JOINT LEARNING OF GRAPH PROCESSES AND GRAPH TOPOLOGIES FOR TIME VERTEX SIGNAL ESTIMATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BERKAY YALDIZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2024

Approval of the thesis:

JOINT LEARNING OF GRAPH PROCESSES AND GRAPH TOPOLOGIES FOR TIME VERTEX SIGNAL ESTIMATION

submitted by **BERKAY YALDIZ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Naci Emre Altun
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Elif Vural
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Abdullah Aydın Alatan
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Elif Vural
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Özlem Tuğfe Demir
Electrical and Electronics Engineering, TOBB ETU _____

Date: 28.08.2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Berkay Yaldız

Signature :

ABSTRACT

JOINT LEARNING OF GRAPH PROCESSES AND GRAPH TOPOLOGIES FOR TIME VERTEX SIGNAL ESTIMATION

Yaldız, Berkay
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Elif Vural

August 2024, 84 pages

In recent years, the analysis of data that evolves over time and across interconnected entities has gained significant interest. Such data, often represented as time-vertex graph signals, encapsulate the dynamic nature of various real-world systems, including social networks, sensor networks, and traffic systems. Traditional methods that separately handle temporal and network dependencies without considering network correctness often fall short in capturing the full complexity of these datasets. To address this, in this thesis, we use a parametric statistical modelling approach called Auto-Regressive Moving Average (ARMA) jointly wide sense stationarity in order to analyze the joint time-vertex behavior of time-varying graph random processes, while we also aim to improve the partially known network topology via the graph Laplacian. We explore ARMA graph process models, where our primary objective is to develop a comprehensive framework that integrates ARMA modeling with graph learning techniques to enhance the analysis of time-varying signals characterized by both temporal and network dependencies. Our study introduces a novel approach where the joint power spectral density derived from the graph ARMA model is used to estimate the covariance matrix of the process. This matrix provides a basis for

our graph learning algorithm, which iteratively refines both the joint process and the graph Laplacian. By integrating the dynamic characteristics of ARMA models with graph learning techniques, the proposed method facilitates the discovery of underlying structures and relationships within time-varying graph signals. Simulations and real-world experiments are conducted to validate the effectiveness of the framework, demonstrating its potential for time-vertex signal analysis. The results indicate improvements in capturing spatiotemporal dependencies for network data with a time-varying structure.

Keywords: Graph Signal Processing, Time-Vertex Signal Processing, Stationary Graph Signal Processing, Graph Laplacian Matrix Learning, Spatiotemporal Signal Processing

ÖZ

ZAMAN-DÜĞÜM SİNYAL KESTİRİMİ İÇİN ÇİZGE SÜREÇLERİ VE ÇİZGE TOPOLOJİLERİNİN ORTAK ÖĞRENİMİ

Yaldız, Berkay
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Elif Vural

Ağustos 2024 , 84 sayfa

Son yıllarda, hem zamanda hem de bir ağ yapısı üzerinde degişim gösteren verilerin analizi önemli ölçüde ilgi gören bir konu olmuştur. Zaman-düğüm çizge sinyalleri olarak temsil edilen bu tür veriler, sosyal ağlar, sensör ağları ve trafik sistemleri gibi çeşitli sistemlerin dinamik yapısından etkilenmektedir. Bağımlılıkları ayrı ayrı ele alan ve ağ doğruluğunu hesaba katmayan geleneksel yöntemler, genellikle bu veri kümelerinin karmaşıklığını yakalamakta yetersiz kalmaktadır. Bu tezde zamanla değişen çizge rastgele süreçlerinin zaman-düğüm noktası davranışını birlikte analiz etmek için ortak geniş anlamda durağanlık adı verilen bir parametrik istatistiksel modelleme yaklaşımı olan Otoregresif Hareketli Ortalama (ARMA) modelleme kullanılmış, aynı zamanda kısmen bilinen Laplacian matrisi aracılığıyla topolojinin de öğrenilmesi hedeflenmiştir. ARMA çizge süreç modelleri incelenmiş; öncelikli olarak hem zamansal hem de ağ bağımlılıklarıyla karakterize edilen zamanda değişen sinyallerin analizi geliştirmek için ARMA modellemesini çizge öğrenme teknikleriyle bütünleştiren kapsamlı bir yöntem geliştirilmesi hedeflenmiştir. Çalışmamız, sürecin kovaryans matrisini tahmin etmek için çizge ARMA modelinden türetilen ortak güç

spektral yoğunluğunun kullanıldığı yeni bir yaklaşım sunmaktadır. Kestirilen kovaryans matrisi, hem ortak süreci hem de çizge Laplacian matrisini yinelemeli olarak iyileştiren çizge öğrenme algoritmamız için bir temel sağlar. ARMA modellerinin dinamik özelliklerini çizge öğrenme teknikleriyle bütünleştirerek önerilen yöntem, zamanda değişen çizge sinyalleri içindeki temel yapıların ve ilişkilerin keşfedilmesini kolaylaştırır. Önerilen yöntemin etkinliğini doğrulamak ve zaman-düğüm sinyal analizindeki potansiyelini göstermek için sentetik ve gerçek veriler üzerinde deneyler yapılmıştır. Sonuçlar, zamanda değişen bir yapıya sahip ağ verileri için zamansal-ağsal bağımlılıkların yakalanmasında ilerleme kaydedilebileceğini göstermektedir.

Anahtar Kelimeler: Çizge Sinyal İşleme, Zaman-Nod Sinyal İşleme, Durağan Çizge Sinyal İşleme, Çizge Laplacian Matris Öğrenimi, Uzay-Zamansal Düzgünlük

To my family

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many individuals. First and foremost, I am deeply grateful to my thesis advisor, Assoc. Prof. Dr. Elif Vural, for her invaluable guidance, unwavering support, and immense patience throughout this journey. Her insights and encouragement have been instrumental in shaping this work.

I would also like to express my gratitude to Eylem Tuğçe Güneyi, whose collaboration during our graduate studies on graph signal processing has been greatly enriching.

A special thanks to Abdullah Canbolat for his illuminating perspectives on problem-solving, which have inspired me throughout this process.

I am also thankful to my colleagues at Meteksan Defence for their understanding and support over the past few months, allowing me to balance both my professional and academic commitments.

This thesis is supported by the TÜBİTAK Scientist Support Programs Presidency (BİDEB) and the 2211-Domestic Postgraduate Scholarship Program under grant 120E246.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

AR	Autoregressive
ARMA	Autoregressive Moving Average
FT	Fourier Transform
CNN	Convolutional Neural Networks
DFT	Discrete Fourier Transform
FIR	Finite Impulse Response Filters
GFT	Graph Fourier Transform
GSP	Graph Signal Processing
GNN	Graph Neural Networks
GAN	Graph Attention Networks
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
CNN	Convolutional Networks
GMRF	Gaussian Markov Random Field
G-VARMA	Graph Polynomial Vector Autoregressive Moving Average Recursions
GP-VAR	Graph Polynomial Vector Autoregressive Recursions
IGFT	Inverse Graph Fourier Transform
IIR	Infinite Impulse Response Filters
IGFT	Inverse Graph Fourier Transform
JPSD	Joint Power Spectral Density
JFT	Joint Fourier Transform
JWSS	Joint Time-Vertex Wide Sense Stationary

k-NN	k nearest neighbors
LMMSE	Linear Minimum Mean Square Error
NMSE	Normalized Mean Square Error
PSD	Power Spectral Density
SNR	Signal-to-Noise Ratio
VAR	Vector Autoregressive
VARMA	Vector Autoregressive Moving Average
MAP	Maximum a posteriori
JS-ARMA	Joint Spectra-ARMA
GL-JS-ARMA	Graph Learning-JS-ARMA

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Classical signal processing excels at analyzing data defined on regular domains like time or space. However, many real-world phenomena naturally exhibit relationships that cannot be captured by these rigid structures, for which Graph Signal Processing (GSP) tools provide useful solutions. GSP leverages the power of graphs to represent complex interactions between data points. Graphs are flexible structures that can model intricate relationships between entities, making them ideal for analyzing data arising from social networks, sensor networks, transportation systems, biological systems, and more. These networks can be effectively represented as graphs, where nodes represent entities and edges depict the connections between them.

GSP seeks to extend traditional signal processing techniques to the analysis of signals defined on graphs. [1, 2]. GSP has a wide range of applications, such as classification [3], image processing [4], interpolation and denoising [5], optimal power flow [6] and smoothing [7].

In traditional signal processing, wide-sense stationarity assumption has a wide range of applications due to its easier applicability to real-world scenarios and it has been classically defined for regular domains. In the recent years, a notion of stationarity for random graph signals has been proposed via the definitions of Graph Fourier Transform (GFT), graph filtering and Power Spectral Density (PSD) in the works [8, 9]. On the other hand, the statistical properties of data may depend not only on the relationships between nodes but also on the specific time instances; therefore, time-varying graph signal models play a vital role in understanding and analyzing

phenomena occurring over complex network structures. Thus, modelling the collected measurements as an ARMA jointly wide-sense stationary (JWSS) time-vertex process on a known graph can help capturing the structural dynamics and interactions better [10, 11].

The covariance structure of the JWSS time-vertex processes can be used in signal interpolation, forecasting and filtering. Assuming the model is a joint time-vertex stationary ARMA process, its Joint Power Spectral Density (JPSD) can be found in terms of the ARMA parameters [11]. Although modelling the data as JWSS has many benefits, most of the current methods assume that the graph topology is perfectly known. In practice, knowing the graph topology is hard and information that appears on the surface at first glance might be misleading. For instance, in a sensor network structure, one common expectation is that nearest sensors should have higher correlations but due to environmental conditions this might not be always the case. Similarly, in a social network friendship connections registered between different users may be out-of-date, as there may be no active communication anymore between two friends in a social network. In order to improve the performance, learning a graph that represents the data better can yield better performance [12]. In this thesis, we aim to learn both the parametric ARMA model and the graph Laplacian matrix together iteratively to achieve better signal inference accuracy.

1.2 Proposed Methods and Models

We consider the problem of jointly learning the JPSD of an ARMA modeled JWSS time-vertex process along with the approximately known graph topology. Although we demonstrate our approach in spatio-temporal interpolation problems, it can be potentially used in various applications such as denoising, forecasting and filtering. Firstly, we assume that the graph Laplacian matrix is approximately known initially and the observed data is composed of realizations of a JWSS time-vertex process. This assumption can be considered realistic in a variety of scenarios. For example, when studying the spread of a pandemic, nodes represent countries or cities and edges represent potential transmission routes or physical proximity; hence, a k-nearest neighborhood (k-NN) graph can be constructed. Although we do not restrict

ourselves to find a graph that preserves the adjacency relations, which corresponds to changing only the edge weight values, we do not try to find a graph that is completely different than the initial one in terms of edge locations.

We fit a joint time-vertex stationary ARMA graph process model to the initial estimate of the JPSD. We then compute the initial JPSD from the covariance matrix with the Joint Fourier Transform (JFT), where the initial estimate of the covariance matrix is obtained with an unbiased sample covariance estimator based on a set of realizations of the process. Next, we obtain the parameters of the graph ARMA process through an optimization problem such that the resulting JPSD is coherent with the initial JPSD estimate at hand. The resulting formulation for learning a parametric ARMA graph process model leads to a non-convex optimization problem [9]. Hence, a convex relaxation is applied in order to put it in a more tractable form [11]. After calculating the graph ARMA process model, which yields a refined estimation of the JPSD, the covariance matrix of this ARMA graph process is also refined through the Inverse Joint Fourier Transform (IJFT) based on the improved JPSD estimate. We finally use this covariance matrix for learning the graph Laplacian. However, due to the nature of the time-vertex process, the dimension of the graph Laplacian is lower than that of the process covariance matrix. Hence, a sub-matrix of this covariance matrix, restricted to a single time instant, which corresponds to the covariance matrix for a vertex stationary process alone, is used effectively. The selection of this covariance sub-matrix is determined from synthetic-data experiments. Then, the stochastic Gaussian Markov Random Field (GMRF) [13, 12] approach is used to fit a valid undirected graph to the estimated covariance matrix. This approach corresponds to the maximum-likelihood estimation of the precision matrix for a GMRF from sampled data, which is the Laplacian matrix. Then this Laplacian matrix is used to fit an ARMA graph process model again to increase the estimation performance.

1.3 Contributions

The primary objective of this thesis is to develop a method for learning stochastic process models from realizations of time-vertex processes in a scenario where the graph topology is not perfectly known. The key contributions include:

- An empirical analysis of jointly learning a time-varying graph ARMA models and the graph Laplacian.
- An algorithm for iteratively learning time-varying graph ARMA models and graph structures.
- Validation of the algorithm through synthetic and real-world data sets, demonstrating its potential, as well as its shortcomings, regarding its practical applicability.

1.4 The Outline of the Thesis

In Chapter 2, we provide a review of the existing literature. It begins with an introduction to the fundamentals of graph signal processing. Graph structures, the definition of a graph signal, the GFT and graph filters are mentioned. Then, the concepts of joint time-vertex graph signal processing, JFT, JPSD and ARMA processes are covered. The chapter then reviews traditional methods of graph construction, followed by modern data-driven approaches. We review different kinds of approaches in time-vertex signal modelling and graph learning in Chapter 2. Finally, the challenges faced in real-world scenarios are discussed.

In Chapter 3, our proposed method for learning time-vertex graph ARMA processes jointly with the graph Laplacian matrix is explained. The definitions of stationarity for time-vertex graph signals and the ARMA graph process model are presented. Subsequently, in Section 3.3.4, we overview the result that learning the Laplacian matrix from data actually corresponds to learning a precision matrix for a Gaussian Markov Random Field (GMRF), in which the probability density of the data is a zero mean n-variate Gaussian distribution [13].

In Chapter 4, we assess the performance of the given algorithm using real-time and synthetic time-vertex data sets.

In Chapter 5, we summarize the findings, potential applications, and directions for future research.

CHAPTER 2

RELATED WORK

In Section 2.1, we provide a review of Graph Signal Processing. Then, in Section 2.2, we discuss joint time-vertex signal processing concepts. In Section 2.3, we cover wide-sense stationary processes for both graph and time-varying graph signals. In Section 2.4, we explore graph learning approaches. Finally, in Section 2.5, we integrate these aforementioned concepts and discuss time-vertex signal models, graph networks, and their applications.

2.1 Fundamentals of Graph Signal Processing

2.1.1 Graph Basics and Notation

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ consists of a set of vertices $\mathcal{V}$ (or nodes), a set of edges $\mathcal{E}$ that connect pairs of vertices, and a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$. The number of vertices is denoted by $N = |\mathcal{V}|$. Each edge $e = (u, v) \in \mathcal{E}$ can be either undirected or directed, depending on the nature of the relationship between the nodes u and v. For undirected graphs, each edge $e = (v, u)$ is also included in $\mathcal{E}$.

In the context of spectral graph theory, undirected graphs have a well-defined notion of frequency. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ of a graph $\mathcal{G}$ indicates the presence of an edge between nodes i and j with $\mathbf{A}_{ij}$, making it a binary matrix. For unweighted graphs, the weight matrix and the adjacency matrix are identical. For weighted graphs, the weight matrix $\mathbf{W}$ assigns a real value, which may be different than 1, to these edges, indicating the strength of the connection between corresponding nodes. Figure 2.1 provides an example of a graph signal.

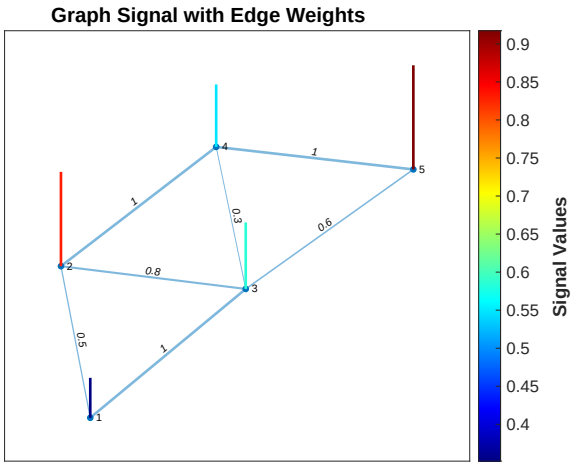

Figure 2.1: Weighted graph signal

To construct a weight matrix, the k-NN algorithm and the Gaussian kernel in (2.1) can be used. We start by defining the weight between each pair of nodes based on their distance. For nodes i and j with positions $\mathbf{p}_i$ and $\mathbf{p}_j$ in some feature space, the distance between these nodes can be computed as $\text{dist}(i,j) = \|\mathbf{p}_i - \mathbf{p}_j\|_2$, where the $\|.\|_2$ operator represents the ℓ_2 norm. The weight matrix $\mathbf{W}$ is then formed by applying a Gaussian kernel to these distances. Specifically, the weight between nodes i and j is given by:

$$\mathbf{W}_{ij} = \exp\left(-\frac{\text{dist}(i,j)^2}{2\sigma^2}\right), \tag{2.1}$$

where $\sigma \in \mathbb{R}$ is a scaling parameter that controls the width of the Gaussian function. This weight matrix $\mathbf{W}$ captures the similarity between nodes, with larger weights as-

signed to pairs of nodes that are closer together in the feature space. The use of Gaussian weights ensures that the influence of distant nodes is exponentially suppressed, leading to a more localized and realistic representation of the graph structure.

2.1.2 Graph Laplacian

Given a graph, the combinatorial Laplacian matrix $\mathcal{L}_\mathcal{G} \in \mathbb{R}^{N \times N}$ is defined as [1]

$$\mathcal{L}_\mathcal{G} = \mathbf{D} - \mathbf{W} \tag{2.2}$$

whereas the symmetrically normalized graph Laplacian is defined as

$$\mathcal{L}_\mathcal{G} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2} \tag{2.3}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix, defined as

$$\mathbf{D}_{ii} = \sum_{j \in \mathcal{V}(i)} \mathbf{W}_{ij} \tag{2.4}$$

and $\mathcal{V}(i)$ denotes the set of neighbors of node i. This formulation ensures that the Laplacian matrix $\mathcal{L}_\mathcal{G}$ is symmetric and positive semi-definite, meaning all its eigenvalues are non-negative and real.

The eigenvalues and eigenvectors of the Laplacian matrix provide valuable insight into the graph's structure. The eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{N-1}$ satisfy

$$0 = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{N-1}. \tag{2.5}$$

The smallest eigenvalue λ_0 is always 0, with the corresponding eigenvector being the constant vector $\mathbf{1} \in \mathbb{R}^N$ (assuming the graph is connected). The multiplicity of the zero eigenvalue indicates the number of connected components in the graph.

2.1.3 Graph Signals

A graph signal $\mathbf{x}$ is a function that assigns a scalar value $\mathbf{x}_i$ to each node i in the graph, represented as a vector $\mathbf{x} \in \mathbb{R}^N$. For example, in a social network, $\mathbf{x}_i$ could represent the activity level of user i.

2.1.4 Connection to Classical Signal Processing

In classical continuous signal processing, the Laplacian operator ∇^2 is a second-order differential operator. For a function $f(x, y)$, the Laplacian is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{2.6}$$

This operator measures the divergence of the gradient of a function, which can be interpreted as the difference between the value at a point and the average value around it [14]. It is used in various applications, including solving partial differential equations and analyzing heat flow.

In the discrete graph domain, the graph Laplacian serves a similar purpose. For a graph signal $\mathbf{x} \in \mathbb{R}^N$, the graph Laplacian applied to $\mathbf{x}$ is given by

$$\mathcal{L}_{\mathcal{G}}\mathbf{x} = \mathbf{D}\mathbf{x} - \mathbf{W}\mathbf{x}. \tag{2.7}$$

For a node i:

$$(\mathcal{L}_{\mathcal{G}}\mathbf{x})_i = \mathbf{D}_{ii}\mathbf{x}_i - \sum_{j \in \mathcal{V}(i)} \mathbf{W}_{ij}\mathbf{x}_j. \tag{2.8}$$

This expression measures the difference between the signal value at node i and the weighted average of the signal values at its neighboring nodes.

In the context of graph signal processing, the gradient operator measures the difference between signal values at connected nodes. For an edge $(i, j) \in \mathcal{E}$ of a non-normalized Laplacian, the gradient of the signal $\mathbf{x}$ along the edge is given by [15]

$$\nabla_{ij}\mathbf{x} = \sqrt{\mathbf{W}_{ij}}\,(\mathbf{x}_i - \mathbf{x}_j). \tag{2.9}$$

The divergence operator, which is the (negative) adjoint of the gradient operator, aggregates these differences at each node. For a signal $\mathbf{x}$, the divergence at node i can be expressed as [16]

$$(\text{div}\mathbf{x})_i = \sum_{j \in \mathcal{V}(i)} (\mathbf{x}_i - \mathbf{x}_j). \tag{2.10}$$

Combining these, the graph Laplacian can be viewed as the composition of the divergence and gradient operators [16]

$$\mathcal{L}_{\mathcal{G}}\mathbf{x} = \text{div}(\nabla\mathbf{x}). \tag{2.11}$$

2.1.5 Graph Fourier Transform

The Graph Fourier Transform (GFT) generalizes the concept of the Fourier transform to graphs. It leverages the eigendecomposition of the graph Laplacian. Let $\mathcal{L}_\mathcal{G} = \mathbf{U}_\mathcal{G}\Lambda_\mathcal{G}\mathbf{U}_\mathcal{G}$, where $\mathbf{U}_\mathcal{G} \in \mathbb{R}^{N\times N}$ is the matrix of eigenvectors and $\Lambda_\mathcal{G} \in \mathbb{R}^{N\times N}$ is the diagonal matrix of eigenvalues. The GFT and the inverse GFT of a graph signal $\mathbf{x}$ are defined as [17]

$$\hat{\mathbf{x}} = \mathbf{U}_\mathcal{G}^\intercal \mathbf{x}, \tag{2.12}$$

$$\mathbf{x} = \mathbf{U}_\mathcal{G}\hat{\mathbf{x}}. \tag{2.13}$$

2.1.6 Graph Filters

In classical signal processing, the basic building block of filters is time shift or delay [18]. On the other hand, shifting a graph signal in GSP involves redistributing the signal values according to the structure of the graph. This operation is analogous to time-shifting in classical signal processing, but it respects the topology of the graph [19]. The shift operator $\mathbf{S}$ can be defined in several ways, but a common choice is the Laplacian matrix $\mathcal{L}_\mathcal{G}$ itself and the shifted graph signal is given by $\mathcal{L}_\mathcal{G}\mathbf{x}$. The primary reason for using the graph Laplacian is that it integrates both adjacency information and node degrees, offering a unique perspective on signal redistribution during the shift operation.

This idea of shifting can be extended to graph filters as [20]

$$\bar{\mathbf{y}} = g(\mathcal{L}_\mathcal{G})\mathbf{x} = \mathbf{U}_\mathcal{G} g(\Lambda_\mathcal{G})\mathbf{U}_\mathcal{G}{}^\intercal \mathbf{x}. \tag{2.14}$$

where $g : \{0\} \cup \mathbb{R}^+ \to \mathbb{R}$ denotes a filter kernel, $g(\mathcal{L}_\mathcal{G}) \in \mathbb{R}^{N\times N}$ is a graph filter.

Filter kernels can be defined in polynomial form to define polynomial graph filters [21]

$$g(\mathcal{L}_\mathcal{G}) = \sum_{k=0}^{K-1} g_k \mathcal{L}_\mathcal{G}^k \mathbf{x} = g_0\mathbf{x} + g_1\mathcal{L}_\mathcal{G}\mathbf{x} + g_2\mathcal{L}_\mathcal{G}^2\mathbf{x} + \cdots + g_{K-1}\mathcal{L}_\mathcal{G}^{K-1}\mathbf{x} \tag{2.15}$$

where g_k for $k = 0, 1, \cdots K-1$ are polynomial coefficients.

Additionally, a graph filter $\mathbf{H} \in \mathbb{R}^{N \times N}$ can be expressed as a function of the Laplacian eigenvalues in the spectral domain

$$\mathbf{H} = g(\mathcal{L}_\mathcal{G}) = \mathbf{U}_\mathcal{G} g(\Lambda_\mathcal{G}) \mathbf{U}_\mathcal{G}^\intercal, \tag{2.16}$$

where $g(\mathcal{L}_\mathcal{G}) \in \mathbb{R}^{N \times N}$ is the graph filter and $g(\Lambda_\mathcal{G}) \in \mathbb{R}^{N \times N}$ is the diagonal matrix with the filter function applied to each eigenvalue λ_i.

Applying the graph filter to a signal $\mathbf{x}$, we get the filtered signal $\mathbf{y} \in \mathbb{R}^N$

$$\mathbf{y} = \mathbf{H}\mathbf{x} = \mathbf{U}_\mathcal{G} g(\Lambda_\mathcal{G}) \mathbf{U}_\mathcal{G}^\intercal \mathbf{x}. \tag{2.17}$$

The shift operation is linear for any two signals $\mathbf{x}$ and $\mathbf{y}$, and scalars α and β:

$$\mathbf{S}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\mathbf{S}\mathbf{x} + \beta\mathbf{S}\mathbf{y} \tag{2.18}$$

The shift operation is shift invariant. Applying the shift operator multiple times corresponds to multiplying the shift matrix multiple times. For instance, shifting the signal twice when $\mathbf{S} = \mathbf{A}$ is

$$\mathbf{S}^2\mathbf{x} = \mathbf{A}^2\mathbf{x}. \tag{2.19}$$

Consider a simple graph with three nodes and the following adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

For a graph signal $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, the shifted signal $\mathbf{A}\mathbf{x}$ is

$$\mathbf{S}\mathbf{x} = \mathbf{A}\mathbf{x} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_1 + x_3 \\ x_2 \end{pmatrix}.$$

In this example, the signal value at node 1 is shifted to node 2, the value at node 2 is distributed to nodes 1 and 3, and the value at node 3 is shifted to node 2.

2.2 Joint Time-Vertex Signal Processing

2.2.1 Joint Laplacian Matrix

Time-varying graph signal processing is a framework that extends the classical signal processing to handle data that varies over both time and graph structures. The joint Laplacian matrix is the essential tool that we use for the analysis of time-varying graph signals, as it allows for the simultaneous consideration of both the temporal and graph domains. It extends the concept of the Laplacian matrix, which traditionally captures the structure of a static graph, to handle the additional complexity introduced by temporal variations.

The temporal Laplacian matrix $\mathcal{L}_T$ is constructed to reflect the connectivity between consecutive time steps, capturing the temporal relationships. Matrix $\mathbf{A}_T$ corresponds to the adjacency matrix of a cyclic graph with T time steps, and it is given by [22, 23]

$$\mathbf{A}_T = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}. \tag{2.20}$$

The corresponding temporal Laplacian matrix $\mathcal{L}_T$ is defined as

$$\mathcal{L}_T = \mathbf{D}_T - \mathbf{A}_T \tag{2.21}$$

where $\mathbf{D}_T$ is the degree matrix of the cyclic graph for which each node has degree 2, so $\mathbf{D}_T$ is

$$\mathbf{D}_T = \begin{pmatrix} 2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 2 \end{pmatrix}. \tag{2.22}$$

Thus, the temporal Laplacian matrix $\mathcal{L}_T$ for a cyclic graph is

$$\mathcal{L}_T = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ -1 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}. \tag{2.23}$$

To incorporate both the graph and time domains, we construct a joint graph-time matrix. The joint Laplacian matrix $\mathcal{L}_J$ is defined as [24]

$$\mathcal{L}_J = \mathcal{L}_T \oplus \mathcal{L}_\mathcal{G} = \mathcal{L}_T \otimes \mathbf{I}_N + \mathbf{I}_T \otimes \mathcal{L}_\mathcal{G} \tag{2.24}$$

where $\mathcal{L}_\mathcal{G}$ is the graph Laplacian, $\mathcal{L}_T$ is the temporal Laplacian, $\oplus$ denotes the Kronecker sum, $\otimes$ denotes the Kronecker product, and $\mathbf{I}_T \in \mathbb{R}^{T\times T}$ and $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ are identity matrices respectively.

The Kronecker product $\otimes$ combines the network and temporal components, ensuring that the joint Laplacian captures interactions both within the graph and across time. The term $\mathbf{I}_T \otimes \mathcal{L}_\mathcal{G}$ ensures that the graph relationships are maintained across all time instances, while $\mathcal{L}_T \otimes \mathbf{I}_N$ ensures that the temporal relationships are maintained independently for each node in the graph.

2.2.2 Time-Varying Graph Signals

A time-varying graph signal is represented as $\mathbf{X} \in \mathbb{R}^{N\times T}$. It is a signal defined on the nodes of a graph $\mathcal{G}$ at each time instance t. This signal can be viewed as a sequence of graph signals, $\begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_T\end{bmatrix}$ where $\mathbf{x}_t$ is the graph signal at time t.

2.2.3 Joint Fourier Transform

The Discrete Fourier Transform (DFT) is used to analyze the time-domain characteristics of a signal. For a time signal $\mathbf{x}(t)$, the Discrete Fourier Transform is [25]

$$\hat{\mathbf{x}}(m) = \frac{1}{\sqrt{T}} \sum_{t=0}^{T-1} \mathbf{x}(t) e^{-j2\pi mt/T}, \quad m = 0, 1, \ldots, T-1. \tag{2.25}$$

In the context of time-varying graph signals, since each row is a time signal, temporal DFT can be found by [24]

$$DFT\{\mathbf{X}\} = \mathbf{X}conj(\mathbf{U}_T) \tag{2.26}$$

where $conj$ represents the element-wise conjugate of a matrix and, $\mathbf{U}_T$ is the conjugate of the normalized DFT matrix given by

$$\mathbf{U}_T(t,\tau) = \frac{e^{j\omega_\tau t}}{\sqrt{T}}, \quad \omega_\tau = \frac{2\pi(\tau-1)}{T} \text{ for } t,\tau = 1,\cdots,T. \tag{2.27}$$

Indeed, $\mathbf{U}_T$ in (2.27) contains the eigenbasis of the cyclic graph $\mathcal{L}_T$ and its eigenvalues are [23]

$$\Lambda_T(\tau,\tau) = 2(1-\cos\omega_\tau). \tag{2.28}$$

Thus, $\mathcal{L}_T$ can be written as

$$\mathcal{L}_T = \mathbf{U}_T\Lambda_T\mathbf{U}_T^H \tag{2.29}$$

where $(\cdot)^H$ denotes the Hermitian (transpose complex-conjugate) of a matrix.

The eigendecomposition of the joint Laplacian matrix $\mathcal{L}_J$ can also be written in terms of the eigenbases of the graph Laplacian $\mathcal{L}_\mathcal{G}$ and cyclic Laplacian $\mathcal{L}_T$ [23]

$$\mathcal{L}_J = (\mathbf{U}_T \otimes \mathbf{U}_\mathcal{G})(\Lambda_T \oplus \Lambda_\mathcal{G})(\mathbf{U}_T \otimes \mathbf{U}_\mathcal{G})^H. \tag{2.30}$$

Moreover, if $\mathbf{X}$ is considered as a series of T graph signals organized in its columns, the GFT of these signals can be computed as

$$GFT\{\mathbf{X}\} = \mathbf{U}_\mathcal{G}^\intercal\mathbf{X}. \tag{2.31}$$

The Joint Fourier Transform (JFT) combines the GFT and DFT to analyze signals in both domains simultaneously.

For a time-varying graph signal $\mathbf{x}(t)$, the JFT is defined as [23]

$$\widehat{\mathbf{X}}(l,k) = \frac{1}{\sqrt{T}}\sum_{n=1}^{N}\sum_{t=1}^{T}\mathbf{X}(n,t)u_l^\intercal(n)e^{-j\omega_k t} \tag{2.32}$$

where l and k represent the indices of the graph and temporal frequencies, respectively. The expression above can be conveniently reformulated in matrix notation as [23]

$$\widehat{\mathbf{X}} = DFT\{GFT\{\mathbf{X}\}\} = GFT\{DFT\{\mathbf{X}\}\} = \mathbf{U}_\mathcal{G}^\intercal\mathbf{X}conj(\mathbf{U}_T) \tag{2.33}$$

which is equal to in matrix vector multiplication form [23]

$$\hat{\bar{\mathbf{x}}} = (\mathbf{U}_J)^H \bar{\mathbf{x}} \tag{2.34}$$

where $\mathbf{U}_J = \mathbf{U}_T \otimes \mathbf{U}_\mathcal{G}$ and $\bar{\mathbf{x}} = \mathrm{vec}\,(\mathbf{X})$ and, vec is an operator that transforms a matrix into a column vector by vertically stacking the columns of the matrix.

2.2.4 Joint Filtering of Time-Vertex Signals

Similar to (2.16), we can define the joint graph filter for time-vertex signals. For a joint graph filter $g(\mathcal{L}_J)$, the connection between the input and output time-vertex signals, $\mathbf{X}$ and $\mathbf{Y}$, can be described using their column-wise vectorized forms as follows

$$\bar{\mathbf{y}} = g(\mathcal{L}_J)\bar{\mathbf{x}} = \mathbf{U}_J\, g(\Lambda_\mathcal{G}, \Omega)\, (\mathbf{U}_J)^H \bar{\mathbf{x}}. \tag{2.35}$$

Here, $g(\Lambda_\mathcal{G}, \Omega) \in \mathbb{R}^{NT \times NT}$ represents the vectorized form of the joint kernel matrix [10]

$$g(\Lambda_\mathcal{G}, \Omega) = \mathrm{diag}\left(\mathrm{vec}\left(\begin{bmatrix} g(\lambda_1, \omega_1) & \cdots & g(\lambda_1, \omega_T) \\ \vdots & \ddots & \vdots \\ g(\lambda_N, \omega_1) & \cdots & g(\lambda_N, \omega_T) \end{bmatrix}\right)\right) \tag{2.36}$$

where $\mathrm{diag}(\cdot)$ returns a square diagonal matrix with the elements of the input vector on the main diagonal.

2.3 Wide-Sense Stationary Processes

2.3.1 Wide-Sense Stationary of Time Signals

In the analysis of time signals, the concept of stationarity plays a crucial role in simplifying and understanding the underlying properties of the signal. A time signal is said to be stationary if its statistical properties do not change over time. Specifically, wide-sense stationarity (WSS) is a less stringent form of stationarity that focuses on the first two statistical moments, the mean and the autocovariance function [26]. Let t be a one-dimensional time variable, and let $x(t)$ be a real-valued stochastic process. Then, wide-sense stationarity of this process depends on two features:

1. **Constant Mean**: The mean $E[x(t)]$ of the signal is constant and does not depend on time t.

$$E[x(t)] = \mu, \quad \forall t$$

where $E[\cdot]$ denotes the expectation operator and μ is a constant.

2. **Autocovariance Function Depends Only on Time Difference**: The autocovariance function $C_x(t_1, t_2)$ depends only on the time difference $\tau = t_2 - t_1$, not on the specific times t_1 and t_2.

$$C_x(t_1, t_2) = E[(x(t_1) - \mu)(x(t_2) - \mu)] = C_{\mathbf{x}}(\tau) \quad \text{where} \quad \tau = t_2 - t_1.$$

The autocovariance function measures the linear dependence between two observations separated by a time lag τ. In a WSS signal, the autocovariance only depends on the lag and not on the specific time instances chosen.

Wide-sense stationarity implies that the behavior of the signal is predictable to some extent because its statistical properties are invariant over time. This invariance simplifies the analysis and processing of the signal. For example:

- **Power Spectral Density (PSD)**: For a WSS signal, the power spectral density, which provides a frequency-domain representation of the signal's power distribution, can be defined and analyzed more easily.
- **Filtering**: When a WSS signal is passed through a linear time-invariant (LTI) system, the output remains WSS, which allows an easier analysis of the impact of the system on the signal.
- **Estimation and Prediction**: WSS properties enable more effective estimation and prediction techniques, such as using the autocovariance function to design optimal filters.

White Noise: A common example of a WSS signal is white noise, where the mean is zero and the autocovariance function is an impulse function (indicating no correlation between different time instances).

$$C_x(\tau) = \sigma^2 \delta(\tau)$$

where $\sigma^2 \in \mathbb{R}$ is the variance of the noise and $\delta(\tau)$ is the Dirac delta function.

2.3.2 Wide-Sense Stationary in Graph Processes

Wide-sense stationarity (WSS) in graph processes extends the classical concept of stationarity to the realm of graph signal processing. A graph process is a collection of random variables defined on the nodes of a graph, where the graph structure introduces dependencies among these variables. A graph process is said to be wide-sense stationary if its statistical properties, specifically the mean and the covariance, are invariant under the shift operator defined by the graph Laplacian or adjacency matrix.

A graph process $\mathbf{x} = \{x_i\}_{i\in\mathcal{V}}$ defined on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ with N nodes is wide-sense stationary if the following two conditions are satisfied [8]:

1. **Constant Mean**: The mean $E[\mathbf{x}_i]$ of the signal is constant over the vertex set and does not depend on a specific node i.

   $$E[\mathbf{x}_i] = \mu \quad \forall i \in \mathcal{V}$$

   where $E[\cdot]$ denotes the expectation operator and $\mu \in \mathbb{R}$ is a constant.

2. **Covariance as a Graph Filter**: The covariance between two nodes i and j is the result of localizing a graph kernel. If the graph shift operator is the $\mathcal{L}_\mathcal{G}$, then an unbiased estimate of the empirical covariance matrix of the graph process can be written as

   $$\mathbf{\Sigma_x} = g(\mathcal{L}_\mathcal{G})$$

   where the covariance matrix $\mathbf{\Sigma_x} \in \mathbb{S}_+^N$ is an element of the $N \times N$ dimensional symmetric positive semidefinite cone and g is a non-negative function that represents the power spectral density.

For a WSS graph process, the covariance matrix $\mathbf{\Sigma_x}$ can be diagonalized by the eigenbasis of the $\mathcal{L}_\mathcal{G}$

$$\mathbf{\Sigma_x} = \mathbf{U}_\mathcal{G} g(\Lambda_\mathcal{G}) \mathbf{U}_\mathcal{G}^\intercal. \tag{2.37}$$

This implies that the covariance function is invariant under the graph shift and depends only on the eigenvalues of the graph Laplacian, making the process WSS in the graph spectral domain.

The power spectral density of a WSS graph process provides a spectral representation and is defined as

$$h(\mathcal{L}_{\mathcal{G}}) = \mathbf{U}_{\mathcal{G}}^{\intercal} \mathbf{\Sigma}_{\mathbf{x}} \mathbf{U}_{\mathcal{G}}. \tag{2.38}$$

The first condition is similar to classical signal processing; the mean of each random variable on the vertex set $\mathcal{V}$ is constant and the same. On the other hand, the second condition requires more explanation since there is no straightforward way to define a vertex shift, which substitutes the time shift in classical signal processing, on graph structures. To address this, the second condition uses graph filters to express covariance stationarity. Specifically, it requires that the covariance between samples at two nodes on the graph be described by how a graph filter, centered at one node, responds at the other node. The nature of this graph filter then defines the covariance structure of the graph process [8]. It enables the design of graph filters, spectral estimation methods, and other signal processing tools that leverage the underlying graph structure.

2.3.3 Jointly Wide-Sense Stationary Processes

A graph process is said to be JWSS if its first two statistical properties, which are mean and covariance, are invariant under the joint time-vertex shift operator. This extends the concept of wide-sense stationarity to both temporal and graph domains, making it suitable for analyzing time-varying graph signals.

A time-varying graph signal $\bar{\mathbf{x}} = \mathrm{vec}\,(\mathbf{X})$ is JWSS if and only if the following conditions are satisfied [10]:

1. **Constant Mean**: The first moment of the process is constant.

$$E[\bar{\mathbf{x}}] = c\mathbf{1}_{NT} \tag{2.39}$$

   where $c \in \mathbb{R}$ is the constant mean value and $\mathbf{1}_{NT} \in \mathbb{R}^{NT}$ represents the vector whose elements consist of ones.

2. **Covariance as a Joint Filter**: The covariance matrix of the process can be written as a joint filter of the joint Laplacian $\mathcal{L}_J$.

$$\mathbf{\Sigma}_{\bar{\mathbf{x}}} = h(\mathcal{L}_J) = h(\mathcal{L}_{\mathcal{G}}, \mathcal{L}_T) = \mathbf{U}_J h(\Lambda_{\mathcal{G}}, \Omega) {\mathbf{U}_J}^H \tag{2.40}$$

where $\mathbf{\Sigma}_{\bar{\mathbf{x}}} \in \mathbb{S}_+^{NT}$ and $h(\Lambda_{\mathcal{G}}, \Omega)$ is the matrix that represents the two dimensional JPSD.

2.3.4 Autoregressive Moving Average Time-Vertex Processes

The extension of ARMA filters from classical signal processing to graph domains has been explored in various studies [27, 28]. A JWSS time-vertex process $\mathbf{X}$ can be represented as an ARMA graph process if it is derived by applying an ARMA graph filter to a zero-mean white process. Specifically, consider an input white process $\mathbf{w}_t \sim \mathcal{N}(0, I_N)$, where the instances $\mathbf{w}_t$ at different time points t are independent. The graph process $\mathbf{x}_t$ at time t is then related to its past values $\mathbf{x}_{t-p}$ and the input process $\mathbf{w}_t$ as follows [28]

$$\mathbf{x}_t = -\sum_{p=1}^{P} a_p(\mathcal{L}_{\mathcal{G}})\mathbf{x}_{t-p} + \sum_{q=0}^{Q} b_q(\mathcal{L}_{\mathcal{G}})\mathbf{w}_{t-q}, \tag{2.41}$$

where $a_p(\mathcal{L}_{\mathcal{G}})$ and $b_q(\mathcal{L}_{\mathcal{G}})$ are graph filters. If these filters are polynomials of the form $a_p(\mathcal{L}_{\mathcal{G}}) = \sum_k a_{pk}\mathcal{L}_{\mathcal{G}}^k$ and $b_q(\mathcal{L}_{\mathcal{G}}) = \sum_m b_{qm}\mathcal{L}_{\mathcal{G}}^m$, where $\mathcal{L}_{\mathcal{G}}^k$ represents the k-th power of the graph Laplacian, the ARMA process model on the graph becomes [27, 28]:

$$\mathbf{x}_t = -\sum_{p=1}^{P}\sum_{k=0}^{K} a_{pk}\mathcal{L}_{\mathcal{G}}^k\mathbf{x}_{t-p} + \sum_{q=0}^{Q}\sum_{m=0}^{M} b_{qm}\mathcal{L}_{\mathcal{G}}^m\mathbf{w}_{t-q}. \tag{2.42}$$

In this model, a_{pk} and b_{qm} are the coefficients of the ARMA graph filters. Given that the input process $\mathbf{w}_t$ is Gaussian, the resulting process $\mathbf{x}_t$ is also Gaussian. By applying the JFT to the process $\mathbf{X}$, the spectral domain representation of the graph filter in (2.42) can be expressed as [28]:

$$H(\lambda_n, \omega_\tau) = \frac{\sum_{q=0}^{Q}\sum_{m=0}^{M} b_{qm}\lambda_n^m e^{-j\omega_\tau q}}{1 + \sum_{p=1}^{P}\sum_{k=0}^{K} a_{pk}\lambda_n^k e^{-j\omega_\tau p}}. \tag{2.43}$$

2.4 Graph Learning Approaches

Graph learning aims to uncover the hidden structure of a graph from observed data. This involves estimating the adjacency matrix $\mathbf{A}$ or the graph Laplacian $\mathcal{L}_{\mathcal{G}}$ from the observations. This is essential for applications where the underlying graph structure

is not explicitly given but must be learned to effectively model, analyze, and process data.

2.4.1 Introduction to Graph Learning

To determine the optimal graph topology that best represents the relationships between data points, estimating the weight matrix or the graph Laplacian is the key approach [12, 13, 29, 30, 31, 32]. These matrices encode the connections and weights between the nodes. The learned graph structure can then be used for various tasks such as clustering, semi-supervised learning, and estimation tasks.

The sparse inverse covariance estimation method [13, 30, 31] is based on the idea that the precision matrix (inverse of the covariance matrix) of a multivariate Gaussian distribution, parametrized with a positive semidefinite precision matrix $\mathbf{\Theta} \in \mathbb{S}_+^N$ defining a Gaussian-Markov Random Field (GMRF), can be used to infer the graph structure. The non-zero entries in the precision matrix correspond to the edges in the graph, which represent the partial correlations between corresponding random variables given all the other random variables in the data set. Formally, given a graph signal data matrix $\mathbf{X} \in \mathbb{R}^{N \times R}$ where N is the number of nodes and R is the number of observations, whose columns $\mathbf{x}_i$ for $i = 1, \ldots, R$ are independent and identically distributed samples with $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ from a GMRF, the unbiased estimation of the covariance matrix $\mathbf{\Sigma}$ is given as

$$\mathbf{\Sigma} \approx \frac{1}{NR-1}\mathbf{X}\mathbf{X}^\intercal. \tag{2.44}$$

The precision matrix $\mathcal{L}_\mathcal{G} = \mathbf{\Theta} = \mathbf{\Sigma}^{-1}$ is then estimated by solving the following optimization problem [13]

$$\begin{aligned} \mathcal{L}_\mathcal{G}^* = \arg\min_{\mathcal{L}_\mathcal{G}} \left(\mathrm{Tr}(\mathbf{\Sigma}\mathcal{L}_\mathcal{G}) - \log|\mathcal{L}_\mathcal{G}|\right) \\ \text{subject to} \quad & \mathcal{L}_\mathcal{G} = \mathcal{L}_\mathcal{G}^\intercal, \\ & \mathcal{L}_\mathcal{G} \succeq 0, \\ & \mathcal{L}_\mathcal{G}\mathbf{1} = \mathbf{0}, \\ & (\mathcal{L}_\mathcal{G})_{ij} \leq 0, \qquad \text{if } (\mathbf{A})_{ij} = 1 \\ & (\mathcal{L}_\mathcal{G})_{ij} = 0 \qquad \text{if } (\mathbf{A})_{ij} = 0 \text{ for } i \neq j \end{aligned} \tag{2.45}$$

where operator $\mathrm{Tr}(.)$ represents the trace operation, $|.|$ denotes the pseudo-determinant (multiplication of the all non-zero eigenvalues of a square matrix), the fourth and the fifth constraints impose the non-positivity of non-diagonal elements of the Laplacian matrix (i.e., nonnegativity of edge weights). The constraints in (2.45) actually state that $\mathcal{L}_\mathcal{G}$ is a singular symmetric, positive-semi definite matrix whose first eigenvalue is 0 and first eigenvector is the $\mathbf{1}$ vector. The graph learning problem in (2.45) can be probabilistically formulated as a parameter estimation problem for GMRF's from data. The likelihood of a candidate graph Laplacian $\mathcal{L}_\mathcal{G}$ can be written as

$$\prod_{i=1}^{R} \mathbf{p}(\mathbf{x}_i \mid \mathcal{L}_\mathcal{G}) = (2\pi)^{-\frac{RN}{2}} |\mathcal{L}_\mathcal{G}^\dagger|^{-\frac{R}{2}} \prod_{i=1}^{R} \exp\left(-\frac{1}{2}\mathbf{x}_i{}^\intercal \mathcal{L}_\mathcal{G} \mathbf{x}_i\right) \tag{2.46}$$

where $(.)^\dagger$ represents the pseudo-inverse. The maximization of the likelihood function in (2.46) can be equivalently formulated as the minimization of the negative log-likelihood, that is [13]

$$\begin{aligned} \mathcal{L}_{\mathcal{G}_{\mathrm{ML}}}^* &= \arg\min_{\mathcal{L}_\mathcal{G}} \left\{ -\frac{R}{2}\log|\mathcal{L}_\mathcal{G}| + \frac{1}{2}\sum_{i=1}^{R} \mathrm{Tr}\left(\mathbf{x}_i{}^\intercal \mathcal{L}_\mathcal{G} \mathbf{x}_i\right) \right\} \\ &= \arg\min_{\mathcal{L}_\mathcal{G}} \left\{ \mathrm{Tr}(\mathcal{L}_\mathcal{G}\boldsymbol{\Sigma}) - \log|\mathcal{L}_\mathcal{G}| \right\} \end{aligned} \tag{2.47}$$

where $\mathcal{L}_{\mathcal{G}_{\mathrm{ML}}}^*$ denotes the maximum likelihood estimate of $\mathcal{L}_\mathcal{G}$.

Another approach that can be used for learning a graph is that it is often assumed that the signals are smooth over the graph [1, 7, 2, 33, 34]. A smooth signal on a graph varies slowly between connected nodes, meaning that adjacent nodes have similar signal values [1]. Mathematically, the smoothness of a signal $\mathbf{x}$ on a graph can be quantified using the quadratic form of the Laplacian matrix $\mathbf{x}^\intercal \mathcal{L}_\mathcal{G} \mathbf{x}$ which is equal to

$$\sum_{(i,j)\in\mathcal{E}} \mathbf{W}_{ij}(x_i - x_j)^2 \tag{2.48}$$

where x_i and x_j are the signal values at nodes i and j, respectively. The smoothness indicator comes from equation (2.48). If the large differences between the signal values of adjacent nodes with higher weights are penalized, then an optimization problem can be formulated as [35, 36]

$$\mathcal{L}_\mathcal{G}^* = \arg\min_{\mathcal{L}_\mathcal{G}} \mathbf{x}^\intercal \mathcal{L}_\mathcal{G} \mathbf{x} + \alpha \|\mathcal{L}_\mathcal{G}\|_F^2 \tag{2.49}$$

where $\alpha \in \mathbb{R}_+$ is a regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm.

The same constraints can be applied to equation (2.49) as in equation (2.45). Learning the graph Laplacian from only the observed data samples is not always the case. Prior information for the graph Laplacian is sometimes available. In these scenarios, we may add the term $\beta\|\mathcal{L}_{\mathcal{G}} - \mathcal{L}_{\mathcal{G}_0}\|_F^2$, where $\beta \in \mathbb{R}_+$ is a regularization parameter and $\mathcal{L}_{\mathcal{G}_0}$ is an initial guess or prior for the Laplacian.

Although learning or improving the graph Laplacian may potentially increase the performance, there are key points that need to be addressed. The optimization problem in equation (2.45) mainly relies on the covariance matrix of the observed samples and if the number of realizations is low then the new learned Laplacian $\mathcal{L}_{\mathcal{G}}^*$ can yield worse performance because of the bad estimation of the covariance matrix. Although the optimization problem in equation (2.49) may not suffer from the covariance estimation, it may suffer from high noise, and this may guide the algorithm wrongly. Also the $\alpha\|\mathcal{L}_{\mathcal{G}}\|_F^2$ term does not necessarily have a direct relation to sparsity. Moreover, learning large-scale graphs can be computationally intensive. Also, dependencies between random variables may extinguish if the regularization parameters are not chosen properly due to sparsity-promoting terms, so parameter tuning can be challenging for some data sets.

2.5 Time-Vertex Signals and Graph Learning

Although the methods in Section 2.4 have achieved some success, they all ignore the characteristics of signals in the time domain. Therefore, more recent works have focused on graph learning with time-varying graph signals. For example, Liu et al. [37] propose estimating a graph from temporal weighted difference signals, where the temporal dynamics are captured by the proposed time-varying graph signal model. Li et al. [38] derive a representation that promotes the smoothness property of the joint graph signal. By decoupling the joint graph, they formulate the graph learning framework as a joint optimization problem that includes signal denoising and the simultaneous learning of time and vertex graphs. Javaheri et al. [39] consider the problem of semi-blind recovery of time-varying graph signals where the underlying graph model is unknown. They assume the Laplacian GMRF model for the temporal difference of the time-vertex signal by using spatio-temporal smoothness [40]. Kadambari et

al. [41] propose a framework for estimating the graph Laplacian matrices of the factors of the product graph from the multidomain training data. They assume that the product graph is constructed from the Cartesian graph product of two smaller factor graphs and the problem of learning the product graph is transformed into the task of estimating the Laplacian matrices of the factor graphs.

2.6 Graph Neural Networks and Graph Attention Networks

2.6.1 Introduction to Graph Neural Networks (GNNs)

A different approach for processing graph signals is inspired from the Convolutional Neural Networks (CNN). Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. These networks leverage the graph structure to capture the dependencies between nodes, making them particularly powerful for tasks such as node classification, link prediction, and graph classification. GNNs typically involve the propagation of node features through the graph, updating each node's representation based on its neighbors' features.

One of the foundational works in this field is the Graph Convolutional Network (GCN) introduced by Kipf and Welling [42]. GCN extends the concept of convolution neural networks to graph-structured data by aggregating feature information from the neighbors of a node. This approach allows the network to learn topological dependencies in the graph, which are crucial for many graph-based tasks. Hamilton et al. proposed GraphSAGE, which learns node embeddings by sampling and aggregating features from the local neighborhood neighborhoods of nodes [43]. Xu et al. proposed GIN, which is powerful enough to distinguish different graph structures and thus improves the discriminative capability of GNNs [44].

2.6.2 Graph Attention Networks (GAT)

Graph Attention Networks (GAT) extend GNNs by incorporating an attention mechanism that allows the model to assign different importance values to different nodes in a neighborhood. This mechanism helps the model to focus on the most relevant

parts of the graph, improving performance in tasks where some connections are more important than others.

The architecture of GAT consists of multiple attention heads, which can be aggregated to form the final node representation. Velickovic et al. introduced the GAT model, demonstrating its effectiveness on several benchmark tasks [45]. By computing attention coefficients for each pair of connected nodes, GATs can dynamically weight the influence of neighbors during the message-passing process. Abu-El-Haija et al. proposed an attention mechanism that models random walks over the graph to capture more global context [46].

2.6.3 Graph Inference and Estimation Problems

GNNs and GATs can address a variety of graph inference and estimation problems. Common tasks include node classification, where the goal is to predict the label of each node, and link prediction, where the objective is to infer missing edges in the graph. Other problems include community detection and graph clustering, which involve identifying groups of closely related nodes.

These tasks are typically formulated as optimization problems, where the network learns to minimize a loss function based on the observed data. For example, in node classification, where GNNs are used to classify nodes in a graph based on their features and the graph structure [42], the loss function may measure the difference between the predicted and true labels of the nodes. In link prediction, GNNs can predict the existence of edges between pairs of nodes, which is useful in social networks and recommender systems [47]. In such applications, the loss function may measure the likelihood of observing the existing edges given the learned embeddings of the nodes.

2.6.4 Graph Learning with GNNs

Graph learning involves using GNNs to learn useful representations of graphs and their components. This can include learning embeddings for nodes, edges, or entire graphs, which can be used for downstream tasks such as clustering or classification.

GNNs can also be used to generate new graphs or to optimize existing graph structures to improve performance on specific tasks.

Advanced techniques in this area include self-supervised learning, where the network learns to generate labels based on the structure of the graph itself, and the integration of GNNs into other machine learning models to enhance their capabilities. For instance, Veličković et al. proposed the method DGI (Deep Graph Infomax) for learning node representations using self-supervised learning from unlabeled data [48]. Grover et al. proposed the method Graph Embeddings for learning low-dimensional representations of nodes or entire graphs for various tasks [49]. You et al. proposed the method Graphrnn using deep auto-regressive models which an be applied in fields like drug discovery [50].

CHAPTER 3

PROPOSED METHOD

This chapter describes the proposed method for jointly learning a graph ARMA process and a graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ from time-vertex signals. In Section 3.1, we demonstrate our process model along with the assumptions. In Section 3.2, we describe the problem and how we aim to solve, detailing the objectives. In Section 3.3, we give the details of the proposed method and each step of the algorithm. In Section 3.4, we show an application of how the proposed method can be used for the spatiotemporal interpolation problem.

3.1 Process Model

In this section, we present the process model and preliminaries necessary for understanding the proposed method, which aims to jointly learn an ARMA graph process model, $\mathbf{a}$ and $\mathbf{b}$ coefficients to construct the JPSD h, and the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. We begin with the signal model and the assumptions underlying our approach.

3.1.1 Signal Model

We consider a time-vertex signal $\mathbf{X} \in \mathbb{R}^{N \times T}$ that is observed on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$. We assume that the random time-vertex signal $\mathbf{X}$ is a JWSS process whose properties are described in the Section 2.3.3. Furthermore, we model this JWSS time-vertex process as an ARMA graph process as in Section 2.3.4.

3.1.2 Laplacian Matrix

The Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ of the graph $\mathcal{G}$ is modeled as deterministic. We assume that there exists an initial guess for $\mathcal{L}_{\mathcal{G}}$, that is $\mathcal{L}_{\mathcal{G}_0}$. $\mathcal{L}_{\mathcal{G}_0}$ is used to construct an ARMA graph process model initially. Then, we use the approach in equation (2.47) to update $\mathcal{L}_{\mathcal{G}}$ based on our estimation of the ARMA graph process model.

3.2 Problem Formulation

The primary objective of this work is to jointly learn an ARMA graph process model and the graph Laplacian matrix such that the estimated model and the Laplacian matrix accurately represent the observed JWSS time-vertex process. We aim to enhance our modeling of data dependencies by representing the observed data as a JWSS ARMA graph process. Additionally, we refine the Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ to gain a deeper understanding of the underlying dynamics, thereby enabling more accurate estimations and predictions.

3.2.1 A priori Information

1. **Available Realizations**: In this work, we consider a scenario where L realizations $\{\mathbf{X}^l\}_{l=1}^{L}$ of the time-vertex process $\mathbf{X}$ are available. Each realization $\mathbf{X}^l \in \mathbb{R}^{N\times T}$ is assumed to be fully observed, although our algorithm can also be used in a scenario where only partial observations of the realizations are available. The motivation behind the full observation assumption of the available realizations is that we aim to learn both the graph topology and the process parameters, so training the model on fully available observations is expected to improve the model estimation accuracy.

2. **Initial Erroneous Laplacian Matrix**: In this work, we assume that an initial Laplacian matrix $\mathcal{L}_{\mathcal{G}_0}$, although it might be erroneous, is available.

3.2.2 Objective

The primary objective of this work is to learn an ARMA graph process model and its underlying graph structure through the graph Laplacian matrix $\mathcal{L}_\mathcal{G}$ jointly. Given the fully observed realizations of the time-vertex process $\{\mathbf{X}^l\}_{l=1}^L$, our goal is to refine the Laplacian matrix $\mathcal{L}_\mathcal{G}$ to better understand the inherent structure of the graph, and accurately model the dependencies in the data by fitting an ARMA graph process.

Let $\hat{\mathbf{X}}(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G})$ denote the modeled process. Fitting $\mathbf{X}$ to $\hat{\mathbf{X}}(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G})$ is a jointly non-convex optimization problem. Although learning the ARMA model and the graph Laplacian jointly is challenging, it is a rewarding task. We can express our learning task in form of a general optimization problem as follows

$$\min_{\{\mathbf{a},\mathbf{b},\mathcal{L}_\mathcal{G}\}} \quad f\left(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G}, \tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}\right) + r(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G}) \tag{3.1}$$

where $f(.)$ represents a data fidelity term and $r(.)$ represents a regularization term. Optimizing the functions in (3.1) is difficult. The objective function exhibits a fourth-order dependency when fitting the parameters $\mathbf{a}$ and $\mathbf{b}$ to the initial estimate of the JPSD, as discussed in [11]. Moreover, the optimization problem involves powers of the eigenvalues of $\mathcal{L}_\mathcal{G}$, making the objective function dependent on $\mathcal{L}_\mathcal{G}$ in a non-convex manner. Hence, we propose to split the parts learning the ARMA coefficients $\mathbf{a}$ and $\mathbf{b}$, and learning the Laplacian matrix $\mathcal{L}_\mathcal{G}$. In this case, the optimization problem can be reformulated as follows

$$\begin{aligned} \min_{\{\mathbf{a},\mathbf{b},\mathcal{L}_\mathcal{G}\}} \quad & f_1\left(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G}, \tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}\right) + r_1(\mathbf{a}, \mathbf{b}) \\ & + f_2\left(\mathcal{L}_\mathcal{G}, \mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})\right) + r_2(\mathcal{L}_\mathcal{G}). \end{aligned} \tag{3.2}$$

The term $f_1\left(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G}, \tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}\right)$ captures the coherence between the process parameters $\mathbf{a}$, $\mathbf{b}$ and the data statistics for a given graph topology, while $r_1(\mathbf{a}, \mathbf{b})$ serves as the regularization term in ARMA graph process fitting. Similarly, $f_2\left(\mathcal{L}_\mathcal{G}, \mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})\right)$ represents the data fidelity component related to the learning of the graph Laplacian matrix $\mathcal{L}_\mathcal{G}$, with $r_2(\mathcal{L}_\mathcal{G})$ acting as its corresponding regularization term. We give the explicit forms of these terms in Section 3.3.

3.3 The Overall Algorithm

The joint learning of the two components in equation (3.2) can be achieved via an iterative algorithm. In the first step, the f_1 and r_1 functions are minimized together to update the ARMA coefficients while $\mathcal{L}_\mathcal{G}$ is fixed. In this step, we always fix the covariance matrix to its initial value $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ calculated from the data. In the second step, when minimizing f_2 and r_2, an updated version of the covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$ is calculated according to the coefficient vectors $\mathbf{a}$ and $\mathbf{b}$, which are fixed to their values learnt in the first step. This updated version of the covariance matrix is used in the optimization of $\mathcal{L}_\mathcal{G}$. Then f_2 and r_2 can be minimized together to update the $\mathcal{L}_\mathcal{G}$ matrix. This optimized $\mathcal{L}_\mathcal{G}$ is then used in subsequent iterations instead of the initial guess to update the parameters of the graph ARMA process. These steps can be repeated until a predetermined number of iterations is reached. We call this method Graph Learning JS-ARMA (GL-JS-ARMA).

Algorithm 1 Iterative ARMA Graph Process and Laplacian Matrix Optimization. GL-JS-ARMA

Inputs: Initial Laplacian matrix $\mathcal{L}_{\mathcal{G}_0}$, initial covariance matrix $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$

Set iteration counter $k = 1$

WHILE k is smaller than maximum iteration number

DO

Update ARMA coefficients: Fix $\mathcal{L}_\mathcal{G} = \mathcal{L}_{\mathcal{G}_k}$.

$$(\mathbf{a}_k, \mathbf{b}_k) = \arg\min_{\mathbf{a},\mathbf{b}} f_1\left(\mathbf{a}, \mathbf{b}, \mathcal{L}_\mathcal{G}, \tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}\right) + r_1(\mathbf{a}, \mathbf{b})$$

Calculate the new covariance matrix: Using the updated $\mathbf{a}_k$, $\mathbf{b}_k$, calculate the covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$ via equations (2.40) and (3.14).

Update Laplacian matrix: Fix $\mathbf{a} = \mathbf{a}_k$ and $\mathbf{b} = \mathbf{b}_k$.

$$\mathcal{L}_{\mathcal{G}_{k+1}} = \arg\min_{\mathcal{L}_\mathcal{G}} f_2\left(\mathcal{L}_\mathcal{G}, \mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})\right) + r_2(\mathcal{L}_\mathcal{G})$$

$k = k + 1$

ENDWHILE

Output: Optimized Laplacian matrix $\mathcal{L}_\mathcal{G}$, optimized ARMA coefficients $\mathbf{a}$ and $\mathbf{b}$

In each iteration of the optimization, we alternate between fixing $\{\mathbf{a}, \mathbf{b}\}$ and optimizing $\mathcal{L}_\mathcal{G}$, and fixing $\mathcal{L}_\mathcal{G}$ and optimizing $\{\mathbf{a}, \mathbf{b}\}$. Between these two steps, we calculate the $\mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$ according to the updated values of the coefficient vectors $\{\mathbf{a}, \mathbf{b}\}$.

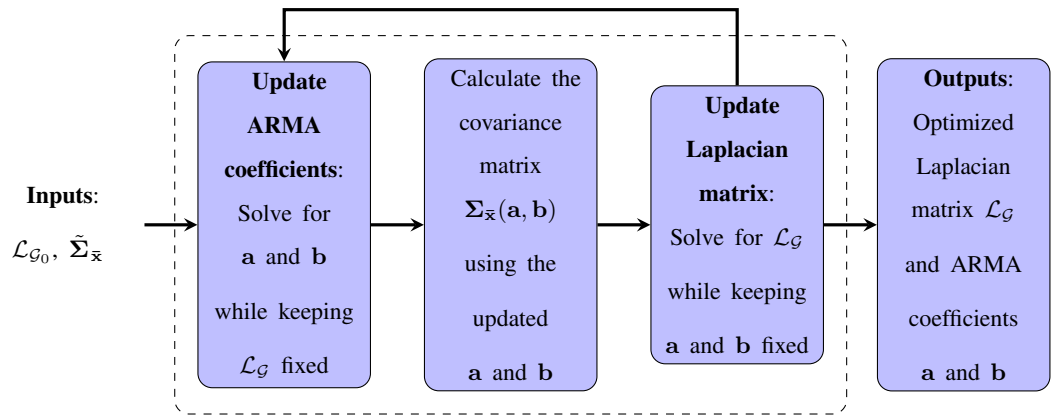

Figure 3.1: The flow chart of GL-JS-ARMA algorithm in 1

Each step in Algorithm 1 is elaborated upon in the following subsections. We begin with Subsection 3.3.1, where the calculation of the initial covariance matrix $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ is explained. This is followed by Subsection 3.3.2, which details the computation of the graph ARMA process parameters, $\mathbf{a}$ and $\mathbf{b}$. The algorithm subsections conclude with the computation of the matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$, followed by the learning of the graph Laplacian matrix.

3.3.1 Initial Covariance Matrix and JPSD

We recall from Chapter 2 that a time-vertex stochastic process modeled as a graph ARMA process (2.42) is also JWSS, as described in the equations (2.39) and (2.40). Thus, the process $\bar{\mathbf{x}}$ has a constant mean, and its covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ can be jointly diagonalized with the joint Laplacian $\mathcal{L}_J$. Consequently, the spectral domain representation of the graph filter is in the form of (2.43).

The covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ of a zero-mean time-vertex process $\mathbf{X}$ is given by

$$\mathbf{\Sigma}_{\bar{\mathbf{x}}} = E[\bar{\mathbf{x}}\bar{\mathbf{x}}^{\intercal}] = \begin{bmatrix} \mathbf{\Sigma}_{1,1} & \mathbf{\Sigma}_{1,2} & \cdots & \mathbf{\Sigma}_{1,T} \\ \mathbf{\Sigma}_{2,1} & \mathbf{\Sigma}_{2,2} & \cdots & \mathbf{\Sigma}_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Sigma}_{T,1} & \mathbf{\Sigma}_{T,2} & \cdots & \mathbf{\Sigma}_{T,T} \end{bmatrix} \tag{3.3}$$

where $\mathbf{\Sigma}_{t,u} = E[\mathbf{x}_t\, \mathbf{x}_u^{\intercal}]$ stands for the covariance matrix of the values of the process at time instants t and u. When $\mathbf{X}$ is a JWSS process, the covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ has a

special property that simplifies its estimation. Each covariance matrix $\mathbf{\Sigma}_{t,u}$ is a graph filter $\mathbf{\Sigma}_{t,u} = g_{t,u}(\mathcal{L}_{\mathcal{G}})$, which depends only on the time difference $t-u$. This results in a block-Toeplitz structure in $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ [10]. Given that any graph filter $g(\mathcal{L}_{\mathcal{G}})$ must be symmetric, and that the overall covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ is also symmetric, it follows that $\mathbf{\Sigma}_{t,u} = \mathbf{\Sigma}_{u,t}$. Therefore, estimating $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ reduces to estimating the smaller matrices $\tilde{\mathbf{\Sigma}}_{\Delta} \in \mathbb{R}^{N\times N}$ for $\Delta = 0, 1, \ldots, T-1$, where $\tilde{\mathbf{\Sigma}}_{t,u} = \tilde{\mathbf{\Sigma}}_{\Delta}$ with $\Delta = |t-u|$. We estimate the covariance matrix $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ based on the estimation of the smaller covariance matrices $\tilde{\mathbf{\Sigma}}_{\Delta}$'s. The unbiased covariance estimate $\tilde{\mathbf{\Sigma}}_{\Delta}$ for $\Delta = \tau$ with circular shift for a zero mean time-vertex JWSS process is given by

$$\tilde{\mathbf{\Sigma}}_{\tau}(i,j) = \frac{1}{RT-1}\sum_{r=1}^{R}\sum_{t=1}^{T}\mathbf{X}_r(i,t)\,\mathbf{X}_r(j,(t+\tau) \bmod T) \quad \text{for } \tau = 0, 1, \ldots, T-1 \tag{3.4}$$

where $\mathbf{X}_r(i,t)$ is the value of the time-vertex signal at node i, time t, and realization r. The circular shift reorders the time series while preserving the intrinsic structure of the JWSS processes. The initial estimate $\tilde{h}(\lambda_n, \omega_\tau)$ of the JPSD can be calculated by extracting the diagonal entries of the matrix

$$\tilde{h}(\Lambda_{\mathcal{G}}, \Omega) = \mathbf{U}_J^H \tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}} \mathbf{U}_J. \tag{3.5}$$

Another estimation of the JPSD, which is based on the Wiener-Khinchin theorem [51], is given by

$$\tilde{h}(\lambda_n, \omega_\tau) = \frac{1}{R}\sum_{r=1}^{R} |\widehat{\mathbf{X}_r}(n,\tau)|^2 \tag{3.6}$$

where $\widehat{\mathbf{X}_r}(n,\tau)$ is the JFT of the r^{th} realization of $\mathbf{X}$, R is the number of realizations, and $\tilde{h}(\lambda_n, \omega_\tau)$ is the estimate of JPSD of the time-vertex process $\mathbf{X}$. The reason why we select equation (3.4) for the calculation of $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ is justified in the results in Table 3.1. A synthetic data set is studied with the parameters $N = 33, T = 30, P = 1, K = 1, Q = 1, M = 0$ with different numbers of realizations. The graph is constructed from a sensor network from the GSP toolbox [52]. The resulting PSD can be seen below.

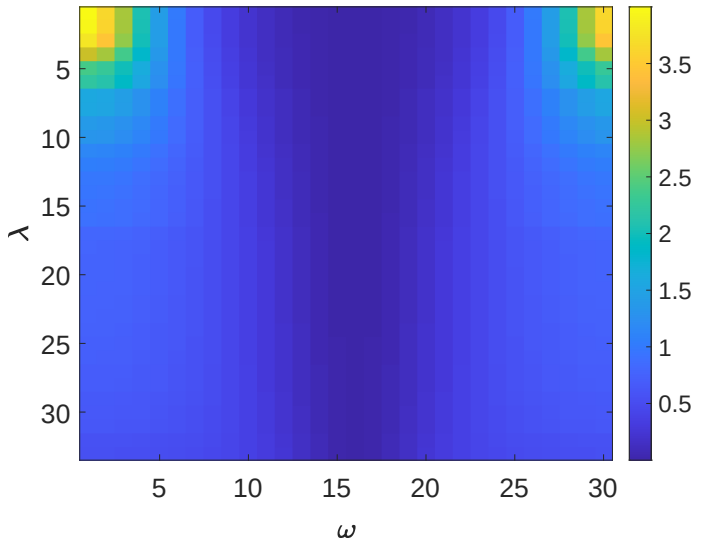

Figure 3.2: Low-Pass PSD: $h(\lambda, \omega)$

Here, λ denotes the graph eigenvalues and ω denotes the time frequencies. The first column of this figure represents the DC frequency in the spectral domain and due to symmetric property of the DFT we can observe the symmetry between columns $2, 3, \cdots, \frac{T}{2}$ and $(\frac{T}{2}+2), \cdots, T$. We compare the normalized errors of the estimation methods in (3.4) and (3.6) against the ground truth covariance matrix. The normalized estimation error is defined as

$$\frac{\|\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}} - \mathbf{\Sigma}_{\bar{\mathbf{x}}}\|_F}{\|\mathbf{\Sigma}_{\bar{\mathbf{x}}}\|_F} \tag{3.7}$$

where $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ represents the estimated covariance matrix from the observed realizations and $\mathbf{\Sigma}_{\bar{\mathbf{x}}}$ represents the ground truth covariance matrix of the time-vertex process. We vary the number of realizations in order to observe its effect over the estimation error

and report the results in Table 3.1. The same parameters are used for each realization.

Normalized Covariance Estimation Error for N = 33, T = 30		
Number of Realizations	**Equation** (3.6) **is used**	**Equation** (3.4) **is used**
R = 50	0.151	0.141
R = 150	0.0789	0.0752
R = 1e4	0.0122	0.0114
R = 5e4	0.00492	0.00457

Table 3.1: Normalized covariance estimation errors for different realizations using different estimation methods

The advantage of the covariance estimate in (3.4) over (3.6) can be seen from Table 3.1, where the unbiased circular covariance estimator performs better than the JFT approach.

3.3.2 Updating the ARMA Coefficients

This part of the algorithm makes use of the work of Guneyi et al which we call Joint Spectra-ARMA (JS-ARMA), [11]. We fit the initial JPSD (3.5) to the JPSD of an ARMA graph process and learn the parameters $\mathbf{a}$ and $\mathbf{b}$. We first rewrite the filter spectrum in (2.43) as

$$H(\lambda_n, \omega_\tau) = \frac{\mathbf{b}^H \mathbf{u}_{n,\tau}}{1 + \mathbf{a}^H \mathbf{v}_{n,\tau}} \tag{3.8}$$

where the vectors $\mathbf{a} \in \mathbb{R}^{P(K+1)\times 1}$ and $\mathbf{b} \in \mathbb{R}^{(Q+1)(M+1)\times 1}$ respectively consist of the filter coefficients a_{pk} and b_{qm} as

$$\begin{aligned} \mathbf{a} &= [a_{10}\ a_{11}\ \cdots a_{pk}\ \cdots a_{PK}]^H \\ \mathbf{b} &= [b_{00}\ b_{01}\ \cdots b_{qm}\ \cdots b_{QM}]^H. \end{aligned} \tag{3.9}$$

The vectors $\mathbf{v}_{n,\tau} \in \mathbb{C}^{P(K+1)\times 1}$ and $\mathbf{u}_{n,\tau} \in \mathbb{C}^{(Q+1)(M+1)\times 1}$ consist of the constant coefficients

$$\begin{aligned} \mathbf{v}_{n,\tau} &= [\lambda_n^0 e^{j\omega_\tau 1}\ \lambda_n^1 e^{j\omega_\tau 1}\ \cdots\ \lambda_n^k e^{j\omega_\tau p}\ \cdots\ \lambda_n^K e^{j\omega_\tau P}]^H \\ \mathbf{u}_{n,\tau} &= [\lambda_n^0 e^{j\omega_\tau 0}\ \lambda_n^1 e^{j\omega_\tau 0}\ \cdots\ \lambda_n^m e^{j\omega_\tau q}\ \cdots\ \lambda_n^M e^{j\omega_\tau Q}]^H \end{aligned} \tag{3.10}$$

where λ_n^k denotes the k-th power of the n-th graph eigenvalue λ_n, and the frequency variables ω_τ are as defined in (2.27).

The JPSD of the process is equal to the magnitude square of the filter spectrum [8],

$$h(\lambda_n, \omega_\tau) = |H(\lambda_n, \omega_\tau)|^2 = \left| \frac{\mathbf{b}^H \mathbf{u}_{n,\tau}}{1 + \mathbf{a}^H \mathbf{v}_{n,\tau}} \right|^2. \tag{3.11}$$

Then, the estimation of the ARMA graph process model parameters from the initially estimated JPSD can be expressed as

$$\min_{\mathbf{a},\mathbf{b}} \sum_{n=1}^{N} \sum_{\tau=1}^{T} \left| \left| \frac{\mathbf{b}^H \mathbf{u}_{n,\tau}}{1 + \mathbf{a}^H \mathbf{v}_{n,\tau}} \right|^2 - \tilde{h}(\lambda_n, \omega_\tau) \right|^2. \tag{3.12}$$

The optimization problem in (3.12) is nonconvex with non-unique minima, and hence difficult to solve. If the equation (2.43) is reformulated as

$$H(\lambda_n, \omega_\tau) = \frac{\sum_{q=0}^{Q} \sum_{m=0}^{M} b_{qm} \lambda_n^m e^{-j\omega_\tau q}}{\sum_{p=0}^{P} \sum_{k=0}^{K} a_{pk} \lambda_n^k e^{-j\omega_\tau p}}. \tag{3.13}$$

where the coefficients are set as $a_{00} = 1$ and $a_{0k} = 0$ for $k = 1, 2, \cdots, K$, then the JPSD of the process has a simpler form

$$h(\lambda_n, \omega_\tau) = \left| \frac{\mathbf{b}^H \mathbf{u}_{n,\tau}}{\tilde{\mathbf{a}}^H \tilde{\mathbf{v}}_{n,\tau}} \right|^2. \tag{3.14}$$

The vectors $\tilde{\mathbf{a}} \in \mathbb{R}^{(P+1)(K+1)\times 1}$ and $\tilde{\mathbf{v}}_{n,\tau} \in \mathbb{C}^{(P+1)(K+1)\times 1}$ are defined to be augmented versions of $\mathbf{a}$ and $\mathbf{v}_{n,\tau}$ as [11]

$$\begin{aligned} \tilde{\mathbf{a}} &= [a_{00}\ a_{01}\ \cdots\ a_{pk}\ \cdots\ a_{PK}]^H \\ \tilde{\mathbf{v}}_{n,\tau} &= [\lambda_n^0 e^{j\omega_\tau 0}\ \lambda_n^1 e^{j\omega_\tau 0}\ \cdots\ \lambda_n^k e^{j\omega_\tau p}\ \cdots\ \lambda_n^K e^{j\omega_\tau P}]^H. \end{aligned} \tag{3.15}$$

In addition to this change of variables, the term in the denominator is removed as proposed in [11] and the final form of the optimization problem becomes

$$\min_{\tilde{\mathbf{a}},\mathbf{b}} \sum_{n=1}^{N} \sum_{\tau=1}^{T} \left| \mathbf{u}_{n,\tau}^H \mathbf{b}\mathbf{b}^H \mathbf{u}_{n,\tau} - \tilde{\mathbf{v}}_{n,\tau}^H \tilde{\mathbf{a}}\tilde{\mathbf{a}}^H \tilde{\mathbf{v}}_{n,\tau}\ \tilde{h}(\lambda_n, \omega_\tau) \right|^2. \tag{3.16}$$

The objective function in (3.16) acts as an alternative to the one in (3.12), though they typically produce different solutions. However, if the denominator term in (3.12) does not become arbitrarily small, indicating that the spectrum has a bounded magnitude, minimizing (3.16) is likely to provide a satisfactory estimate for the solution of (3.12). On the other hand, the optimization function is still nonconvex because of the vectors

$\tilde{\mathbf{a}}$ and $\mathbf{b}$. Therefore, the study in [11] proposes using matrices $\mathbf{A}$ and $\mathbf{B}$, defined as $\mathbf{A} \triangleq \tilde{\mathbf{a}}\tilde{\mathbf{a}}^H$ and $\mathbf{B} \triangleq \mathbf{b}\mathbf{b}^H$. For these definitions to hold, the matrices $\mathbf{A}$ and $\mathbf{B}$ must be rank-1 and positive semidefinite. Thus, the optimization problem becomes

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{B}} \sum_{n=1}^{N}\sum_{\tau=1}^{T} \eta(\lambda_n,\omega_\tau)\left|\mathbf{u}_{n,\tau}^H\mathbf{B}\,\mathbf{u}_{n,\tau} - \tilde{\mathbf{v}}_{n,\tau}^H\mathbf{A}\,\tilde{\mathbf{v}}_{n,\tau}\,\tilde{h}(\lambda_n,\omega_\tau)\right|^2 \\ &\text{subject to} \quad \operatorname{rank}(\mathbf{A})=1,\ \operatorname{rank}(\mathbf{B})=1, \\ &\qquad\qquad \mathbf{A}\in\mathbb{S}_+^{(P+1)(K+1)},\ \ \mathbf{B}\in\mathbb{S}_+^{(Q+1)(M+1)}, \\ &\qquad\qquad a_{00}=1,\ \ a_{0k}=0 \text{ for } k=1,2,\cdots,K \end{aligned} \tag{3.17}$$

where $\eta(\cdot,\cdot)$ stands for an optional weight function for adaptively penalizing the error at particular zones of the joint spectrum, which can be chosen as $\mu(\lambda_n,\omega_\tau)=1$ under no priors. It can be chosen so as to guide the optimization problem such that the resulting graph ARMA process has a low-pass JPSD, which is a realistic assumption for most applications. The rank one constraints are nonconvex, so instead of directly applying the rank one constraints the positive semidefinite matrices $\mathbf{A}$ and $\mathbf{B}$ can be pushed to be low-rank by minimizing the sums of their singular values, or equivalently, their traces $\operatorname{Tr}(\mathbf{A})$ and $\operatorname{Tr}(\mathbf{B})$ as in the work [11]. The final form of the optimization problem is obtained as

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{B}} \sum_{n=1}^{N}\sum_{\tau=1}^{T} \eta(\lambda_n,\omega_\tau)\left|\mathbf{u}_{n,\tau}^H\mathbf{B}\,\mathbf{u}_{n,\tau} - \tilde{\mathbf{v}}_{n,\tau}^H\mathbf{A}\,\tilde{\mathbf{v}}_{n,\tau}\,\tilde{h}(\lambda_n,\omega_\tau)\right|^2 \\ &+\mu_A\operatorname{Tr}(\mathbf{A})+\mu_B\operatorname{Tr}(\mathbf{B}), \quad \text{subject to} \quad \mathbf{A}\in\mathbb{S}_+^{(P+1)(K+1)}, \\ &\mathbf{B}\in\mathbb{S}_+^{(Q+1)(M+1)},\ \ a_{00}=1,\ \ a_{0k}=0 \text{ for } k=1,2,\cdots,K \end{aligned} \tag{3.18}$$

where μ_A and μ_B are non-negative weight parameters. The objective function in (3.18) is quadratic and jointly convex in $\mathbf{A}$ and $\mathbf{B}$. Therefore, the f_1 term in equation (3.2) can be expressed as $\sum_{n=1}^{N}\sum_{\tau=1}^{T}\eta(\lambda_n,\omega_\tau)\left|\mathbf{u}_{n,\tau}^H\mathbf{B}\,\mathbf{u}_{n,\tau} - \tilde{\mathbf{v}}_{n,\tau}^H\mathbf{A}\,\tilde{\mathbf{v}}_{n,\tau}\,\tilde{h}(\lambda_n,\omega_\tau)\right|^2$, and the r_1 term can be written as $\mu_A\operatorname{Tr}(\mathbf{A})+\mu_B\operatorname{Tr}(\mathbf{B})$.

The constraint set consists of linear equality constraints, and the positive semidefiniteness of the $\mathbf{A}$ and $\mathbf{B}$ matrices. Hence, (3.18) is a convex problem that can be solved using convex optimization techniques [53, 54] relying on semidefinite quadratic linear programming [55]. Once the matrices $\mathbf{A}$ and $\mathbf{B}$ are computed by solving (3.18), the ARMA model parameter vectors $\mathbf{a}$ and $\mathbf{b}$ can be recovered through rank-1 decompositions of $\mathbf{A}$ and $\mathbf{B}$.

3.3.3 Calculation of the Covariance Matrix $\boldsymbol{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$

The covariance matrix $\boldsymbol{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$ is determined as a result of solving the optimization problem (3.18). Once the coefficients $\mathbf{a}$ and $\mathbf{b}$ are obtained, the JPSD of the process is computed using equation (3.14). Following this, $\tilde{\boldsymbol{\Sigma}}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$ can be derived from equation (2.40). This intermediate covariance matrix is then used in Section 3.3.4. For a fixed time t in an ARMA graph process, the covariance matrix of the process corresponds to $\boldsymbol{\Sigma}_{\Delta}$ with $\Delta = 0$. Thus, we utilize the first $N \times N$ block of the covariance matrix $\boldsymbol{\Sigma}_{\bar{\mathbf{x}}}(\mathbf{a}, \mathbf{b})$.

3.3.4 Updating the Laplacian Matrix

This part of the algorithm builds on the work of Egilmez et al, [13]. In order to update the graph Laplacian $\mathcal{L}_{\mathcal{G}}$ coherently with the learned graph ARMA process, the sparse inverse covariance estimation method in Section 2.4.1 is used. The graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ can be updated via equation (2.45), with the addition of a regularization term as [13]

$$
\begin{aligned}
\mathcal{L}_{\mathcal{G}}^{\star} = \arg\min_{\mathcal{L}_{\mathcal{G}}} & \Big(\mathrm{Tr}(\tilde{\boldsymbol{\Sigma}}_0 \mathcal{L}_{\mathcal{G}}) - \log|\mathcal{L}_{\mathcal{G}}| + \|\mathcal{L}_{\mathcal{G}} \odot \mathbf{F}\|_1 \Big) \\
\text{subject to} \quad & \mathcal{L}_{\mathcal{G}} = \mathcal{L}_{\mathcal{G}}^{\mathsf{T}}, \\
& \mathcal{L}_{\mathcal{G}} \succeq 0, \\
& \mathcal{L}_{\mathcal{G}} \mathbf{1} = \mathbf{0}, \\
& (\mathcal{L}_{\mathcal{G}})_{ij} \leq 0, \qquad \text{if } (\mathbf{A})_{ij} = 1 \\
& (\mathcal{L}_{\mathcal{G}})_{ij} = 0 \qquad \text{if } (\mathbf{A})_{ij} = 0 \text{ for } i \neq j
\end{aligned}
\tag{3.19}
$$

where $\|\mathcal{L}_{\mathcal{G}} \odot \mathbf{F}\|_1$ is the sparsity promoting weighted ℓ_1-regularization term multiplying $\mathcal{L}_{\mathcal{G}}$ and $\mathbf{F}$ element-wise. $\mathbf{F} \in \mathbb{R}^{N \times N}$ is a symmetric regularization matrix and the sparsity promoting term can be compactly written as

$$
\|\mathcal{L}_{\mathcal{G}} \odot \mathbf{F}\|_1 = \mathrm{Tr}(\mathcal{L}_{\mathcal{G}} \mathbf{F}). \tag{3.20}
$$

In this work, the Laplacian $\mathcal{L}_{\mathcal{G}}$ is modeled as deterministic. As a side note, in previous studies relying on a Bayesian approach [13, 31], the sparsity-promoting regularization term has been shown to provide the following insightful interpretation: The maximum

a posteriori (MAP) estimate of $\mathcal{L}_\mathcal{G}$ has been obtained in these works by incorporating the prior knowledge about $\mathcal{L}_\mathcal{G}$ into a prior distribution $\mathbf{p}(\mathcal{L}_\mathcal{G})$ as

$$\hat{\mathcal{L}_\mathcal{G}}_{\text{MAP}} = \arg\min_{\mathcal{L}_\mathcal{G}} \left\{ \text{Tr}(\tilde{\mathbf{\Sigma}}_0 \mathcal{L}_\mathcal{G}) - \log|\mathcal{L}_\mathcal{G}| - \log(\mathbf{p}(\mathcal{L}_\mathcal{G})) \right\}. \tag{3.21}$$

For example, the following M-variate exponential prior distribution is chosen to promote the sparsity of the edge weight vector $\mathbf{w}$

$$(2\alpha)^M \exp(-2\alpha \mathbf{1}^\top \mathbf{w}) \quad \text{for } \mathbf{w} \geq 0, \tag{3.22}$$

then the log-likelihood of $\mathbf{w}$ is captured by the term $2\alpha\|\mathbf{w}\|_1 = \alpha\|\mathcal{L}_\mathcal{G}\|_{1,\text{off}}$ where $\|.\|_{1,\text{off}}$ denotes the ℓ_1 norm of the vectorized form of a square matrix by excluding its diagonal elements. In this case, (3.21) can be written as follows

$$\hat{\mathcal{L}_\mathcal{G}}_{\text{MAP}} = \arg\min_{\mathcal{L}_\mathcal{G}} \left\{ \text{Tr}(\mathcal{L}_\mathcal{G} \tilde{\mathbf{\Sigma}}_0) - \log|\mathcal{L}_\mathcal{G}| + \alpha\|\mathcal{L}_\mathcal{G}\|_{1,\text{off}} \right\}. \tag{3.23}$$

Therefore, the f_2 term in equation (3.2) can be expressed as $\text{Tr}(\mathcal{L}_\mathcal{G}\tilde{\mathbf{\Sigma}}_0) - \log|\mathcal{L}_\mathcal{G}|$, and the r_2 term can be written as $\alpha\|\mathcal{L}_\mathcal{G}\|_{1,\text{off}}$. The standard ℓ_1 regularization term with a parameter α can be written as

$$\alpha\|\mathcal{L}_\mathcal{G}\|_{1,\text{off}} = \text{Tr}(\mathcal{L}_\mathcal{G}\mathbf{F}), \quad \mathbf{F} = \alpha(2\mathbf{I} - \mathbf{1}\mathbf{1}^\intercal) \tag{3.24}$$

where α can be regarded as a non-negative regularization parameter that controls the sparsity of the Laplacian (precision) matrix.

In order to solve the optimization problem in (3.19), some reformulation steps are applied as in the work [13]. Since the operator Tr is linear, the problem in (3.19) can be rewritten as

$$\arg\min_{\mathcal{L}_\mathcal{G}} \{\text{Tr}(\mathcal{L}_\mathcal{G}\mathbf{K}) - \log|\mathcal{L}_\mathcal{G}|\} \text{ where } \mathbf{K} = \tilde{\mathbf{\Sigma}}_0 + \mathbf{F}. \tag{3.25}$$

Since the graph Laplacian matrix $\mathcal{L}_\mathcal{G}$ is a rank deficient matrix whose determinant is 0, the term $\log|\mathcal{L}_\mathcal{G}|$ leads to an inconvenience for the problem (3.25). To cope with this difficulty, the optimization problem can be reformulated as

$$\begin{aligned}
\mathcal{L}_\mathcal{G}^\star = \arg\min_{\mathcal{L}_\mathcal{G}} & \left(\text{Tr}(\mathcal{L}_\mathcal{G}(\mathbf{K} + \mathbf{J})) - \log\det(\mathcal{L}_\mathcal{G} + \mathbf{J})\right) \\
\text{subject to} \quad & \mathcal{L}_\mathcal{G} = \mathcal{L}_\mathcal{G}^\intercal, \\
& \mathcal{L}_\mathcal{G} \succeq 0, \\
& \mathcal{L}_\mathcal{G}\mathbf{1} = \mathbf{0}, \\
& (\mathcal{L}_\mathcal{G})_{ij} \leq 0, \qquad \text{if } (\mathbf{A})_{ij} = 1 \\
& (\mathcal{L}_\mathcal{G})_{ij} = 0 \qquad \text{if } (\mathbf{A})_{ij} = 0 \text{ for } i \neq j
\end{aligned} \tag{3.26}$$

where $\mathbf{J} = \mathbf{u}_1\mathbf{u}_1^\intercal = (1/N)\mathbf{1}\mathbf{1}^\intercal$ such that $\mathbf{u}_1$ is the eigenvector corresponding to the zero eigenvalue of the combinatorial graph Laplacian matrix. The optimization problem in (3.26) is convex and can be solved by an iterative block-coordinate descent algorithm [13, 56]. The prior knowledge about the graph structure is built into the choice of the adjacency matrix $\mathbf{A}$, determining the structural constraints. For example, the edges of a $\mathcal{L}_{\mathcal{G}_0}$ and the highly correlated indices of $\tilde{\mathbf{\Sigma}}_0$ can be used to guide the connectivity structure of $\mathcal{L}_{\mathcal{G}}$ along with the parameter α until the desired level of sparsity is achieved. These structural constraints guide the algorithm by defining the permissible edges, which is a crucial detail for determining the edge places.

3.3.5 Complexity Analysis

In this subsection, we analyze the computational complexity of the algorithm, focusing on the distinct complexities of learning the graph ARMA model and the graph Laplacian matrix. The ARMA model involves solving for the coefficients $\mathbf{a}$ and $\mathbf{b}$, calculating the joint eigenbasis $\mathbf{U}_J$, and estimating the covariance matrix $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$. The complexity depends on factors such as the model order, the time series length T, the node dimension N, and the number of realizations R. First, computing the estimated covariance matrix $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ has a complexity of $O(N^2TR)$ due to its block-Toeplitz structure. Next, calculating the joint eigenbasis $\mathbf{U}_J$ involves computing $\mathbf{U}_{\mathcal{G}}$ with a complexity of $O(N^3)$, and the Kronecker product with $\mathbf{U}_T$ incurs an additional complexity of $O(N^2T^2)$. To solve equation (3.18), the initial JPSD $\tilde{h}$ must be calculated, which has a complexity of $O(N^3T^3)$. Assuming that the model parameters P, K, Q, M are significantly smaller than N and T, the optimization problem (3.18) can be solved using the HKM algorithm [11, 55], with a complexity of $O(NT)$. Therefore, the overall complexity of learning the graph ARMA coefficients, assuming the model parameters are much smaller than the process dimensions, is $O(N^3T^3)$.

The learning of the graph Laplacian matrix in this work employs a block-coordinate descent algorithm, as described in [13, 56]. The computational complexity of this algorithm is generally $O(N^3)$, where N represents the number of nodes in the graph. This cubic complexity arises from the fact that in each iteration, the algorithm updates one block (or subset) of the inverse of the $\mathcal{L}_{\mathcal{G}}$'s variables, solving for the optimal

Laplacian matrix based on the current estimates of the other variables. However, the actual complexity can be significantly reduced depending on the sparsity of the graph structure. If the graph is sparse, meaning that each node is connected to only a few other nodes, the complexity of the algorithm scales with the maximum number of edges connected to any node. In such cases, the complexity is reduced to $O(s^3)$, where s is the maximum degree of a node, i.e., the largest number of edges incident to any single node in the graph. Since in many real-world applications, graphs tend to be sparse (with $s \ll N$), the computational cost of learning the graph Laplacian can be considerably lower than the worst-case $O(N^3)$ scenario. This sparsity-based optimization allows the algorithm to be scalable and more efficient for large-scale graphs, where only a fraction of the possible edges are present.

A single iteration of the proposed method has a computational complexity of $O(N^3T^3)$. This complexity arises from the steps involved in calculating the joint spectral basis, the covariance matrices, and solving for the ARMA coefficients and the graph Laplacian. Given that the method typically involves multiple iterations, the overall computational complexity scales with the total number of iterations. Thus, if L represents the maximum number of iterations, the total complexity of the algorithm can be expressed as $O(N^3T^3L)$. In practice, L is generally much smaller than the number of nodes or the time series length, but it still contributes to the overall runtime of the method.

3.4 Application of the Proposed Algorithm in Spatiotemporal Interpolation Problems

In this section, we demonstrate the usage of our algorithm proposed in Section 3.3 in spatiotemporal signal interpolation applications. After jointly optimizing the ARMA graph process coefficients $\mathbf{a}$ and $\mathbf{b}$ along with the Laplacian matrix $\mathcal{L}_\mathcal{G}$, we use these parameters to estimate the missing entries of a partially observed process. An improved estimate of $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$, denoted as $\mathbf{\Sigma}^\star_{\bar{\mathbf{x}}}$, is obtained using the fitted parameters $\mathbf{a}$, $\mathbf{b}$ and the eigenvalues of the optimized Laplacian $\mathcal{L}^\star_\mathcal{G}$. This is achieved by firstly calculating the improved JPSD estimate $h^\star(\lambda_n, \omega_\tau)$ by (3.14). Then, $h^\star(\Lambda_\mathcal{G}, \Omega)$ is constructed as

in (2.36). Lastly, $\mathbf{\Sigma}^{\star}_{\bar{\mathbf{x}}}$ is calculated as

$$\mathbf{\Sigma}^{\star}_{\bar{\mathbf{x}}} = \mathbf{U}^{\star}_{J}\, h^{\star}(\Lambda_{\mathcal{G}}, \Omega)\, (\mathbf{U}^{\star}_{J})^{H} \tag{3.27}$$

where $\mathbf{U}^{\star}_{J}$ is the matrix containing the joint eigenbasis that is constructed from the optimized Laplacian $\mathcal{L}^{\star}_{\mathcal{G}}$ and the cyclic Laplacian $\mathcal{L}_{T}$ with equation (2.30). Our setting for the interpolation application is as follows.

We consider a scenario where L realizations $\{\mathbf{X}_l\}_{l=1}^{L}$ of the time-vertex process $\mathbf{X}$ are available for training, and $R - L$ realizations $\{\mathbf{X}_l\}_{l=L+1}^{R}$ are available for testing. During the training phase, we apply the proposed method to learn the vectors $\mathbf{a}$ and $\mathbf{b}$ and optimize the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. Each realization $\mathbf{X}_l \in \mathbb{R}^{N \times T}$ is fully observed, although the algorithm can also handle scenarios with partial observations, in which case the covariance matrix $\tilde{\mathbf{\Sigma}}_{\bar{\mathbf{x}}}$ in equation (3.4) can still be calculated by ignoring the missing values in the observations. The motivation behind assuming fully observed realizations is to separate the training and learning phase from the estimation and prediction phase, allowing us to accurately learn a large number of parameters.

We use 80% of the training realizations for learning and 20% for validation. The validation realizations, while known, are used for calculating the validation error. This error is utilized for tuning the parameters μ_A, μ_B, and α of the proposed method. Then, the performance of the proposed method is evaluated at test realizations. For a realization l, the known samples of a time-vertex graph signal can be represented by the set $\mathcal{I}_l$. Specifically, let $\mathcal{I}_l = \{(i,t) \mid \mathbf{X}_l(i,t) \text{ is observed}\}$ denote the index set of the available entries in realization l, for $l = 1, 2, ..., R$. Similarly, the complement of this set, $\bar{\mathcal{I}}_l = \{(i,t) \mid \mathbf{X}_l(i,t) \text{ is missing}\}$ denotes the index set of the missing entries in realization l, for $l = 1, ..., R$.

Let $\bar{\mathbf{x}}_l$ be the vectorized form of the realization l for the time-vertex signal $\mathbf{X}_l$. Let $\bar{\mathbf{y}}_l$ and $\bar{\mathbf{z}}_l$ vectors be the known and missing entries of $\bar{\mathbf{x}}_l$ respectively, i.e., the sample values in the sets $\mathcal{I}_l$ and $\bar{\mathcal{I}}_l$. The vector of missing process values $\bar{\mathbf{z}}_l$ can then be estimated with minimum mean square error (MMSE) estimation approach, which is the same as the linear MMSE (LMMSE) estimate since $\bar{\mathbf{y}}_l$ and $\bar{\mathbf{z}}_l$ are jointly Gaussian [8]

$$(\bar{\mathbf{z}}_l)^{\star} = (\mathbf{\Sigma}_{\bar{\mathbf{z}}_l \bar{\mathbf{y}}_l})^{\star}\, ((\mathbf{\Sigma}_{\bar{\mathbf{y}}_l})^{\star})^{-1}\, \bar{\mathbf{y}}_l. \tag{3.28}$$

Here $(\mathbf{\Sigma}_{\bar{\mathbf{z}}_l \bar{\mathbf{y}}_l})^\star$ and $(\mathbf{\Sigma}_{\bar{\mathbf{y}}_l})^\star$ respectively denote the estimates of the cross-covariance matrix of $\bar{\mathbf{z}}_l$ and $\bar{\mathbf{y}}_l$, and the covariance matrix of $\bar{\mathbf{y}}_l$. These matrices can be formed by extracting the corresponding entries of $\mathbf{\Sigma}^\star_{\tilde{\mathbf{x}}}$ for each realization $\mathbf{X}_l$. The normalized mean of the error is calculated as

$$\text{NME} = \left(\frac{1}{L} \sum_{l=1}^{L} \frac{\|(\bar{\mathbf{z}}_l)^\star - (\bar{\mathbf{z}}_l)\|_2^2}{\|(\bar{\mathbf{z}}_l)\|_2^2} \right)^{\frac{1}{2}}. \tag{3.29}$$

We express our parameter selection method in Appendix A.

CHAPTER 4

EXPERIMENTS

In this chapter, we evaluate the performance of the proposed method. We begin by conducting a series of experiments on synthetic data to assess the accuracy of the model calculations. These experiments also allow us to examine the impact of various parameters on the model's accuracy. Following this, we perform comparative experiments on real data sets to further validate our approach. We conduct 8 experiments and present the average errors for each subsection.

4.1 Synthetic data set Experiments

To thoroughly assess the model computation accuracy of our algorithm, we present results using a synthetically generated ARMA graph process. The underlying graph topology is derived from a sensor network by using the GSP toolbox [52]. Using this topology, we synthetically generate realizations of an ARMA graph process as described in (2.42). In each experiment, different realizations of the graph ARMA process are generated by using the parameters described below.

The synthetic data experiments aim to evaluate the accuracy of the proposed algorithm in learning the model parameters $\mathbf{a}$ and $\mathbf{b}$, as well as the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. Therefore, we perturb the weights of $\mathcal{L}_{\mathcal{G}}$ to generate a noisy observation $\mathcal{L}_{\mathcal{G}_0}$ of the graph Laplacian, provide $\mathcal{L}_{\mathcal{G}_0}$ as input to our algorithm, and measure the accuracy of the model parameters across different SNR levels. We assess the model's accuracy by comparing four different estimated parameters which are the JPSD $h(\lambda_n, \omega_\tau)$, $\mathbf{a}$,

$\mathbf{b}$ and $\mathcal{L}_{\mathcal{G}}$. The estimation error of JPSD is calculated as

$$\frac{\|h^{\star}(\lambda_n, \omega_\tau) - h(\lambda_n, \omega_\tau)\|_F}{\|h(\lambda_n, \omega_\tau)\|_F} \tag{4.1}$$

where $h^{\star}(\lambda_n, \omega_\tau)$ represents the estimated JPSD matrix, $h(\lambda_n, \omega_\tau)$ represents the ground truth JPSD matrix, and $\|.\|_F$ denotes the Frobenius norm. The errors for the parameters $\mathbf{a}$ and $\mathbf{b}$ are calculated as follows

$$\frac{\|\mathbf{a}^{\star} - \mathbf{a}\|}{\|\mathbf{a}\|} \tag{4.2}$$

$$\frac{\|\mathbf{b}^{\star} - \mathbf{b}\|}{\|\mathbf{b}\|} \tag{4.3}$$

where $\mathbf{a}^{\star}$ and $\mathbf{b}^{\star}$ represent the estimated vectors for the parameters $\mathbf{a}$ and $\mathbf{b}$, respectively. The operator $\|.\|$ stands for the ℓ_2-norm. We compare the estimation errors of these three methods with the approach described in [11] as we call it JS-ARMA, which does not account for the error in $\mathcal{L}_{\mathcal{G}}$. Our focus is on the impact of $\mathcal{L}_{\mathcal{G}}$'s accuracy on the model parameters and how precisely we estimate both the parameters and $\mathcal{L}_{\mathcal{G}}$. The error of $\mathcal{L}_{\mathcal{G}}$ is calculated as follows

$$\frac{\|\mathcal{L}_{\mathcal{G}}^{\star} - \mathcal{L}_{\mathcal{G}}\|_F}{\|\mathcal{L}_{\mathcal{G}}\|_F} \tag{4.4}$$

where $\mathcal{L}_{\mathcal{G}}^{\star}$ is the estimated graph Laplacian matrix.

We begin by observing the effect of the number of iterations on the estimation performance in Section 4.1.1. As we will see in Section 4.1.1, the estimation error of the model parameters and $\mathcal{L}_{\mathcal{G}}$ tend to start increasing or destabilize after the second iteration. Hence, we select the maximum number of iterations as 3 and do not process further after 2. Then, we move on to analyze the impact of the noise on $\mathcal{L}_{\mathcal{G}}$ on the performance. We consider two scenarios. In Section 4.1.2, the graph ARMA process realizations are produced according to a ground truth $\mathbf{W}$ but the proposed method only has access to perturbed weight matrix, $\mathbf{W}_{\text{Noisy}}$. We only perturb the existing edges of the weight matrix $\mathbf{W}$ without adding or removing any edges. In Section 4.1.3, we allow for both the addition and removal of edges, as well as perturbations to existing edges in the weight matrix $\mathbf{W}$. We conduct these experiments for different SNR values. The SNR in dB is defined as

$$20 \log_{10} \left(\frac{\|\mathbf{W}\|_F}{\|\mathbf{W}_{\text{Noisy}} - \mathbf{W}\|_F} \right) \tag{4.5}$$

where $\mathbf{W}_{\text{Noisy}}$ is the perturbed weight matrix.

In Section 4.1.4, we examine how the number of realizations impacts estimation performance. We assume the presence of noise in both the observed signals and $\mathbf{W}$, as our focus is on evaluating the algorithm's performance under these noisy conditions, particularly with respect to the noise in $\mathbf{W}$. We do not address model complexity and mismatch effects, as these factors have already been explored in the work by Guneyi et al. [11] where we expect to observe similar findings. We describe the experiment environment and parameters as follows.

We select number of nodes and time length as $N = 30$ and $T = 25$, respectively. We use $P = 1$, $K = 1$, $Q = 1$, $M = 0$ as our graph ARMA process parameters and set the parameter vectors as $\mathbf{a} = \begin{bmatrix} 1 & -0.5 & 0 & 0.5 \end{bmatrix}^H$ and $\mathbf{b} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^H$. In these experiments, the ground truth values for P, K, Q and M are provided to the algorithm. The regularization parameters μ_A, μ_B and α are found by the validation procedure described in Appendix A. The resulting JPSD can be seen in Figure 4.1.

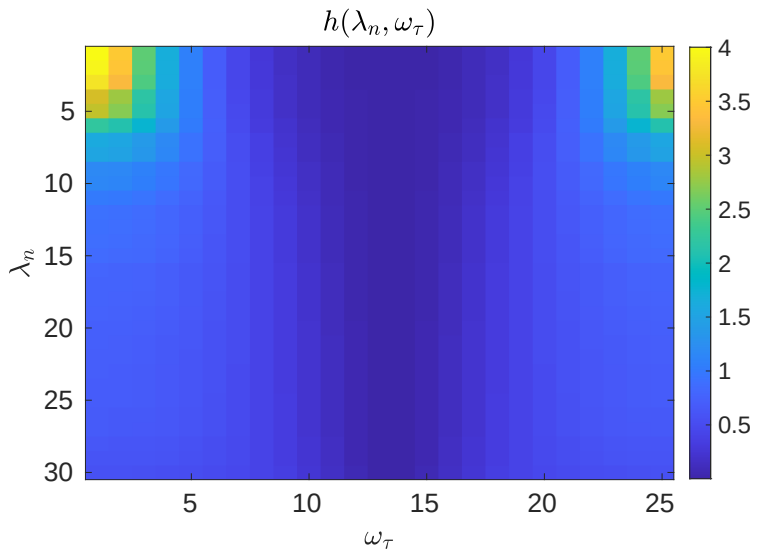

Figure 4.1: Synthetic data set JPSD

It exhibits low-pass characteristics in both the time and node domains; therefore, the generated realizations are expected to vary smoothly across the graph and over time.

4.1.1 Performance Analysis Across Iterations

In this section, we study the behavior of the algorithm in terms of the estimated JPSD, the parameters $\mathbf{a}$ and $\mathbf{b}$ of the graph ARMA process, and the graph Laplacian matrix. We generate $R = 200$ realizations and keep the observations noise-free. The SNR of the weight matrix is 10 dB.

1. JPSD Estimation:

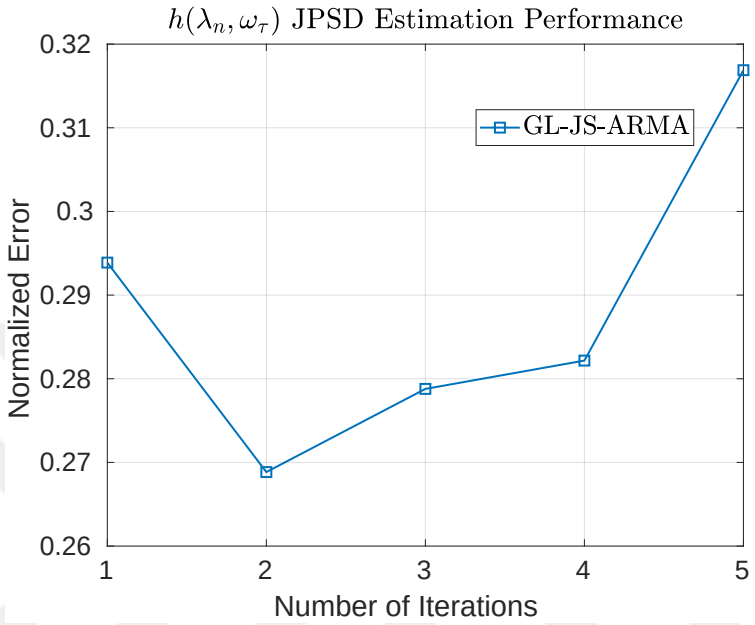

Figure 4.2: Variation of the JPSD estimation error with iterations

In Figure 4.2, after the second iteration, the accuracy of the estimated JPSD begins to deteriorate. While the first two iterations contribute to aligning the estimated JPSD with the ground truth, subsequent iterations introduce errors that result in an overfitting effect. This overfitting is likely due to the model attempting to fit to noise rather than capturing the true underlying structure of the time-vertex process.

2. Graph ARMA Parameters a and b:

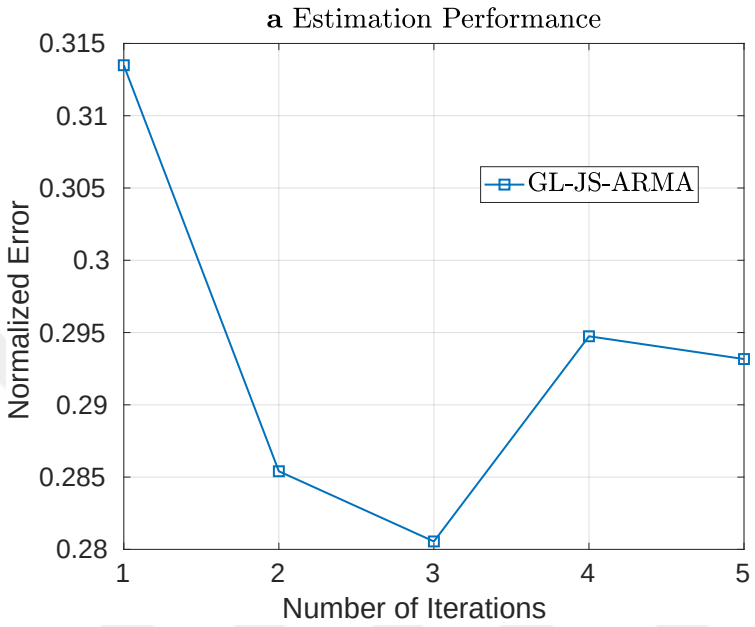

Figure 4.3: Variation of the estimation error of parameter **a** with iterations

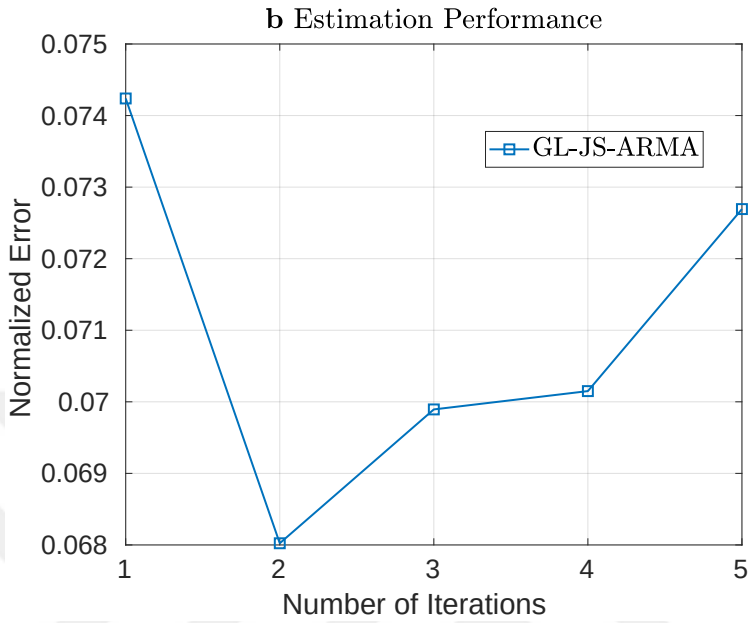

Figure 4.4: Variation of the estimation error of parameter $\mathbf{b}$ with iterations

In Figures 4.3 and 4.4, the errors in estimating the parameters $\mathbf{a}$ and $\mathbf{b}$ show a consistent pattern of improvement during the first two iterations. However, after the second iteration, the optimization procedure appears to become unstable, leading to fluctuations in the parameter estimates. This instability suggests that the model parameters are becoming increasingly sensitive to minor inaccuracies in the estimation process, potentially causing them to diverge from the true values. As mentioned in Section 3.2, our primary goal is to minimize the function $f_1 + r_1 + f_2 + r_2$. However, since this is a challenging task, we take an iterative approach by first minimizing the function $f_1 + r_1$ by optimizing variables $\mathbf{a}$ and $\mathbf{b}$, and then minimizing the function $f_2 + r_2$ by optimizing $\mathcal{L}_\mathcal{G}$. This choice is due to the fact that the overall cost function is not

jointly convex with respect to the variables we aim to optimize. As a result, even reaching a local minimum can be challenging, if not impossible. In addition, the parameters $\mathbf{a}$ and $\mathbf{b}$ are approximated using a convex relaxation procedure, as described in equation (3.18). While this relaxation provides initial estimates for the parameters, the approximation starts to deviate from the ground truth values after the second iteration. This divergence indicates that the approximated solutions lose accuracy in representing the underlying model, which could be due to the inherent non-convexity of the original optimization problem. Therefore, although the convex relaxation offers some computational advantages, its accuracy tends to degrade as the optimization progresses. Furthermore, a fundamental issue here arises when calculating the Laplacian matrix $\mathcal{L}_\mathcal{G}$ by minimizing $f_2 + r_2$, as we ignore the dependency of the f_1 term on $\mathcal{L}_\mathcal{G}$. Ideally, we should be minimizing $f_1 + f_2 + r_2$ together. However, the dependency of f_1 on $\mathcal{L}_\mathcal{G}$ is quite complex as f_1 involves polynomials of its eigenvalues, which we ignore for the sake of ease of computation. This approach can lead to inconsistent behavior, where the algorithm may fail to converge towards a solution over iterations. It is highly likely that this assumption disrupts the algorithm's convergence behavior, preventing it from stabilizing on a consistent solution.

3. Graph Laplacian Matrix $\mathcal{L}_{\mathcal{G}}$:

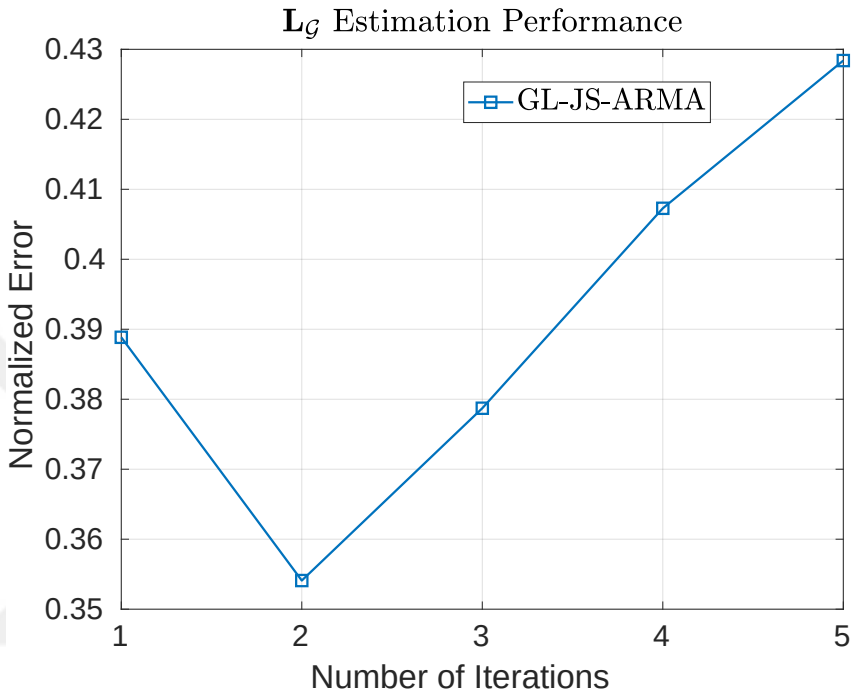

Figure 4.5: Variation of the estimation error of parameter $\mathcal{L}_{\mathcal{G}}$ with iterations

In Figure 4.5, the estimation of the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ also suffers from instability after the second iteration. Initially, the algorithm successfully refines $\mathcal{L}_{\mathcal{G}}$ to better reflect the underlying graph structure. However, as with the JPSD and the ARMA parameters, further iterations result in the introduction of noise into the graph structure, leading to a less accurate Laplacian matrix that can negatively impact the overall model.

General Observations: The instability observed after the second iteration suggests that while the algorithm is effective in the early stages of optimization, it may be prone to overfitting as the iterations progress. This highlights the need for potential safeguards, such as early stopping after the second iteration as we did.

4.1.2 Effect of Perturbation of Edge Weights with Edge Preservation

The results obtained from the experiments provide an analysis of the proposed method's performance in estimating the JPSD, the parameters of the graph ARMA process $\mathbf{a}$, $\mathbf{b}$ and the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. We start by adding zero-mean white Gaussian noise to $\mathbf{W}$. Since in this section we aim to preserve the edges, we assign $\epsilon = 2.2204\text{e}{-10}$ to any of the weights that become non-positive due to noise addition. Then, $R = 50$ process realizations $\mathbf{X}_l$ are generated by filtering the realizations of a zero-mean white Gaussian time-vertex process.

1. JPSD Estimation:

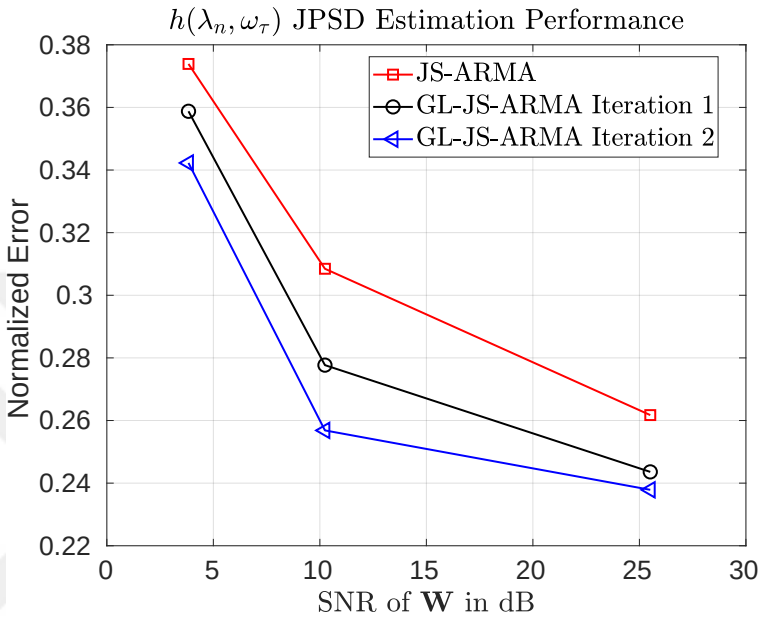

Figure 4.6: Variation of the estimation error of JPSD with SNR under edge preservation

In Figure 4.6, we present the variation of the JPSD estimation error with the SNR of the graph topology. The JPSD estimation results reveal the accuracy of the proposed method in capturing the spectral characteristics of the time-vertex signals. The comparison between the estimated and ground truth JPSD matrices demonstrates that our method effectively reconstructs the joint spectrum, even under noisy conditions for $\mathcal{L}_\mathcal{G}$. The JS-ARMA method [11], which does not attempt to learn a new $\mathcal{L}_\mathcal{G}$ but instead estimates the parameters $\mathbf{a}$ and $\mathbf{b}$ from the initial, possibly noisy, Laplacian

matrix, serves as the baseline for our comparisons. As illustrated in Figure 4.6, the estimation error of the JPSD decreases over iterations, aligning with the results discussed in Section 4.1.1.

2. Parameters of Graph ARMA Process $\mathbf{a}$ and $\mathbf{b}$:

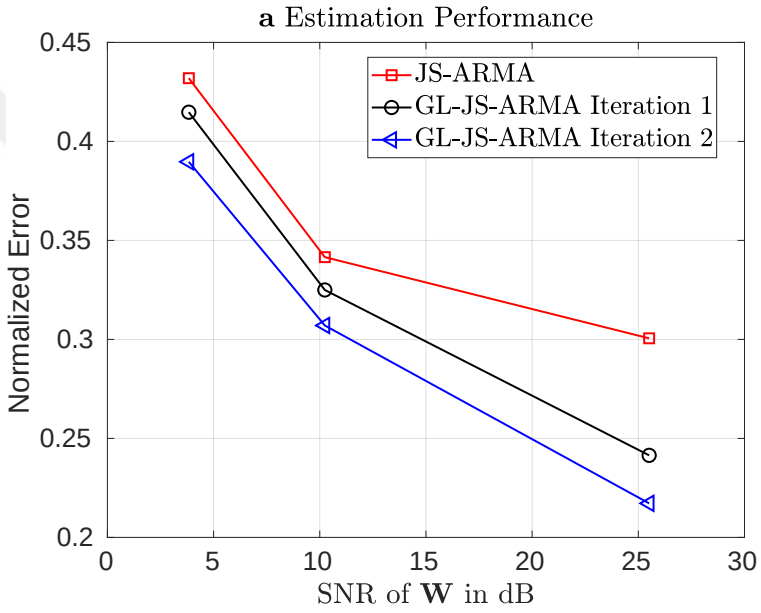

Figure 4.7: Variation of the estimation error of parameter $\mathbf{a}$ with SNR under edge preservation

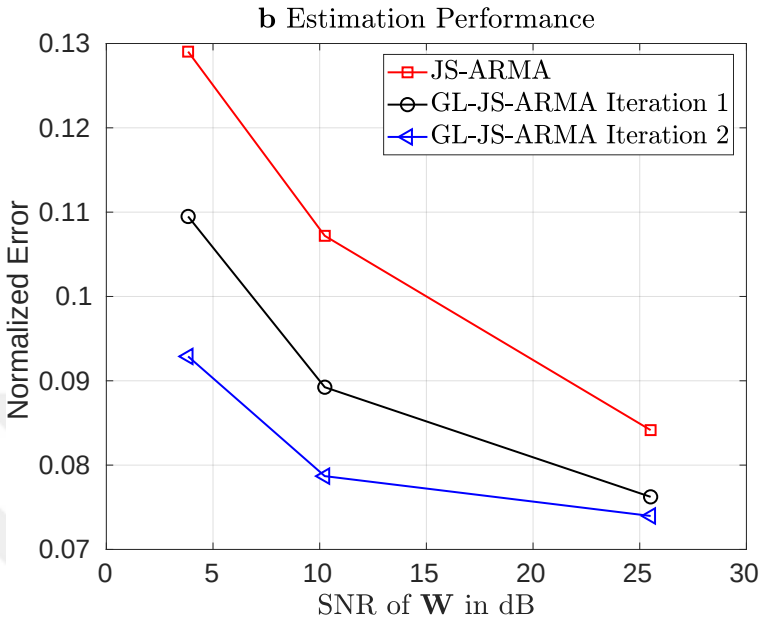

Figure 4.8: Variation of the estimation error of parameter $\mathbf{b}$ with SNR under edge preservation

The estimation errors for the parameters $\mathbf{a}$ and $\mathbf{b}$ of the graph ARMA process in Figures 4.7 and 4.8, obtained through the iterative optimization procedure, consistently decrease with increasing SNR as expected.. These parameters directly influence the accuracy of the signal reconstruction and JPSD estimation, and the results indicate that the proposed method achieves lower JPSD error compared to the baseline method JS-ARMA. This confirms the effectiveness of our approach in capturing the temporal and spatial dependencies of the graph signals by taking into account the possible deviation between the observed graph topology and the actual graph topology.

3. Graph Laplacian Matrix $\mathcal{L}_{\mathcal{G}}$:

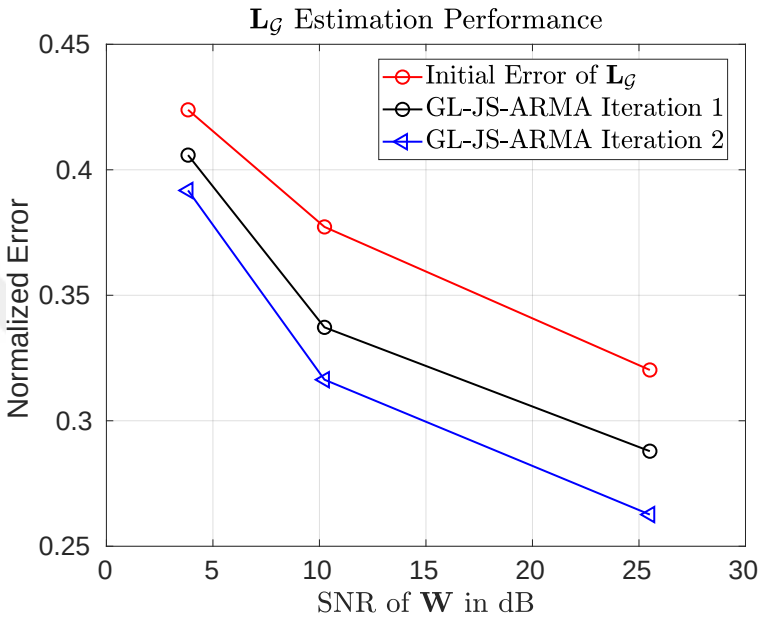

Figure 4.9: Variation of the estimation error of $\mathcal{L}_{\mathcal{G}}$ with SNR under edge preservation

In Figure 4.9, we present the variation of the estimation error of the graph topology with SNR. The optimization of the graph Laplacian matrix plays a critical role in the overall performance of the method. The iterative updates to $\mathcal{L}_{\mathcal{G}}$ ensure that the learned graph structure is well-aligned with the underlying data characteristics. The experiments demonstrate that the resulting $\mathcal{L}_{\mathcal{G}}$ successfully captures the essential connections between nodes, leading to better estimation performance. The adaptability of $\mathcal{L}_{\mathcal{G}}$ to varying levels of noise further highlights the robustness of the proposed method.

In conclusion, the experiments confirm that the proposed method improves the estimation accuracy of both the process parameters and the graph structure. The joint learning of the JPSD, $\mathbf{a}$, $\mathbf{b}$, and $\mathcal{L}_{\mathcal{G}}$ leads to a more precise representation of time-vertex graph signals, making the method effective for practical applications in signal processing and data analysis.

4.1.3 Effect of Noise on Possibly All Edges of $\mathcal{L}_{\mathcal{G}}$

In this subsection, we analyze the effect of introducing noise to the weight matrix $\mathbf{W}$, where noise is applied not only to the existing edges but also to potential edges that were initially non-existent. We explore how this noise impacts the SNR and consequently the estimation performance of the proposed algorithm.

SNR Degradation: The introduction of noise to the weight matrix $\mathbf{W}$ significantly impacts the SNR, particularly when noise is allowed to affect all possible edges, including those with zero initial weights. This setting creates a scenario where the underlying graph structure becomes increasingly perturbed, leading to a reduction in the SNR, which takes much lower values than those in Section 4.1.2. The decrease in the SNR correlates with a decline in the algorithm's ability to accurately estimate the graph Laplacian matirx $\mathcal{L}_{\mathcal{G}}$.

1. JPSD Estimation:

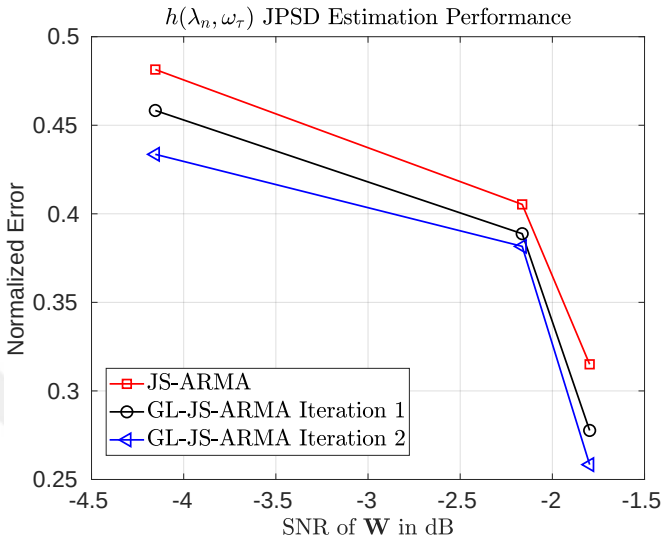

Figure 4.10: Variation of the JPSD estimation error under noise with possibility of edge modification of $\mathbf{W}$

In Figure 4.10, we present the variation of the JPSD estimation error with SNR. The noisy weight matrix scenario with the possibility of edge modification results in a degradation in the accuracy of the estimated JPSD. In this challenging setting, it is relatively difficult to differentiate between signal and noise, especially when noise is introduced to edges that do not correspond to any actual connections in the graph. The JPSD estimation error in this setting is seen to be higher than that in Section 4.1.2 by more than 10%. This leads to an inaccurate representation of the JPSD, due to the reduced SNR.

2. Graph ARMA Parameters a and b:

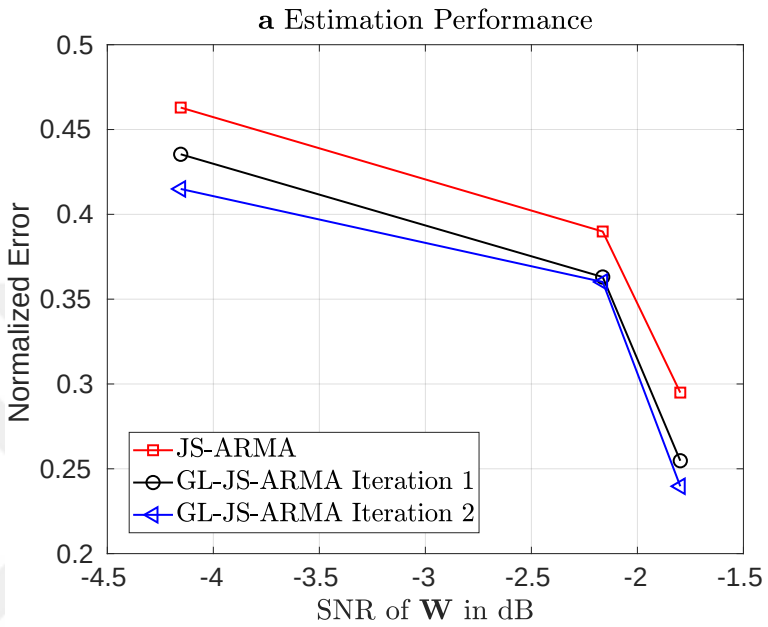

Figure 4.11: Variation of the **a** estimation error under noise with possibility of edge modification of **W**

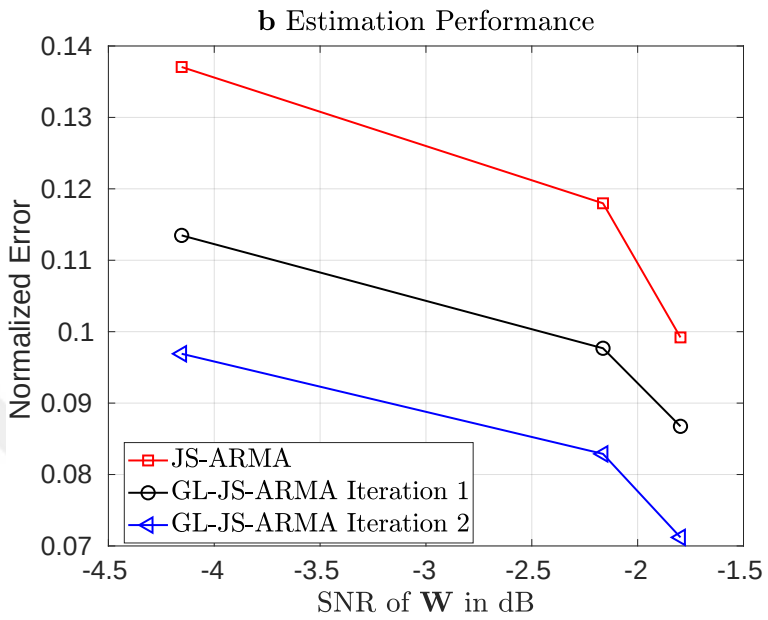

Figure 4.12: Variation of the $\mathbf{b}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

The estimation of the graph ARMA parameters $\mathbf{a}$ and $\mathbf{b}$ is also affected by the low SNR values in this setting. as they can be seen in Figures 4.11 and 4.12. The presence of noise across all edges creates additional challenges in accurately estimating these parameters. As a result, the parameters $\mathbf{a}$ and $\mathbf{b}$ exhibit greater variability, and their estimated values are less reliable, compared to the setting with higher SNR in Section 4.1.2. For example, for the parameter $\mathbf{a}$, the error is calculated to be at least 10% higher. The noise introduces inconsistencies in the optimization process, leading to less accurate parameter estimates.

3. Graph Laplacian Matrix $\mathcal{L}_{\mathcal{G}}$:

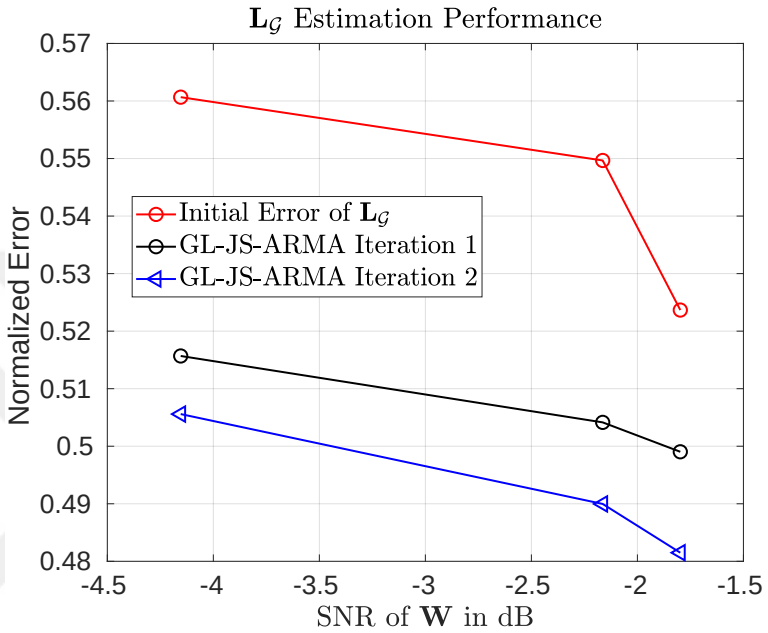

Figure 4.13: Variation of the $\mathcal{L}_{\mathcal{G}}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

The noisy weight matrix significantly impacts the accuracy of the estimated graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ as it can be seen in Figure 4.13. As the noise is spread across both existing and non-existing edges, the estimated $\mathcal{L}_{\mathcal{G}}$ becomes less reflective of the true underlying graph structure. The introduction of noise on non-existent edges leads to spurious connections in the graph, which ultimately degrades the model's performance.

General Observations:

Overall, the experiments demonstrate that when noise is introduced to the weight matrix, particularly when it affects all possible edges, the SNR decreases significantly. This reduction in SNR adversely affects the accuracy of the estimated JPSD, the ARMA parameters, and the graph Laplacian matrix. The results emphasize that the initially known graph topology should not be too far from the actual graph topology in order to ensure reliable estimation of graph-based models.

4.1.4 Impact of the Number of Realizations on Estimation Performance

In this section, we analyze the impact of the number of realizations on the estimation performance of our proposed method. The performance metrics are evaluated for an SNR of 5 dB for the weight matrix $\mathbf{W}$. Moreover, we also add zero-mean white Gaussian noise to the observations which yields an SNR of 12 dB for realizations. Observation SNR in dB scale for a realization l is calculated as

$$20 \log_{10} \left(\frac{\|\mathbf{X}_l\|_F}{\|\mathbf{X}_l - \mathbf{X}_{l_{\text{Noisy}}}\|_F} \right). \tag{4.6}$$

1. JPSD Estimation:

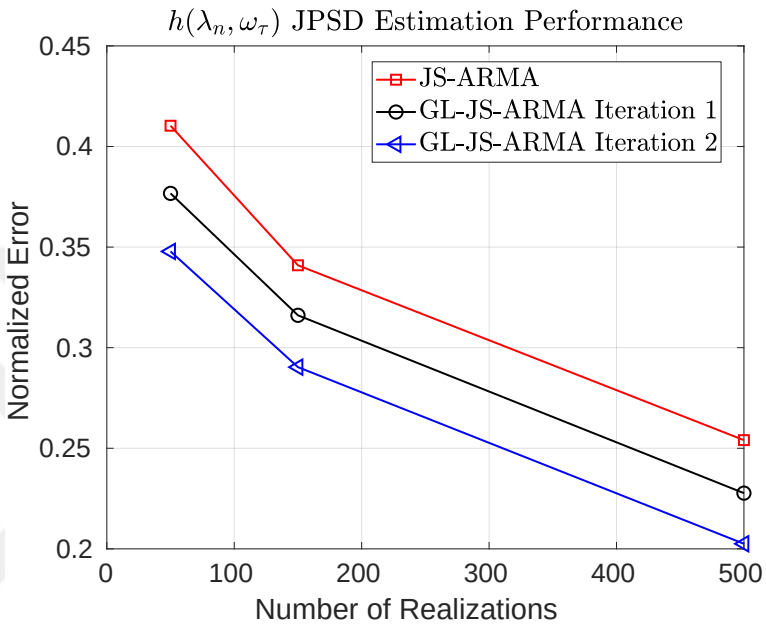

Figure 4.14: Variation of the $\mathcal{L}_{\mathcal{G}}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

In Figure 4.14, as the number of realizations increases, the estimation errors for the JPSD decrease, reflecting the improved accuracy of the spectral representation of the time-vertex signal. This trend is evident in Figure 4.14, where the estimated JPSD becomes increasingly closer to the ground truth with more realizations, confirming the effectiveness of the iterative method in capturing the joint time-vertex spectral properties of the process.

2. Graph ARMA Parameters $\mathbf{a}$ and $\mathbf{b}$:

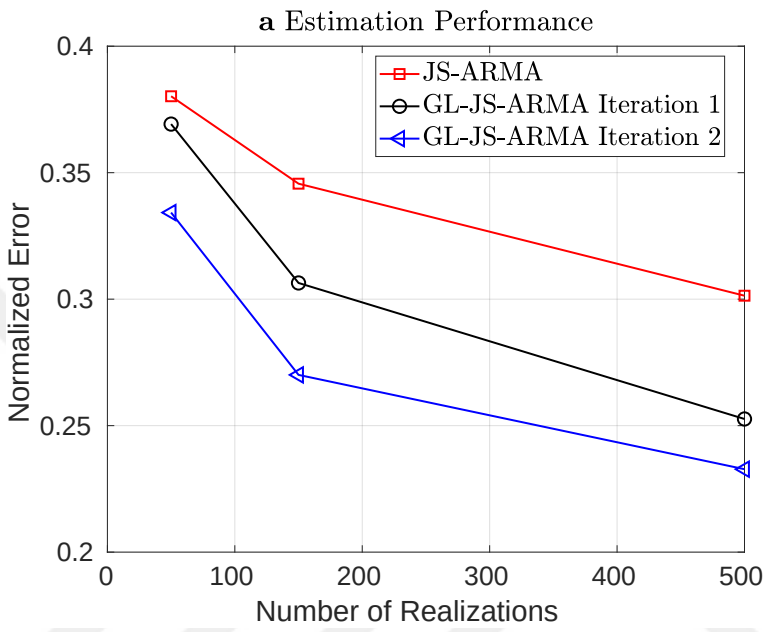

Figure 4.15: Variation of the $\mathcal{L}_{\mathcal{G}}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

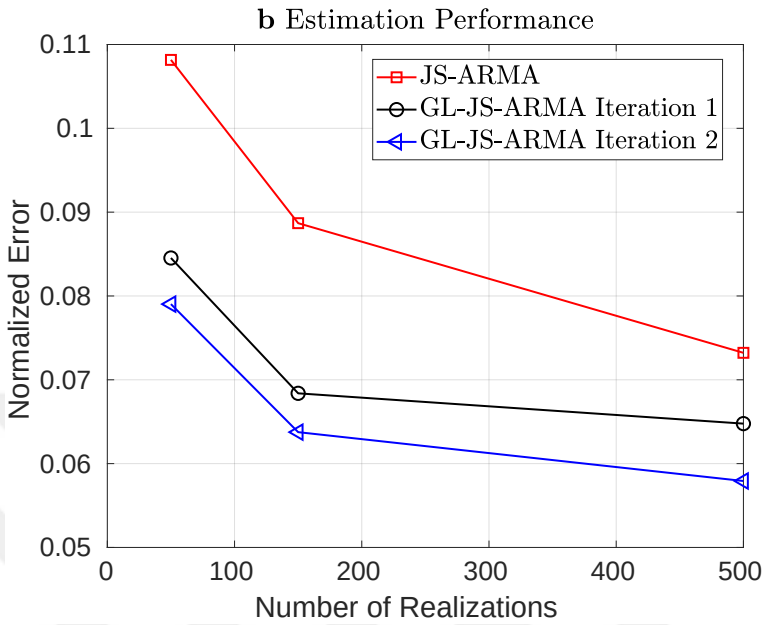

Figure 4.16: Variation of the $\mathcal{L}_{\mathcal{G}}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

The estimation errors for the parameters $\mathbf{a}$ and $\mathbf{b}$ exhibit a consistent decline as the number of realizations increases, as shown in Figures 4.15 and 4.16. With a limited number of realizations, the errors in these parameters are relatively high, reflecting the challenges in capturing the underlying dynamics of the process. However, as the number of realizations increases, the estimation errors decrease, indicating that the additional data provides better information for learning these parameters.

The iterative optimization process leverages the additional information provided by the increasing number of realizations to refine the parameter estimates, leading to improved model accuracy. This is crucial for ensuring that the graph ARMA process

model accurately captures the underlying dynamics of the time-vertex signal.

3. Graph Laplacian Matrix $\mathcal{L}_{\mathcal{G}}$:

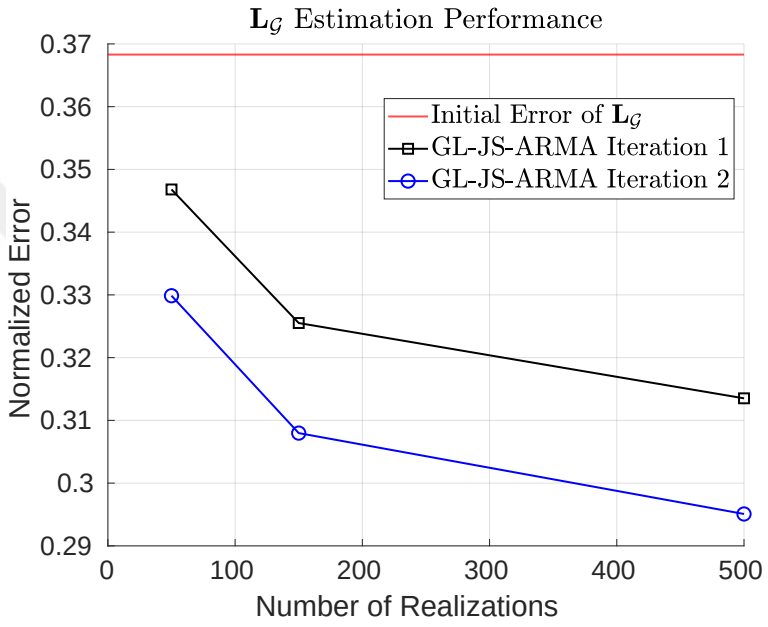

Figure 4.17: Variation of the $\mathcal{L}_{\mathcal{G}}$ estimation error under noise with possibility of edge modification of $\mathbf{W}$

The graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ also benefits from improved estimation accuracy with more realizations, as illustrated in Figure 4.17. This trend highlights the importance of sufficient data for accurately capturing the graph topology and, consequently, for improving the overall model performance.

General Observations:

Overall, the results demonstrate that increasing the number of realizations significantly enhances the estimation performance across all key components of the model, including the JPSD, the parameters $\mathbf{a}$ and $\mathbf{b}$, and the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. This underscores the importance of utilizing sufficient realizations in practical applications to achieve robust and accurate model estimation.

4.2 Sensivity Analysis of α

To determine the appropriate range for the regularization parameter α, which determines the sparsity of the graph topology by weighting the ℓ_1 norm of the graph Laplacian matrix, we conduct a sensitivity analysis. The aim is to identify the values of α that optimize the performance of the algorithm across different range of values and COVID-19 data set is used for this purpose. In Table 4.1, we report the NME obtained at different values of the α parameter for missing observation ratios 10%, 50% and 80%, to observe the effect of the choice of α in different scenarios. The analysis indicates that choosing α within the range $[10^{-7}, 10^{-4}]$ is particularly effective in achieving optimal results. This range was selected based on its consistent performance in minimizing the NME, as shown in Table 4.1. The results suggest that α values within this range provide a good balance between the sparsity of $\mathcal{L}_{\mathcal{G}}$ and dependencies between nodes, ensuring that the algorithm remains stable while effectively capturing the underlying structure of the data. The optimal values for α decrease as the ratio of missing observations increases, indicating that a denser graph better captures dependencies and leads to improved estimation results.

α **VALUES**	**10% is missing**	**50% is missing**	**80% is missing**
0	0.131	0.166	0.295
1.00e-09	0.128	0.154	0.258
1.00e-08	0.120	0.154	**0.254**
1.00e-07	0.115	0.151	**0.252**
1.00e-06	0.114	0.151	0.264
1.00e-05	0.112	**0.147**	0.264
1.00e-04	**0.103**	**0.148**	0.271
1.00e-03	**0.104**	0.150	0.273
1.00e-02	0.107	0.152	0.274

Table 4.1: NME for different α values across missing data scenarios

4.3 Comparative Experiments

In this section, we evaluate the performance of our method for two different time-vertex data sets. The first dataset is the Molène weather dataset. This experiment uses hourly weather measurements collected in the Brittany region of France during January 2014 [57]. We focus on temperature measurements taken from $N = 37$ different weather stations, each represented as a graph node. A 10-NN graph with Gaussian edge weights given in equation (2.1) is constructed to model the connections between the stations. Each 24-hour measurement sequence is treated as one realization of a time-vertex graph process $\mathbf{X}$ with a graph size of $N = 37$ and a time length of $T = 24$, resulting in a total of $R = 31$ realizations.

The second dataset is related to the COVID-19 pandemic. This experiment is conducted using data on the number of daily new cases per country between February 15, 2020, and July 5, 2021 [58]. We include the $N = 37$ most populous European countries in the experiment, where each country is treated as a graph node. A 4-NN graph is constructed with Gaussian edge weights based on a hybrid distance measure that considers both geographical proximities and the number of flights (sourced from [59]) between countries. The number of daily new cases is normalized by each country's population and smoothed using a moving average filter over a 7-day win-

Dataset	Time Stationarity	Vertex Stationarity	Time-Vertex Stationarity
Molène	0.8955	0.9365	0.9203
COVID-19	0.9963	0.7608	0.7525

Table 4.2: Stationarity ratios of the data sets

dow. The time length of the process is taken as $T = 21$ days (three weeks), and the experiments are conducted on $R = 23$ realizations of the process.

We study the signal estimation problem in a scenario where missing observations occur at randomly and independently selected time-vertex pairs. The performances of the algorithms are compared with respect to the normalized mean error (NME) of the estimates of the missing observations as defined in equation (3.29). Since real data already has some natural deviation from the process model considered in our study, no extra noise is added to the data.

To better interpret our estimation results, we begin by analyzing the joint time-vertex stationarity, vertex stationarity, and time stationarity of each dataset. As noted in (3.5), the covariance matrix of a time-vertex stationary process must be diagonalizable with the eigenvectors of the joint Laplacian. We therefore compute the time-vertex stationarity ratio for each dataset as $\|\text{diag}(\tilde{h}(\Lambda_{\mathcal{G}}, \Omega))\|/\|\tilde{h}(\Lambda_{\mathcal{G}}, \Omega)\|_F$. The vertex stationarity ratio and time stationarity ratio are calculated similarly, by restricting the covariance matrix to the vertex or time domain in (3.5) and substituting the eigenvector matrix with $\mathbf{U}_{\mathcal{G}}$ or $\mathbf{U}_T$, respectively. The stationarity ratios for each dataset are presented in Table 4.2.

The proposed method is compared with several established methods in the literature. The first of these methods is the AR time process models method [26], which is employed for modeling univariate time series and predicts future values based on the past values of the series itself individually at each node. Then, the study compares against the Vector Autoregressive (VAR) process models method [60], which is a multivariate time series approach that models the dynamic relationships between multiple variables, generating predictions based on the past values of each variable. The Graph Vector Autoregressive Moving Average Recursions (G-VARMA) and the Graph Polynomial Vector Autoregressive Recursions (GP-VAR) methods [28] leverage the con-

cept of time-vertex stationarity and use the model in equation (2.42), where the the GP-VAR method drops the MA part. The JWSS method [24] uses a non-parametric JPSD estimator that corresponds to the initial estimate of the JPSD in our approach. The JS-ARMA method [11] uses a parametric JPSD estimator that corresponds to first block of our method in Figure 3.1, which does not learn a new $\mathcal{L}_{\mathcal{G}}$.

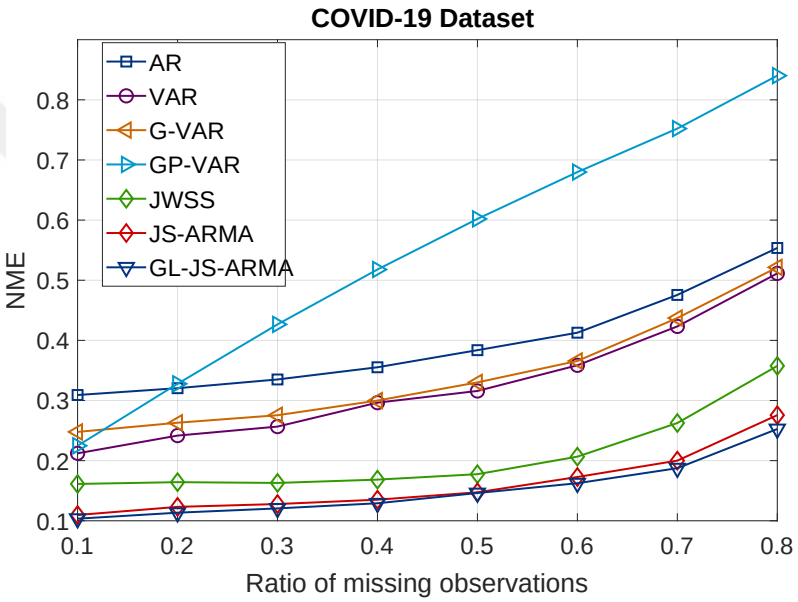

Figure 4.18: NME of COVID-19 data set

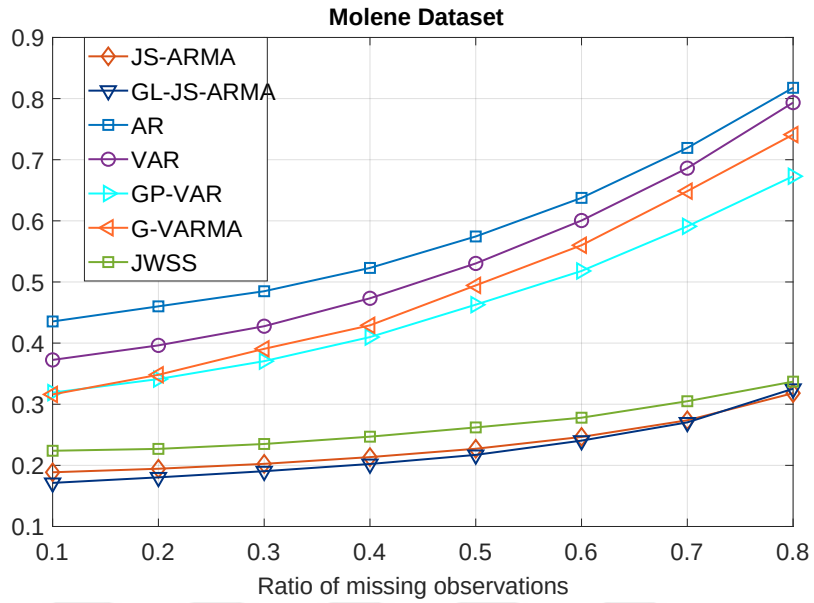

Figure 4.19: NME of Molène data set

In the COVID-19 data set, we use the model parameters $P = 1$, $K = 1$, $Q = 0$, $M = 1$, $\mu_A = 0.1$, $\mu_B = 0.1$. Motivated by our findings in Section 4.2, we adapt the parameter α to the ratio of missing observations, such that it takes the values $\begin{bmatrix} 10^{-4} & 10^{-4} & 10^{-6} & 10^{-6} & 10^{-6} & 10^{-7} & 10^{-7} & 10^{-7} \end{bmatrix}$ as the ratio of missing observations varies between 0.1 and 0.8. In the Molène data set, we use the model parameters $P = 2$, $K = 2$, $Q = 1$, $M = 0$, $\mu_A = 10^{-3}$, $\mu_B = 0$. The parameter α is again adapted to the ratio of missing observations as in the COVID-19 dataset, taking the values $\begin{bmatrix} 10^{-3} & 10^{-3} & 10^{-5} & 10^{-5} & 10^{-5} & 10^{-5} & 10^{-6} & 10^{-6} \end{bmatrix}$.

We find that the proposed GL-JS-ARMA method delivers the best estimation performance among the methods that utilize stochastic process models. Generally, graph

process models outperform the VAR and AR models, which do not account for graph topology. However, an interesting exception arises with the COVID-19 dataset, where the AR and VAR methods surpass the graph-based GP-VAR method. This outcome aligns with the high time stationarity and weaker vertex or time-vertex stationarity observed in the COVID-19 dataset. Notably, the proposed GL-JS-ARMA method, JS-ARMA method and the JWSS method, which leverage the time-vertex joint spectrum of the process, consistently outperform G-VARMA, GP-VAR, AR, and VAR, which do not utilize this information. This holds true even for the COVID-19 dataset, underscoring that the joint time-vertex spectrum of a time-varying graph signal reveals critical characteristics that cannot be captured by vertex-only or time-only frequency analysis. Moreover, the proposed GL-JS-ARMA method consistently outperforms the JS-ARMA method at nearly every point, except for the case of the 80% missing ratio in the Molène dataset. This result highlights the crucial role that the underlying graph structure plays in capturing essential information and dependencies within a dataset. Learning a new graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}^{\star}$ with the proposed method provides better performance than the JS-ARMA method, which uses the fixed topology of the initial $\mathcal{L}_{\mathcal{G}_0}$ and only learns the graph ARMA process parameters $\mathbf{a}$ and $\mathbf{b}$ once. The parameter α plays a crucial role in the GL-JS-ARMA method, as it directly influences the sparsity level of the learned Laplacian matrix, $\mathcal{L}_{\mathcal{G}}$. As the missing observation ratio changes, the sparsity of $\mathcal{L}_{\mathcal{G}}$ significantly impacts the NME. For both datasets, the parameter α consistently decreases with the increasing missing ratio of observations, leading to a denser graph in the GL-JS-ARMA method as stated in Section 4.2. A denser graph topology tends to impose the dependencies between the observed and unobserved samples in a stricter way, which turns out to be advantageous at higher ratios of missing observations by facilitating the diffusion of the available information to other nodes under severe lack of data.

CHAPTER 5

CONCLUSION

In this thesis, we have proposed an iterative method for jointly learning a time-vertex graph ARMA processes and graph topologies from observations of the process. Our approach begins by obtaining an empirical estimate of the JPSD of the process, leveraging the block-Toeplitz structure of its covariance matrix through an unbiased circular covariance matrix estimator. We propose to jointly learn the process parameters and the graph topology in an alternating manner. We formulate the learning of graph ARMA process parameters as an optimization problem, with the objective of minimizing the error between the parametric JPSD and its empirical estimate. Although this initial problem is non-convex, a relaxation is applied so that a convex objective function is obtained by expressing the graph ARMA process parameters in terms of low-rank positive semi-definite matrices. After estimating the process parameters, we update the covariance matrix of the process to further optimize the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$. This iterative method continues until a predetermined number of iterations is reached. The proposed method estimates the optimum parameters $\mathbf{a}$ and $\mathbf{b}$ with the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ in this manner, which can potentially be used in several time-vertex inference tasks. In our study, we apply the learnt model for the task of estimating the unavailable observations of a time-vertex process using the LMMSE approach.

The proposed method has been evaluated on both synthetic and real-world data sets. Our experiments demonstrate that the enhanced graph ARMA process model, combined with an optimized graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$, improves the estimation performance. Specifically, our method consistently outperforms existing models, particularly in scenarios where the graph topology plays a critical role in capturing the

underlying dependencies of the data. The results validate the effectiveness of our iterative approach in learning both the graph structure and the ARMA model parameters simultaneously, leading to more accurate and robust signal estimation.

In this thesis, we demonstrate that enhancing the graph ARMA process model with an optimized graph topology can improve estimation performance in real-world data sets. For future work, conducting more extensive experiments on diverse real-world data sets may be helpful for further exploring the advantages of modeling time-varying graph signals as JWSS ARMA graph process models. While this study leverages the covariance matrix of the graph ARMA process, a promising direction for future research is to investigate optimizing the graph Laplacian matrix $\mathcal{L}_{\mathcal{G}}$ in relation to the graph ARMA filter, which is inherently a function of $\mathcal{L}_{\mathcal{G}}$. As an example, the methodology proposed by Egilmez et al. in [31] can be adapted to time-vertex signals, where they aim to learn the graph structure alongside the graph filter for vertex signals. By refining the graph structure with consideration of the filter, it is possible to obtain a more accurate estimate of $\mathcal{L}_{\mathcal{G}}$, ultimately leading to improved model parameters and enhanced estimation performance, particularly to reduce the impact of noisy weight matrices with edge modifications. Furthermore, an important area for future research involves exploring scenarios where the graph topology changes dynamically over time. Current models often assume a static graph structure, which may not adequately capture real-world phenomena where the relationships or interactions stem from an evolving graph. By extending the model to accommodate time-varying graph topologies, we can better capture the dynamics of the underlying processes. An intriguing avenue for another future research path is to explore scenarios where the statistical properties of the processes change over time or across different regions of the graph. Current models typically assume stationary or uniform statistical properties, which may not fully capture the complexity of processes that exhibit temporal or spatial variability. A key area for another future research involves developing adaptive signal processing solutions for scenarios where data arrives sequentially rather than all at once. Traditional signal processing methods often assume batch processing, which may not be suitable for real-time or streaming data environments where the data is received incrementally.

REFERENCES

[1] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, pp. 83–98, May 2013.

[2] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, pp. 808–828, May 2018.

[3] S. Itani and D. Thanou, "A graph signal processing framework for the classification of temporal brain data," in *2020 28th European Signal Processing Conference (EUSIPCO)*, pp. 1180–1184, 2021.

[4] Y.-L. Qiao, Y. Zhao, C.-Y. Song, K.-G. Zhang, and X.-Z. Xiang, "Graph wavelet transform for image texture classification," *IET Image Processing*, vol. 15, no. 10, pp. 2372–2383, 2021.

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

[6] D. Owerko, F. Gama, and A. Ribeiro, "Optimal power flow using graph neural networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5930–5934, 2020.

[7] J. Miettinen, S. A. Vorobyov, E. Ollila, and X. Wang, "Correlation-based graph smoothness measures in graph signal processing," in *2023 31st European Signal Processing Conference (EUSIPCO)*, pp. 1848–1852, 2023.

[8] N. Perraudin and P. Vandergheynst, "Stationary signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3462–3477, 2017.

[9] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, "Stationary graph processes and spectral estimation," *IEEE Transactions on Signal Processing*, vol. 65, no. 22, pp. 5911–5926, 2017.

[10] N. Perraudin, A. Loukas, F. Grassi, and P. Vandergheynst, "Towards stationary time-vertex signal processing," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3914–3918, 2017.

[11] E. T. Güneyi, B. Yaldız, A. Canbolat, and E. Vural, "Learning graph arma processes from time-vertex spectra," *IEEE Transactions on Signal Processing*, vol. 72, pp. 47–56, 2024.

[12] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021.

[13] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under laplacian and structural constraints," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 825–841, 2017.

[14] J. S. Ovall, "The laplacian and mean and extreme values," *The American Mathematical Monthly*, vol. 123, pp. 287 – 291, 2016.

[15] J. Pang and G. Cheung, "Graph laplacian regularization for image denoising: Analysis in the continuous domain," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 1770–1785, 2017.

[16] H. M. Schey, *Div, Grad, Curl, and All That: An Informal Text on Vector Calculus*. New York: W.W. Norton & Company, 4th ed., 1997.

[17] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[18] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. USA: Prentice Hall Press, 3rd ed., 2009.

[19] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[20] E. Isufi, F. Gama, D. I. Shuman, and S. Segarra, "Graph filters for signal processing and machine learning on graphs," *IEEE Transactions on Signal Processing*, pp. 1–32, 2024.

[21] S. Segarra, G. Mateos, A. G. Marques, and A. Ribeiro, "Blind identification of graph filters," *IEEE Transactions on Signal Processing*, vol. 65, no. 5, pp. 1146–1159, 2017.

[22] H. Sevi, G. Rilling, and P. Borgnat, "Harmonic analysis on directed graphs and applications: From fourier analysis to wavelets," *Applied and Computational Harmonic Analysis*, vol. 62, pp. 390–440, 2023.

[23] F. Grassi, A. Loukas, N. Perraudin, and B. Ricaud, "A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 817–829, 2018.

[24] A. Loukas and N. Perraudin, "Stationary time-vertex signal processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2019, no. 1, p. 36, 2019.

[25] B. Osgood, *Lectures on the Fourier Transform and Its Applications*. Providence, Rhode Island: American Mathematical Society, 2019.

[26] M. Hayes, *Statistical Digital Signal Processing and Modeling*. Wiley India Pvt. Limited, 2009.

[27] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Separable autoregressive moving average graph-temporal filters," in *Proc. 24th EUSIPCO*, pp. 200–204, 2016.

[28] E. Isufi, A. Loukas, N. Perraudin, and G. Leus, "Forecasting time series with VARMA recursions on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 18, pp. 4870–4885, 2019.

[29] E. Pavez, H. E. Egilmez, and A. Ortega, "Learning graphs with monotone topology properties and multiple connected components," *IEEE Transactions on Signal Processing*, vol. 66, no. 9, pp. 2399–2413, 2018.

[30] J. V. d. M. Cardoso and D. P. Palomar, "Learning undirected graphs in financial markets," in *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pp. 741–745, 2020.

[31] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from filtered signals: Graph system and diffusion kernel identification," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 2, pp. 360–374, 2019.

[32] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[33] X. Pu, S. L. Chau, X. Dong, and D. Sejdinovic, "Kernel-based graph learning from smooth signals: A functional viewpoint," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 192–207, 2021.

[34] J. Guo, S. Moses, and Z. Wang, "Graph learning from signals with smoothness superimposed by regressors," *IEEE Signal Processing Letters*, vol. 30, pp. 942–946, 2023.

[35] T. Liao, W.-Q. Wang, B. Huang, and J. Xu, "Learning laplacian matrix for smooth signals on graph," in *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pp. 1–5, 2019.

[36] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Laplacian matrix learning for smooth graph signal representation," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3736–3740, 2015.

[37] Y. Liu, L. Yang, K. You, W. Guo, and W. Wang, "Graph learning based on spatiotemporal smoothness for time-varying graph signal," *IEEE Access*, vol. 7, pp. 62372–62386, 2019.

[38] R. Li, J. Wang, W. Xu, J. Lin, and H. Qiu, "Graph laplacian matrix learning from smooth time-vertex signal," *China Communications*, vol. 18, no. 3, pp. 187–204, 2021.

[39] A. Javaheri, A. Amini, F. Marvasti, and D. P. Palomar, "Joint signal recovery and graph learning from incomplete time-series," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 13511–13515, 2024.

[40] X. Mao, K. Qiu, T. Li, and Y. Gu, "Spatio-temporal signal recovery based on low rank and differential smoothness," *IEEE Transactions on Signal Processing*, vol. 66, no. 23, pp. 6281–6296, 2018.

[41] S. K. Kadambari and S. Prabhakar Chepuri, "Learning product graphs from multidomain signals," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5665–5669, 2020.

[42] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[43] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[44] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2019.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[46] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn: Multi-scale graph convolution for semi-supervised node classification," 2018.

[47] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," 2018.

[48] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," 2018.

[49] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 855–864, Association for Computing Machinery, 2016.

[50] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," 2018.

[51] A. Papoulis and S. Pillai, *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill series in electrical engineering: Communications and signal processing, McGraw-Hill, 2002.

[52] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, "GSPBOX: A toolbox for signal processing on graphs," *ArXiv e-prints*, Aug. 2014.

[53] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," Mar. 2014.

[54] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, pp. 95–110, Springer-Verlag Limited, 2008.

[55] K. Toh, R. Tutuncu, and M. Todd, "On the implementation of sdpt3 (version 3.1) - a matlab software package for semidefinite-quadratic-linear programming," in *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pp. 290–296, 2004.

[56] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, p. 3–34, jun 2015.

[57] B. Girault, "Stationary graph signals using an isometric graph translation," in *2015 23rd European Signal Processing Conference (EUSIPCO)*, pp. 1516–1520, 2015.

[58] "COVID-19 coronavirus pandemic data."

[59] "Eurostat: An official website of the European Union."

[60] H. Lütkepohl, *New introduction to multiple time series analysis*. Springer, 2005.

APPENDIX A

VALIDATION PROCEDURE FOR THE SELECTION OF ALGORITHM HYPERPARAMETERS

In this section, the selection of the regularization parameters μ_A, μ_B and α is inspected. Although various search methods can be employed to find the optimal parameters, we use a strategy similar to a greedy search. In our approach, the model parameters P, K, Q, and M of f_1, along with the regularization terms μ_A and μ_B of r_1, are searched jointly, while the regularization term α of r_2 is searched separately. This search method is employed because the two blocks in Figure 3.1 can be viewed as distinct optimization problems. The parameter α is determined by evaluating a range of values, as discussed in the sensitivity analysis in Section 4.2, and selecting the one with the lowest validation error. On the other hand, choosing the parameters P, K, Q, M, μ_A, and μ_B requires a more sophisticated approach due to the high dimensionality involved.

We search for optimal values of μ_A and μ_B by testing them with specific combinations of P, K, Q, and M, recording the lowest possible validation error. Once the best possible set of P, K, Q, and M values is identified, we fix these values for the remaining iterations and we only search for the optimal μ_A and μ_B parameters. The search method of μ_A and μ_B is as follows.

At each step, we fix a specific value of μ_A and increase μ_B by 10 times and record the validation error. The first step where the validation error starts to increase is recorded. At this point, two scenarios are possible, the error may continue to rise, or it may decrease up to a certain point and start to increase again. If the error continues to increase, the search is terminated, and the parameters corresponding to the lowest error are saved. If the error decreases, the search continues until another increase

is detected. At that point, the two local minima are compared, and the parameters corresponding to the lowest error are selected. The same approach is then applied by every possible value of μ_A, which increased 10 times at each step. This search method is faster than brute-force search in terms of time complexity and yields better results than the greedy search. Additionally, it can be parallelized to further accelerate the search process. To illustrate this process, a couple of scenarios are presented below.

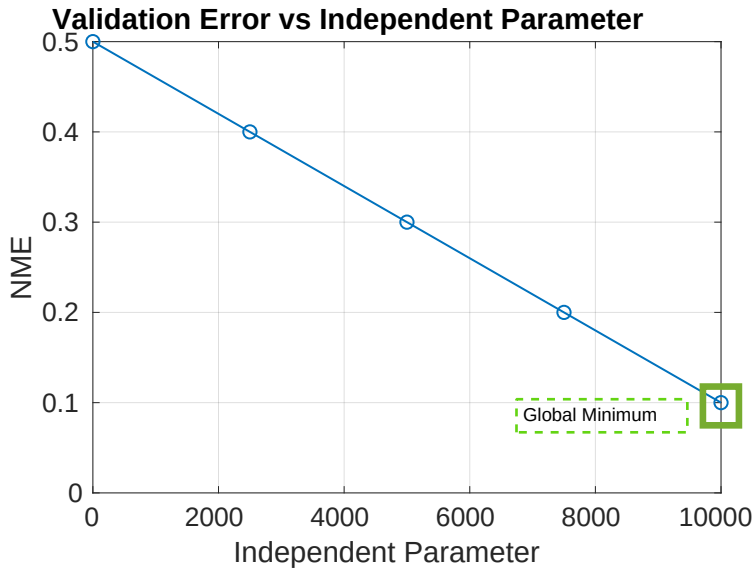

Figure A.1: Monotonic decrease scenario

In Figure A.1, the decision mechanism continues up to the highest parameter value in the set because the validation error decreases monotonically.

Figure A.2: Monotonic increase scenario

In Figure A.2, the decision mechanism selects the smallest parameter value in the set because the validation error increases monotonically. It continues although the error increases because there may be a sharp decrease in the error at some parameter values that has not been tried yet. The parameter value with the first local minimum is selected.

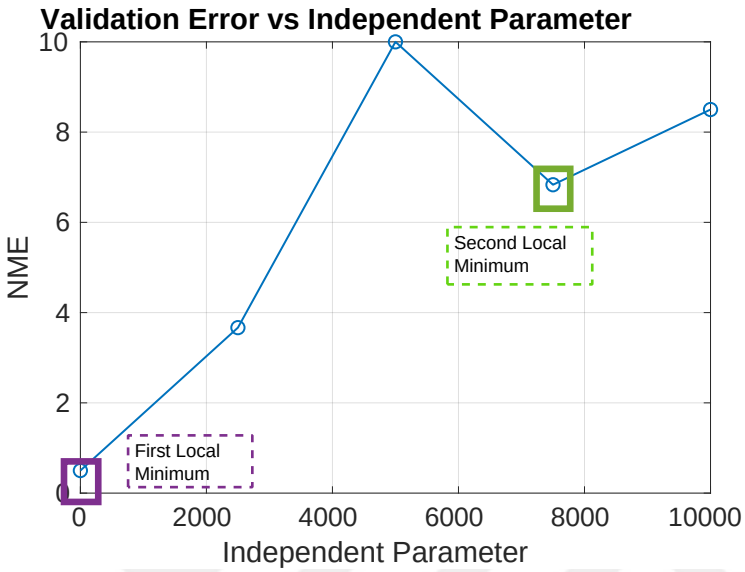

Figure A.3: Decrease and increase scenario

In Figure A.3, the validation error increases up to a point, then decreases and starts to increase again. Since a second increase occurs in the error at larger parameter values, the decision mechanism terminates at that point, compares the two local minima and decides the parameter value that corresponds to the first local minimum.

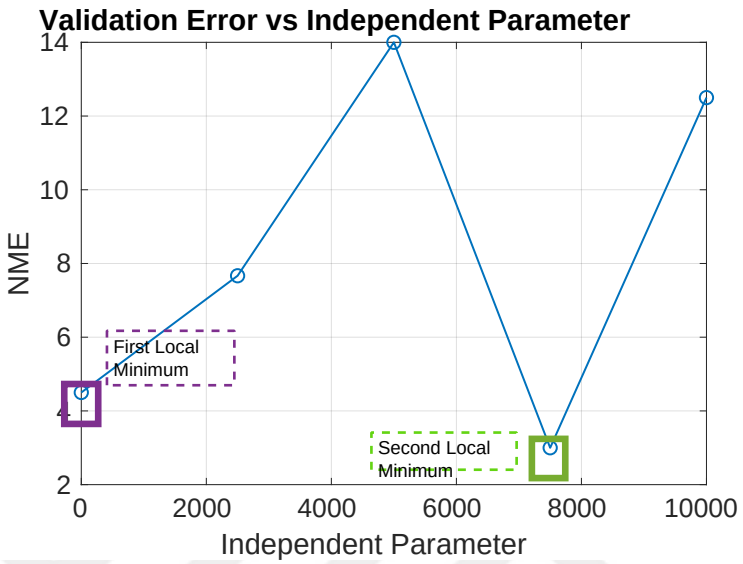

Figure A.4: Decrease and increase scenario - 2

Figures A.3 and A.4 are very similar except that the second local minimum is smaller than the first. Hence, the parameter value corresponding to the second local minimum is selected.

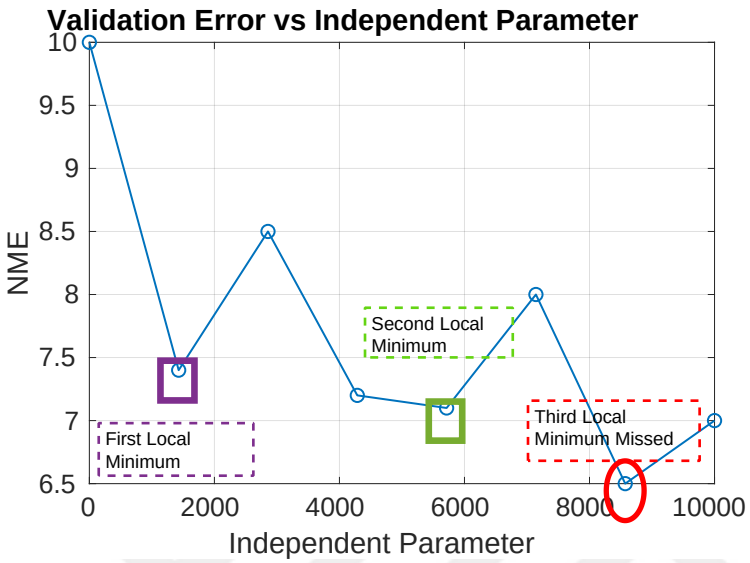

Figure A.5: Decrease and increase scenario - 3

In Figure A.5, the decision mechanism stops at the second increase step and misses the parameter with the third local minimum. It selects the parameter corresponding to the second local minimum.