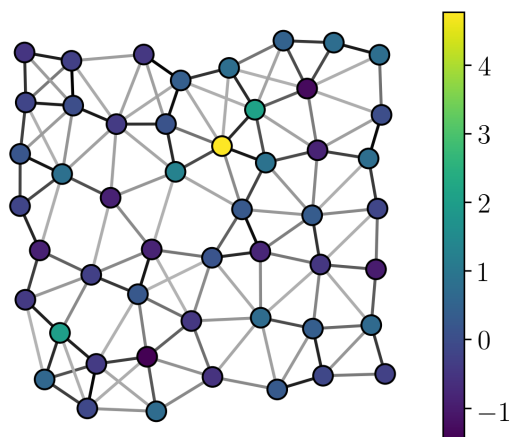
Let $\mathbb{R}_+ = [0, \infty[$ be the graph frequency domain. We call any continuous function $\hat{h} : \mathbb{R}_+ \rightarrow \mathbb{R}$ a graph filter. Given any graph \mathcal{G} and its Laplacian’s eigendecomposition



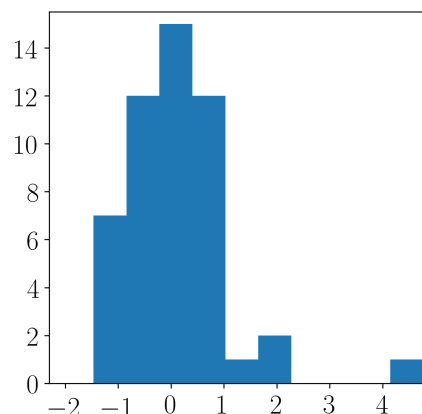
(a) Plot of a realization of the random graph signal $Uc = \sum_l c_l u_l$, where the coefficients c_l are independent and identically distributed, drawn randomly from the set $\{-1, +1\}$. This is a simulation of a sum of out-of-phase complex exponentials.



(b) Histogram of signal sample values for the graph signal at the left. The values are clustered around zero, showing no indication of a single outlier sample that could be interpreted as an impulse.



(c) Plot of the signal $Uc = \sum_l c_l u_l$ where, this time, the coefficients c_l are deterministically chosen to make each $c_l u_l$ positive at one specific (arbitrarily determined) node, seen in yellow in the figure. This simulates in-phase complex-exponentials.



(d) Histogram of signal sample values for the graph signal at the left. The variance of the values around zero is smaller than in Figure 2.5b above, and this time there is clearly an outlier, indicating that this signal could be interpreted as an approximation of an impulse signal. Here, the node on which the impulse is supported is responsible for $\approx 46\%$ of the energy of the signal.

Figure 2.5: Using the same graph as in Figure 2.4, with the inverse-square kernel, we attempt to mimic the Euclidean-domain phenomenon of constructing an impulse signal out of in-phase complex exponentials.

```

# Randomize polarity of each column of U:
U_randomized = U * ((np.random.random((len(graph),)) >= .5)*2-1)
signal = gsp.GraphSignal(U_randomized.sum(axis=1), graph)
signal.plot(edge_color='grayscale', colorbar=False)
plt.figure()
plt.hist(U_randomized.sum(axis=1))

# Choose column polarities so that U[0, i] > 0:
U_deterministic = U_randomized
for i in range(len(graph)):
    if U_deterministic[0, i] < 0:
        U_deterministic[:, i] *= -1
signal = gsp.GraphSignal(U_deterministic.sum(axis=1), graph)
signal.plot(edge_color='grayscale', colorbar=False)
plt.figure()
plt.hist(U_deterministic.sum(axis=1))

```

Listing 4: Instructions to generate Figure 2.5, using the same graph and \mathbf{U} objects as in Listing 3.

$\mathbf{L} = \mathbf{U}\mathbf{A}\mathbf{U}^T$, we consider the linear transformation

$$\hat{\mathbf{H}} = \hat{h}(\mathbf{L}) = \mathbf{U}\hat{h}(\mathbf{\Lambda})\mathbf{U}^T = \mathbf{U} \begin{bmatrix} \hat{h}(\lambda_0) & & & \\ & \hat{h}(\lambda_1) & & \\ & & \dots & \\ & & & \hat{h}(\lambda_{N-1}) \end{bmatrix} \mathbf{U}^T.$$

The action of $\hat{\mathbf{H}}$ on a graph signal \mathbf{x} is to apply successively the matrices \mathbf{U}^T , $\hat{h}(\mathbf{\Lambda})$, \mathbf{U} to \mathbf{x} . These linear transformations represent, respectively, a change of basis from x to the frequency-domain $\hat{x}(\lambda_l)$, a frequency-wise multiplication of each $\hat{x}(\lambda_l)$ by the filter gain $\hat{h}(\lambda_l)$, and a change of basis back to the graph domain. In Figure 2.6 we show an example.

2.3.2.2 Convolution

The algebraic definition of convolution in the Euclidean domain relies on its intrinsic regularity. The definition of a convolution aims to answer the question of what is the response of a time-invariant linear system given an input signal; however, there's no graph signal processing analogy of time-invariance, since there is no notion of translation.

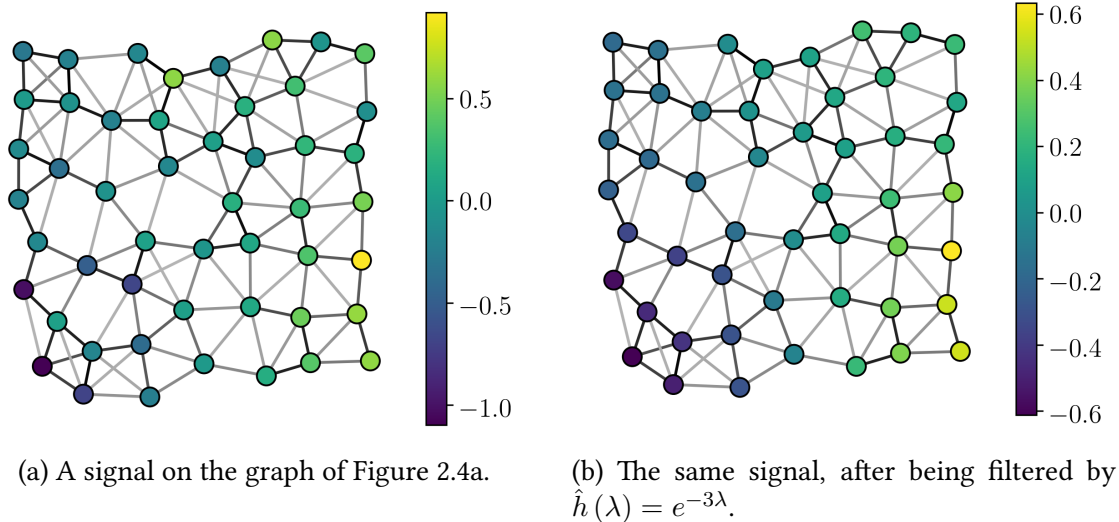


Figure 2.6: An example of frequency-domain filtering.

Still, convolution can be defined by generalizing the multiplicative property in the frequency-domain. It must be noted, however, that this is dependent on the directions chosen for the eigenvectors, as mentioned above.

Another notion close to the idea of convolution is modulation. In Euclidean domains, modulation is dual to convolution in the sense that while convolution in time translates as pointwise multiplication in the frequency domain, modulation, being represented by pointwise multiplication in the time-domain, translates as convolution in the frequency domain. In [12] it is shown that if a graph signal with a prevalence of low frequencies (usually called a “low-pass signal” in some contexts, since its graph Fourier transform is supported near zero) is pointwise multiplied (in the graph domain) by a Laplacian eigenvector \mathbf{u}_l , the Fourier transform of the resulting vector is supported near λ_l .

2.3.3 Wavelets

Multiple authors have defined wavelet transforms and multiresolution representations on graphs [13–16]. In many cases, designing wavelets on graphs boils down to defining, for each node and each scale, a wavelet signal “centered” on that node. This can be done, e.g., by designing a frequency-domain filter for each scale, and defining the wavelets to be filtered impulses, as in [16]. Alternatively, the design can be done in the graph vertex domain, choosing signal sample values directly based on known wavelet values (e.g., the “mexican hat” wavelet [17]) and the distance from each vertex to the one on which the wavelet is to be centered. Here, “distance” is to be defined by the application.

In Figure 2.7, we can see an example of wavelets generated by the Spectral Graph Wavelet Transform (SGWT), a set of wavelets designed on the frequency domain, as proposed on [16]. On the graph we have been using as example, we plot the scaling

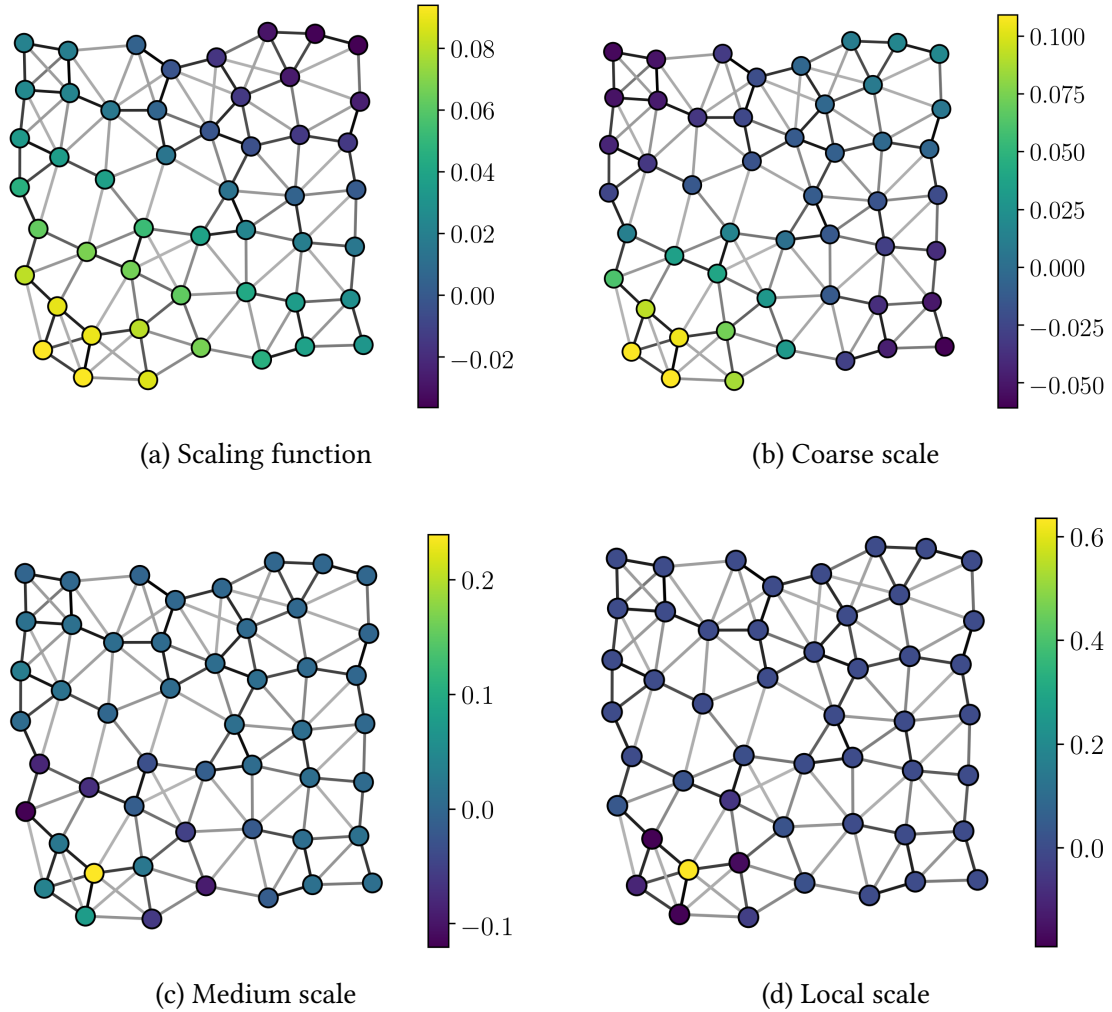


Figure 2.7: Wavelet examples.

function and three scale levels of the wavelet centered around a given node. In this section, we give a brief description of the SGWT transform.

2.3.3.1 Review of classical wavelet theory

In the classical theory of wavelets, one possible strategy for constructing a wavelet transform is to define a “mother” wavelet signal ψ , and define the convolution of a given signal x with different versions ψ_s of the mother wavelet, one for each scale. For instance, if $\psi(t)$ is the signal that represents the “mexican hat” wavelet, then we define

$$\psi_s(t) = \frac{1}{s} \psi\left(\frac{t}{s}\right)$$

and then, for each scale factor $s > 0$, we consider the convolution $\psi_s * x$. The collection of signals $\{\psi_s * x \mid s > 0\}$ is the first ingredient of the wavelet transform of the signal x . If $\hat{\psi}$ and \hat{x} are the Fourier transforms of the signals ψ and x , the Fourier transform of the signal $y_s = \psi_s * x$ is given by $\hat{y}_s(\xi) = \hat{\psi}(s\xi) \hat{x}(\xi)$.

Since the objective of each scaled wavelet ψ_s is to extract features of an original signal x at a specific resolution (small scale factors s yielding fine resolutions, and bigger factors s yielding coarser resolutions), the mother wavelet function ψ is designed to behave in a band-pass manner in the frequency domain. In this way, if the energy of the frequency representation $\hat{\psi}(\xi)$ is concentrated around $|\xi| = 1$, then the energy of $\hat{\psi}(s\xi)$ will be concentrated around $|\xi| = 1/s$, and that's why the parameter s controls the resolution of the transform: it controls which frequency regions will be filtered away (those far from $1/s$) and which ones will be present in $\psi_s * x$ (those close to $|\xi| = 1/s$).

Because the mother wavelet ψ is chosen to act as a bandpass filter, it has no component at $\xi = 0$, that is, $\hat{\psi}(0) = 0$. This means that the information about the mean value of x gets lost in each of the convolutions mentioned above. To solve this problem, we define another signal φ , called the scaling function of the wavelet transform, which acts as a low-pass filter. The convolution $y_0 = \varphi * x$ therefore encodes the information that was lost in the convolutions with ψ_s , and the complete wavelet transform of the signal x is the set of signals $\{\psi_s * x, \varphi * x\} = \{y_s \mid s \geq 0\}$. In this last notation, we write $y_s = \psi_s$ for $s > 0$, and $y_0 = \varphi$.

2.3.3.2 The SGWT

Wavelets defined on euclidean spaces, as described above, can be designed on the time-domain—i.e., the functions ψ and φ can be designed directly, and their Fourier transforms can be calculated afterwards. As another possibility, they can be designed in the frequency domain—in which case we first define $\hat{\psi}$ and $\hat{\varphi}$, and then calculate their inverse transforms as needed. For wavelets defined on graphs, the graph-domain formula “ $(1/s)\psi(t/s)$ ” is hard to work with because of the arithmetic operation t/s ; now that the time-domain has become the graph-domain, t represents a graph node instead of a time instant, and it is not clear what t/s should mean.

Defining a graph wavelet transform in the graph domain has been done; Crovella and Kolaczyk [14] succeeded in doing so. For each graph node t_0 used as the center of the convolution

$$\psi_s * x(t_0) = \frac{1}{s} \int \psi\left(\frac{t_0 - t}{s}\right) x(t) dt,$$

they interpret the factor $t_0 - t$ as the geodesic distance on the graph from node t_0 to node t , thus being able to carry out the needed arithmetic. However, we will describe here the approach used by Hammond et al. which define a graph wavelet transform designed directly on the frequency domain.

The frequency domain is easier to work with, since the graph frequencies are real numbers, and therefore the arithmetic in the formula $\hat{\psi}(s\xi)\hat{x}(\xi)$ can be carried out directly. Following the classical description given above, the definition of the SGWT is

based on two mother functions:

$$\begin{aligned} g &: \mathbb{R}^+ \rightarrow \mathbb{R} \\ &\lambda \mapsto g(\lambda); \text{ and} \\ h &: \mathbb{R}^+ \rightarrow \mathbb{R} \\ &\lambda \mapsto h(\lambda). \end{aligned}$$

The function g takes the place of ψ : it satisfies $g(0) = g(\infty) = 0$, showing a band-pass behaviour. Meanwhile, the function h is a lowpass filter, decreasing its value from $h(0) > 0$ to $h(\infty) = 0$. In Figures 2.8a and 2.8b, we show the mother functions g and h proposed in [16]. Having defined the functions g and h , we proceed to defining, for a given graph signal \mathbf{x} , the functions \mathbf{y}_s which will comprise its SGWT transform according to the considerations of the previous section. By mimicking those previous formulas, we define \mathbf{y}_s on the frequency domain:

$$\begin{aligned} \hat{\mathbf{y}}_s(\lambda_l) &= g(s\lambda_l) \hat{\mathbf{x}}(\lambda_l) \\ \hat{\mathbf{y}}_0(\lambda_l) &= h(\lambda_l) \hat{\mathbf{x}}(\lambda_l) \end{aligned}$$

Using the eigendecomposition of the graph Laplacian, $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, we define

$$g(s\mathbf{\Lambda}) = \begin{bmatrix} g(s\lambda_0) & & & \\ & g(s\lambda_1) & & \\ & & \ddots & \\ & & & g(s\lambda_{N-1}) \end{bmatrix},$$

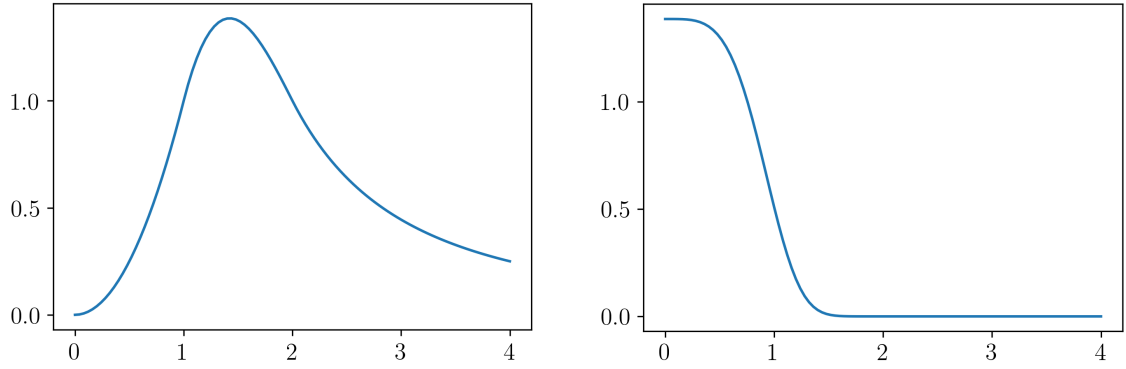
so that we can write directly in vector notation:

$$\begin{aligned} \hat{\mathbf{y}}_s &= g(s\mathbf{\Lambda}) \hat{\mathbf{x}} \\ \hat{\mathbf{y}}_0 &= h(\mathbf{\Lambda}) \hat{\mathbf{x}}. \end{aligned}$$

Using the Fourier transform and inverse transform matrices \mathbf{U}^T and \mathbf{U} , we can write those equations in the graph domain:

$$\begin{aligned} \mathbf{y}_s &= \mathbf{U}g(s\mathbf{\Lambda})\mathbf{U}^T \mathbf{x} = g(s\mathbf{L}) \mathbf{x} \\ \mathbf{y}_0 &= \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^T \mathbf{x} = h(\mathbf{L}) \mathbf{x}. \end{aligned} \tag{2.1}$$

We have thus described the SGWT transform in terms of the linear transformations defined by $g(s\mathbf{L}) = \mathbf{U}g(s\mathbf{\Lambda})\mathbf{U}^T$ and $h(\mathbf{L}) = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^T$. Given a graph signal $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$,



(a) g -function for defining bandpass wavelet signals. Designed so that $g(1) = g(2) = 1$, and $g(0) = 0$.

(b) h -function for defining (lowpass) scaling functions. Designed to have the same maximum value as g .

Figure 2.8: Mother functions that generate an SGWT.

the signals y_s and y_0 defined above comprise the SGWT transform of \mathbf{x} . A simple way of interpreting this transform is as follows. The value $y_s(i)$ of the signal y_s at the i -th node of the graph is given by the inner product between \mathbf{x} and $g(s\mathbf{L})\boldsymbol{\delta}_i$, where $\boldsymbol{\delta}_i$ is the impulse vector at the i -th node, and similarly for $y_0(i)$. This means that the action of g and h on any graph vector \mathbf{x} can be completely specified by the graph vectors $\{g(s\mathbf{L})\boldsymbol{\delta}_i, h(\mathbf{L})\boldsymbol{\delta}_i \mid s > 0\}$.

The final step in defining the SGWT for a given graph is choosing a discrete set of scales. As defined above, the SGWT transform of a signal \mathbf{x} is an infinite set $\{y_s\}$, since s can be any non-negative real number. By choosing a set of scales $\{s_1, \dots, s_J\}$, the SGWT transform of a signal \mathbf{x} becomes the set of $J + 1$ signals $\{y_0, y_1, \dots, y_J\}$, where we have written y_i in place of y_{s_i} in order to simplify the notation. In Figure 2.9 we show an example of four filters $g(s\lambda)$, one for each of four scales s , and another filter $h(\lambda)$, which together define an SGWT transform on a graph.

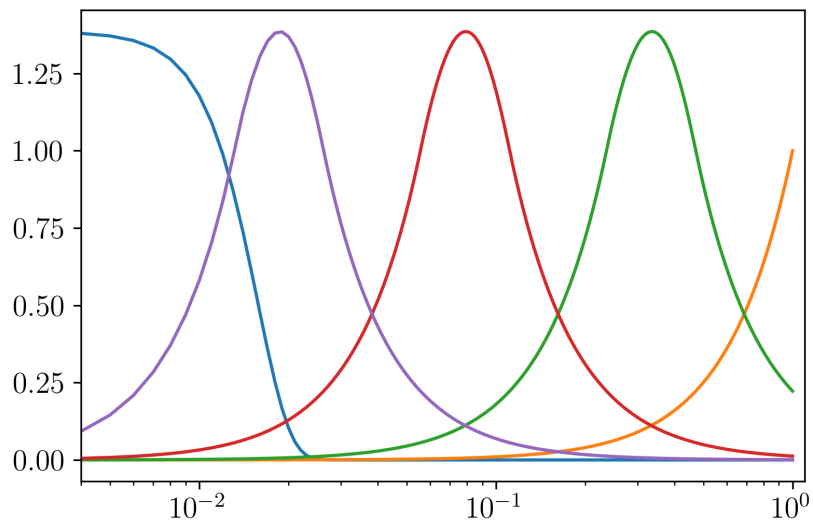


Figure 2.9: An example of a set of generating functions for a Spectral Graph Wavelet Transform. The scales are uniformly spaced in the log-domain, hence the log-scale in the horizontal axis. In order to handle graphs with edges normalized for $\lambda_{\max} = 1$, the smallest (finest) scale, s_J , shown in yellow (rightmost curve), has been chosen so that $g(s_J \lambda_{\max}) = 1$; that is, $s_J = 1$.

Chapter 3

The Least Mean Squares algorithm

In applications such as networks of sensors, time-varying graph signals can represent the dynamic nature of the information recorded by such sensors, while at the same time respecting the irregular structure of the domain. In this chapter, we study time-varying graph signals, and in particular the estimation algorithms described in [18] and [19].

The simplest way to pose the problem is as follows. The goal is to estimate a signal \mathbf{x}_0 on a graph $\mathcal{G} = (\mathcal{V}, \mathbf{W})$, and the information available is a sequence of noisy measurements $\mathbf{x}_0 + \mathbf{v}[n]$. The signal \mathbf{x}_0 is assumed to be band-limited to a set \mathcal{F} of the graph frequencies, and \mathcal{F} is assumed to be known. It will be shown that, because the desired signal is band-limited, it is enough to sample \mathbf{x}_0 on a subset $\mathcal{S} \subset \mathcal{V}$ of the graph vertices.

In the network-of-sensors example, the assumptions above can be justified: if the quantity measured by the sensors is a smooth function of the sensor position, then adjacent vertices on the graph will correspond to geographically close sensors, and therefore the values of \mathbf{x}_0 on such vertices will be similar. In this case, it is reasonable to assume that only low graph-frequencies are present in \mathbf{x}_0 . It is also reasonable to assume that recording a sample value from a sensor requires power, and therefore that we would like to minimize the amount of sensor readings at each time instant. This explains why it is desirable to sample \mathbf{x}_0 only in the vertices belonging to some small subset \mathcal{S} .

An example application of the LMS algorithm to networks of sensors is [20], in which the authors apply the method described here to estimating temperatures in some of Brazil's weather stations. Although recovering information that was lost, or would be expensive to measure directly, is the canonical application of this technique, [18] cites others such as increasing situational awareness in cognitive radio networks with known reliability.

3.1 The algorithm

As seen in Chapter 2, the main difficulty in translating signal processing techniques and algorithms is to pose or rewrite the technique in such a way that a generalization becomes apparent. This is true of filtering—which, as we’ve seen, are more easily handled in the frequency domain—and even more so when dealing with adaptive filters, which rely heavily on the time-like uniformity of the signal domains. From what we have gathered from a small literature review [18, 19], the path for designing adaptive filtering techniques for graph signal processing is not clear. Since the field of signal processing on graphs is still incipient, and in particular the workforce of trying to derive adaptive filtering algorithms, most of the work is ad hoc in nature, devoid of a pattern that could be inferred as a possible answer to the question “How to develop adaptive filtering technique for graph signal processing?”

The algorithm that will be described in this chapter, for instance, bypasses some of the difficulty of dealing with the transformation of time domains to graph domains by working with both at once. This seems to be a common theme: to consider a linear and uniform time dimension, and model the spatial dimension using graphs. In this way, the theory of graph signal processing can be applied since each sample in time—when thinking in terms of classical adaptive filtering—has now become a whole graph signal. Other common themes seem to be the form of the algorithms, which remain always similar to their classical counterparts, and the theoretical analysis¹, which rely on analysing the behaviour of the mean-square error when $n \rightarrow \infty$.

We begin by defining the sparsifying operators \mathbf{D} and \mathbf{B} . Given some set $\mathcal{S} \subset \mathcal{V}$ of vertices of a graph, we define \mathbf{D} to be the diagonal matrix with ones at the entries corresponding to elements of \mathcal{S} , and zeros everywhere else. The operator \mathbf{D} is an orthogonal projection on the space of graph signals supported on \mathcal{S} . Analogously, given a set of graph frequencies $\mathcal{F} \subset \sigma(\mathbf{L})$, where $\sigma(\mathbf{L}) = \{\lambda_0, \dots, \lambda_{N-1}\}$ is the spectrum of the graph Laplacian, we define \mathbf{B} to be the orthogonal projection on the space of signals band-limited to \mathcal{F} , that is, the space spanned by $\{\mathbf{u}_l \mid l \in \mathcal{F}\}$.

The space of graph signals band-limited to \mathcal{F} , called the Paley-Wiener space for \mathcal{F} , is the space of all $\mathbf{x} \in \mathbb{C}^{\mathcal{G}}$ such that $\mathbf{B}\mathbf{x} = \mathbf{x}$. Similarly, the space of signals supported on \mathcal{S} is the space all \mathbf{x} satisfying $\mathbf{D}\mathbf{x} = \mathbf{x}$.

It is shown in [21] that such spaces can admit non-trivial intersection. In particular, we have the following theorem.

Theorem 3.1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathbf{W})$, a subset of vertices $\mathcal{S} \subset \mathcal{V}$, and a subset of frequencies $\mathcal{F} \subset \sigma(\mathbf{L})$, together with the corresponding matrices \mathbf{D} and \mathbf{B} , the following conditions are equivalent:*

¹Means square error and steady-state performance analysis will not be a subject of this text.

1. *There is a non-zero graph signal $\mathbf{x} \in \mathbb{C}^G$ that's perfectly localized both over vertex set \mathcal{S} and over frequency set \mathcal{F} (that is, $\mathbf{x} = \mathbf{B}\mathbf{x} = \mathbf{D}\mathbf{x}$);*
2. *\mathbf{BDB} has an eigenvalue equal to 1;*
3. *\mathbf{DBD} has an eigenvalue equal to 1;*
4. *$\|\mathbf{BDB}\| = 1$;*
5. *$\|\mathbf{DBD}\| = 1$;*
6. *\mathbf{BD} has an eigenvalue equal to 1;*
7. *\mathbf{DB} has an eigenvalue equal to 1;*
8. *$\|\mathbf{DB}\| = 1$;*
9. *$\|\mathbf{BD}\| = 1$.*

If either of them holds, and therefore all others also do, then the eigenvector corresponding to eigenvalue 1 can be taken to be the same in all matrices above, and represents a signal x satisfying $\mathbf{B}\mathbf{x} = \mathbf{D}\mathbf{x} = \mathbf{x}$.

Since \mathbf{D} and \mathbf{B} are orthogonal projections, they always satisfy $\|\mathbf{D}\| = \|\mathbf{B}\| = 1$, and so all their products satisfy $\|\mathbf{DB}\| \leq \|\mathbf{D}\| \cdot \|\mathbf{B}\| = 1$. If such products have eigenvalues equal to 1, then they also have norm 1, and the corresponding eigenvectors \mathbf{x} satisfy $\mathbf{DB}\mathbf{x} = \mathbf{x}$. This can be written as

$$\mathbf{x} \mapsto \mathbf{B}\mathbf{x} \mapsto \mathbf{DB}\mathbf{x} = \mathbf{x}.$$

Since each step in this chain is an orthogonal projection, orthogonal projections can't increase the norm of a vector, and the final output has the same norm as the initial input, each step must preserve the norm of its input. Therefore, each step, being an orthogonal projection, must preserve the actual input vector, that is, $\mathbf{B}\mathbf{x} = \mathbf{x}$ and $\mathbf{D}\mathbf{x} = \mathbf{x}$.

This phenomenon of perfect localization simultaneously in the vertex and frequency domains does not happen in classical Fourier analysis. In fact, the Paley-Wiener theorem [22], [23] places very restrictive conditions on perfectly-localized, continuous-time, non-periodic signals:

Theorem 3.2. (Paley-Wiener)

Let $f \in L^2(\mathbb{R})$ be a square-integrable, complex-valued function on the real line. Then its Fourier transform \hat{f} is supported on the interval $[-M, M]$ if and only if f can be extended to an entire function on \mathbb{C} such that $|f(x + iy)| = O(e^{M|y|})$.

We recall that an entire function is a function $f : \mathbb{C} \rightarrow \mathbb{C}$ that can be written as a power series that converges everywhere on the complex plane:

$$f(z) = \sum_{n=0}^{\infty} a_n z^n, \quad z \in \mathbb{C}.$$

Of course, by duality the theorem is also valid the other way around: if f is supported in a bounded set, then \hat{f} can be extended to an entire function. Since entire functions can never vanish infinitely many times in a bounded set (let alone vanish in a non-degenerate interval), no continuous-time, non-periodic signal f can be perfectly localized (i.e. supported on a bounded set) in both time and frequency.

In the graph domain, then, the LMS approach gives rise to the following algorithm. Let $\mathbf{y}[n]$ be the signal measured on the vertices \mathcal{S} :

$$\mathbf{y}[n] = \mathbf{D}(\mathbf{x}_0 + \mathbf{v}[n]).$$

Then, given \mathbf{y} , to solve the problem of estimating \mathbf{x}_0 we just need to find the solution to the following optimization problem:

$$\begin{aligned} \min \quad & \mathbb{E}[\|\mathbf{y}[n] - \mathbf{D}\mathbf{B}\mathbf{x}\|^2] \\ \text{s.t.} \quad & \mathbf{B}\mathbf{x} = \mathbf{x} \end{aligned}$$

The squared error can be rewritten as:

$$e^2 = \mathbb{E}[\|\mathbf{y}[n] - \mathbf{D}\mathbf{B}\mathbf{x}\|^2] = \|\mathbb{E}[\mathbf{y}[n]] - \mathbf{D}\mathbf{B}\mathbf{x}\|^2,$$

and then we use the instantaneous error to calculate the gradient, as done in [24]:

$$\nabla_{\mathbf{x}} e^2 = 2\mathbf{B}\mathbf{D}(\mathbf{B}\mathbf{x} - \mathbf{y}[n]).$$

This gradient does not disrupt the condition $\mathbf{B}\mathbf{x} = \mathbf{x}$, which means that if the initial guess $\mathbf{x}[n]$ is already band-limited to \mathcal{F} , applying the algorithm is quite simple:

$$\mathbf{x}[n+1] = \mathbf{x}[n] + \mu\mathbf{B}\mathbf{D}(\mathbf{y}[n] - \mathbf{x}[n]).$$

Let $\bar{\mathcal{S}} = \mathcal{V} \setminus \mathcal{S}$ be the set complement of \mathcal{S} with respect to \mathcal{V} . In order for the original signal \mathbf{x}_0 to actually *be* recoverable from its samples on \mathcal{S} , there must not exist any \mathcal{F} -limited signals that are also perfectly localized on $\bar{\mathcal{S}}$ —if such signals exist, they will be sampled as 0, and \mathbf{D} would lose all information on them. If we let $\bar{\mathbf{D}} = \mathbf{I} - \mathbf{D}$ be the sampling matrix for the set $\bar{\mathcal{S}}$, then the following theorem from [18] summarizes this situation:

Theorem 3.3. *The LMS optimization problem above admits a unique solution (i.e., any \mathcal{F} -limited signal \mathbf{x}_0 can be recovered from its \mathcal{S} -samples $\mathbf{D}\mathbf{x}_0$), if and only if the sets $\bar{\mathcal{S}}$ and \mathcal{F} do not satisfy the conditions of Theorem 3.1.*

A corollary of this theorem is that a necessary condition for the solvability of the reconstruction problem is $|\mathcal{S}| \geq |\mathcal{F}|$. This is required also from the perspective of degrees of freedom: since we need to find all coefficients \hat{x}_l of the graph Fourier transform of \mathbf{x}_0 for $l \in \mathcal{F}$, through a map with domain $\{\mathbf{x} \mid \mathbf{D}\mathbf{x} = \mathbf{y}\}$, the dimensionality $|\mathcal{S}|$ should be at least as large as that of the image of the map.

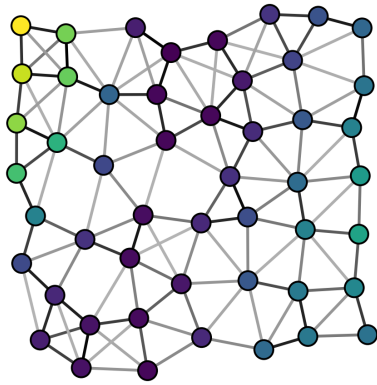
While requiring $|\mathcal{S}| \geq |\mathcal{F}|$ is necessary, it is not sufficient. Even though the condition on Theorem 3.3 might be satisfied, the map from input \mathbf{y} to the estimate \mathbf{x} might be ill-conditioned. We describe here one of the strategies proposed in [18] for choosing the set \mathcal{S} given \mathcal{F} .

Let $\mathbf{U}_{\mathcal{F}}$ be the $N \times |\mathcal{F}|$ matrix with columns \mathbf{u}_l for l in \mathcal{F} , where N is the number of nodes in the graph. Such \mathbf{u}_l form an orthonormal basis for the set of allowed (i.e., \mathcal{F} -limited) signals \mathbf{x}_0 . Since each line of \mathbf{U} (and therefore each line of $\mathbf{U}_{\mathcal{F}}$) corresponds to a graph node (in the order of the labeling of nodes, as discussed in Section 2.1.1), when we sample the measured signals through the sparsifying operator \mathbf{D} , we focus on the lines of \mathbf{U} corresponding to the vertices of \mathcal{S} . Knowing this, the strategy for choosing an optimal set \mathcal{S} is to make these $|\mathcal{S}|$ lines (of length $|\mathcal{F}|$ each) as independent from each other as possible. In other words, these lines must span the subspace of \mathcal{S} -localized vectors, and must do so as stably as possible.

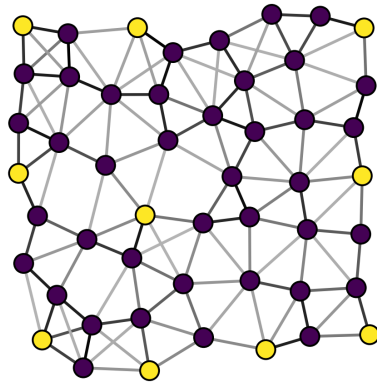
The algorithm then consists in a greedy search for the optimal \mathcal{S} . In a first step, we choose the line in $\mathbf{U}_{\mathcal{F}}$ with highest norm, and call it \mathbf{l}_0 . Then, we choose the line whose component orthogonal to \mathbf{l}_0 has highest norm and call it \mathbf{l}_1 . Inductively, if we have already chosen $\mathbf{l}_0, \dots, \mathbf{l}_{n-1}$, we choose \mathbf{l}_n to be the line in $\mathbf{U}_{\mathcal{F}}$ with greatest component orthogonal to span $\{\mathbf{l}_0, \dots, \mathbf{l}_{n-1}\}$. Since this is equivalent to maximizing the $(n+1)$ -dimensional volume of the parallelepiped with sheared sides $\mathbf{l}_i, i = 0, \dots, n$, it is also equivalent to maximizing $\det_+ (\mathbf{U}_{\mathcal{F}}^T \mathbf{D}^{(n)} \mathbf{U}_{\mathcal{F}})$, where $\mathbf{D}^{(n)}$ the binary diagonal matrix with ones on the entries that have been selected by the algorithm, and \det_+ is the product of the non-zero eigenvalues of a matrix.

In Figure 3.1 we plot an example of evolution of the LMS iteration. For the graph of Figure 2.4a, we generate a random vector, band-limited to the first (lowest) 10 frequencies, and we apply the greedy procedure described above to list the graph nodes in order of importance. We then apply the LMS algorithm, using as sampling set \mathcal{S} the first K nodes listed by the greedy algorithm for $K = 7, 8, 9, 10, 20, 30, 50$. For each value of K , we plot the mean square deviation, defined as

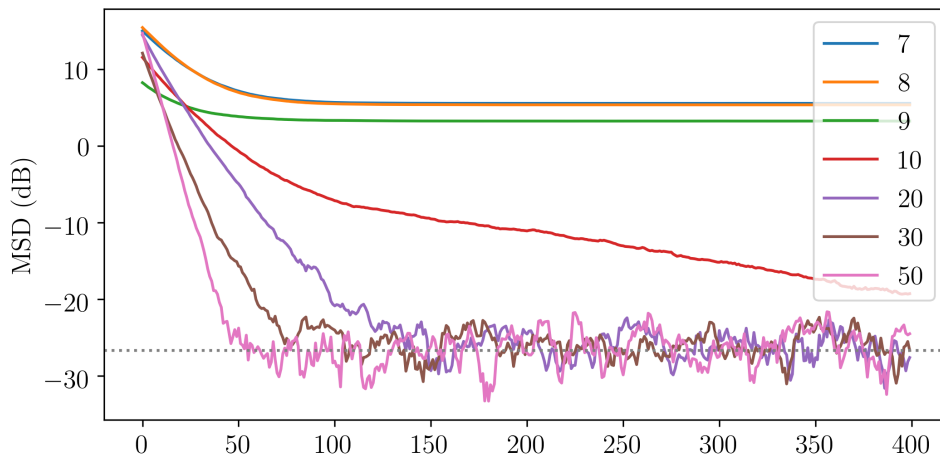
$$\text{MSD} = \|\mathbf{x}[n] - \mathbf{x}_0\|^2.$$



(a) Example of a random band-limited graph signal. Here, the band consists of the first (lowest) 10 frequencies, out of 50.



(b) 10 first samples chosen by the greedy algorithm. We can see that the chosen nodes are mostly those that the LMS iteration would have most difficulty estimating, for they are far from their neighbors, or are weakly linked to them.



(c) We run the LMS iteration for the problem illustrated in Figures 3.1a and 3.1b. Although Figure 3.1b shows only the set \mathcal{S} with exactly 10 nodes, we also plot the results for 7, 8, 9, 10, 20, 30, and 50 measured nodes. The dotted line shows the theoretical steady-state MSD.

Figure 3.1: Results of an LMS example.

It can be seen that, for $|\mathcal{S}| = K < 10 = |\mathcal{F}|$, the algorithm is not able to recover the original signal \mathbf{x}_0 . As expected from intuition, the higher the number K of sampled nodes, the higher the information available, and therefore the faster the convergence. Also, the steady-state MSD is the same for all curves, since it does not depend on the sampling set \mathcal{S} (as long as the original signal is recoverable from its samples in \mathcal{S} , as mentioned in Theorem 3.3).

3.2 Probabilistic sampling

In [19] a more general sampling method was proposed, which we discuss in this section. The motivation is that, while sampling in a limited subset of graph nodes can help reduce costs and energy consumption, consistently ignoring another large subset of the vertices can drastically reduce the convergence rate, as exemplified in Figure 3.1c. Using the method of probabilistic sampling, all graph nodes are potentially sampled, aiming to maintain a sparse sampling pattern while at the same time not losing information from important graph nodes.

In the deterministic framework described in the last section, it was important to choose carefully a set $\mathcal{S} \subset \mathcal{V}$ of graph vertices on which to measure signal values. Specifying such a set is the same as specifying a binary vector

$$\mathbf{p} = [p_0 \ p_1 \ \cdots \ p_{N-1}]^T$$

such that $p_i = 1$ if the graph node v_i belongs to \mathcal{S} , and $p_i = 0$ otherwise. In the probabilistic framework, each p_i is a probability $p_i \in [0, 1]$, and for every iteration, each graph node is sampled or not at random, independently from other graph nodes, with probability p_i . If, for some vertex v_i , p_i is zero, then this vertex is never sampled; if, on the other hand, $p_i = 1$, then v_i is sampled at each time instant.

The task of choosing a sampling set \mathcal{S} has now been transformed into the task of choosing a suitable probability vector \mathbf{p} . The same trade-offs apply here: if $\mathbf{p}^T \mathbf{1}$, the expected number of sampled nodes, is high, then the costs of measuring are high; on the other hand, if $\mathbf{p}^T \mathbf{1}$ is too low, convergence of $\mathbf{x}[n]$ to the original signal \mathbf{x}_0 will not take place.

Since the domain $\mathbf{p} \in [0, 1]^N$ is now continuous, in contrast to the discrete domain $\mathbf{p} \in \{0, 1\}^N$ underlying the deterministic framework, the optimization problem we face is analytic instead of combinatorial, and there is no need for greedy algorithms anymore. The proposed strategy then is to solve an optimization problem for \mathbf{p} , trying to minimize the expected number of measurements per iteration, $\mathbf{p}^T \mathbf{1}$. The two most trivial constraints are $\mathbf{0} \leq \mathbf{p} \leq \mathbf{1}$. More generally, we can specify a vector \mathbf{p}_{\max} of energy or resource limitations on each node, and require $\mathbf{0} \leq \mathbf{p} \leq \mathbf{p}_{\max}$. Unless other con-

strains are imposed, $\mathbf{p} = \mathbf{0}$ would minimize $\mathbf{p}^T \mathbf{1}$ trivially. Therefore, constraints on the convergence rate and on the steady-state MSD value are usually present.

Chapter 4

Further exploratory results

In this chapter, we present the results of two exploratory experiments around the theme of signal processing on graphs. In Section 4.1 we attempt to extend the spectral graph wavelet transform (SGWT) proposed in [16] to allow for a critically sampled wavelet transform as used in uniform euclidean domains [25]. In Section 4.2 we examine the trade-offs of less greedy algorithms for choosing sampling sets \mathcal{S} in the LMS iteration described in Chapter 3.

4.1 Sparse wavelet representation

The Spectral Graph Wavelet Transform described in Section 2.3.3 has one important drawback, which is the complexity of the inversion problem. The product of the SGWT is the collection of signals $\mathbf{y}_1, \dots, \mathbf{y}_J, \mathbf{y}_0$ —results of inner products between the original graph signal \mathbf{x} and all J scaled wavelet functions, plus the scaling function, centered at each vertex. (See equations (2.1).)

The problem of recovering \mathbf{x} from $\mathbf{y}_1, \dots, \mathbf{y}_J, \mathbf{y}_0$ is solved by a least-squares procedure. The wavelet signals are gathered in a matrix

$$\mathbf{\Gamma} = \left[\begin{array}{c|c|c} g(s_1\mathbf{L})\boldsymbol{\delta}_1 & \cdots & g(s_1\mathbf{L})\boldsymbol{\delta}_N \\ \cdots & & \cdots \\ g(s_J\mathbf{L})\boldsymbol{\delta}_1 & \cdots & g(s_J\mathbf{L})\boldsymbol{\delta}_N \\ \hline h(\mathbf{L})\boldsymbol{\delta}_1 & \cdots & h(\mathbf{L})\boldsymbol{\delta}_N \end{array} \right]$$

where $\mathbf{\Gamma} \in \mathbb{R}^{N \times (J+1)N}$. The solution to the inverse problem is the solution to the normal equations:

$$\mathbf{\Gamma}\mathbf{\Gamma}^T\mathbf{x} = \mathbf{\Gamma} \left[\mathbf{y}_1^T \quad \cdots \quad \mathbf{y}_J^T \quad \mathbf{y}_0^T \right]^T.$$

The size of the matrix $\mathbf{\Gamma}$ reveals a trade-off: the more columns $\mathbf{\Gamma}$ has, the higher the complexity of the inverse problem. The number of columns $(J+1)N$ means that the transform of a signal \mathbf{x} with N samples is a collection of signals with a total of $(J+1)N$ coefficients. Besides the complexity of the inverse problem, there is a storage problem: the SGWT transform may reveal a lot of information about the multi-scale properties of

a signal, but this comes at the price of needing $J + 1$ times the memory needed to store the graph-domain signal.

In the other hand, subsampling this supercomplete transform is not trivial: the less coefficients of the transform are stored, the higher the condition number of Γ , which controls the stability of the inverse problem. It is possible to recover \mathbf{x} from a subset of the SGWT coefficients, but if too many coefficients are dismissed, the inverse problem might become unstable.

4.1.1 Proposed algorithm

In this section, we propose a method for choosing a subset of the transform coefficients in a way reminiscent of critical sampling for filterbanks. In applications of multirate systems, one also decimates the different versions of a signal, filtered to represent different scales, at fine-tuned rates, with the objective of obtaining a representation from which a perfect reconstruction can be obtained while not containing redundancy of information.

The heuristic for subsampling the SGWT transform will be the following. Our objective will be to dismiss JN of the $(J + 1)N$ coefficients in such a way as to obtain a linear transform with N coefficients—represented in the graph-vertex domain by a matrix in $\mathbb{R}^{N \times N}$ instead of $\mathbb{R}^{N \times (J+1)N}$ —but which extracts information from all scales, having a stable inverse. As also happens in multirate systems in euclidean 1D domain [25], coarse-scale coefficients vary very smoothly, and this redundancy can be used to reconstruct the signal from just a few samples. Meanwhile, fine-scale coefficients represent very localized features, and therefore vary more quickly. In other words, the higher the scale s_i , the lower will be the number of coefficients needed to represent y_i .

Inspired by dyadic filterbanks, our first experiment is to sample coefficients of the SGWT transform in the following way. We initially suppose that the number N of vertices in the graph is a power of two, $N = 2^K$, and assign 2^{K-1} vertices to the finest scale (s_1), 2^{K-2} of the unused vertices to the second-finest scale, and so on.

We begin by partitioning the set of vertices in two equal parts (each with 2^{K-1} vertices), in the most uniform manner (in the graph domain) as possible. In particular, if the graph is bipartite¹, the partitioning must preserve this structure. How this is done will be specified shortly. In Figure 4.1a we can see an example. Given the partition, we choose one of the vertex sets to be the set on which the finest-scale wavelet coefficients will be sampled. The remaining set will be used for the remaining scales (and scaling function). As to the remaining 2^{K-1} vertices, we partition them further in two equal parts, and assign one of the parts to be used as vertices for sampling the next-finest

¹A bipartite graph is a graph with a vertex set \mathcal{V} which can be partitioned in two subsets $\mathcal{V} = \mathcal{V}_0 \cup \mathcal{V}_1$, $\mathcal{V}_0 \cap \mathcal{V}_1 = \emptyset$ in such a way that, if v is any vertex in \mathcal{V}_0 , then all of the vertices adjacent to v belong to \mathcal{V}_1 , and vice-versa.

scale, s_2 . The remaining 2^{K-2} vertices will be partitioned in two once more, and so on until we have 2^{K-J} vertices left. By then, we will have assigned vertices to all scales (s_1 having received 2^{K-1} vertices and s_J received 2^{K-J}), and the 2^{K-J} remaining can be assigned to the scaling function. Algorithm 1 summarizes the procedure, and Figure 4.1 illustrates it.

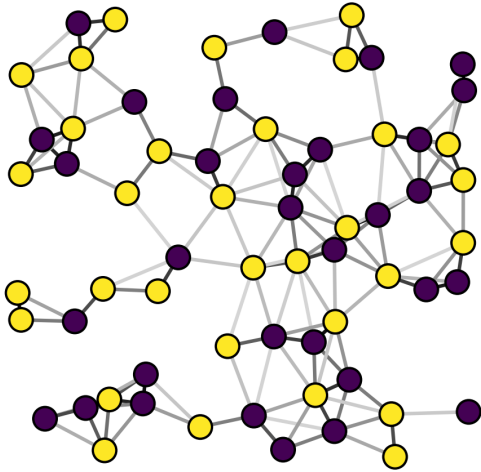
Algorithm 1: Procedure for subsampling the SGWT transform

```

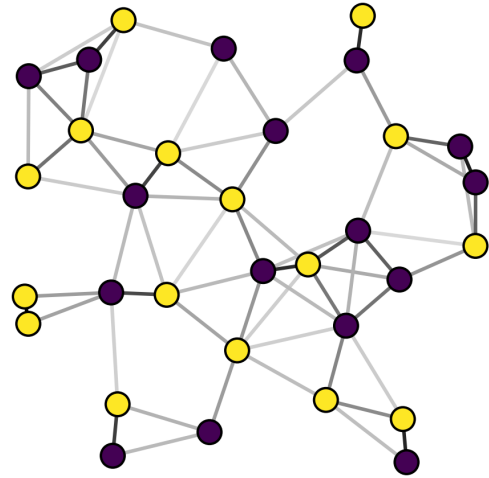
Let  $\mathcal{G}$  be a graph with  $N = 2^K$  nodes
Let  $J \leq K$  be the number of scales
Label all  $2^K$  vertices as unassigned
Let  $\Gamma$  be a real matrix, initially empty with size  $N \times 0$ 
for  $i = 1 \dots J$  do
    Let  $\mathcal{U}_i$  be the set of unassigned vertices
    Comment:  $\mathcal{U}_i$  has  $2^{K-i+1}$  elements
    Sample  $2^{K-i}$  elements from  $\mathcal{U}_i$ , uniformly spaced in the graph domain
    foreach node  $v$  which was sampled in the above step do
        Let  $\ell$  be the index of  $v$  in the graph  $\mathcal{G}$ 
        Assign vertex  $v$  to the scale  $s_i$ 
        Add the vector  $g(s_i \mathbf{L}) \boldsymbol{\delta}_\ell$  as a column to  $\Gamma$ 
        Label  $v$  as assigned
    end
end
Let  $\mathcal{U}_0$  be the set of unassigned vertices
Comment:  $\mathcal{U}_0$  has  $2^{K-J}$  elements
foreach  $v \in \mathcal{U}_0$  do
    Let  $\ell$  be the index of  $v$  in the graph  $\mathcal{G}$ 
    Assign  $v$  to the scaling function
    Add the vector  $h(\mathbf{L}) \boldsymbol{\delta}_\ell$  as a column to  $\Gamma$ 
    Label  $v$  as assigned
end

```

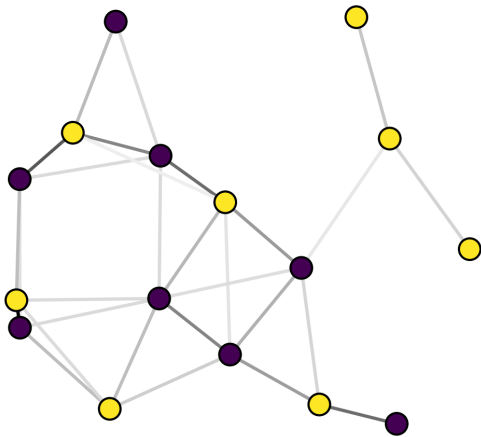
It still remains to be specified the procedure for choosing the nodes at each step of the algorithm. One possible procedure is the recursive spectral bisection (RSB), described in [26]. It consists in using the polarity of the graph Fiedler vector \mathbf{u}_0 —the graph Laplacian eigenvector associated to the least non-zero eigenvalue—to recursively bisect the graph in two. This has the disadvantage of generating a set of “samples” which do not correspond to actual nodes in the original graph, but only areas (sets of nodes). An idea dual to the RSB is [27] to use the polarity of the eigenvector associated to the *highest* eigenvalue, \mathbf{u}_{\max} . This is what we use: we sort the nodes according to their value for



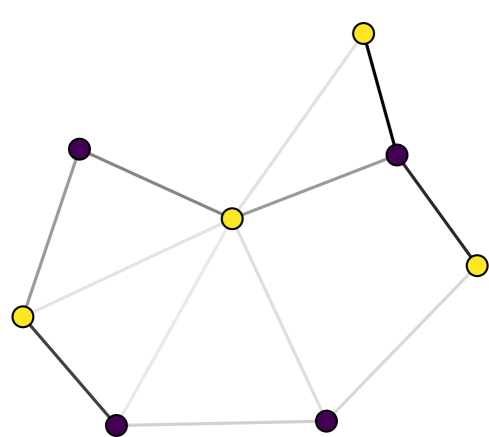
(a) First pass of the algorithm. The nodes shown in dark purple are assigned to the first (finest) scale, s_1 , while the nodes shown in bright yellow stay for the next pass.



(b) Second pass. All nodes present in this image were colored yellow in Figure 4.1a. Nodes colored purple are assigned to s_2 , and the rest go through to the third pass.



(c) Third pass, using only nodes which were colored yellow in Figure 4.1b. The color code is the same as in the previous figures.



(d) Last pass of the algorithm. All nodes present here were colored yellow in the last figure. In this figure, nodes colored purple are assigned to the last scale, s_J , and nodes colored yellow will be used to sample the scaling function coefficients.

Figure 4.1: Example of application of Algorithm 1 in a graph with $N = 64$ vertices, using $J = 4$ scales.

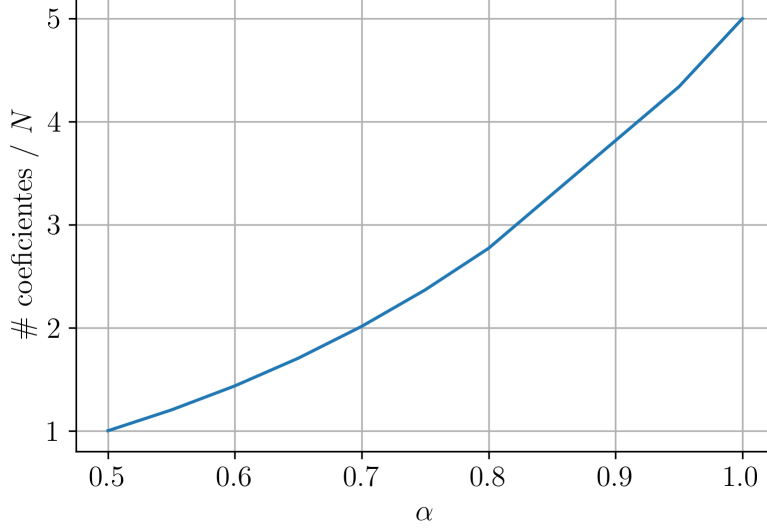


Figure 4.2: Number of coefficients of the sampled SGWT transform, for $J = 4$. Critical sampling ($\alpha = 0.5$) generates a transform with exactly $1 \cdot N = N$ coefficients, while the full transform ($\alpha = 1.0$) needs $(J + 1) N = 5N$ coefficients.

\mathbf{u}_{\max} , and then sample the first half.

The only problem with this approach is that the set of nodes that would go through for the subsequent pass is not a graph itself, but a subset of vertices of \mathcal{G} ; it is not clear how the new vector \mathbf{u}_{\max} will be chosen in the subsequent pass, unless this set of vertices becomes a graph. For example, the edges shown in Figure 4.1b are not the original edges of the graph (shown in Figure 4.1a). In order to populate the new graph with edges, we can calculate resistive distances [28], or repopulate the edges using a kernel function as described in Chapter 2.

4.1.2 Results and stability considerations

When we apply the algorithm described above to the graph of Figure 4.1, we obtain a sampled $\Gamma \in \mathbb{R}^{N \times N}$ matrix, instead of the full $N \times (J + 1) N$. The trade-off is that the condition number of the full matrix is usually of the order of a few units (between 1 and 10), while the condition number of the sampled matrix can reach much higher orders of magnitude. For the graph shown, the condition number becomes $\kappa \approx 10^4$.

In an attempt to mitigate this problem, we can give up the critical sampling, and sample a fraction $\alpha \geq 0.5$ at each pass of the algorithm, where $\alpha = 0.5$ corresponds to critical sampling, and $\alpha = 1.0$ corresponds to the full transform. Figure 4.2 illustrates the impact of α on the size of the transform, while Figure 4.3 illustrates the impact on the conditioning of the problem.

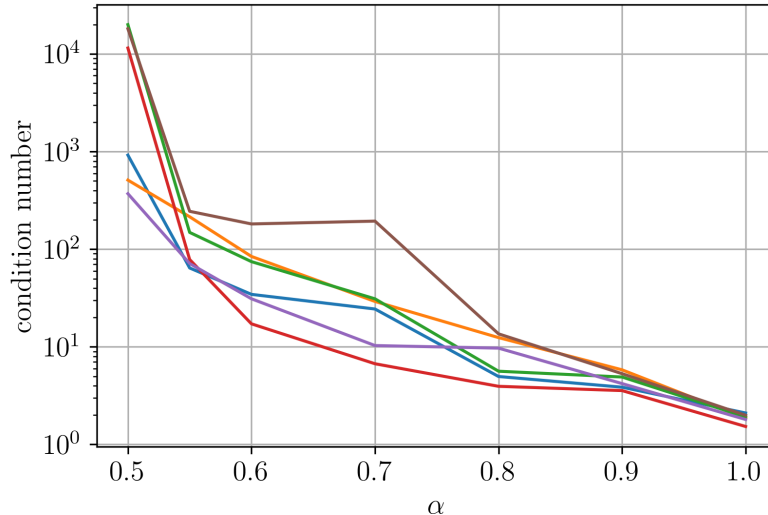


Figure 4.3: Condition number of the problem of inverting the sampled SGWT transform.

4.2 Optimal sampling for Least Mean Squares

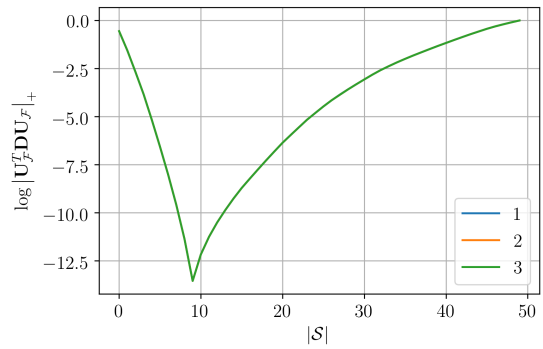
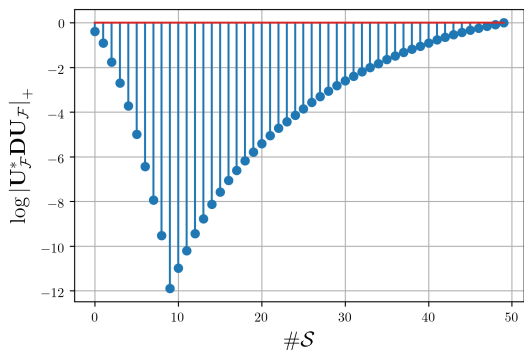
In Chapter 3, we described how the LMS algorithm proposed in [18] depends on the set of vertices on which the bandlimited signal is allowed to be sampled. The algorithm proposed in that paper is a greedy algorithm which, as already described, searches for the best sampling set by choosing nodes one at a time, and sticking to them once they have been chosen. In this section, we propose an improvement which searches for nodes two at a time, or three at a time.

A note on complexity is in order. Choosing the sampling set \mathcal{S} is a combinatorial problem, with a time complexity exponential in the number of vertices of the graph. The advantage of the greedy algorithm is that it breaks the search for the global optimum in $|\mathcal{S}|$ steps, and each step has linear time complexity. When we substitute in a search for nodes two at a time, each step becomes a search for the optimal *pair* of nodes at that step, which has quadratic time complexity. If we search for nodes three at a time, the time complexity of each step becomes cubic; and so on.

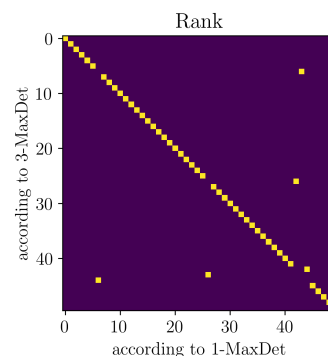
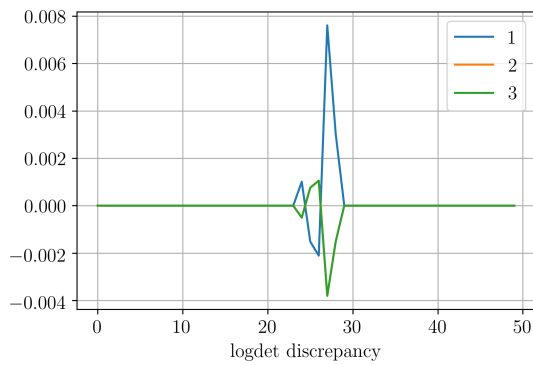
The results of this experiment are summarized in Figure 4.4. We recall that the greedy algorithm that chooses one node at each step tries to maximize the determinant $|\mathbf{U}_{\mathcal{F}}^T \mathbf{D} \mathbf{U}_{\mathcal{F}}|_+$. In Figure 4.4a we see the maximum found at each step for the graph used as example in Chapter 3. Figure 4.4b compares this with the maximum obtained by making the (less greedy) searches for nodes two at a time and three at a time. Since the differences are not visible, we plot the differences explicitly in Figure 4.4c. Analysing this figure we can infer that the difference between the results of the algorithms is that the resulting order of importance of the nodes is almost exactly the same except for a couple of nodes with interchanged positions. Indeed, when we plot a permutation matrix (Figure 4.4d), we see that all nodes have the same order of importance, except

for five of them, which appear permuted. The result is that, in this example, unless the number of vertices to be sampled was between 23 and 29 (see Fig. 4.4c), the set of vectors returned by the different algorithm would be exactly the same. If the number of vertices to be sampled was actually between 23 and 29, Fig. 4.4b indicates that the difference would be negligible.

Finally, the last two figures show the result of a Monte Carlo mean. Figure 4.4e shows the mean of 100 different realizations of Fig. 4.4c, onde for a different random graph. The faint noise in the background is not enough to distinguish this figure from an identity matrix. At last, Figure 4.4f shows the mean of 100 realizations of Fig. 4.4d.

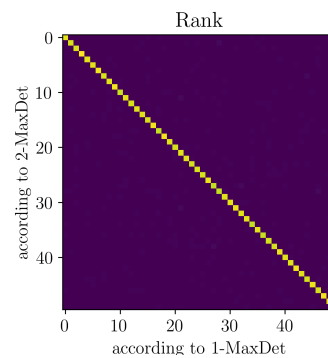
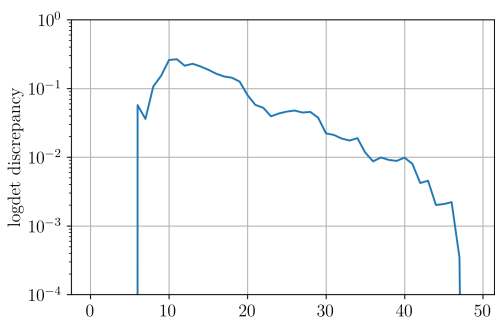


(a) Maximum log-determinant achieved at each step. This makes it clear that, in this example, each step, for the algorithms that search for we are dealing with a signal bandlimited to 10 nodes one, two, and three at a time frequencies.



(c) Discrepancy between the curves 1 and 3 from Figure 4.4b.

(d) Permutation matrix corresponding to Figure 4.4c.



(e) Monte Carlo mean of the log-determinant from Figure 4.4c

(f) Monte Carlo mean of the permutation matrix image in Figure 4.4d

Figure 4.4: Less greedy sampling for the bandlimited LMS algorithm.

Chapter 5

Conclusions

In this work, we have discussed and analyzed a framework for processing signals with irregular, non-uniform domains. Such domains, represented as graphs, are in contrast to the usual continuous Euclidean domain that is ubiquitous in analog signal processing, or the discrete domain with equally-spaced time intervals that features in digital signal processing. We presented basic tools used to deal with such signals defined on graphs in the way we would deal with signals in traditional signal processing. In particular, we defined a graph Fourier transform and examined closely its relationship to the Fourier transform in \mathbb{R}^n . We have focused on being as didactic as possible, with the goal that this work be useful as in introductory text to the field of graph signal processing.

Using the semantics of the graph Fourier transform and of graph frequencies, we discussed band-limited signals on graphs, and how they can be recovered from an incomplete set of samples. We analyzed how Paley-Wiener spaces have strikingly different behaviours in the graph domain and in the Euclidean domain—signals on graphs can be localized in both frequency and graph domains at the same time, while signals on \mathbb{R} that are localized in one domain must exhibit highly restrictive conditions on the other. The interaction between the localization properties of graph signals on the vertex and the frequency domains has helped us understand a Least Mean Squares algorithm for estimating band-limited graph signals given noisy samples.

We have shown and discussed the results of two small experiments that further investigate the ideas presented in the text, in order to raise questions and insights about the behaviour and inner working of the tools that were discussed in the text. The first experiment, regarding wavelets, shows a promising way of transforming the SGWT super-complete transform into a critically-sampled—or almost critically-sampled, depending on the conditioning-complexity trade-off chosen—transform that avoids the many-fold increase in storage needed to store the SGWT representation of a signal. In one example, although the critically sampled transform would avoid a five-fold increase in storage at the cost of multiplying the condition number of the problem by 10^4 , an almost-critically sampled transform would still exchange the $5\times$ increase for a $1.5\times$ increase, but this

time at the cost of a minor loss of 10^2 in the condition number.

In the second experiment, we attempt to enhance the sampling mechanism of the LMS for bandlimited signals. However, the gains obtained did not appear to be significant enough to justify the increase in computational complexity incurred by the enhancement.

Finally, we have developed a Python package focused on fast prototyping and data visualization. Some examples were provided in order to illustrate the use of the library for analysing the structure of graphs and graph signals.

Appendix A

List of selected routines from the graphdsp package

In this section we list some important classes and routines implemented in the graphdsp package. Instructions on how to download and install the package can be found in Chapter 1.

- Class UndirectedWeightedGraph:

- Initializer signature:

```
UndirectedWeightedGraph(  
    *, # all parameters must be provided as keyword parameters  
    W=None, L=None,  
    nodes=None,  
    copy_matrix=True)
```

Exactly one of the W or L parameters must be provided. They should contain the information about the edge weights in the form of a weight matrix, or a Laplacian matrix. The parameter nodes is an optional list of node names (any hashable python object, e.g. strings or integers). The copy_matrix parameter specifies whether to copy the W or L matrix provided into a new structure, or to reuse the memory of the provided array. Example:

```
weights = np.array([  
    [0, 1, 3],  
    [1, 0, 1],  
    [3, 1, 0],  
])  
graph = gsp.UndirectedWeightedGraph(  
    W=weights, nodes=['A', 'B', 'C'], copy_matrix=False)
```

- Property W: returns the weight matrix of the graph.

- Property L: returns the Laplacian matrix of the graph.
 - Special method `__len__`: returns the number of vertices on the graph. Example: `assert len(graph) == 3`.
 - Property nodes: returns the list of nodes of the graph.
 - Property spectrum: the list of Laplacian eigenvalues in increasing order. Calculated only once and then cached. The cache can be cleared with `del graph.spectrum`.
 - Property U: matrix of Laplacian eigenvectors.
 - Method `diag_lambda`: diagonal matrix with graph eigenvalues, in increasing order.
 - Method `spectral_decomposition`: returns a pair (lambda, U), where lambda is the list of eigentvalues in increasing order, and U is the list of eigenvectors in the same order from left to right. Example: `lambda, U = graph.spectral_decomposition()`.
 - Method `gft`: calculate the graph Fourier transform of a signal. Example: `transform = graph.gft(some_signal)`
 - Method `igft`: the same, but for the inverse transform.
- Class `GeometricUWG`. Inherits from `UndirectedWeightedGraph`. Represents a graph whose nodes are associated to points in an Euclidean space.

- Class method `make_random`: returns a randomized graph. Example:

```
make_random(
    cls, preset='default', *,
    N=50,
    lmax=1)
```

The preset parameter must be one of 'default' or 'planar'. The 'planar' option generates a planar graph. N is the number of nodes (default 50), and the graph edges will be normalized so that the largest Laplacian eigenvalue will be lmax.

- `GraphSignalBase`: base class for graph-domain and frequency-domain graph signals. This is an abstract base class, and as such cannot be instantiated directly.

- Initializer:

```
super().__init__(
    values,
    graph=None,
```

```
copy_values=False,  
from_transform=None)
```

Here, `values` is a list of signal samples; `graph` is an optional graph object representing the graph on top of which the signal is to be interpreted; `copy_values` specifies whether the values structure should be copied or its values reused; and finally `from_transform` is a transformed signal, if it is known. Examples: if `signal` is an instance of `GraphSignalBase`, then `len(signal)` returns the number of entries, and `signal[4]` returns the value of the entry number 4.

- Class `GraphSignal`. Inherits from `GraphSignalBase`. Implements one additional method:

```
plot(self, *,  
      cmap=matplotlib.cm.get_cmap('viridis'),  
      title=None,  
      colorbar=True, edge_color='k',  
      show_indexes=False, filename=None,  
      dotcolor=None, highlight=None)
```

The parameter `cmap` is a matplotlib color map; `edge_color` can be one of `'grayscale'`, `'zero-crossing'`, or a matplotlib color (see Figure 2.4); `show_indexes` specifies whether to show the integer index of each node (see Figure 2.1); `dotcolor` is a matplotlib color; and `highlight` is the index of a node which should stand out in the image. The remaining parameters are self-explanatory.

Bibliography

- [1] PUSCHEL, M., MOURA, J. M. F. “Algebraic Signal Processing Theory: Foundation and 1-D Time”, *IEEE Transactions on Signal Processing*, v. 56, n. 8, pp. 3572–3585, Aug 2008.
- [2] PUSCHEL, M., MOURA, J. M. F. “Algebraic Signal Processing Theory: 1-D Space”, *IEEE Transactions on Signal Processing*, v. 56, n. 8, pp. 3586–3599, Aug 2008.
- [3] STEIN, E. M., SHAKARCHI, R. *Fourier Analysis: an introduction*, v. 1, *Princeton Lectures in Analysis*. Princeton (N.J.), Oxford, Princeton University Press, 2003.
- [4] DE FIGUEIREDO, D. G. *Análise de Fourier e Equações Diferenciais Parciais*. Projeto Euclides. Rio de Janeiro, Brasil, Instituto de Matemática Pura e Aplicada, 1977.
- [5] SHUMAN, D. I., NARANG, S. K., FROSSARD, P., et al. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”, *IEEE Signal Processing Magazine*, v. 30, n. 3, pp. 83–98, May 2013.
- [6] SANDRYHAILA, A., MOURA, J. M. F. “Discrete Signal Processing on Graphs”, *IEEE Transactions on Signal Processing*, v. 61, n. 7, pp. 1644–1656, April 2013.
- [7] VON LUXBURG, U. “A tutorial on spectral clustering”, *Statistics and Computing*, v. 17, n. 4, pp. 395–416, Dec 2007.
- [8] CVETKOVIĆ, D., GUTMAN, I. *Selected Topics on Applications of Graph Spectra*. Novi Sad, Zbornik Radova, 2011.
- [9] JONES, E., OLIPHANT, T., PETERSON, P., et al. “SciPy: Open source scientific tools for Python”. 2001–. Available at: <http://www.scipy.org/>. [Online; accessed March 2019].
- [10] GODSIL, C., ROYLE, G. *Algebraic Graph Theory*, v. 207, *Graduate Texts in Mathematics*. New York, NY, Springer, 2001.

- [11] ACKER, F. *Análise Vetorial Clássica*, v. 11, *Coleção Textos Universitários*. Rio de Janeiro, RJ, Sociedade Brasileira de Matemática, 2012.
- [12] SHUMAN, D. I., RICAUD, B., VANDERGHEYNST, P. “A windowed graph Fourier transform”. In: *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pp. 133–136, Aug 2012. doi: 10.1109/SSP.2012.6319640.
- [13] WANG, W., RAMCHANDRAN, K. “Random Multiresolution Representations for Arbitrary Sensor Network Graphs”. In: *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, v. 4, pp. IV–IV, May 2006.
- [14] CROVELLA, M., KOLACZYK, E. “Graph wavelets for spatial traffic analysis”. In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, v. 3, pp. 1848–1857, March 2003.
- [15] SWELDENS, W. “The Lifting Scheme: A Construction of Second Generation Wavelets”, *SIAM Journal on Mathematical Analysis*, v. 29, n. 2, pp. 511–546, 1998.
- [16] HAMMOND, D. K., VANDERGHEYNST, P., GRIBONVAL, R. “Wavelets on graphs via spectral graph theory”, *Applied and Computational Harmonic Analysis*, v. 30, n. 2, pp. 129 – 150, 2011.
- [17] RYAN, H. “Ricker, Ormsby, Klander, Butterworth – A Choice of Wavelets”, *Journal of the Canadian Society of Exploration Geophysicists*, v. 19, n. 7, pp. 8–9, September 1994.
- [18] DI LORENZO, P., BARBAROSSA, S., BANELLI, P., et al. “Adaptive Least Mean Squares Estimation of Graph Signals”, *IEEE Transactions on Signal and Information Processing over Networks*, v. 2, n. 4, pp. 555–568, Dec 2016.
- [19] DI LORENZO, P., BANELLI, P., ISUFI, E., et al. “Adaptive Graph Signal Processing: Algorithms and Optimal Sampling Strategies”, *IEEE Transactions on Signal Processing*, v. 66, n. 13, pp. 3584–3598, July 2018.
- [20] SPELTA, M. J., MARTINS, W. A. “Online Temperature Estimation using Graph Signals”, *XXXVI Simpósio Brasileiro de Telecomunicações*, Setembro 2018.
- [21] PEENSON, I. “Sampling in Paley-Wiener Spaces on Combinatorial Graphs”, *Transactions of the American Mathematical Society*, v. 360, n. 10, pp. 5603–5627, 2008.
- [22] STEIN, E. M., SHAKARCHI, R. *Complex Analysis*, v. 2, *Princeton Lectures in Analysis*. Princeton (N.J.), Oxford, Princeton University Press, 2003.

- [23] STRICHARTZ, R. *A guide to distribution theory and Fourier transforms*. Studies in advanced mathematics. Boca Raton, Florida, USA, CRC Press, 1994.
- [24] WIDROW, B., HOFF, M. E. “Adaptive Switching Circuits”, *IRE WESCON Convention Record*, v. 4, pp. 96–104, 1960.
- [25] DINIZ, P. S. R., DA SILVA, E. A. B., NETTO, S. L. *Digital Signal Processing: System Analysis and Design*. 2 ed. Cambridge, Cambridge University Press, September 2010.
- [26] BARNARD, S. T., SIMON, H. D. “Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems”, *Concurrency: Practice and Experience*, v. 6, n. 2, pp. 101–117, 1994.
- [27] BIYIKOĞLU, T., LEYDOLD, J., STADLER, P. F. *Laplacian Eigenvectors of Graphs*, v. 1915, *Lecture Notes in Mathematics*. Springer-Verlag Berlin Heidelberg, 2007.
- [28] KLEIN, D. J., RANDIĆ, M. “Resistance distance”, *Journal of Mathematical Chemistry*, v. 12, n. 1, pp. 81–95, 1993.