Heriot-Watt University

Doctoral Thesis

Bayesian Reconsruction and Regression with Multivariate Graph Signals

Author:
Edward Antonian

Supervisors:
Prof. Mike Chantler &
Prof. Gareth W. Peters

A thesis submitted in fulfilment of the requirements for the degree of PhD.

in the

School of Mathematical and Computer Sciences

July 2023

Abstract

Graph Signal Processing (GSP) is a rapidly evolving field that combines ideas from spectral graph theory and classical signal processing to analyse and manipulate data residing on an irregular domain. In this thesis, we contribute several advancements to GSP theory, in particular, with regard to Bayesian reconstruction and regression techniques for multivariate graph signals. The first topic we consider is the reconstruction of signals existing on two-dimensional Cartesian product graphs, in the presence of noise and arbitrary missing data. Using numerical methods and the properties of the Kronecker product, we derive two efficient algorithms for computing the posterior mean and show how the optimal choice of technique depends on the model hyperparameters and sparsity of the input data. We then build on this by applying similar algorithms to solve several multivariate graph signal regression models. In particular, we generalise prior work on Kernel Graph Regression (KGR) and Regression with Network Cohesion (RNC), which are relevant when the explanatory variables are exogenous and endogenous respectively, by allowing for arbitrary patterns of missing data in the input signal. Following this, we adapt the reconstruction and regression methods developed prior in the thesis to the Multiway Graph Signal Processing (MWGSP) paradigm. MWGSP is an emerging sub-field that focuses on tensor-valued graph signals, where each axis is described by a unique graph topology. In order to help write effective and efficient MWGSP algorithms, we also present the PyKronecker library which creates an abstracted API for manipulating high-dimensional Kronecker-structured matrices. The next topic we consider is techniques for computing the posterior covariance of our models. First, we propose several algorithms for estimating the marginal posterior variance and compare them to other alternative standard techniques. Combined with an active learning strategy, we demonstrate that our procedure can generate greatly superior estimates, with $R^2 > 0.95$. We also derive an efficient algorithm for sampling directly from the posterior whilst avoiding computationally expensive MCMC-based approaches, using a technique known as perturbation optimisation. Finally, we develop new models that generalise our previous reconstruction and regression models to accommodate binary and categorical tensor graph signals. Each topic in this thesis is also accompanied by a real-world case study to corroborate the utility of the methods or demonstrate their correctness.

Acknowledgements

The acknowledgements and the people to thank go here.

"The more I read, the more I acquire, the more certain I am that I know nothing."

Voltaire

Contents

List of Figures

List of Tables

Abbreviations

GSP	Graph Signal Processing
GFT	Graph Fourier Transform
IGFT	Inverse Graph Fourier Transform
GSR	Graph Signal Reconstruction
KGR	Kernel Graph Regression
RNC	Regression with Network Cohesion
GLS	Generalised Least Squares
DCT	Discrete Cosine Transform
FCT	Fast Cosine Transform
FFT	Fast Fourier Transform
PSD	Positive Semi-Definite
SIM	Stationary Iterative Method
CGM	Conjugate Gradient Method
SNR	Signal to Noise Ratio

Symbols

Unless otherwise specified, the following naming conventions apply.

Integer constants

N	The number of nodes in a graph
T	The number of time points considered
M	The number of explanatory variables
Q	The number of queries

Integer variables

n	The index of a specific node in a graph
t	The index of a specific time point
m	The index of a specific explanatory variable
q	The index of a specific query
i, j, k	Generic indexing variables

Scalar variables

α	An autocorrelation regularisation parameter
β	A hyperparameter characterising a graph filter
γ	A precision parameter
λ	An eigenvalue *or* ridge regression penalty parameter
μ	The mean of a random variable
θ	AR(1) autocorrelation parameter
σ^2	The variance of a random variable

Matrices

$\mathbf{A}$	The graph adjacency matrix
$\mathbf{D}$	A diagonal matrix
$\mathbf{E}$	The prediction residuals
$\mathbf{F}$	A predicted graph signal
$\mathbf{G}$	A spectral scaling matrix
$\mathbf{H}$	A graph filter *or* Hessian matrix
$\mathbf{I}_N$	The $(N \times N)$ identity matrix
$\mathbf{O}_N$	An $(N \times N)$ matrix of ones
$\mathbf{K}$	A kernel (Gram) matrix
$\mathbf{L}$	The graph Laplacian
$\mathbf{S}$	A binary selection matrix
$\mathbf{U}$	Laplacian eigenvector matrix
$\mathbf{V}$	Kernel eigenvector matrix
$\mathbf{X}$	Data matrix of explanatory variables
$\mathbf{Y}$	(Partially) observed graph signal
$\mathbf{\Lambda}$	A diagonal eigenvalue matrix
$\mathbf{\Sigma}$	A covariance matrix
$\mathbf{\Phi}, \mathbf{\Psi}$	Generic eigenvector matrices
$\mathbf{\Omega}$	Log marginal variance matrix

Vectors/tensors

$\mathbf{1}_N$	A length-N vector of ones
$\mathbf{e}$	The prediction residuals
$\mathbf{e}_i$	The i-th unit basis vector
$\mathbf{f}$	The predicted graph signal
$\mathbf{s}$	A binary selection vector/tensor
$\mathbf{x}$	A vector of explanatory variables
$\mathbf{y}$	The observed graph signal
$\boldsymbol{\alpha}$	A flexible intercept vector/tensor
$\boldsymbol{\beta}$	A graph filter parameter vector *or* vector of regression coefficients
$\boldsymbol{\theta}$	A aggregated coefficient vector $[\boldsymbol{\alpha}^\top, \boldsymbol{\beta}^\top]^\top$

Functions

$g(\cdot)$	A graph filter function
$p(\text{statement})$	The probability that a statement is true
$\pi(\cdot)$	A probability density function
$\xi(\cdot)$	Optimisation target function
$\kappa(\cdot, \cdot)$	A kernel function

Operations

$(\cdot)^\top$	Transpose of a matrix/vector
$\|\| \cdot \|\|_\mathrm{F}$	The Frobenius norm
$\mathrm{tr}\,(\cdot)$	The trace of a square matrix
$\mathrm{vec}\,(\cdot)$	Convert a matrix to a vector in column-major order
$\mathrm{vec}_\mathrm{RM}(\cdot)$	Convert a matrix to a vector in row-major order
$\mathrm{mat}\,(\cdot)$	Convert a vector to a matrix in column-major order
$\mathrm{mat}_\mathrm{RM}(\cdot)$	Convert a vector to a matrix in row-major order
$\mathrm{diag}\,(\cdot)$	Convert a vector to a diagonal matrix
$\mathrm{diag}^{-1}\,(\cdot)$	Convert the diagonal of a matrix into a vector
$\otimes$	The Kronecker product
$\oplus$	The Kronecker sum
$\circ$	The Hadamard product

Graphs

$\mathcal{G}$	A graph
$\mathcal{V}$	A vertex/node set
$\mathcal{E}$	An edge set

Miscellaneous

$\hat{(\cdot)}$	The estimator of a matrix/vector/tensor
$O(\cdot)$	The runtime complexity
x_i	A vector element
$\mathbf{X}_i$	A matrix column
$\mathbf{X}_{ij}$	A matrix element

Identities

1	$\operatorname{vec}(\mathbf{AXB})$	$(\mathbf{B}^\top \otimes \mathbf{A})\operatorname{vec}(\mathbf{X})$
2	$\operatorname{tr}\left(\mathbf{A}^\top\mathbf{B}\right)$	$\operatorname{vec}(\mathbf{A})^\top \operatorname{vec}(\mathbf{B})$
3	$\mathbf{AC} \otimes \mathbf{BD}$	$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D})$
4	$(\mathbf{A} \otimes \mathbf{B})^{-1}$	$\mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$
5	$\operatorname{tr}\left(\mathbf{X}^\top\mathbf{AYB}\right)$	$\operatorname{vec}(\mathbf{X})^\top (\mathbf{B}^\top \otimes \mathbf{A})\operatorname{vec}(\mathbf{Y})$
6	$\operatorname{vec}(\mathbf{J} \circ \mathbf{Y})$	$\operatorname{diag}\big(\operatorname{vec}(\mathbf{J})\big)\operatorname{vec}(\mathbf{Y})$
9	$\operatorname{diag}^{-1}\big(\mathbf{A}\operatorname{diag}(\mathbf{x})\,\mathbf{B}\big)$	$(\mathbf{B}^\top \circ \mathbf{A})\,\mathbf{x}$

For/Dedicated to/To my...

Chapter 1

Introduction

In the field of discrete mathematics, the term "graph" denotes a collection of distinct objects that may possess some form of pairwise connection or association. The discrete objects that make up the graph are referred to as nodes (or vertices), and their connections are known as edges (or arcs). This abstract structure can be employed to represent numerous real-world phenomena. For example, in an airline route map, nodes could symbolise airports, and edges could indicate the presence of a direct flight.

Graphs and network models are utilised in many actively researched areas of mathematics, including network processes such as epidemic modelling [Pare et al., 2020], Graph Neural Networks (GNNs) [Zhou et al., 2020], graphical models [Holmes and Jain, 2008], and semi-supervised learning [Chong et al., 2020].

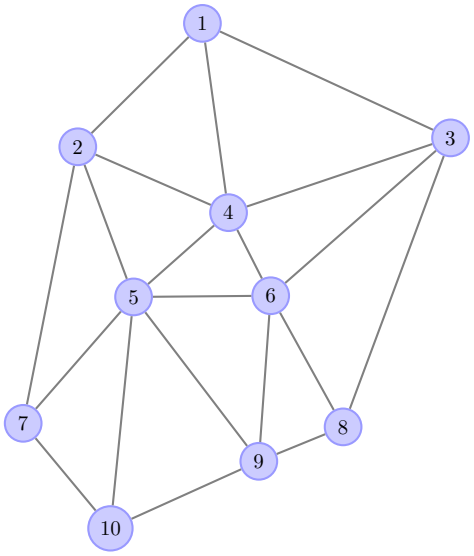


FIGURE 1.1: A visual representation of a simple graph with 10 nodes and 20 edges.

In this thesis, the subject of focus is Graph Signal Processing (GSP), a rapidly evolving field that sits at the intersection of spectral graph theory, statistics, and data science [Ortega et al., 2018]. GSP is an actively researched area devoted to the mathematical analysis of signals that are defined over the nodes of a graph, simply referred to as *graph signals*.

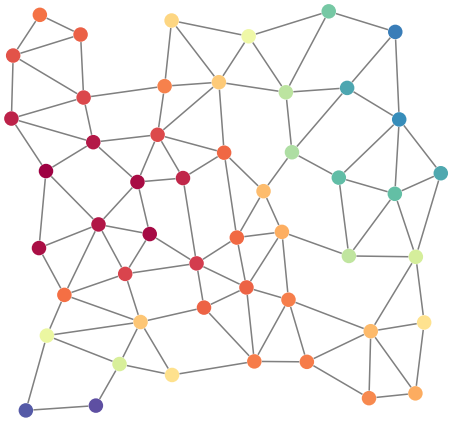


FIGURE 1.2: A graphical depiction of a graph signal. Here, the value of the signal at each node is represented by its color.

A graph signal represents a value that is measured simultaneously at every node in a graph. For example, consider a social media network, where each node represents an individual, and presence of an edge indicates that the two individuals have connected. An example of a graph signal in this context could be the age of each person in the network, or a score indicating their propensity to engage with certain content. In either case, this is a value that could theoretically be measured or estimated across the network, though it may be subject to missing data or noise.

GSP typically uses tools that stem from classical signal processing, where the goal is to manipulate data that resides on a regular domain such as audio, images or radio. GSP utilises spectral graph theory to generalise these classical techniques, leading to graph-based counterparts of algorithms such as filtering, sampling, and reconstruction [Shuman et al., 2013]. Applications of GSP are numerous, ranging from social networks [Dong et al., 2015] and brain connectivity modelling [Huang et al., 2016] to sensor networks [Zhu and Rabbat, 2012], and molecular structures [Kearnes et al., 2016]. As an emerging field, there are many opportunities for the development of theory, and new use-cases to explore, making it an exciting and dynamic area of research.

1.1 Research goals

The goal of this thesis is to provide contributions to the areas of multivariate graph signal reconstruction and regression. A single signal, measured over a network comprising N nodes, is an example of a univariate structure, and can be described by a vector $\mathbf{y} \in \mathbb{R}^N$. Our methods, by contrast, are designed to be applicable to higher-dimensional objects. For example, T samples of a graph signal measured across time can be described by a matrix $\mathbf{Y} \in \mathbb{R}^{N \times T}$. This kind of two-dimensional data can also be viewed as a single graph signal measured on the nodes of a Cartesian product graph [Imrich and Klavžar, 2000]. More generally, a tensor-valued signal $\mathcal{Y} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$ can be understood as a signal on a d-dimensional Cartesian product graph [Stanley et al., 2020].

Our aim is to expand the existing literature on multivariate GSP by presenting several novel Bayesian models for reconstructing partially observed graph signals, with and without additional explanatory variables. Graph signal reconstruction is a well-studied topic

in GSP, however, the majority of published literature to date has focused on univariate, or time-varying signal models that provide point estimates for the missing values. We aim to generalise this in two ways. First, we present a rich Bayesian formulation, that outputs not only a point estimate, but a full probability distribution quantifying prediction uncertainty. Furthermore, our goal is to produce models applicable to graph signals defined on arbitrary Cartesian product graphs, which naturally extend univariate and time-varying models. While the input to a graph signal reconstruction problem is typically only the partially observed graph signal and the topology of the underlying graph, we also consider the situation in which additional explanatory variables are available to aid in the estimation process. Therefore another key objective of ours is to produce graph signal regression models, applicable to a range of different data scenarios, that are robust to missing data.

As mentioned, a key aspect of our models is the Bayesian formulation, which produces a full posterior probability distribution over the unknown reconstructed signal, rather than a single point estimate. Given the high-dimensional nature of the data, the posterior covariance, which contains information regarding the corelation between every element in the multivariate signal $\boldsymbol{\mathcal{Y}}$, is often too large to process with consumer-grade computer hardware. Therefore, another objective of ours is to develop efficient methods for extracting information regarding the prediction uncertainty. In particular, we consider two related but separate tasks: estimating the marginal posterior variance (node-level uncertainty) and drawing samples directly from the posterior.

The majority of GSP models tend to focus on real-valued graph signals, leaving signals with binary or categorical entries relatively unexplored. It is therefore an additional goal of ours to develop new statistical reconstruction and regression algorithms for such graph signals. Although related tasks have been examined within semi-supervised learning in machine learning and GNNs in deep learning, we aim to introduce and showcase novel Bayesian models that are grounded firmly in the theoretical framework of GSP.

The high-dimensional nature of the models explore in this thesis comes with a unique set of challenges concerning computational robustness and efficiency. As such, a core objective is to reduce the computational cost of our algorithms wherever possible and to provide a detailed and transparent analysis of their time and memory complexity. The models we present are designed to be applicable to a wide range of application areas, and as such, throughout this thesis we also provide illustrative examples and case-studies to demonstrate their applicability.

1.2 GSP fundamentals

A graph, $\mathcal{G}$, with N vertices is described by a node set $\mathcal{V}$, and an edge set $\mathcal{E}$. In this thesis, we will be primarily concerned with undirected graphs without self-loops, meaning the edges have no preferential direction and nodes do not connect to themselves. By imposing some arbitrary but consistent ordering on the nodes, this graph can also be described by an $N \times N$ adjacency matrix $\mathbf{A}$, where the entry $\mathbf{A}_{ij} = \mathbf{A}_{ji} \geq 0$ holds the strength of connection between nodes i and j. In the basic case of a non-weighted graph, $\mathbf{A}_{ij}$ is simply one if the corresponding edge exists and zero otherwise. Using the same ordering, a graph signal can be represented by a vector, $\mathbf{y}$, of length N, where $\mathbf{y}_i$ holds the value of the graph signal at node i.

One key property of a graph signal is its *smoothness*. Intuitively speaking, a smooth graph signal should exhibit gentle variation between closely connected nodes, as illustrated in fig. 1.2. Conversely, a rough graph signal might see large jumps in signal value between neighbouring nodes. Mathematically, the smoothness of a signal, $\mathbf{y}$, defined over the nodes of an undirected graph, can be measured in several ways. One simple option is the total square variation (TV_2), which is defined as follows.

$$\mathrm{TV}_2(\mathbf{y}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{A}_{ij}(\mathbf{y}_i - \mathbf{y}_j)^2 \tag{1.1}$$

This metric, also known as Dirichlet energy, sums up the square difference in signal value at each neighbouring node, weighted by the corresponding entry in the adjacency matrix, with the factor of a half adjusting for the double counting of nodes. It can also be written in terms of a single quadratic form, by introducing a new matrix $\mathbf{L}$ - the so-called graph *Laplacian*.

$$\mathrm{TV}_2(\mathbf{y}) = \mathbf{y}^\top\mathbf{L}\mathbf{y} \tag{1.2}$$

The Laplacian, like the adjacency matrix $\mathbf{A}$, is another symmetric $N \times N$ matrix, and is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix. $\mathbf{D}$ is a diagonal operator, where entry $\mathbf{D}_{ii}$ holds the sum of all the edge weights linked to node i, or in other words, the vector along the diagonal of $\mathbf{D}$ is the column (or row) sum of $\mathbf{A}$. The Laplacian, which can be interpreted as a generalisation of the discrete second order derivative operator for an irregular topology, is of central importance to many aspects of GSP [Shuman et al., 2013].

It is straightforward to show that the two expressions for the total square variation given in eqs. (1.1) and (1.2) are equivalent; see for example chapter 3 of Ortega [2022]. A few basic facts stand out about the Laplacian quadratic form.

1. $\mathbf{y}^\top \mathbf{L}\mathbf{y} \geq 0$ for any $\mathbf{y}$. Since $\mathrm{TV}_2(\mathbf{y})$ sums the square difference between the signal at each pair of nodes, weighted by the non-negative entries of the adjacency matrix, the Laplacian quadratic form must be strictly non-negative. By definition, this implies that the matrix $\mathbf{L}$ is positive semi-definite (PSD).

2. $\mathbf{1}^\top \mathbf{L}\mathbf{1} = 0$, where $\mathbf{1}$ is a length-N vector of ones. If the signal of interest is constant over the whole graph, the total square variation must be zero. Moreover, if the graph contains two or more isolated sub-graphs, with edges connecting nodes within a each clique but not between cliques, then any signal that is constant over each sub-graph will have a Laplacian quadratic form of zero.

Since $\mathbf{L}$ is positive semi-definite, its eigenvalues are all real and non-negative, and its eigenvectors can be chosen to be real and orthonormal. Thus, $\mathbf{L}$ can be decomposed as follows.

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \tag{1.3}$$

Here, $\mathbf{U}$ is the orthogonal, $N \times N$ matrix of eigenvectors $\mathbf{u}_i$, and $\mathbf{\Lambda}$ is the diagonal matrix of corresponding eigenvalues λ_i, typically arranged in ascending order. As $\mathbf{U}$ is orthonormal, it holds that $\mathbf{u}_i^\top \mathbf{u}_j = \delta_{ij}$, and that $\mathbf{u}_i$ form a set spanning $\mathbb{R}^N$. Given the definition of the Laplacian quadratic form, the eigenvectors are the unique set of orthonormal vectors that sequentially minimise the total square variation, subject to perpendicularity to all those preceding [Spielman, 2019].

$$\begin{aligned}
\mathbf{u}_1 &= \underset{|\mathbf{u}|^2=1}{\operatorname{argmin}} \quad \mathrm{TV}_2(\mathbf{u}) \\
\mathbf{u}_2 &= \underset{|\mathbf{u}|^2=1,\, \perp\mathbf{u}_1}{\operatorname{argmin}} \quad \mathrm{TV}_2(\mathbf{u}) \\
\mathbf{u}_3 &= \underset{|\mathbf{u}|^2=1,\, \perp\mathbf{u}_1,\mathbf{u}_2}{\operatorname{argmin}} \quad \mathrm{TV}_2(\mathbf{u}) \\
\mathbf{u}_4 &= \quad \dots
\end{aligned}$$

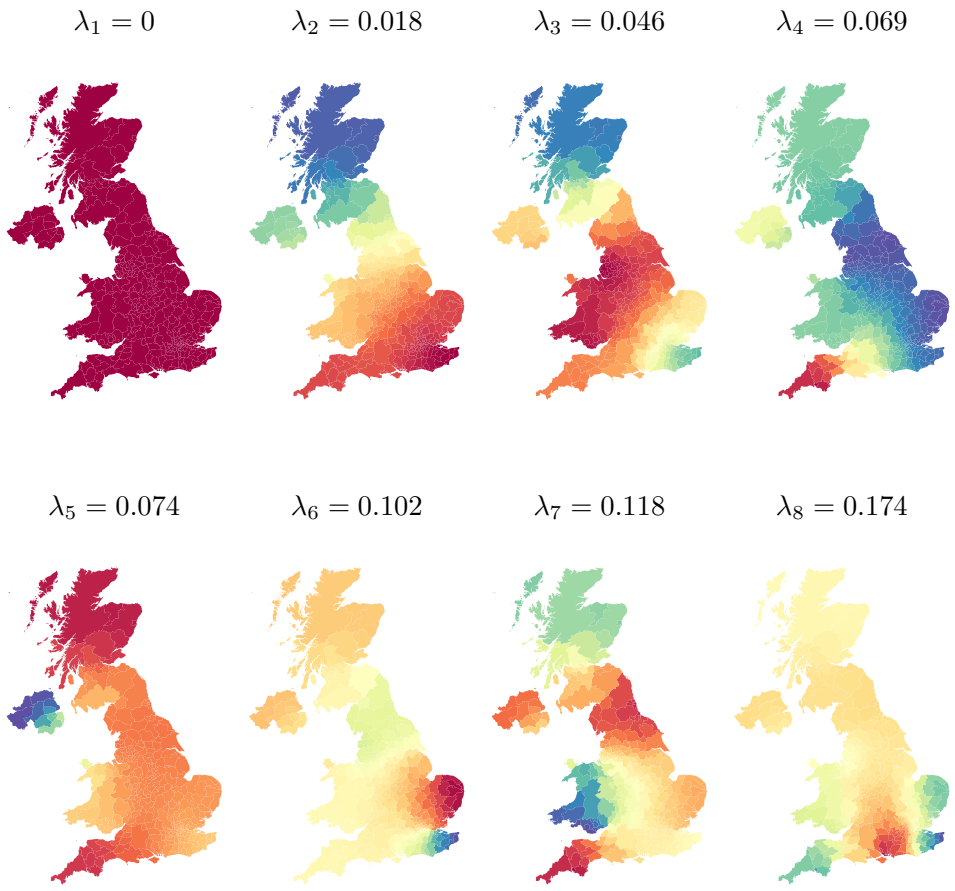


FIGURE 1.3: A visualisation of the first eight Laplacian eigenvectors, alongside their associated eigenvalues, for a network of local authority regions in the UK. Each node represents a region, and each pair of regions share an edge if they border one another (or have a direct ferry crossing). Colour is used to represent the value of the graph signal.

In this way, the eigenvectors of the graph Laplacian can be understood as sequentially less smooth with respect to the topology of the graph. The corresponding eigenvalue, referred to as the frequency, gives a value specifying how "rough" each eigenvector is relative to the others, as measured by TV_2. Note that, for any undirected graph, the first Laplacian eigenvector will always be constant with an eigenvalue of zero. Figure 1.3 gives a visual depiction of the first eight Laplacian eigenvectors and eigenvalues for a network representing local authority regions in the UK. [1]

Since $\mathbf{u}_i$ span the total space of $\mathbb{R}^N$, any graph signal, $\mathbf{y}$, can be decomposed into a weighted sum of the Laplacian eigenvectors. This parallels the classical Fourier transform, which expands signals into the basis of complex exponentials [Sneddon, 1995]. As such, any graph signal, $\mathbf{y}$, has a dual representation, $\mathbf{z}$, in the frequency domain. Transformations between these two representations can be achieved by applying the Graph

[1] Using boundary data published by the Office for National Statistics [ONS, 2019]

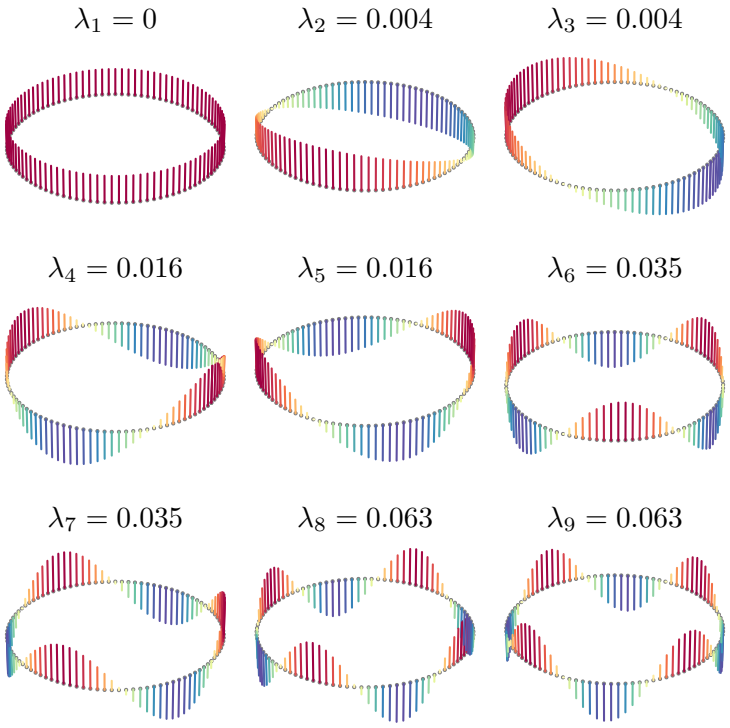


FIGURE 1.4: A visualisation of the first nine Laplacian eigenvectors of a cycle graph with 50 nodes, along with their associated eigenvalue.

Fourier Transform (GFT) and Inverse Graph Fourier Transform (IGFT), which amount to multiplying by the matrices $\mathbf{U}^\top$ and $\mathbf{U}$, respectively.

$$\mathbf{z} = \text{GFT}(\mathbf{y}) = \mathbf{U}^\top \mathbf{y} \tag{1.4}$$

$$\mathbf{y} = \text{IGFT}(\mathbf{z}) = \mathbf{U}\mathbf{z} \tag{1.5}$$

For some simple graphs, the GFT is equivalent to a well-known transform in classical signal processing. For example, the Laplacian of the cycle graph (i.e. a set of nodes connected in a loop) has eigenvectors that can be expressed as sines and cosines (or, in full generality, complex exponentials) as visualised in fig. 1.4. This helps establish a rigorous connection between GSP and classical signal processing, through the study of Algebraic Signal Processing (ASP) [Puschel and Moura, 2003, Sandryhaila and Moura, 2013b]. With the GFT and IGFT defined, a wide range of potential models are opened up, many of which can be understood as direct generalisations of methods from conventional signal processing. Classical tasks such as filtering, denoising, reconstruction, sampling, compression etc. can all be translated into the GSP paradigm, setting the stage for a new era of signal processing that leverages graph structures for more complex and rich data analysis, and opening the door to a wealth of novel applications.

1.3 Example applications

Graph signal processing methods have proven effective in many real-world applications. In this short section, we highlight some use-cases from the literature, serving as motivation for the content of this thesis. Many of these areas are utilised within illustrative examples for the methods developed in this thesis.

1.3.1 Biology

The world of biology provides numerous interesting use cases for graph signal processing methods [Li et al., 2023]. One area which has garnered significant interest is in analysing protein-protein interactions. Here, the concept is to model a set of proteins as nodes in an "interactome", which is the network of physical molecular interactions in a particular cell. Recent work such as Colonnese et al. [2021] used GSP methods to predict new interactions between proteins, while work such as Jha et al. [2022] used graph neural networks for the same task. These models aim to automate the prediction of protein interactions, reducing the time and cost associated with experimental methods. Other work has modelled an individual protein molecule as network of residues connected to each other based on their physical distance in 3D space, with the aim of predicting its biophysical properties [Srivastava et al., 2023].

Another domain in biology that has benefited from GSP methods is brain connectivity modelling. Studies such as Atasoy et al. [2016], Goldsberry et al. [2017], Itani and Thanou [2021], Menoret et al. [2017] have found that the graph-spectral frequency profile of signals derived from fMRI imaging can provide valuable insight into brain activity. Specifically, these and related works have identified specific Laplacian spectral profiles associated with different emotional states, task familiarity, and neurological diseases.

1.3.2 Transportation and infrastructure

Another domain in which GSP methods have been extensively applied is transportation and infrastructure. In Hasanzadeh et al. [2017], the authors propose an adaptive ARMA model in the graph frequency domain for real-time traffic prediction, achieving a substantial performance increase over non-graph-aware alternatives. Traffic prediction was also addressed using GSP methods in Chakraborty [2017] with a technique known as trend filtering [Wang et al., 2016]. Automatic incident detection using spatio-temporally denoised thresholds was applied in [Chakraborty et al., 2019]. In Xiu et al. [2022], the authors used a spatial-temporal multi-graph convolutional wavelet network to predict passenger numbers on a metro system.

GSP has also been employed to analyse patterns of power consumption in electricity grids [Ramakrishna and Scaglione, 2021] and pressure in hydraulic networks [Zhou et al., 2022b]. In He et al. [2018] and Zheng et al. [2022], the authors utilised GSP to address non-intrusive load disaggregation—an active area of research in the field of smart grids and energy conservation where the goal is to estimate each appliance's contribution to total consumption. In Ying et al. [2022], the authors tackle the recovery of harmonic data lost during power transmission, based on spectral graph theory, and in Wang et al. [2022c], the authors use GSP to identify abnormal batteries from cell-level voltage signals in a network of electric bicycle charging stations.

1.3.3 Finance and economics

Networks have been used in financial and economic models for many years [Marti et al., 2021], yet recently GSP has found some interesting use-cases. In Vinicius et al. [2020], the authors employ a spectral model to learn an underlying graph structure from financial time-series data, showing that meaningful physical interpretations related to the market index factor can be extracted from this representation. In [Zhang et al., 2023], the authors consider how GSP can be incorporated into momentum-based financial forecasting models, demonstrating that the additional topological information can enhance the accuracy of such models.

GSP has also been used recently in the context of asset allocation and portfolio theory. The portfolio cut paradigm, introduced in [Dees et al., 2020], is a graph-theoretic portfolio partitioning technique that enables investors to group economically meaningful clusters of assets. In [Arroyo et al., 2022], this concept is extended to include a dynamic portfolio, with time-evolving clusters.

Graph Neural Networks have also been used extensively in financial applications [Wang et al., 2022a].

1.3.4 Sensor networks

Sensor networks are a classic application for GSP methods, as a graph is typically straightforward to construct from the physical positioning of the devices [Jablonski, 2017]. Many applications in this domain focus on distributed algorithms, as sensors and IoT devices are naturally suited to local communication. The aims of such applications are varied, encompassing tasks like compression [Zhu and Rabbat, 2012], denoising [Tay, 2021], reconstruction [Wang et al., 2015b], and distributed data processing [Chi et al., 2022].

Some of the earliest work classified as GSP involved applications with sensor data, for example, Guestrin et al. [2004]. Here, the authors used local basis functions to design a regression algorithm capable of fast convergence with only local communication. In Wagner et al. [2005], the authors develop a distributed wavelet transform and applied it to a signal compression task over a network of sensors.

1.4 Thesis organisation

The core content of this thesis begins in chapter 3, with the reconstruction of signals with two axes, such as network time series data. Chapter 4 introduces several regression techniques for two-dimensional graph signals, and chapter 5 generalises the techniques developed prior in the thesis to d-dimensional data, under the "MultiWay" GSP (MWGSP) framework. This is followed, in chapter 6, by an investigation into the posterior covariance of the models, and finally chapter 7 introduces generalisations for binary and categorical graph signals. Along the way, there are several methodological contributions as well as some detailed case studies. In chapter 2, we define the precise scope of this thesis, review the relevant literature, and present our core contributions.

Chapter 2

Literature Review, Contributions and Scope

The aim of this literature review is to provide an overview of the field of Graph Signal Processing (GSP) with a particular emphasis on multivariate reconstruction and regression models. First, we will give some historical context for the subject, highlighting the key theoretical advancements and canonical examples, before moving towards the contemporary developments relevant to this thesis. This begins with a basic review of graph filters and kernels, which are foundational building blocks of the methods presented in this thesis. Next, we discuss the topic of graph signal reconstruction, before covering different approaches to regression with graph signals. This is followed by a survey the emerging field of multiway graph signal processing, and finally we discuss node classification methods. For each of this topics we explain how the methods presented in this thesis contribute to the field.

By presenting a holistic view of the existing literature, this review aims to set a firm foundation upon which we can build the discussion for our ensuing research questions. In addition, this chapter will also hone in on the specific scope of this thesis and define the boundaries of our research. In particular, we aim to identify the areas of interest that remain under-explored or incomplete in the current state of research, as well as clearly demarcate the topics which are not directly relevant to our research focus.

2.1 A historical perspective on GSP

Though GSP as a field in its own right is generally understood to have been established in the early 2000s, its underlying conceptual framework draws upon the well-established

fields of Spectral Graph Theory (SGT), and Digital Signal Processing (DSP), which both date back several decades. SGT is a branch of mathematics that studies the eigenvalues and eigenvectors associated with a graph's adjacency or Laplacian matrix to gain insight into its structural properties [Chung, 1997]. Early work on SGT can be traced back to the 1950s and '60s [Collatz and Sinogowitz, 1957, Hoffman, 1969], although many of the concepts had already been studied in parallel within quantum chemistry [Huckel, 1931]. One of the foundational results in SGT is the multiplicity of the Laplacian's zero eigenvalue gives the number of connected components in the graph [Cvetkovic et al., 1980]. Another key result is Cheeger's inequality, which relates the second smallest eigenvalue of the Laplacian (also known as the algebraic connectivity) to the isoperimetric number of the graph (a measure of a graph's bottleneck) [Cheeger, 1971].

Digital Signal Processing (DSP), sometimes considered a branch of electrical engineering, utilises digital computation to analyse, transform, or filter signals, which can be in forms such as sound, images, and sensor data [Rabiner and Gold, 1975]. The core principles of DSP are grounded in linear algebra, calculus, differential equations, and statistics. This theoretical underpinning has led to a plethora of practical applications across varied fields, such as telecommunications, audio processing, image and video processing, astronomy, and seismology, to name a few. Within DSP, the Discrete Fourier Transform (DFT) and its fast algorithmic implementation, the Fast Fourier Transform (FFT), play a central role, with many tasks such as reconstruction, denoising, compression, and filtering requiring analysis and manipulation of the frequency content of signals [Duhamel and Vetterli, 1990].

GSP began to take shape as a separate discipline around the start of the 21st century, as the proliferation of data and advancements in computer technology gave rise to more complex, irregular, and high-dimensional data structures. The theoretical framework underpinning GSP emerged from work on Algebraic Signal Processing (ASP), with several papers published by Puschel and Moura from 2003 to 2008 establishing an axiomatic approach to discrete time signal processing [Puschel and Moura, 2003, 2006, 2008]. This mathematical formalism, based on the concept of shift operators, established a unifying framework for several concepts from classical signal processing. For example, under the ASP paradigm, the Discrete Cosine Transform (DCT) and DFT are understood as generated from different discrete shift operators (a chain and cycle respectively) [Isufi et al., 2022].

Meanwhile, in the data science and machine learning community, concepts from SGT were being applied to nonlinear dimensionality reduction using the Laplacian eigenbasis [Belkin and Niyogi, 2003, Donoho and Grimes, 2003, Roweis and Saul, 2000]. This was

followed by work such as Kondor and Lafferty [2002] and Smola and Kondor [2003], who studied the topic of graph kernels. This work was subsequently applied in semi-supervised learning, where the goal is to maximise the utility of unlabelled data using its topology in feature space [Belkin et al., 2004, Zhou and Schölkopf, 2004, Zhu et al., 2003]. We return to the topic of graph kernels in greater detail in section 2.2.

Around the same time, in the signal processing community, authors were working on both distributed and global algorithms for denoising and regression on sensor networks Guestrin et al. [2004], Wagner et al. [2006, 2005].

In the early 2010s, two distinct yet complementary approaches to GSP emerged, as discussed in Leus et al. [2023]. The first approach, often credited to Sandryhaila and Moura [2013b,c], adopted an algebraic perspective, building upon existing work in Algebraic Signal Processing. This methodology primarily focused on defining the Graph Fourier Transform and related concepts using the foundational operation of graph shifts. Conversely, the second approach, as proposed by Hammond et al. [2011], Shuman et al. [2013], endorsed the use of the graph Laplacian as the core component for GSP algorithms. This second paradigm, which aligns closely with the concepts presented in section 1.2, is also the primary approach utilised in this thesis.

2.2 Graph kernels and graph filters

In GSP, two distinct but closely-related concepts are graph filters and graph kernels. In the Laplacian-based GSP paradigm, both can understood as spectral operators which modify the frequency profile of a graph signal, providing a direct generalisation of conventional filters in classical signal processing. (It is worth noting that, in this context, 'graph kernel' should not be confused with another concept, bearing the same name, which refers to distance measures between pairs of graphs [Kriege et al., 2020].) Graph filters/kernels play a fundamental role in GSP, and are at the heart of numerous algorithms including reconstruction [Romero et al., 2017b], anomaly detection [Xiao et al., 2021], GNNs [Gama et al., 2020] and graph learning [Mateos et al., 2019].

A graph filter or kernel is characterised by a function $g(\lambda)$, which is applied to the frequency profile of a signal $\mathbf{y}$. This is performed by initially transforming $\mathbf{y}$ into the frequency domain via the GFT, then subsequently applying $g(\cdot)$ to each frequency component independently, and finally transforming back to the node domain via the IGFT. Consequently, it can be represented by a matrix operator $\mathbf{H} \in \mathbb{R}^{N \times N}$ as follows.

$$\mathbf{H} = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^\top \tag{2.1}$$

Filter	$g(\lambda;\, \beta)$
1-hop random walk	$(1+\beta\lambda)^{-1}$
Diffusion	$\exp(-\beta\lambda)$
ReLu	$\max(1-\beta\lambda, 0)$
Sigmoid	$2\big(1+\exp(\beta\lambda)\big)^{-1}$
Gaussian	$\exp\big(-(\beta\lambda)^2\big)$
Bandlimited	1, if $\beta\lambda \leq 1$ else 0

TABLE 2.1: Some example low-pass graph filter functions

Equivalently, $\mathbf{H}$ can be understood as originating from the direct application of the function $g(\cdot)$ to the Laplacian, represented as $\mathbf{H} = g(\mathbf{L})$. Here, this operation should be interpreted as an analytic function of a matrix, that is, a series sum of matrix powers [Bhatia, 1997]. The function $g(\cdot)$ can be set such that it promotes certain properties in the output signal. For example, a low-pass filter (as those specified in table 2.1) is a monotonically decreasing function, attenuating the high-frequency content of a signal. The choice of filter will depend on the application of interest, and may rely on expert domain knowledge about the spectral profile of the graph signals.

In the GSP community, the term 'filter' is typically preferred when referring to this class of operator [Shuman et al., 2013], although some favour the term 'kernel' [Ioannidis et al., 2016, Romero et al., 2017a]. Often, the primary practical distinction is that a kernel is an increasing function, used to construct penalty terms in regularised problems, whereas a filter is a decreasing one, interpreted as providing spectral transformations on graph signals. In Romero et al. [2017b], the authors advocate for the preferential use of the term kernel, arguing that many GSP methods have an elegant interpretation in terms of Reproducing Kernel Hilbert Spaces (RKHSs). Within the machine learning community, where kernel methods and RKHSs are common, the kernel interpretation is typically favoured [Kondor and Lafferty, 2002, Smola and Kondor, 2003]. Regardless, either convention can be used, and in this thesis, we predominantly describe our methods in terms of filters since they provide a more direct correlation with conventional signal processing techniques, offering an intuitive way to manipulate and reason about the spectral content of graph signals.

2.2.1 Probability distributions over graph signals

Spectral operators can be used to define probability distributions over smooth graph signals. An early example of this approach can be found in Gadde and Ortega [2015], where the authors propose a Gaussian Random Field (GRF) model for generating graph signals. In particular, they propose a method for vertex sampling that treats signals as random variables drawn from the following underlying multivariate Gaussian distribution.

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, (\mathbf{L} + \delta\mathbf{I})^{-1}\right) \tag{2.2}$$

This probabilistic interpretation offers a principled way of reasoning about graph signals, making the connection between GSP and specific methods from semi-supervised learning more apparent [Zhu et al., 2003]. A more general probabilistic framework for graph signal processing is proposed in Zhang et al. [2015], where distributions are parametrised by graph filters, offering increased flexibility. Once again, graph signals are interpreted as Gaussian Random Fields, however, the covariance structure is expanded to encompass a greater variety of spectral profiles. In particular, the distribution is given by

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \gamma^{-1}\mathbf{H}^2\right) \tag{2.3}$$

where $\mathbf{H}$ is a generic graph filter. Since $\mathbf{H}^2 = \mathbf{U}g^2(\mathbf{\Lambda})\mathbf{U}^\top$, this distribution can be interpreted as an independent multivariate Gaussian in the Laplacian eigenbasis, where the covariance of each basis vector is given by $g^2(\lambda_i)$. Since, for a low pass filter, $g(\cdot)$ is monotonically decreasing, this naturally results in higher probability density placed on smoothly varying graph signals.

Further insight into this model can be gained from the following consideration. First, start with an independent multivariate normally distributed random variable, denoted as $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$. Applying a graph filter yields a new signal, $\mathbf{Hy}$, which has its high frequency content attenuated in accordance with the properties of the filter. This filtered signal will also be normally distributed, according to $\mathbf{Hx} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{H}^2\right)$. Therefore the process of filtering white Gaussian noise is equivelent to drawing from a distribution with mean zero and covariance $\mathbf{H}^2$, meaning $\mathbf{H}^2$ can be interpreted as a covariance matrix for generating smooth, normally distributed graph signals. This flexible approach permits the encoding of various beliefs regarding the frequency profile of graph signals. Figure 2.1 displays an example of a random signal drawn from the distribution of eq. (2.3) using a bandlimited filter.

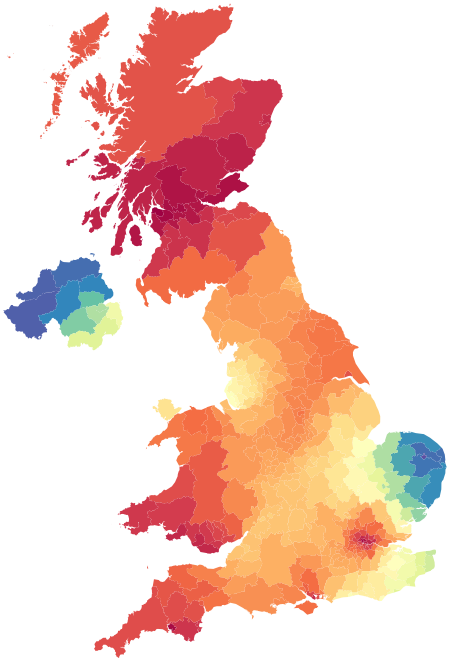


FIGURE 2.1: An example of a random smooth signal on the UK district graph, sampled from $\mathcal{N}(\mathbf{0}, \mathbf{H}^2)$ using a bandlimited filter.

This approach is also embraced in Venkitaraman et al. [2020], where the authors use graph filters to construct a Gaussian process regression model over graphs. In particular, they use a GRF to specify the prior distribution over their weight matrix, resulting in predictions which are smooth with respect to the graph topology. The filter-based GRF model also provides interpretation for the work of Gadde and Ortega [2015], which can be seen as a special case of this more general formulation. In particular, their chosen covariance matrix can be written in the form $(\mathbf{L} + \delta\mathbf{I})^{-1} = \mathbf{U}g^2(\mathbf{\Lambda})\mathbf{U}^\top$, where the filter function is given by $g(\lambda) = 1/\sqrt{\lambda + \delta}$. In a similar manner, [Perraudin and Vandergheynst, 2017] suggest a flexible class of Laplacian-based covariance matrices, however, their focus is on stationary signals with potentially non-Gaussian distributions. In other work such as Dong et al. [2016], the authors take a Bayesian approach to the problem of graph learning, using similar graph-spectral priors.

In general, many types of GSP models utilising some form of quadratic Laplacian-based regularisation can be interpreted in Bayesian terms, although this often goes unnoted. In this thesis, we follow the example of Zhang et al. [2015] in describing signal distributions with GRFs. However, unlike any of the aforementioned work, we extend this to multidimensional graph signals (two-dimensional matrix signals in chapter 3, and d-dimensional tensor signals in chapter 5). This allows us to build a rich class of Bayesian models for tasks such as multivariate reconstruction and regression.

2.2.2 Approximating graph filters with Chebyshev polynomials

So far, we have presented graph filters as spectral operators which are constructed by applying a function to the eigenvalues of the graph Laplacian. However, this formulation is predicated on the notion that it is practical to compute and store the eigendecomposition of $\mathbf{L}$. Since the time-complexity of this operation scales as $O(N^3)$, this can quickly become infeasible for large-scale graphs with a high number of nodes. In addition, applying the matrix $\mathbf{H}$ to a graph signal has complexity $O(N^2)$, in general requiring information from all nodes to compute the filtered signal value at each node.

On the other hand, a typical feature of graphs is sparsity, since the degree of each node is often independent of the size of the graph. This means that multiplication of a graph signal $\mathbf{y}$ by the Laplacian can often be achieved with complexity $O(N|\mathcal{E}|) \ll O(N^2)$. Furthermore, in the context of a physically realised graph such as a sensor network, computation of $\mathbf{Ly}$ can be executed 'locally', in the sense that the result at any particular node depends only on the signal value at the direct neighbours of that node.

One popular and longstanding technique that takes advantage of both these ideas involves approximating a graph filter using Chebyshev polynomials [Shuman et al., 2018]. In general, if a graph filter can be approximated as a polynomial of the graph Laplacian, then we can take advantage of the sparsity of $\mathbf{L}$ to efficiently filter signals without performing eigendecomposition. Consider the following standard polynomial approximation to a graph filter.

$$g(\lambda) \approx \sum_{k=1}^{K} c_k \lambda^k \quad \Longleftrightarrow \quad \mathbf{H} \approx \sum_{k=1}^{K} c_k \mathbf{L}^k \tag{2.4}$$

The Chebyshev polynomial approximation is analogous, but rather than summing powers of $\mathbf{L}$ directly, we sum order-K polynomials, $\bar{T}_k(\mathbf{L})$.

$$g(\lambda) \approx \sum_{k=1}^{K} c_k \bar{T}_k(\lambda) \quad \Longleftrightarrow \quad \mathbf{H} \approx \sum_{k=1}^{K} c_k \bar{T}_k(\mathbf{L}) \tag{2.5}$$

Standard Chebyshev polynomials form an orthonormal basis with respect to the product

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) \frac{dx}{\sqrt{1-x^2}} \tag{2.6}$$

and can be used to approximate arbitrary functions within this domain. They also have a number well-documented advantages in terms of computational robustness and

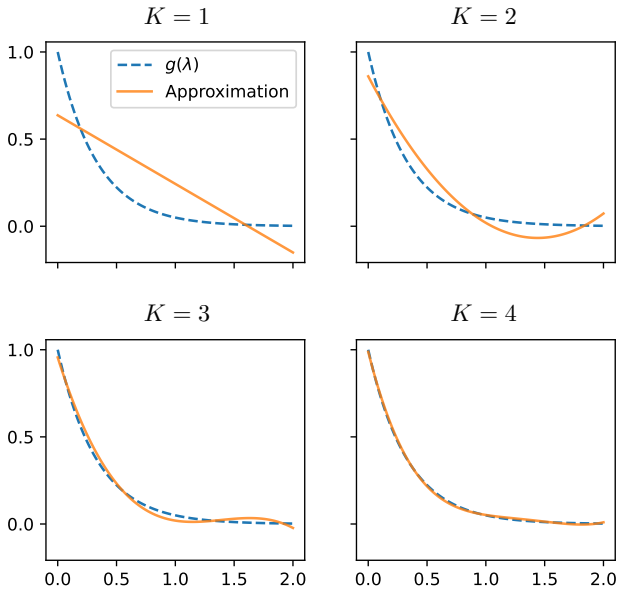


FIGURE 2.2: Successive approximations of the filter function $g(\lambda) = \exp(-3\lambda)$ using Chebyshev polynomials.

error over standard polynomial approximations that arise from a Taylor series expansion [Rivlin, 2020]. While the standard Chebyshev polynomials $\{T_k\}$ are valid on the domain $[-1, 1]$, in the context of graph filters, they must be applicable over the range $[0, \lambda_{\max}]$, leading to the transformed variant $\{\bar{T}_k\}$ appearing in eq. (2.5). These can be defined recursively as follows.

$$\bar{T}_k(\mathbf{L}) = \left(\frac{4\mathbf{L}}{\lambda_{\max}} - 2\right)\bar{T}_{k-1}(\mathbf{L}) - \bar{T}_{k-2}(\mathbf{L}) \tag{2.7}$$

subject to the initial conditions

$$\bar{T}_0(\mathbf{L}) = \mathbf{I}_N, \quad \bar{T}_1(\mathbf{L}) = \frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}_N \tag{2.8}$$

Note that this definition ensures only repeated multiplications by $\mathbf{L}$ when applying $\bar{T}_k(\mathbf{L})$ to a graph signal. To approximate an arbitrary filter function $g(\lambda)$, the coefficients c_k are given by

$$c_k = \begin{cases} \dfrac{1}{\pi}\displaystyle\int_0^\pi g\left(\frac{\lambda_{\max}}{2}(\cos(\theta)+1)\right) d\theta & \text{if} \quad k = 0, \\ \dfrac{2}{\pi}\displaystyle\int_0^\pi \cos(k\theta)\, g\left(\frac{\lambda_{\max}}{2}(\cos(\theta)+1)\right) d\theta & \text{otherwise.} \end{cases} \tag{2.9}$$

and can be quickly evaluated numerically.

Chebyshev polynomials appeared in early papers on GSP, such as [Hammond et al., 2011], in which the authors were concerned with defining wavelet transforms on graphs, also defined in terms of Laplacian-based spectral operators $g(\mathbf{L})$. Since then, they have been widely used in distributed applications [Shuman et al., 2018], and have also been extensively employed in Graph Neural Networks [Gama et al., 2020]. In Defferrard et al. [2017], Chebyshev polynomials form the basis of convolutional filters with learned parameters, in direct analogy with traditional Convolutional Neural Networks (CNNs). This approach was further simplified in Kipf and Welling [2017] by reducing the number of terms in the polynomial to mitigate overfitting.

In Rimleanscaia and Isufi [2020], Tseng [2020], the authors propose an extension to polynomial filters, which considers a more general class of rational filters, i.e. the ratio between two polynomials. As they describe, in certain circumstances, rational designs can approximate a given frequency response using lower-order polynomials. Notably, when the response is band-limited, rational filters were found to have significantly lower error than standard Chebyshev polynomials, which tend to struggle when approximating the sharp cut-off. In general, rational filtering requires solving a linear system $\mathbf{y}' = P(\mathbf{L})^{-1}Q(\mathbf{L})\mathbf{y}$, where P and Q are the polynomials occurring in the numerator and denominator, respectively. This incurs an additional computational cost. Furthermore, solving for the coefficients of each polynomial no longer has a closed-form solution and requires addressing a more challenging non-convex constrained problem.

As noted, polynomial approximations are primarily useful because multiplications of a graph signal by the Laplacian can be executed efficiently. In all the papers discussed so far, efficiency has been achieved through distributed execution and/or leveraging Laplacian sparsity. However, another way to exploit the Laplacian structure is if it has a representation in terms of a tensor factorisation. This is one of the approaches we take in the subsequent sections, focusing on how tensor factorisation can further optimise the computation time and resource usage, especially in cases with high-dimensional data or large-scale product graphs.

2.3 Graph Signal Reconstruction

Graph Signal Reconstruction (GSR), sometimes also known as graph signal interpolation, is a classic problem in GSP [Ortega et al., 2018]. The task can be stated as, given a partially observed graph signal, estimate the signal value at the unobserved vertices using only the topology of the graph. Closely related, is the topic of signal sampling, which seeks to find an informative set of vertices to measure a graph signal at, in order to perform an accurate reconstruction [Tanaka et al., 2020]. Typically, GSR is posed as

an optimisation problem where one seeks to identify a smooth underlying graph signal that simultaneously minimises a measure of error across the observed nodes, as well as a penalty term that enforces some kind of signal smoothness or bandlimited assumption.

Early work on GSR, such as Anis et al. [2016], Narang et al. [2013a,b], Wang et al. [2015a], mainly focused on the reconstruction of bandlimited graph signals. Here, the goal is to find the optimal reconstruction of a signal using only the first K eigenvectors of the graph Laplacian, with a frequency less than ω (the so-called Paley-Wiener space). In Narang et al. [2013a], this is achieved using an Iteratively Reweighted Least Squares (IRLS) algorithm, with a Chebyshev polynomial approximation to the bandlimited filter. Further work such as [Romero et al., 2017b] proposes using a more flexible specification of the reconstructed signal profile based on graph kernels.

Several methods have also been proposed for the reconstruction of time-varying graph signals. In Qiu et al. [2017], the penalty term in the optimisation objective promotes both smoothness across the graph using a Laplacian penalty, and consistency across time, by introducing a temporal difference operator $\mathbf{D}$. The problem is then solved for both the noiseless and noisy case using a gradient projection algorithm with back-tracking line search. A similar method is adopted in Giraldo et al. [2022], but with a regularised version of the Laplacian for improved computational robustness. In Ioannidis et al. [2016], this approach is modified to incorporate a more dynamic penalty that can adjust the strength of regularisation across the graph and across time independently, using so-termed "space-time kernels". Furthermore, in this paper, as well as Ioannidis et al. [2018], Romero et al. [2017a], the authors propose an online algorithm that can accommodate dynamic graphs (i.e. graphs for which the edge weights can change over time), although this is only possible for a special class of kernel which have a tri-diagonal inverse.

Recently, some methods have also been proposed for distributed reconstruction of time varying graph signals. In Chi et al. [2022], as well as [Zhou et al., 2022a], the authors propose distributed algorithms that draw upon the work of Qiu et al. [2017] using the temporal difference operator. In these two papers, which have similar goals but were developed separately, the authors use a modified Newton method to solve the optimisation problem, by making use of an approximation to the Hessian.

In this thesis, we take inspiration from space-time kernels developed in Ioannidis et al. [2016], as well as the graph spectral priors of Venkitaraman et al. [2020], to develop a flexible Bayesian formulation of the reconstruction problem. In addition, while the problem of time-varying graph signal reconstruction has been examined by several papers, in chapter 3 we propose reconstruction on two-dimensional Cartesian products graphs, of which time varying graph signals can be considered a special case. Furthermore, we

offer two alternative optimisation algorithms, one a Stationary Iterative Method (SIM) using matrix splitting, and the other based on the Conjugate Gradient Method (CGM) with a specialised graph-spectral preconditioner. By conducting a detailed asymptotic analysis, we compare and contrast the convergence behaviour of these two algorithms. Whilst convergence is discussed in many of the aforementioned papers, we provide fresh insight into the effect of the model hyperparameters, and present synthetic data studies to confirm our theoretical findings. Furthermore, we show how the SIM can be conducted in a distributed manner, and how the fraction of missing data in the original graph signal affects which of these two methods is preferable. In chapter 5, we extend our methods to encompass GSR on Cartesian products graphs with an arbitrary number of dimensions, under the multiway GSP paradigm. To the best of our knowledge, these all represent novel contributions to the field.

2.4 Graph Signal Regression

Another key topic that runs through this thesis is graph signal regression. One of the earliest papers to reference regression in the context of GSP is Guestrin et al. [2004]. Here, the authors propose a distributed algorithm for computing a function that approximates a time-varying graph signal, with applications aimed specifically at sensor networks. This paper, which predates much of the work on Laplacian-based GSP, is primarily focused on efficient algorithms for communication between sensors, and in modern parlance is more akin to graph signal compression, as the authors seek a low-dimensional representation of the original signal. In can also be considered a particular class of distributed spatial regression, as they consider the x-y coordinates of each sensor, rather than modelling the network in terms of nodes and edges.

In this thesis, the definition of graph signal regression we use differs somewhat from this formulation. For our purposes, we consider graph signal regression to be the task of estimating a graph signal as a function of additional explanatory variables. In particular, we highlight two different scenarios which we term exogenous (global) and endogenous (local) regression. For the exogenous case, we examine work on Kernel Graph Regression (KGR) [Venkitaraman et al., 2019] and related variants such as Gaussian Process on Graphs (GPoG) Venkitaraman et al. [2020]. For the endogenous case, we examine work on Regression with Network Cohesion (RNC) [Li et al., 2019].

2.4.1 Exogenous case: Kernel Graph Regression and Gaussian Processes on Graphs

The task of graph regression with endogenous explanatory variables can be stated as follows. Consider a series of T real-valued graph signals, each measured on a graph comprising N nodes, $\{\mathbf{y}_t \in \mathbb{R}^N\}_{t=1}^T$. Each of these signals has a corresponding associated length-M vector of explanatory variables $\{\mathbf{x}_t\}_{t=1}^T$. The goal is to learn a function that maps the explanatory variables to the graph signals, such that if a new explanatory vector $\mathbf{x}'$ is supplied, we should be able to make an estimate for the corresponding graph signal $\mathbf{y}'$. By penalising predictions that are not smooth with respect to the topology of the graph, the intention is to produce an estimator that outperforms non graph-aware alternatives. Note that, while we use the index t to denote each sample, the set $\{\mathbf{x}_t, \mathbf{y}_t\}$ can simply be regarded as training pairs and need not represent time-series data (although they often may).

In this context, the variables are 'exogenous' or global in the sense that they are not associated with any node in particular, and the signal at each node in the graph can learn a unique functional response. For example, consider a time-varying graph signal consisting of net profits for a network of businesses connected based on industry or supply chain. In this case, the explanatory variables could be factors in the broader economy such as GDP growth, inflation or unemployment which may affect each company differently.

In Venkitaraman et al. [2019], the authors propose Kernel Graph Regression (KGR) for this purpose. Here, each node in the observed graph signal $\mathbf{y}_t$ is modelled as a unique linear combination of a basis function representation of the explanatory variables $\boldsymbol{\phi}(\mathbf{x}_t) \in \mathbb{R}^P$, such that $\mathbf{y}_t = \mathbf{W}^\top \boldsymbol{\phi}(\mathbf{x}_t)$, where $\mathbf{W} \in \mathbb{R}^{P \times N}$ is a matrix of coefficients. To find the optimal value of $\mathbf{W}$, they propose the following cost function.

$$C(\mathbf{W}) = \sum_{t=1}^{T} ||\mathbf{y}_t - \mathbf{W}^\top \boldsymbol{\phi}(\mathbf{x}_t)||_2^2 + \alpha \text{tr}\left(\mathbf{W}^\top \mathbf{W}\right) + \beta \sum_{t=1}^{T} \text{TV}_2\big(\mathbf{W}^\top \boldsymbol{\phi}(\mathbf{x}_t)\big) \tag{2.10}$$

The first term in this objective seeks to minimise the square prediction error, the second provides regularisation for the coefficient matrix, and the third utilises the total square variation, of eq. (1.2), to promote signal smoothness. Next, the authors leverage the so-called 'kernel trick' to transform this to a non-parametric regression problem. This is achieved by introducing a $T \times T$ kernel matrix $\mathbf{K}$, where $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ [Rasmussen and Williams, 2005], which results in the following in-sample prediction.

$$\text{mat}\left(\big(\mathbf{I}_N \otimes \mathbf{K}\big)\big(\mathbf{I}_N \otimes (\mathbf{K} + \alpha \mathbf{I}_T) + \beta \mathbf{L} \otimes \mathbf{K}\big)^{-1} \text{vec}\,(\mathbf{Y})\right) \tag{2.11}$$

where $\mathbf{Y} \in \mathbb{R}^{T\times N}$ is the matrix formed by stacking each observed signals $\{\mathbf{y}_t\}$, $\text{vec}\,(\cdot)$ represents column-major vectorisation mapping $\mathbb{R}^{T\times N} \mapsto \mathbb{R}^{TN}$, and $\text{mat}\,(\cdot)$ is the reverse operation $\mathbb{R}^{TN} \mapsto \mathbb{R}^{T\times N}$. The result is the prediction made over the training samples $\{\mathbf{x}_t\}_{t=1}^{T}$. This can also be adjusted to make a prediction for an unseen explanatory variable $\mathbf{x}'$. Some recent extensions have been proposed to KGR such as Elias et al. [2022], where the authors use an algorithm based on random Fourier features to improve computational robustness.

In Venkitaraman et al. [2020], the same authors propose a new model which takes a Bayesian perspective to define Gaussian Processes over Graphs (GPG). Here, a modified approach is used, where a graph-spectral prior is placed over the coefficient matrix to induce smoothness in the output signal. This results in not only a point estimate for the signal, but a multivariate Gaussian probability distribution with a mean and covariance. In particular, the authors define an online algorithm where the distribution over a signal at time $T+1$, given $\mathbf{x}_{T+1}$ is given by

$$\mathcal{N}\left(\boldsymbol{\mu} = \mathbf{D}^\top \mathbf{C}_T^{-1}\,\text{vec}\,(\mathbf{Y}),\ \ \boldsymbol{\Sigma} = \mathbf{F} - \mathbf{D}^\top \mathbf{C}_T^{-1}\mathbf{D}\right) \tag{2.12}$$

with

$$\mathbf{C}_T = \mathbf{H}^2 \otimes \mathbf{K} + \beta^{-1}\mathbf{I}_N \otimes \mathbf{I}_T \tag{2.13}$$

$$\mathbf{D} = \mathbf{H}^2 \otimes \left[\kappa(\mathbf{x}_1, \mathbf{x}_{T+1}),\ \kappa(\mathbf{x}_2, \mathbf{x}_{T+1}),\ \ldots,\ \kappa(\mathbf{x}_N, \mathbf{x}_{T+1})\right]^\top \tag{2.14}$$

$$\mathbf{F} = \kappa(\mathbf{x}_{T+1}, \mathbf{x}_{T+1})\mathbf{H}^2 + \beta^{-1}\mathbf{I}_N \tag{2.15}$$

Several papers have also proposed extensions to this model. For example, in Miao et al. [2022], the authors present a modified algorithm that simultaneously learns the underlying graph structure. In Zhi et al. [2023], the authors propose parametrising the filter function as a finite polynomial of the graph Laplacian, and simultaneously learn the polynomial coefficients, extending their earlier work in Pu et al. [2021].

In chapter 4, we present a hybrid model that incorporates aspects of graph signal reconstruction, as well as Gaussian processes over graphs. In particular, we are interested in reconstructing time-varying partially observed graph signals when additional exogenous explanatory variables exist. Notably, all the models discussed in this subsection assume that readings across all vertices are available as inputs into the model. However, in many real-world applications, node-level data may be corrupted by equipment failure or difficult/expensive to reliably collect. Whilst one option may be to simply remove

problematic nodes from the problem entirely, this can waste valuable data that may help in the estimation problem. In chapter 5, we take this further by developing a model that is capable of reconstructing d-dimensional multiway graph signals with exogenous variables. To the best of our knowledge, none of these issues have been addressed in the available literature.

2.4.2 Endogenous case: Regression with Network Cohesion

The second graph regression setting we wish to highlight is the case when endogenous variables exist to aid with the estimation process. Consider a single graph signal $\mathbf{y} \in \mathbb{R}^N$, with elements existing on the nodes of a known graph. In this scenario, each node, n, has an associated vector of explanatory variables $\mathbf{x}_n \in \mathbb{R}^M$. The goal is to build a predictive model which utilised these local explanatory variables, whilst also incorporating the underlying structure of the network.

In Li et al. [2019], the authors propose Regression with Network Cohesion (RNC). Here, they propose modelling each element of the observed graph signal $\mathbf{y}_n$ as a linear combination of an intercept term and the explanatory variables at node n. This is summarised by the following statistical model.

$$\mathbf{y} = \mathbf{c} + \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon} \tag{2.16}$$

$\mathbf{X} \in \mathbb{R}^{N \times M}$ is the matrix formed by stacking each vector of explanatory variables, $\mathbf{w} \in \mathbb{R}^M$ is a coefficient vector, and $\mathbf{c} \in \mathbb{R}^N$ is a flexible intercept term of individual node effects. The model error $\boldsymbol{\epsilon} \in \mathbb{R}^N$ is assumed to have a zero mean and a variance of $\sigma^2\mathbf{I}_N$. The model seeks to learn an optimal value for $\mathbf{c}$ and $\mathbf{w}$ by minimising the following objective function.

$$C(\mathbf{c}, \mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w} - \mathbf{c}||_2^2 + \lambda \mathrm{TV}_2(\mathbf{c}) \tag{2.17}$$

The addition of the total square variation term, which the authors name the cohesion penalty, is intended to promote smoothness over the flexible intercept with respect to the topology of the graph. The justification for this is that closely connected nodes are likely to have similar baseline effects, meaning the model simultaneously learns a shared coefficient vector $\mathbf{w}$, and a unique but smooth intercept. The optimal values can be computed by introducing a combined parameter vector $\boldsymbol{\theta} = \begin{bmatrix} \mathbf{c} \\ \mathbf{w} \end{bmatrix}$, with an estimator given by

$$\hat{\boldsymbol{\theta}} = \begin{bmatrix} \mathbf{I}_N + \lambda \mathbf{L} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{X}^\top \mathbf{X} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I}_N & \mathbf{X} \end{bmatrix} \mathbf{y} \tag{2.18}$$

In order to make a prediction on a test set, where the new explanatory variables $\mathbf{X}' \in \mathbb{R}^{N' \times M}$ are available, the authors use the following.

$$\hat{\mathbf{y}}' = \hat{\mathbf{c}}' + \mathbf{X}' \hat{\mathbf{w}} \tag{2.19}$$

where $\hat{\mathbf{w}}$ is the optimal value found by minimising eq. (2.17), and $\hat{\mathbf{c}}'$ is found by minimising the following objective.

$$\hat{\mathbf{c}}' = \underset{\mathbf{c}'}{\operatorname{argmin}} \quad \begin{bmatrix} \hat{\mathbf{c}} \\ \mathbf{c}' \end{bmatrix}^\top \begin{bmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ \mathbf{L}_{12}^\top & \mathbf{L}_{22} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{c}} \\ \mathbf{c}' \end{bmatrix} = -\mathbf{L}_{22}^{-1} \mathbf{L}_{12}^\top \hat{\mathbf{c}} \tag{2.20}$$

where the new Laplacian characterising the full graph containing all train and test nodes has been broken into blocks as shown. Note that as long as the two parts of the graph are mutually connected, the inverse of $\mathbf{L}_{22}$ will exist. As noted by the authors, there may be some issues surrounding computational robustness within this model. In particular, the system matrix found in eq. (2.18) may in general be singular. To overcome this the authors propose adding a small term $\gamma \mathbf{I}_N$ to the graph Laplacian, which fixes this issue.

One limitation of the RNC model is that the total variation penalty, while simple and easy to interpret, lacks the flexibility to specify more generic expectations about the spectral profile of the observed graph signal.

Le and Li [2022]

2.5 Multiway Graph Signal Processing

Multiway Graph Signal Processing (MWGSP) is an emerging subfield that extends Graph Signal Processing to analyse multiway data represented by multidimensional arrays and tensors [Stanley et al., 2020]. In most GSP applications, a graph signal is represented by a vector $\mathbf{y} \in \mathbb{R}^N$, which is interpreted as existing on the nodes of a single graph. MWGSP extends this by considering tensor-valued signals $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$, where each axis is described by an independent graph. As a whole, the tensor can be understood as existing on the nodes of a single product graph.

By incorporating underlying graphs in each data axis, MWGSP handles higher dimensional data sets, which are typically unaddressed by classical GSP. This new approach offers promising applications in real-world situations, such as climate modelling, epidemic spread, and complex biological systems.

Zhao et al. [2023]

Li et al. [2012]

2.6 Graph signal classification

Tran et al. [2020]

Sandryhaila and Moura [2013a]

[Ahmed et al., 2017]

No one has done it on multiway graphs. Limited regression models. No one using graph filters.

in ML: [Belkin and Niyogi, 2002]

2.7 Related topics

For the sake of clarity, in this section we give a high-level overview of some important sub-fields of GSP which are adjacent to the work in this thesis, but not directly relevant to the core subject matter. While not addressed directly in this thesis, some of these topics offer potentially useful avenues for extension of our work.

2.7.1 Graph learning

When GSP was originally formulated, the graph of interest was typically assumed to be known a priori, with the subsequent methods following from the existence of this basic structure. In real-world applications, this can often be the case, for example, in a social, transport or citation network it is clear how objects should be connected given the nature of the problem. In other applications, for example in a protein-protein interactome, domain specific knowledge can be incorporated in a principled way to construct a graph [Li et al., 2023]. In others, such as a sensor network, k-nearest neighbour, ε-ball, or permuted minimum spanning tree techniques can be used to construct a reasonable graph [Qiao et al., 2018].

However, in other circumstances, it may not be so clear how to derive or construct a graph, and instead one must be learned from the data. An example of this may be the stock price returns of a network of companies. While it may be possible to construct a graph using information such as industry, supply chain, or strategic alliance [Cheng and Li, 2021, Gao et al., 2021], this information may be difficult to gather or otherwise unavailable. Another approach would be simply to learn a graph directly from the returns data. In general, the graph learning problem can be formulated as, given a set of graph signals $\{\mathbf{y}_t \in \mathbb{R}^N\}_{t=1}^T$, find an appropriate sparse $N \times N$ adjacency matrix [Dong et al., 2019]. From the perspective of GSP, it is assumed that these observations are generated from some network process, operating on a latent underlying graph structure.

One of the oldest and simplest approaches, predating work on GSP, is so-called correlation networks. Here, a graph can be constructed by considering the pairwise correlation between the signal at nodes i and j. For example, one approach is to set $\mathbf{A}_{ij} = 1$ if $\rho_{ij} > 0$ and zero otherwise [Mateos et al., 2019]. While such approaches are simple and have an intuitive notion of pairwise interaction, correlations may be due to latent network effects rather than from direct pairwise influence. For example, two nodes may be interacting via a third node k, in which case it is more prudent to set $\mathbf{A}_{ik} = \mathbf{A}_{jk} = 1$, and $\mathbf{A}_{ij} = 0$. While there are techniques to overcome such confounding of variables (see, Kolaczyk [2009], section 7), in general this can be problematic from the perspective of graph learning.

An alternative prominent technique is the Graphical Lasso (GL) [Friedman et al., 2007]. The GL is a sparse penalised maximum likelihood estimator for the precision matrix $\mathbf{P}$ (the inverse of the covariance matrix $\mathbf{\Sigma}$) of a multivariate Gaussian random variable. The estimated value of $\mathbf{P}$ is given by

$$\hat{\mathbf{P}} = \underset{\mathbf{P} \succcurlyeq 0}{\operatorname{argmin}} \left[\operatorname{tr}\left(\mathbf{P}\hat{\mathbf{\Sigma}}\right) - \log\det\left(\mathbf{P}\right) + \lambda \sum_{i \neq j} |\mathbf{P}_{ij}| \right] \tag{2.21}$$

where $\hat{\mathbf{\Sigma}}$ is the sample covariance, with the L_1 norm promoting sparsity in $\mathbf{P}$. The graph Laplacian is then derived by assuming $\mathbf{P}$ is a regularised version of $\mathbf{L}$ [Lake and Tenenbaum, 2010].

Other approaches favour more GSP-oriented estimation models by incorporating signal smoothness assumptions. For example, in Hu et al. [2015], the authors propose learning thee Laplacian matrix directly by solving the following optimisation problem.

$$\begin{aligned} \hat{\mathbf{L}} &= \text{argmin} \left[\text{tr} \left(\mathbf{Y}^\top \mathbf{L} \mathbf{Y} \right) - \beta ||\mathbf{L}||_F^2 \right], \\ \text{s.t.} \quad & \text{tr}\left(\mathbf{L}\right) = N, \quad \mathbf{L}\mathbf{1} = \mathbf{0}, \quad \mathbf{L}_{ij} = \mathbf{L}_{ji} > 0 \end{aligned} \tag{2.22}$$

where $\mathbf{Y} \in \mathbb{R}^{N \times T}$ is the matrix obtained by stacking each observed graph signal $\mathbf{y}_t$. This was modified slightly in Dong et al. [2016] by introducing a new matrix $\mathbf{X}$, of the same dimensions as $\mathbf{Y}$, meant to be a smooth approximation of $\mathbf{Y}$. Their optimisation objective was then given by

$$\begin{aligned} \hat{\mathbf{L}}, \hat{\mathbf{X}} &= \text{argmin} \left[||\mathbf{X} - \mathbf{Y}||_F^2 + \alpha \text{tr} \left(\mathbf{X}^\top \mathbf{L} \mathbf{X} \right) - \beta ||\mathbf{L}||_F^2 \right], \\ \text{s.t.} \quad & \text{tr}\left(\mathbf{L}\right) = N, \quad \mathbf{L}\mathbf{1} = \mathbf{0}, \quad \mathbf{L}_{ij} = \mathbf{L}_{ji} > 0 \end{aligned} \tag{2.23}$$

A more computationally efficient approach was proposed in Kalofolias [2016], but using the adjacency matrix rather than the Laplacian. There has been a substantial amount of additional work on graph learning from a GSP perspective. For a detailed review, see Dong et al. [2019], Mateos et al. [2019].

2.7.2 GSP on directed graphs

The work presented in this thesis, and indeed the majority of published literature on GSP, is focused on undirected graphs. The benefit of this framework is that the undirected graph Laplacian naturally gives rise to a simple definition of signal smoothness. It is also a well-behaved Hermitian operator, with real eigenvalues and orthonormal eigenvectors that serve as a well-motivated and coherent Fourier basis [Ortega et al., 2018]. However, there is also a rich literature regarding GSP for directed graphs (or digraphs) [Marques et al., 2020]. Directed graphs, while less commonly found in GSP, are more suitable for modelling certain applications such as web links, citation networks, trade flows and follower-model social networks, since they can accommodate both incoming and outgoing edges. However, extending the Graph Fourier Transform framework to signals on digraphs less straightforward and a number of alternative proposals exist.

One originates from some of the earliest work on GSP from Sandryhaila and Moura, where the GFT for a general directed graph is constructed from the total variation of a graph signal $\mathbf{y}$ defined as follows.

$$\text{TV}_1(\mathbf{y}) = ||\mathbf{y} - \mathbf{A}^{\text{norm}} \mathbf{y}||_1 \tag{2.24}$$

where $\mathbf{A}^{\text{norm}}$ is the adjacency matrix (or, more generally, any graph shift operator) divided by its spectral radius to ensure that the transformed signal is appropriately scaled for comparison with the original signal [Sandryhaila and Moura, 2013c]. A Fourier basis $\{\mathbf{u}_i\}$ is then defined by first taking an eigendecomposition of the adjacency matrix, and then ordering the eigenvectors according to their total variation. Where the adjacency matrix is non-diagonalisable, one can instead resort to the generalised eigenvectors using Jordan decomposition.

While this definition is attractive from a theoretical standpoint, it suffers from several drawbacks. In general, the eigenvectors are complex-valued and may be more difficult to interpret, for example, constant signals no longer have zero variation. Furthermore, the eigenvectors are typically non-orthonormal, meaning Parseval's identity no longer holds and signal power is not preserved when transforming between the vertex and spectral domains. Other work has attempted to remedy this by proposing a new definition for variation on directed graphs. In Sardellitti et al. [2017], the authors define the Graph Directed Variation (GDV) as follows.

$$\text{GDV}(\mathbf{y}) = \sum_{i,j}^{N} \mathbf{A}_{ij}[\mathbf{y}_i - \mathbf{y}_j]_+ \tag{2.25}$$

where $[\mathbf{y}_i - \mathbf{y}_j]_+ = \max(\mathbf{y}_i - \mathbf{y}_j, 0)$. The search for the GFT basis is then given by the optimisation problem of finding the set of orthonormal vectors that subsequently minimise the GDV subject to perpendicularity with all those preceding. Under this framework, smooth signals on digraphs represent a flow from a smaller value to a larger one over a directed edge. Hence, the signal at two nodes i and j is only penalised if $\mathbf{y}_i > \mathbf{y}_j$. For example, if there were a set of temperature sensors over a mountain range, a directed graph would connect stations based on elevation, since readings at higher altitudes are expected to be lower.

A similar strategy was used in Shafipour et al. [2019], where here the variation was defined as follows.

$$\text{GDV}_2(\mathbf{y}) = \sum_{i,j}^{N} \mathbf{A}_{ij}[\mathbf{y}_i - \mathbf{y}_j]_+^2 \tag{2.26}$$

In order to generate a Fourier basis that is roughly evenly spread with respect to the variation measure, these authors proposed solving the following optimisation objective.

$$\begin{aligned} \mathbf{U}^\star = \operatorname{argmin} \sum_{i}^{N-1} \Big(\mathrm{GDV}_2(\mathbf{u}_{i+1}) - \mathrm{GDV}_2(\mathbf{u}_i) \Big)^2 \\ \text{s.t.} \quad \mathbf{U}^\top \mathbf{U} = \mathbf{I}, \quad \mathbf{u}_1 \propto \mathbf{1}, \quad \mathbf{u}_N = \operatorname*{argmin}_{|\mathbf{u}|=1} \mathrm{GDV}_2(\mathbf{u}) \end{aligned} \tag{2.27}$$

These approaches, whilst offering a simple and meaningful definition of directed variation are potentially limited in their applicability, since the underlying assumption that a graph signal should flow from lower to higher values over directed edges may not be suitable for all scenarios. It also implies that any signal that monotonically increases over the direction of the network has the same variation, namely zero. Furthermore, there is a significant additional computation cost to solving these optimisation problems that makes scaling to large graphs difficult.

One other interesting and novel proposal for the GFT of a directed graph is to use the so-termed "magnetic Laplacian" [DeResende and Costa, 2020, Zhang et al., 2021]. The magnetic Laplacian is a complex-valued Hermitian matrix which generalises the standard undirected Laplacian for directed graphs and is defined as follows.

$$\mathbf{L}_M = \mathbf{D} - \mathbf{\Gamma} \circ \mathbf{A}_S \tag{2.28}$$

Here, $\mathbf{D}$ is the diagonal degree matrix defined as $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, $\mathbf{A}_S$ is the symmetric adjacency matrix, defined as $\mathbf{A}_S = \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$, and $\mathbf{\Gamma}$ is a Hermitian matrix given by

$$\mathbf{\Gamma} = \exp\left(2\pi \mathrm{i} q \left(\mathbf{A} - \mathbf{A}^\top \right) \right) \tag{2.29}$$

Here, i is the imaginary unit and q is a parameter known as the *charge*. The exponentiation is element-wise. Each element $\mathbf{\Gamma}_{ij}$ represents a complex phase, where the angle is proportional to the net outflow between nodes i and j. The effect is that $\mathbf{L}_M$ is a Hermitian matrix that captures both the magnitude and the direction of the edges at each node. This operator arises in quantum mechanics to describe the Hamiltonian of a charged particle on a graph subject to the action of a magnetic field [Shubin, 1994]. Note that, for an undirected graph, the magnetic Laplacian reduces to the standard Laplacian.

As a complex Hermitian operator, $\mathbf{L}_M$ has eigenvectors given by unitary matrix $\mathbf{U}$, and real-valued eigenvalues.

$$\mathbf{L}_M = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^\dagger \tag{2.30}$$

where $\mathbf{U}^\dagger$ is the Hermitian adjoint of $\mathbf{U}$. Therefore, the GFT of a signal $\mathbf{y}$ is given by $\mathbf{U}^\dagger\mathbf{y}$ and the IGFT by $\mathbf{U}\mathbf{y}$. This elegant formulation maintains many of the desirable properties of the undirected GFT, such as power preservation. Early results indicate that the magnetic Laplacian may be effective in tasks such as community detection and denoising on directed graphs [Fanuel et al., 2017, Furutani et al., 2020]. However, some uncertainty remains on how to interpret the complex-valued signals after transformations via the GFT and IGFT. In Furutani et al. [2020], the authors opt to take the real part of the signal only, however best practices have not yet been established.

In general, spectral methods on directed graphs present a number of additional challenges and there is no clear front-runner for how to handle them. For this reason, as well as the additional computational challenges associated with many of the methods, the authors of Marques et al. [2020] suggest that node-domain algorithms may generally be preferable when it comes to directed graphs.

2.7.3 Graph neural networks

Another are that is strongly related to GSP, although has arguably evolved into a field in its own right, is graph neural networks.

2.8 Summary of contributions

Chapter 3

Signal Reconstruction on Cartesian Product Graphs

In this chapter, we are concerned with the reconstruction of signals defined over the nodes of a Cartesian product graph. In particular, we are pose the reconstruction problem in terms of Bayesian inference, where a smooth underlying signal must be inferred given a noisy partial observation. The key goal of this chapter is formulate a Bayesian model for this purpose, and to examine scalable techniques for obtaining the posterior mean.

Whilst the topic of Graph Signal Reconstruction (GSR) on one-dimensional graphs has been studied fairly extensively, work on time-varying graph signals and general Cartesian product graphs is less complete. Furthermore, the problem is usually not framed in Bayesian terms and a thorough complexity analysis is often missing, which is particularly important in the context of product graphs, where the number of nodes under consideration can grow rapidly. In this chapter, we devote substantial attention to the computational complexity of our GSR model. Since our solution involves iterative methods, we center much of the discussion around how to reduce the number of iterations required for convergence and how each iteration can be completed in a maximally efficient fashion.

We begin in section 3.1 by revisiting the concept of a graph product, and explaining our rationale for focusing on the Cartesian product in particular. Moreover, we provide a review of how concepts from conventional one-dimensional graph signal processing, such as the Graph Fourier Transform (GFT) and spectral filtering, can be extended to encompass the two-dimensional case. In section 3.2, we present our statistical model for graph signal reconstruction on a Cartesian product graph, and derive two alternative algorithms for solving the posterior mean. These encompass a Stationary Iterative

Method (SIM) and a Conjugate Gradient Method (CGM). In both instances, we demonstrate how graph-spectral considerations can be employed to improve their convergence rate, and utilise the properties of the Kronecker product to facilitate the efficient execution of each iteration. Finally, in section 3.3, we engage in an in-depth analysis of the convergence properties of each method, and deduce how the rate of convergence is influenced by the hyperparameters. Specifically, we establish how the optimal selection of a method is contingent upon the value of these hyperparameters and propose practical guidelines for method selection.

3.1 Graph Products

In this chapter, we will be primarily concerned with signal processing on *Cartesian product graphs*. This special class of graph finds applications in numerous areas, such as video, hyper-spectral image processing and network time series problems. However, the Cartesian product is not the only way to consistently define a product between two graphs. In this section we formally introduce the concept of a graph product, examine some prominent examples, and explain why we choose to look specifically at the Cartesian graph product.

3.1.1 Basic definitions

In the general case, consider two undirected graphs $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)$ and $\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B)$ with vertex sets given by $\mathcal{V}_A = \{a \in \mathbb{N} \,|\, a \leq A\}$ and $\mathcal{V}_B = \{b \in \mathbb{N} \,|\, b \leq B\}$ respectively. (In this context we do not regard zero to be an element of the natural numbers). A new graph $\mathcal{G}$ can be constructed by taking the product between $\mathcal{G}_A$ and $\mathcal{G}_B$. This can be generically written as follows.

$$\mathcal{G} = \mathcal{G}_A \diamond \mathcal{G}_B = (\mathcal{V}, \mathcal{E}) \tag{3.1}$$

For all definitions of a graph product, the new vertex set $\mathcal{V}$ is given by the Cartesian product of the vertex sets of the factor graphs, that is

$$\mathcal{V} = \mathcal{V}_A \times \mathcal{V}_B = \{(a, b) \in \mathbb{N}^2 \,|\, a \leq A \text{ and } b \leq B\} \tag{3.2}$$

Typically, vertices are are arranged in lexicographic order, in the sense that $(a, b) \leq (a', b')$ iff $a < a'$ or $(a = a'$ and $b \leq b')$ [Harzheim, 2005]. Each consistent rule for

constructing the new edge set $\mathcal{E}$ corresponds to a different definition of a graph product. In general, there are eight possible conditions for deciding whether two nodes (a, b) and (a', b') are to be connected in the new graph.

$$
\begin{array}{lccc}
1. & [a, a'] \in \mathcal{E}_A & \text{and} & b = b' \\
2. & [a, a'] \notin \mathcal{E}_A & \text{and} & b = b' \\
3. & [a, a'] \in \mathcal{E}_A & \text{and} & [b, b'] \in \mathcal{E}_B \\
4. & [a, a'] \notin \mathcal{E}_A & \text{and} & [b, b'] \in \mathcal{E}_B \\
5. & [a, a'] \in \mathcal{E}_A & \text{and} & [b, b'] \notin \mathcal{E}_B \\
6. & [a, a'] \notin \mathcal{E}_A & \text{and} & [b, b'] \notin \mathcal{E}_B \\
7. & a = a' & \text{and} & [b, b'] \in \mathcal{E}_B, \\
8. & a = a' & \text{and} & [b, b'] \notin \mathcal{E}_B
\end{array}
$$

Each definition of a graph product corresponds to the union of a specific subset of these conditions, thus, there exist 256 different types of graph product [Barik et al., 2015]. Of these, the Cartesian product (conditions 1 or 7), the direct product (condition 3), the strong product (conditions 1, 3 or 7) and the lexicographic product (conditions 1, 3, 5 or 7) are referred to as the standard products and are well-studied [Imrich and Klavžar, 2000]. A graphical depiction of the standard graph products is shown in figure 3.1. In each of these four cases, the adjacency and Laplacian matrices of the product graph can be described in terms of matrices relating to the factor graphs [Barik et al., 2018, Fiedler, 1973]. This is shown in table 3.1.

Given these definitions, it may seem that all the standard graph products are non-commutative in the sense that $\mathbf{A}_A \oplus \mathbf{A}_B \neq \mathbf{A}_B \oplus \mathbf{A}_A$ etc. However, the graphs $\mathcal{G}_A \diamond \mathcal{G}_B$ and $\mathcal{G}_B \diamond \mathcal{G}_A$ are in fact isomorphically identical in the case of the Cartesian, direct and

	Adjacency matrix	Laplacian
Cartesian	$\mathbf{A}_A \oplus \mathbf{A}_B$	$\mathbf{L}_A \oplus \mathbf{L}_B$
Direct	$\mathbf{A}_A \otimes \mathbf{A}_B$	$\mathbf{D}_A \otimes \mathbf{L}_B + \mathbf{L}_A \otimes \mathbf{D}_B - \mathbf{L}_A \otimes \mathbf{L}_B$
Strong	$\mathbf{A}_A \otimes \mathbf{A}_B + \mathbf{A}_A \oplus \mathbf{A}_B$	$\mathbf{D}_A \otimes \mathbf{L}_B + \mathbf{L}_A \otimes \mathbf{D}_B - \mathbf{L}_A \otimes \mathbf{L}_B + \mathbf{L}_A \oplus \mathbf{L}_B$
Lexicographic	$\mathbf{I}_A \otimes \mathbf{A}_B + \mathbf{A}_A \otimes \mathbf{O}_A$	$\mathbf{I}_A \otimes \mathbf{L}_B + \mathbf{L}_A \otimes \mathbf{O}_B + \mathbf{D}_A \otimes (\lvert\mathcal{V}_B\rvert \mathbf{I}_B - \mathbf{O}_B)$

TABLE 3.1: The adjacency and Laplacian matrices for the standard graph products. Here, $\mathbf{D}_A$ and $\mathbf{D}_B$ are the diagonal degree matrices, i.e $\mathbf{D}_A = \text{diag}\,(\mathbf{A}_A \mathbf{1})$. $\mathbf{I}_A$ and $\mathbf{O}_A$ are the $(A \times A)$ identity matrix and matrix of ones respectively.

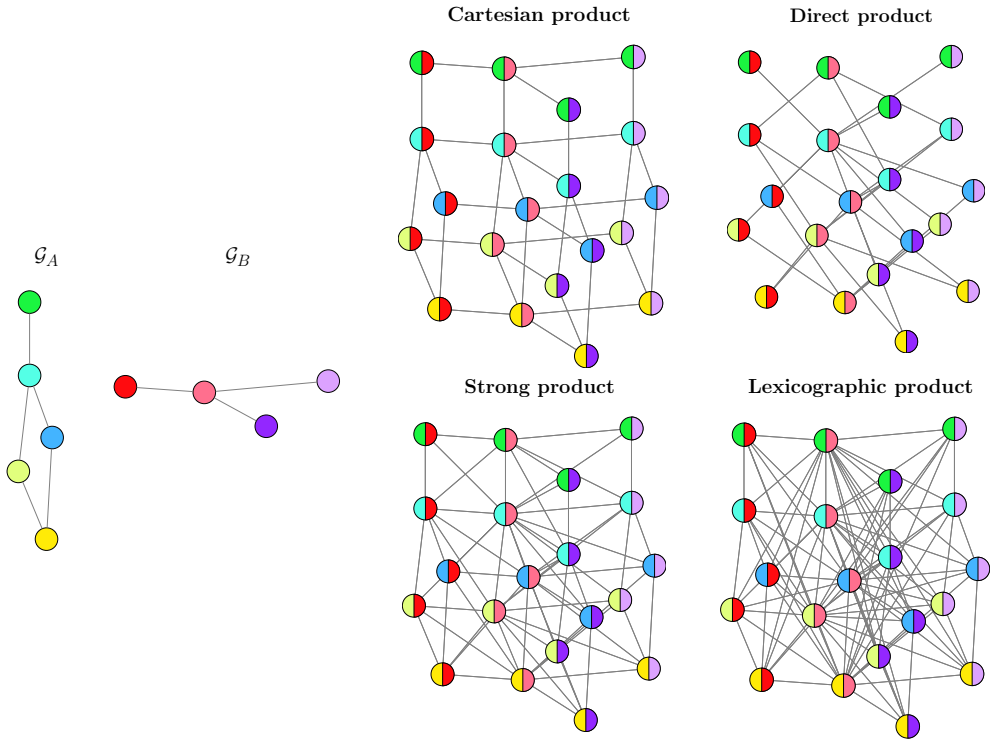


FIGURE 3.1: A graphical depiction of the four standard graph products

strong products. This is not the case for the Lexicographic product [Imrich and Klavžar, 2000].

3.1.2 The spectral properties of graph products

In the field of graph signal processing, we are often concerned with analysing the properties of graphs via eigendecomposition of the graph Laplacian [Mieghem, 2010]. In the case of product graphs, it is greatly preferable if we are able to fully describe the spectrum of $\mathcal{G}_A \diamond \mathcal{G}_B$ in terms of the spectra of $\mathcal{G}_A$ and $\mathcal{G}_B$ alone. This is because direct decomposition of a dense $\mathbf{L}$ has time-complexity $O(A^3B^3)$, whereas decomposition of the factor Laplacians individually has complexity $O(A^3 + B^3)$. As the graphs under considerations become medium to large, this fact quickly makes direct decomposition of the product graph Laplacian intractable. However, in the general case, only the spectra of the Cartesian and lexicographic graph products can be described in this way [Barik et al., 2018]. In the case of the direct and strong product, it is possible to estimate the spectra without performing the full decomposition (see [Sayama, 2016]). However, in general, the full eigendecomposition of the product graph Laplacian can only be described in terms of the factor eigendecompositions when both factor graphs are regular.

Consider the eigendecompositions of $\mathbf{L}_A$ and $\mathbf{L}_B$.

	Eigenvalues	Eigenvectors
Cartesian	$\lambda_a^{(A)} + \lambda_b^{(B)}$	$(\mathbf{U}_A)_a \otimes (\mathbf{U}_B)_b$
Direct*	$r_A\lambda_b^{(B)} + r_B\lambda_a^{(A)} - \lambda_a^{(A)}\lambda_b^{(B)}$	$(\mathbf{U}_A)_a \otimes (\mathbf{U}_B)_b$
Strong*	$(1+r_A)\lambda_b^{(B)} + (1+r_B)\lambda_a^{(A)} - \lambda_a^{(A)}\lambda_b^{(B)}$	$(\mathbf{U}_A)_a \otimes (\mathbf{U}_B)_b$
Lexicographic†	$B\lambda_a^{(A)}$	$(\mathbf{U}_A)_a \otimes \mathbf{1}_B$
	$\lambda_b^{(B)} + B\deg(a)$	$\mathbf{e}_a \otimes (\mathbf{U}_B)_b$

TABLE 3.2: Eigendecomposition of the Laplacian of the standard graph products. Here, a and b are understood to run from 1 to A and 1 to B respectively. $\star$ only for r_A and r_B-regular factor graphs. $\dagger$ note that the b runs from 2 to B in the lower row.

$$\mathbf{L}_A = \mathbf{U}_A\mathbf{\Lambda}_A\mathbf{U}_A^\top, \quad \text{and} \quad \mathbf{L}_B = \mathbf{U}_B\mathbf{\Lambda}_B\mathbf{U}_B^\top \tag{3.3}$$

where $\mathbf{U}_A$ and $\mathbf{U}_B$ are the respective orthonormal eigenvector matrices, and $\mathbf{\Lambda}_A$ and $\mathbf{\Lambda}_B$ are the diagonal eigenvalue matrices given by

$$\mathbf{\Lambda}_A = \operatorname{diag}\left(\begin{bmatrix}\lambda_1^{(A)} & \lambda_2^{(A)} & \dots & \lambda_A^{(A)}\end{bmatrix}\right), \quad \text{and} \quad \mathbf{\Lambda}_B = \operatorname{diag}\left(\begin{bmatrix}\lambda_1^{(B)} & \lambda_2^{(B)} & \dots & \lambda_B^{(B)}\end{bmatrix}\right)$$

Given these definitions, table 3.2 gives information about the spectral decomposition of the standard graph products.

3.1.3 GSP with Cartesian product graphs

While both the direct and strong products do find uses in certain applications (for example, see [Kaveh and Alinejad, 2011]), they are both less common and more challenging to work with in a graph signal processing context due to their spectral properties described in the previous subsection. In practice, being limited to regular factor graphs means the majority of practical GSP applications are ruled out. The lexicographic product does not share this drawback, however it is also significantly less common than the Cartesian product in real-world applications. For this reason, in the following, we focus primarily on the Cartesian product.

Given the spectral decomposition of the Cartesian graph product stated in table 3.2, we can write the Laplacian eigendecomposition in matrix form as follows.

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \quad \text{where} \quad \mathbf{U} = \mathbf{U}_A \otimes \mathbf{U}_B \quad \text{and} \quad \mathbf{\Lambda} = \mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B \tag{3.4}$$

This motivates the following definitions for the Graph Fourier Transform (GFT) and its inverse (IGFT). Consider a signal defined over the nodes of a Cartesian product graph expressed as a matrix $\mathbf{Y} \in \mathbb{R}^{B \times A}$. We can perform the GFT as follows.

$$\text{GFT}(\mathbf{Y}) = \text{mat}\left(\left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \text{vec}\left(\mathbf{Y}\right)\right) = \mathbf{U}_B^\top \mathbf{Y} \mathbf{U}_A \tag{3.5}$$

Correspondingly, we can define the IGFT acting on a matrix of spectral components $\mathbf{Z} \in \mathbb{R}^{B \times A}$ as follows.

$$\text{IGFT}(\mathbf{Z}) = \text{mat}\left(\left(\mathbf{U}_A \otimes \mathbf{U}_B\right) \text{vec}\left(\mathbf{Z}\right)\right) = \mathbf{U}_B \mathbf{Z} \mathbf{U}_A^\top \tag{3.6}$$

Product graph signals: repseprentation and vectorisation

It is natural to assume that signals defined on the nodes of a Cartesian product graph $\mathcal{G}_A \,\square\, \mathcal{G}_B$ could be represented by matrices (order two tensors) of shape $(A \times B)$. Since product graph operators, such as the Laplacian $\mathbf{L}_A \oplus \mathbf{L}_B$, act on vectors of length AB, we must define a consistent function to map matrix graph signals $\in \mathbb{R}^{A \times B}$ to vector graph signals $\in \mathbb{R}^{AB}$. The standard mathematical operator for this purpose is the $\text{vec}\,(\cdot)$ function, along with its reverse operator $\text{mat}\,(\cdot)$. However, this is somewhat problematic since $\text{vec}\,(\cdot)$ is defined to act in *column-major* order, that is

$$\text{vec}\left(\begin{bmatrix} \mathbf{Y}_{(1,1)} & \mathbf{Y}_{(1,2)} & \cdots & \mathbf{Y}_{(1,B)} \\ \mathbf{Y}_{(2,1)} & \mathbf{Y}_{(2,2)} & \cdots & \mathbf{Y}_{(2,B)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Y}_{(A,1)} & \mathbf{Y}_{(A,2)} & \cdots & \mathbf{Y}_{(A,B)} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{Y}_{(1,1)} \\ \mathbf{Y}_{(2,1)} \\ \vdots \\ \mathbf{Y}_{(A-1,B)} \\ \mathbf{Y}_{(A,B)} \end{bmatrix}$$

As is visible, this does not result in a lexicographic ordering of the matrix elements when the graph signal has shape $(A \times B)$. Therefore, to avoid this issue and to be consistent with standard mathematical notation, we will assume that graph signals are represented by matrices of shape $(B \times A)$ when considering the product between two graphs $\mathcal{G}_A \,\square\, \mathcal{G}_B$. For graph signals of this shape, the first index represents traversal of the nodes in $\mathcal{G}_B$, and the second index represents traversal of the nodes in $\mathcal{G}_A$. This ensures that matrix elements are correctly mapped to vector elements when using the column-major $\text{vec}\,(\cdot)$ function.

Given these definitions, we can define a spectral operator (usually a low-pass filter) $\mathbf{H}$ which acts on graph signals according to a spectral scaling function $g(\lambda\,;\,\beta)$ such as one of those defined in table **??**. As with regular non-product graphs, the action of this operator can be understood as first transforming a signal into the Laplacian frequency domain via the GFT, then scaling the spectral components according to some function, and finally transforming back into the vertex domain via the IGFT.

$$\begin{aligned}\mathbf{H} &= g(\mathbf{L}_A \oplus \mathbf{L}_B) \\ &= \left(\mathbf{U}_A \otimes \mathbf{U}_B\right) g\left(\mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B\right) \left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \\ &= \left(\mathbf{U}_A \otimes \mathbf{U}_B\right) \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right) \left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \\ &= \mathbf{U}\mathbf{D}_{\mathbf{G}}\mathbf{U}^\top \end{aligned} \tag{3.7}$$

The matrix $\mathbf{G} \in \mathbb{R}^{B\times A}$, which we refer to as the spectral scaling matrix, holds the value of the scaling function applied to the sum of pairs of eigenvalues, such that

$$\mathbf{G}_{ba} = g\left(\lambda_a^{(A)} + \lambda_b^{(B)}; \beta\right) \tag{3.8}$$

We observe that defining the filtering operation in this manner implies that the intensity is equal across both $\mathcal{G}_A$ and $\mathcal{G}_B$. We refer to filters of this type as *isotropic*. This can be further generalised by considering an *anisotropic* graph filter, which offers independent control over the filter intensity in each of the two dimensions. In this case, we define $\mathbf{G}$ as follows.

$$\mathbf{G}_{ba} = g\left(\lambda_a^{(A)}, \lambda_b^{(B)};\, \beta_a, \beta_b\right) \tag{3.9}$$

where now g is chosen to be an anisotropic graph filter such as one of those listed in table 3.3. Note that the original parameter β is now replaced by two parameters β_a and β_a which offer control over the filter intensity in each dimension. Filters of this kind appear often in image processing literature [Aubert and Kornprobst, 2006], however, their use in graph signal processing is so far limited. Figure 3.2 depicts an anisotropic graph filter applied to an image, which is a special case of a 2D Cartesian product graph signal.

Filter	$g(\lambda_a, \lambda_b; \beta_a, \beta_b)$
1-hop random walk	$(1 + \beta_a\lambda_a + \beta_b\lambda_b)^{-1}$
Diffusion	$\exp(-\beta_a\lambda_a - \beta_b\lambda_b)$
ReLu	$\max(1 - \beta_a\lambda_a - \beta_b\lambda_b, 0)$
Sigmoid	$2\big(1 + \exp(\beta_a\lambda_a + \beta_b\lambda_b)\big)^{-1}$
Bandlimited	1, if $\beta_a\lambda_a + \beta_b\lambda_b \leq 1$ else 0

TABLE 3.3: Anisotropic graph filter functions in two dimensions

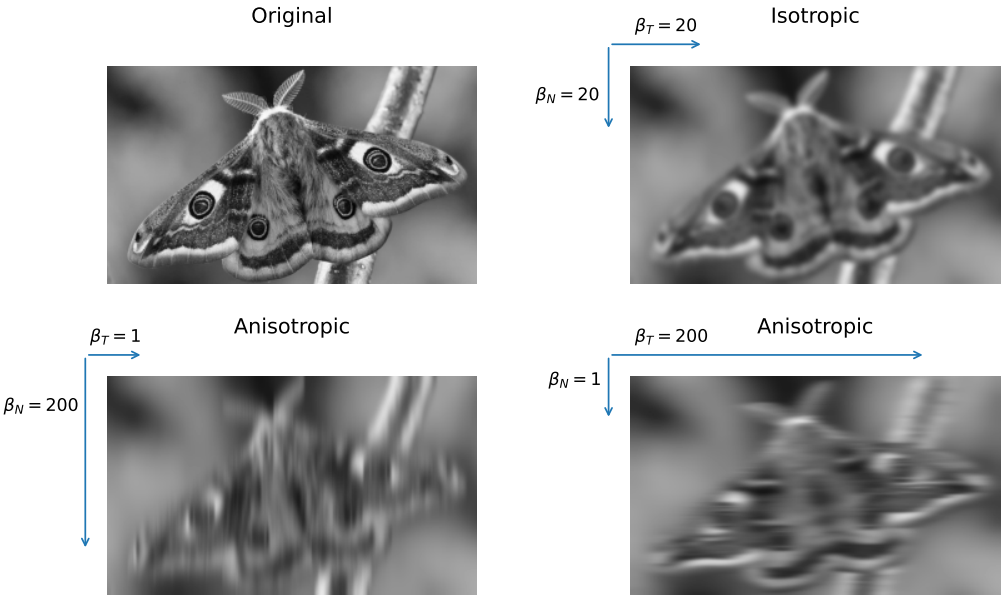


FIGURE 3.2: A visual representation of applying an isotropic and anisotropic graph filter. Here, the graph signal is an image, which can be thought of as existing on the nodes of a Cartesian product graph constructed by taking the product of two chain graphs.

3.2 Graph Signal Reconstruction on Cartesian Product Graphs

We now turn our attention to the task of signal reconstruction on Cartesian product graphs. In the following, we will replace the factor graph labels A and B with T and N respectively. The reason for this is that one application of particular interest is graph time-series problems, where we seek to model a network of N nodes across a series of T discrete time points. These so called "time-vertex" (T-V) problems have garnered significant interest recently in the context of GSP [Grassi et al., 2018, Isufi et al., 2017, Loukas and Foucard, 2016]. T-V signals can be understood as existing on the nodes of a Cartesian product graph $\mathcal{G}_T \,\square\, \mathcal{G}_N$. In particular, we can conceptualise T repeated measurements of a signal defined across the nodes of a N-node graph as a

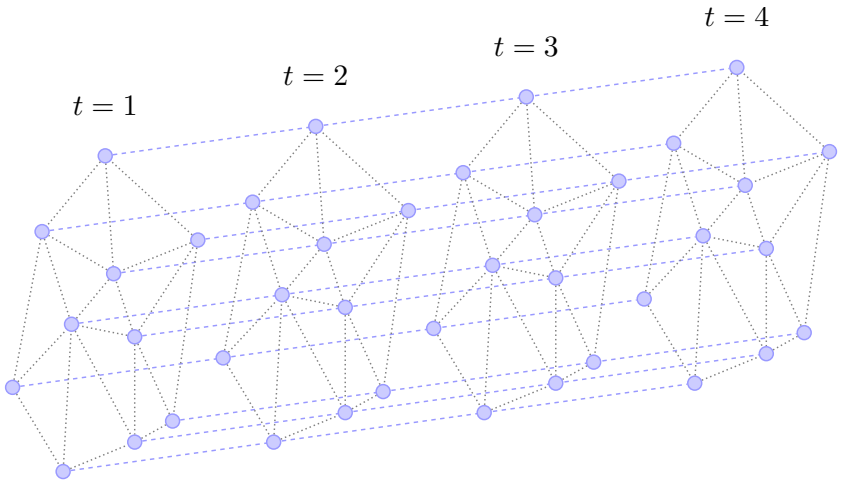


FIGURE 3.3: A graphical depiction of a time-vertex Cartesian product graph.

single measurement of a signal defined on the nodes of $\mathcal{G}_T \,\square\, \mathcal{G}_N$, where $\mathcal{G}_T$ is a simple path graph.

On the Laplacian spectrum of the path graph

When considering time-vertex problems with uniformly spaced time intervals, $\mathcal{G}_T$ will be described by a path graph with equal weights on each edge. This special case of a graph has vertices given by $\mathcal{V}_T = \{t \in \mathbb{N} \,|\, t \leq T\}$ and edges given by $\mathcal{E}_T = \{\,[t, t+1] \,|\, t < T\}$. The Laplacian matrix of the path graph is therefore given by

$$\mathbf{L}_T = \begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{bmatrix}$$

The eigenvalues and eigenvectors of this Laplacian are well-known and can be expressed in closed-form [Jiang, 2012]. In particular,

$$\lambda_t^{(T)} = 2 - 2\cos\left(\frac{t-1}{T}\pi\right)$$

and

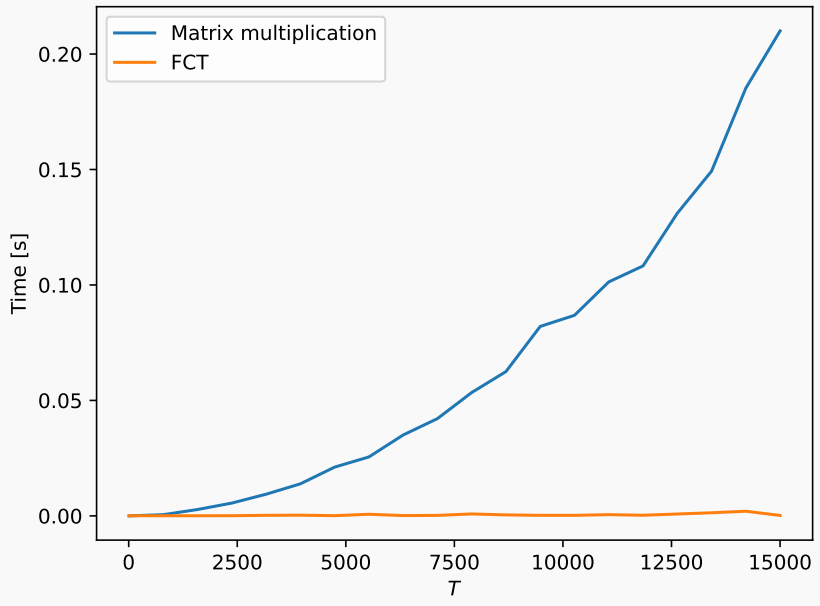
$$(\mathbf{U}_T)_{ij} = \cos\left(\frac{j-1}{T}(i - \frac{1}{2})\pi\right)$$

where the columns of $\mathbf{U}$ are appropriately normalised such that the magnitude of each eigenvector is one. Furthermore, this implies that that the graph Fourier transform of a signal $\mathbf{y} \in \mathbb{R}^T$ is given by the orthogonal type-II Discrete Cosine Transform (DCT) [Ahmed et al., 1974]. This is of significance, as it means we can leverage Fast Cosine Transform (FCT) algorithms [Makhoul, 1980] which operate in a similar manner to the well-known Fast Fourier Transform (FFT) [Cooley and Tukey, 1965]. See chapter 4 of Rao and Yip [1990] for an overview of FCT algorithms. In general, FFT-type algorithms are challenging to derive for the GFT [LeMagoarou and Gribonval, 2016], however, this is a simple case where it can be achieved.

In particular, this reduces both of the following procedures

$$\text{GFT}(\mathbf{y}) = \mathbf{U}_T^\top \mathbf{y} \quad \text{and} \quad \text{IGFT}(\mathbf{y}) = \mathbf{U}_T \mathbf{y}$$

from $O(T^2)$ operations to $O(T \log T)$ operations, which can be significant for large time-series problems. The figure below compares the time to compute the graph Fourier transform of a random signal using the matrix multiplication method vs the FCT implementation. In particular, we varied T from 10 to 15,000 in 20 equally spaced increments, and measured the mean time to compute $\mathbf{U}_T^\top \mathbf{y}$ across five independent trials using both the standard matrix multiplication and the Fast Cosine Transform method. As is visible, the difference becomes extremely pronounced as T grows large.



Note that, despite the observation that $\mathcal{G}_T$ is often a path graph in the context of T-V problems, the methods introduced in this section are valid for the Cartesian product between arbitrary undirected factor graphs.

3.2.1 Problem statement

The goal of Graph Signal Reconstruction (GSR) is to estimate the value of a partially observed graph signal at nodes where no data was collected. In the context of GSR on a Cartesian product graph, the available data is an observed signal $\mathbf{Y} \in \mathbb{R}^{N\times T}$ where only a partial set $\mathcal{S} = \{(n_1, t_1), (n_2, t_2), \dots\}$ of the signal elements were recorded. All other missing elements of $\mathbf{Y}$ are set to zero. To track which elements were missing from $\mathbf{Y}$, we also define $\mathbf{S} \in \{0,1\}^{N\times T}$, which is referred to as the binary sensing matrix. It has entries given by

$$\mathbf{S}_{nt} = \begin{cases} 1 & \text{if } (n,t) \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

As such, the data available for input into the GSR problem is as follows.

$$\text{input data} = \left\{ \mathbf{Y} \in \mathbb{R}^{N\times T},\ \ \mathbf{S} \in \{0,1\}^{N\times T},\ \ \mathbf{A} \in \mathbb{R}^{NT\times NT} \right\}$$

Our model is based on the assumption that $\mathbf{Y}$ is a noisy partial observation of an underlying signal $\mathbf{F} \in \mathbb{R}^{N\times T}$, which is assumed to be smooth with respect to the graph topology. Specifically, we assume that the observed matrix $\mathbf{Y}$ is generated according to the following statistical model.

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{F} + \mathbf{E}\big) \tag{3.11}$$

The matrix $\mathbf{E}$ represents the model error and is assumed to have an independent normal distribution with unit variance. Therefore, the probability distribution of $\mathbf{Y}$ given the latent signal $\mathbf{F}$ is

$$\text{vec}\,(\mathbf{Y}) \mid \mathbf{F} \sim \mathcal{N}\Big(\text{vec}\,(\mathbf{S} \circ \mathbf{F}),\ \text{diag}\,(\text{vec}\,(\mathbf{S}))\Big) \tag{3.12}$$

Note that the covariance matrix $\text{diag}\,(\text{vec}\,(\mathbf{S}))$ is semi-positive definite by construction. This naturally reflects the constraint that some elements of $\mathbf{Y}$ are zero with probability 1.

In order to estimate the latent signal $\mathbf{F}$, we must provide a prior distribution describing our belief about its likely profile ahead of time. In general, we expect $\mathbf{F}$ to be smooth with respect to the topology of the graph. This can be expressed by setting the covariance matrix in its prior to be proportional to $\mathbf{H}^2$, where $\mathbf{H}$ is a graph filter as defined in equation (3.7). For now, in the absence of any further information, we assume that the prior mean for $\mathbf{F}$ is zero across all elements.

$$\operatorname{vec}(\mathbf{F}) \sim \mathcal{N}\big(\mathbf{0},\, \gamma^{-1}\mathbf{H}^2\big) \tag{3.13}$$

Next, given an observation $\mathbf{Y}$, we use Bayes' rule to find the posterior distribution over $\mathbf{F}$. This is given by

$$\pi\big(\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{\pi\big(\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F}\big)\pi(\mathbf{F})}{\pi(\mathbf{Y})}. \tag{3.14}$$

where we use the notation $\pi(\cdot)$ to denote a probability density function.

The posterior distribution for $\mathbf{F}$ is given by

$$\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\big(\mathbf{P}^{-1}\operatorname{vec}(\mathbf{Y}),\, \mathbf{P}^{-1}\big) \tag{3.15}$$

where $\mathbf{P}$ is the posterior precision matrix, given by

$$\mathbf{P} = \operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma\mathbf{H}^{-2} \tag{3.16}$$

A proof of this can be found in the appendix, theorem A.1. In this chapter, we are primarily interested in computing the posterior mean, which is the solution to the following linear system.

$$\operatorname{vec}(\mathbf{F}) = \Big(\operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma\mathbf{H}^{-2}\Big)^{-1}\operatorname{vec}(\mathbf{Y}) \tag{3.17}$$

We return to the question of sampling from the posterior and estimating the posterior covariance directly in chapter 6.

Two significant computational challenges arise when working with non-trivial graph signal reconstruction problems, where the number of vertices in the product graph is large. First, although the posterior mean point estimator given in eq. (3.17) has an exact closed-form solution, its evaluation requires solving an $NT \times NT$ system of equations,

which is impractical for all but the smallest of problems. Second, since the eigenvalues of $\mathbf{H}$ can be close to or exactly zero, $\mathbf{H}^{-2}$ may be severely ill-conditioned and even undefined. This means the condition number of the coefficient matrix may not be finite, making basic iterative methods to numerically solve the linear system, such as steepest descent, slow or impossible. The models proposed in this paper aim to overcome these problems.

Since the coefficient matrix defining the system is of size $NT \times NT$, direct methods such as Gaussian elimination are assumed to be out of the question. In such cases, one often resorts to one of three possible solution approaches: stationary iterative methods; Krylov methods; and multigrid methods. Each are part of the family of iterative methods which are most commonly found in applications of sparse matrices, such as finite element methods [Brenner et al., 2008]. In the following, we propose a stationary iterative method and a Krylov method and compare their relative behaviour. In both cases, we show that each step of the iterative process can be completed in $O(N^2T+NT^2)$ operations, making a solution feasible for relatively large graph problems. First, we present each of the methods in isolation. Then, the convergence behaviour of each is derived theoretically and verified numerically.

3.2.2 A stationary iterative method

In this section, we demonstrate a technique for obtaining the posterior mean by adopting a classic approach to solving linear systems, known as *matrix splitting*, which sits within the family of Stationary Iterative Methods (SIMs) [Saad, 2003]. The general splitting strategy is to break the coefficient matrix into the form $\mathbf{M} - \mathbf{N}$, such that

$$\text{vec}\,(\mathbf{F}) = (\mathbf{M} - \mathbf{N})^{-1}\text{vec}\,(\mathbf{Y}) \tag{3.18}$$

By noting that

$$\mathbf{M}\text{vec}\,(\mathbf{F}) = \mathbf{N}\text{vec}\,(\mathbf{F}) + \text{vec}\,(\mathbf{Y}) \tag{3.19}$$

$$\text{vec}\,(\mathbf{F}) = \mathbf{M}^{-1}\mathbf{N}\text{vec}\,(\mathbf{F}) + \mathbf{M}^{-1}\text{vec}\,(\mathbf{Y}) \tag{3.20}$$

we devise an iterative scheme given by

$$\text{vec}\,(\mathbf{F}_{k+1}) = \mathbf{M}^{-1}\mathbf{N}\text{vec}\,(\mathbf{F}_k) + \mathbf{M}^{-1}\text{vec}\,(\mathbf{Y}) \tag{3.21}$$

When $\mathbf{M}$ is a simple matrix that is easy to invert, this update function can be vastly more efficient to compute. Common approaches to finding a suitable value for $\mathbf{M}$ and $\mathbf{N}$ include the Jacobi, Gauss-Seidel and successive over-relaxation methods, each of which represent a different strategy for splitting the coefficient matrix [Saad, 2003]. However, whilst these techniques are well-studied, they are not appropriate for use in the case of graph signal reconstruction. This is because, for each of these methods, the coefficient matrix is split according to its diagonal and off-diagonal elements in some way. Consequently, this would require the evaluation of $\mathbf{H}^{-2}$ directly which, as we have discussed, may be large, severely ill-conditioned and possibly ill-defined.

Instead, we require a custom splitting that avoids direct evaluation of $\mathbf{H}^{-2}$, and allows the right hand side of eq. (3.21) to be computed efficiently. The main contribution of this subsection is the identification of appropriate values for $\mathbf{M}$ and $\mathbf{N}$, and an investigation of the consequences of that choice.

In the following, we set

$$\mathbf{M} = \gamma \mathbf{H}^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}'\right)\right). \tag{3.22}$$

where $\mathbf{S}'$ is the binary matrix representing the complement of the set of selected nodes, i.e.

$$\mathbf{S}'_{nt} = \begin{cases} 1 & \text{if } (n,t) \notin \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

In this way, the update equation is given by

$$\operatorname{vec}\left(\mathbf{F}_{k+1}\right) = \left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}'\right)\right) \operatorname{vec}\left(\mathbf{F}_k\right) + \left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{vec}\left(\mathbf{Y}\right) \tag{3.24}$$

Note that this splitting is valid since $\left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1}$ is guaranteed to exist. It can also be readily computed as we already have the eigendecomposition of $\mathbf{H}$. Noting the decomposed definition of $\mathbf{H}$ given in eq. (3.7), this can be written as

$$\begin{aligned}\mathbf{M}^{-1} &= \left(\gamma\mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \\ &= \left(\gamma\big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\,\mathrm{diag}\left(\mathrm{vec}\left(\mathbf{G}\right)\right)^{-2}\,\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big) + \mathbf{I}\right)^{-1} \\ &= \big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\left(\gamma\,\mathrm{diag}\left(\mathrm{vec}\left(\mathbf{G}\right)\right)^{-2} + \mathbf{I}\right)^{-1}\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big) \\ &= \big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\,\mathrm{diag}\left(\mathrm{vec}\left(\mathbf{J}\right)\right)\,\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big) \end{aligned} \tag{3.25}$$

where $\mathbf{J} \in \mathbb{R}^{N\times T}$ has elements defined by

$$\mathbf{J}_{nt} = \frac{\mathbf{G}_{nt}^2}{\mathbf{G}_{nt}^2 + \gamma}. \tag{3.26}$$

Note that the update formula can be computed with $O(N^2T + NT^2)$ complexity at each step.

$$\mathbf{F}_{k+1} = \mathbf{U}_N\left(\mathbf{J} \circ \left(\mathbf{U}_N^\top\left(\mathbf{S}' \circ \mathbf{F}_k\right)\mathbf{U}_T\right)\right)\mathbf{U}_T^\top + \mathbf{F}_0 \tag{3.27}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N\left(\mathbf{J} \circ \left(\mathbf{U}_N^\top\,\mathbf{Y}\,\mathbf{U}_T\right)\right)\mathbf{U}_T^\top \tag{3.28}$$

Furthermore, this is reduced to $O(N^2T + NT\log T)$ in the case of T-V problems, and to $O\big(NT\log NT\big)$ for data residing on a grid (see section 3.2).

It is well-know that a given splitting will be convergent if the largest eigenvalue $\lambda_{\max}$ of the matrix $\mathbf{M}^{-1}\mathbf{N}$ has an absolute value of less than one. This attribute, $\rho = |\lambda_{\max}|$, is known as the spectral radius.

Whilst the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$ cannot be computed directly, we can derive an upper bound based on the properties of $\mathbf{M}$ and $\mathbf{N}$ individually.

Consider the spectral radius of $\mathbf{M}^{-1}$. By directly inspecting eq. (3.25), it is clear that $\rho(\mathbf{M}^{-1})$ will be the maximum entry in the matrix $\mathbf{J}$ since $\mathbf{M}^{-1}$ is already diagonalised in the basis $\mathbf{U}_T \otimes \mathbf{U}_N$. Consider now the definition of $\mathbf{J}$ given in eq. (3.26). By definition, $g(\cdot)$ has a maximum value of one on the non-negative reals, achieved when its argument is zero. Since the graph Laplacian is guaranteed to have at least one zero eigenvalue, the maximum entry in the matrix $\mathbf{J}$, and therefore the spectral radius of $\mathbf{M}^{-1}$, is surely given by

$$\rho(\mathbf{M}^{-1}) = \frac{1}{1+\gamma} \tag{3.29}$$

Next, consider the spectral radius of $\mathbf{N}$. This can be extracted directly as 1, since it is a diagonal binary matrix. Since both $\mathbf{M}^{-1}$ and $\mathbf{N}$ are positive semi-definite, we can apply the theorem

$$\rho(\mathbf{AB}) \leq \rho(\mathbf{A})\,\rho(\mathbf{B}) \tag{3.30}$$

[Bhatia, 1997]. Therefore, the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$ is guaranteed to be less than or equal to $1/(1+\gamma)$. Since γ is strictly positive, this is less than one and, as such, convergence is guaranteed. We return to the question of convergence more thoroughly in section 3.3.

Finally, the update formulas given in eqs. (3.27) and (3.28) can be written equivalently as

$$\Delta\mathbf{F}_0 = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{3.31}$$

$$\Delta\mathbf{F}_{k+1} = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \Delta\mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{3.32}$$

In this form, the iterations can be easily terminated when $|\Delta\mathbf{F}_k|$ is sufficiently small. The complete procedure is given in algorithm 1.

3.2.2.1 An eigendecomposition-free distributed implementation

In the previous section, we introduced the SIM algorithm, premised on the assumption that the matrices $\mathbf{L}_T$ and $\mathbf{L}_N$ could be decomposed into $\mathbf{U}_T\mathbf{\Lambda}_T\mathbf{U}_T^\top$ and $\mathbf{U}_N\mathbf{\Lambda}_N\mathbf{U}_N^\top$ respectively. However, it is also feasible to implement the SIM in a manner that avoids the eigendecomposition of both Laplacians, and instead only requires the repeated multiplication of vectors by $\mathbf{L}_T \oplus \mathbf{L}_N$ in the node domain. If, as is often the case, the original graphs are sparse, this can be achieved with complexity $O(T|\mathcal{E}_N| + N|\mathcal{E}_T|)$. This alternative is particularly beneficial when working with large factor graphs, since the complexity involved in decomposition generally scales at $O(N^3 + T^3)$.

Moreover, in certain contexts like sensor or IoT networks, nodes may possess the capability to communicate and compute locally. In such instances, a distributed approach to the signal reconstruction problem, employing a message-passing algorithm, may be more desirable. In this section, we will discuss on how these two objectives can be realised by utilising Chebyshev polynomials [Rivlin, 2020].

Algorithm 1 Stationary iterative method with matrix splitting

Input: Observation matrix $\mathbf{Y} \in \mathbb{R}^{N \times T}$
Input: Sensing matrix $\mathbf{S} \in \{0,1\}^{N \times T}$
Input: Space-like graph Laplacian $\mathbf{L}_N \in \mathbb{R}^{N \times N}$
Input: Time-like graph Laplacian $\mathbf{L}_T \in \mathbb{R}^{T \times T}$
Input: Regularisation parameter $\gamma \in \mathbb{R}^+$
Input: Graph filter function $g(\,\cdot\,; \boldsymbol{\beta} \in \mathbb{R}^2)$

Decompose $\mathbf{L}_N$ into $\mathbf{U}_N \boldsymbol{\Lambda}_L \mathbf{U}_N^\top$ and $\mathbf{L}_T$ into $\mathbf{U}_T \boldsymbol{\Lambda}_T \mathbf{U}_T^\top$
Compute $\mathbf{G} \in \mathbb{R}^{N \times T}$ as $\mathbf{G}_{nt} = g\left(\lambda_n^{(N)}, \lambda_t^{(T)}; \boldsymbol{\beta}\right)$
Compute $\mathbf{J} \in \mathbb{R}^{N \times T}$ as $\mathbf{J}_{nt} = \mathbf{G}_{nt}^2 / (\mathbf{G}_{nt}^2 + \gamma)$
$\mathbf{S}' \leftarrow \mathbf{1} \in \mathbb{R}^{N \times T} - \mathbf{S}$
$\Delta\mathbf{F} \leftarrow \mathbf{U}_N \big(\mathbf{J} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T)\big) \mathbf{U}_T^\top$
$\mathbf{F} \leftarrow \Delta\mathbf{F}$
while $|\Delta\mathbf{F}| >$ tol **do**
  $\Delta\mathbf{F} \leftarrow \mathbf{U}_N \Big(\mathbf{J} \circ \big(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \Delta\mathbf{F}\right) \mathbf{U}_T\big)\Big) \mathbf{U}_T^\top$
  $\mathbf{F} \leftarrow \mathbf{F} + \Delta\mathbf{F}$
end while

Output: $\mathbf{F}$

First, note that each iteration of the SIM algorithm is computed by multiplying some vector, $\mathrm{vec}\,(\mathbf{Z})$, by the matrix $\mathbf{M}^{-1}$, which is given by eq. (3.25). Crucially, since $\mathbf{M}^{-1}$ has eigenvectors $\mathbf{U}_T \otimes \mathbf{U}_N$, it can be understood as function applied to a weighted Kronecker sum of the factor graph Laplacians. In particular,

$$\mathbf{M}^{-1} = J(\beta_T \mathbf{L}_T \oplus \beta_N \mathbf{L}_N) \tag{3.33}$$

where

$$J(x) = \frac{g^2(x)}{g^2(x) + \gamma}$$

and $g(\cdot)$ represents the original filter function used, with parameters β_T, β_N. (Here, we interpret the application of $J(x)$ to a matrix in terms of a power series rather than element-wise.) Thus, eigendecomposition can be entirely bypassed by approximating the function $J(x)$ using shifted Chebyshev polynomials [Isufi et al., 2022]. This requires knowledge of the largest eigenvalues of $\mathbf{L}_T$ and $\mathbf{L}_N$, $\lambda_T^{(\max)}$ and $\lambda_N^{(\max)}$ respectively, but these can also be computed efficiently using methods that take advantage of sparsity such as Arnoldi iterations.

Assuming an order K approximation to the function $J(x)$ is used, with shifted Chebyshev polynomials $\bar{T}_k(x)$ defined over the interval $[0, \bar{\lambda}]$, where $\bar{\lambda} = \beta_T \lambda_T^{(\max)} + \beta_N \lambda_N^{(\max)}$, the approximation is given by

$$J(x) \approx \sum_{k=0}^{K} c_k \bar{T}_k(x) \tag{3.34}$$

where the coefficients, c_k, are computed numerically via the integral given in eq. (2.9) of section 2.2.2. The action of $\mathbf{M}^{-1}$ on an arbitrary vector $\mathrm{vec}\,(\mathbf{Z})$ can then be approximately computed as

$$\mathbf{M}^{-1}\mathrm{vec}\,(\mathbf{Z}) \approx \sum_{k=0}^{K} c_k\, \bar{T}_k(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) \tag{3.35}$$

where $\bar{\mathbf{L}} = \beta_T \mathbf{L}_T \oplus \beta_N \mathbf{L}_N$. The result of $\bar{T}(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z})$ is defined recursively as

$$\bar{T}_k(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) = \left(\frac{4\bar{\mathbf{L}}}{\bar{\lambda}} - 2\right) \bar{T}_{k-1}(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) - \bar{T}_{k-2}(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) \tag{3.36}$$

with the initial conditions

$$\bar{T}_0(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) = \mathrm{vec}\,(\mathbf{Z})\,, \quad \text{and} \quad \bar{T}_1(\bar{\mathbf{L}})\mathrm{vec}\,(\mathbf{Z}) = \frac{2}{\bar{\lambda}}\bar{\mathbf{L}}\mathrm{vec}\,(\mathbf{Z}) - \mathrm{vec}\,(\mathbf{Z}) \tag{3.37}$$

In addition, the action of $\bar{\mathbf{L}}$ on $\mathrm{vec}\,(\mathbf{Z})$ can be efficiently computed as

$$\mathrm{mat}\left(\bar{\mathbf{L}}\mathrm{vec}\,(\mathbf{Z})\right) = \beta_N \mathbf{L}_N \mathbf{Z} + \beta_T \mathbf{Z}\mathbf{L}_T \tag{3.38}$$

with complexity $O(T|\mathcal{E}_N| + N|\mathcal{E}_T|)$. When performed in a distributed manner, this operation can be executed at each node utilising information about the value of $\mathrm{vec}\,(\mathbf{Z})$ at its direct neighbours only. This implies that, to compute an order K polynomial, information will need to be gatherer from nodes that are a maximum of K hops away via graph edges.

In the context of a graph time series problem, this method also lends itself to an online implementation. In particular, since each node requires information from nodes no further than K hops away, the signal at node n at time t can be reconstructed when the observed signal at time $t + K$ becomes available.

One caveat worth noting is that the accuracy of the Chebyshev approximation depends not only on the order of the polynomial but also on the filter used and its parameter(s). Figure 3.4 demonstrates how an order-3 approximation to $J(x)$, with $\gamma = 0.05$, varies across several different filter types and parameter settings. For filter functions that exhibit slow variation, typically corresponding to smaller values of β, the fit is usually quite accurate. However, in certain other contexts, it deviates significantly from the true filter function. Another consideration is that the spectral radius of $\mathbf{M}^{-1}$ is no longer guaranteed to be less than one. For instance, in the case of the bandlimited filter, the function clearly reaches higher values. To guarantee convergence, the polynomial coefficients should be adjusted such that the approximation remains within the range of $[-1, 1]$.

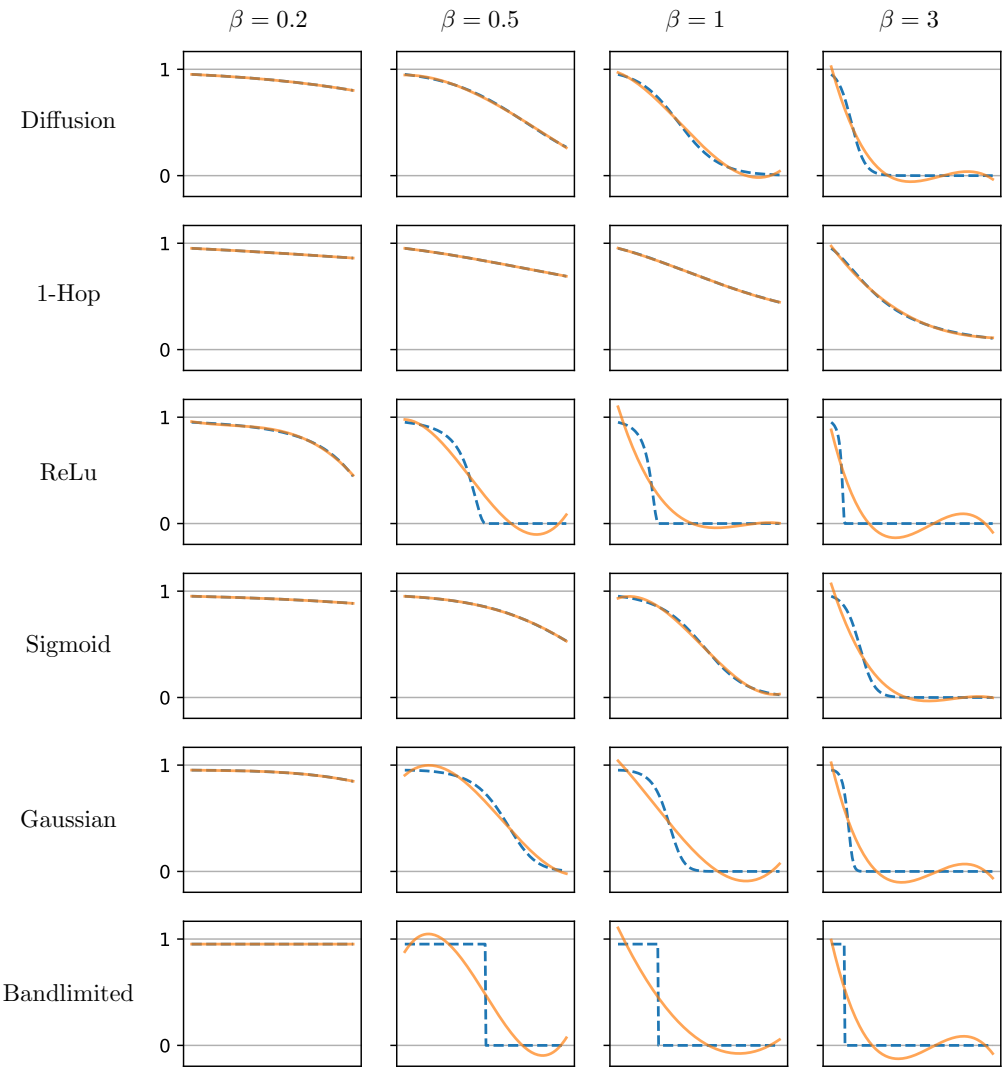


FIGURE 3.4: An order-3 Chebyshev polynomial approximation (orange) is compared to the true output of $J(x)$ (dashed blue) for several different filter types across various parameter settings.

3.2.3 A conjugate gradient method

The second approach we consider for computing the posterior mean is to use the Conjugate Gradient Method (CGM). First proposed in 1952, the CGM is part of the Krylov subspace family, and is perhaps the most prominent iterative algorithm for solving linear systems [Hestenes and Stiefel, 1952]. In computational terms, the method only requires repeated forward multiplication of vectors by the coefficient matrix which, in the standard CGM, bust be PSD. It is therefore effective in applications where this process can be performed efficiently.

In brief, the CGM seeks to solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ by minimising, at the k-th iteration, some measure of error in the affine space $\mathbf{x}_0 + \mathcal{K}_k$ where $\mathcal{K}_k$ is the k-th Krylov subspace given by

$$\mathcal{K}_k = \text{span}\big(\mathbf{r}_0,\ \mathbf{A}\mathbf{r}_0,\ ...,\ \mathbf{A}^{k-1}\mathbf{r}_0\big)$$

The residual $\mathbf{r}_k$ is given by

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}$$

and the k-th iterate of the CGM minimises

$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{x}^\top \mathbf{b}$$

over $\mathbf{x}_0 + \mathcal{K}_k$ [Kelley, 1995].

The CGM works best when the coefficient matrix $\mathbf{A}$ has a low condition number κ (that is, the ratio between the largest and smallest eigenvalue is small) and, as such, a preconditioning step is often necessary. The purpose of a preconditioner is to reduce κ by solving an equivalent transformed problem. This can be achieved by right or left multiplying the linear system by a preconditioning matrix $\mathbf{\Psi}$. However, this likely means the coefficient matrix is no longer PSD, meaning the CGM cannot be used in its basic form. (Other approaches modified for non-PSD matrices exists, e.g. the CGNE or GIMRES [Elman, 1982, Saad and Schultz, 1986]). A preconditioner can also multiply the coefficient matrix on the right by a preconditioner $\mathbf{\Psi}^\top$ and the left by $\mathbf{\Psi}$. This preserves the symmetry meaning we can can continued to use the regular CGM.

In our case, where the coefficient matrix is given by $\left(\text{diag}\left(\text{vec}\left(\mathbf{S}\right)\right) + \gamma\mathbf{H}^{-2}\right)$, preconditioning will be essential for convergence. To see why, consider the definition of $\mathbf{H}$ in equation (3.7). A low-pass filter function $g(\cdot)$ may be close to zero when applied to the high-Laplacian frequency eigenvalues of the graph Laplacian, meaning elements of $\text{diag}\left(\text{vec}\left(\mathbf{G}\right)\right)^{-2}$ may be very high. In the worst case, for example with a band-limited filter, the matrix $\mathbf{H}$ will be singular, no matrix $\mathbf{H}^{-2}$ will exist, and the condition number of the coefficient matrix will be, in effect, infinite. Therefore, the primary purpose of this subsection is to find a preconditioner that maintains efficient forward multiplication and is effective at reducing the condition number of the coefficient matrix.

References such as [Saad, 2003] give a broad overview of the known approaches to finding a preconditioner. Standard examples include the Jacobi preconditioner which is given by the inverse of the coefficient matrix diagonal and is effective for diagonally dominant matrices, and the Sparse Approximate Inverse preconditioner [Grote and Huckle, 1997]. However, such preconditioners generally require direct evaluation of parts of the coefficient matrix or are computationally intensive to calculate.

In the following, we derive an effective symmetric preconditioner that allows forward multiplication of the coefficient matrix to be performed efficiently. First consider the transformed variable $\mathbf{Z}$, related to $\mathbf{F}$ in the following way.

$$\mathbf{F} = \mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top \tag{3.39}$$

Here, $\mathbf{Z}$ can be interpreted as set of Laplacian frequency coefficients, which are subsequently scaled according to the graph filter function, and then reverse Fourier transformed back into the node domain. Matrices $\mathbf{Z}$ which are distributed according to a spherically symmetric distribution, result in signals $\mathbf{F}$ which are smooth with respect to the graph topology. Since this transform filters out the problematic high-Laplacian frequency Fourier components, the system defined by this transformed variable $\mathbf{Z}$ is naturally far better conditioned.

By substituting this expression for $\mathbf{F}$ back into the likelihood in equation (3.12), and the prior of equation (3.13), one can derive a new expression for the posterior mean of $\mathbf{Z}$. This is done explicitly in theorem A.2. The end result is that the new linear system for the transformed variable $\mathbf{Z}$ is given by

$$\text{vec}\left(\mathbf{Z}\right) = \left(\mathbf{D}_\mathbf{G}\left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right)\mathbf{D}_\mathbf{S}\left(\mathbf{U}_T \otimes \mathbf{U}_N\right)\mathbf{D}_\mathbf{G} + \gamma\mathbf{I}_{NT}\right)^{-1} \text{vec}\left(\mathbf{G} \circ \left(\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T\right)\right) \tag{3.40}$$

where we have abbreviated diag (vec (**G**)) and diag (vec (**S**)) as $\mathbf{D_G}$ and $\mathbf{D_S}$ respectively. Note that the conditioning of the coefficient matrix is greatly improved from the untransformed problem, as we will discuss in greater detail in section 3.3. Note also that multiplication of a vector vec (**R**) by the coefficient matrix can be computed efficiently as

$$\begin{aligned}\operatorname{mat}\Big(\Big(\mathbf{D_G}\left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right)\mathbf{D_S}\left(\mathbf{U}_T \otimes \mathbf{U}_N\right)\mathbf{D_G} + \gamma \mathbf{I}_{NT}\Big)\operatorname{vec}(\mathbf{R})\Big) \\ = \gamma \mathbf{R} + \mathbf{G} \circ \Big(\mathbf{U}_N^\top\Big(\mathbf{S} \circ \left(\mathbf{U}_N(\mathbf{G} \circ \mathbf{R})\mathbf{U}_T^\top\right)\Big)\mathbf{U}_T\Big)\end{aligned} \tag{3.41}$$

This has $O(N^2T + NT^2)$ complexity at each step which may be reduced to $O(N^2T + NT\log T)$ in the case of T-V problems, and to $O\big(NT\log NT\big)$ for data residing on a grid (see section 3.2).

The linear system defined eq. (3.40) can be understood as a two-sided symmetrically preconditioned version of the original linear system given in eq. (3.17). In particular, the new expression can be constructed by modifying the original system in the following way.

$$\Big(\mathbf{\Psi}^\top\left(\mathbf{D_S} + \gamma \mathbf{H}^{-2}\right)\mathbf{\Psi}\Big)\Big(\mathbf{\Psi}^{-1}\operatorname{vec}(\mathbf{F})\Big) = \mathbf{\Psi}^\top \operatorname{vec}(\mathbf{Y}), \tag{3.42}$$

where

$$\mathbf{\Psi} = \left(\mathbf{U}_T \otimes \mathbf{U}_N\right)\mathbf{D_G}. \tag{3.43}$$

Since preconditioning of the coefficient matrix on the left is achieved with $\mathbf{\Psi}^\top$ and on the right with $\mathbf{\Psi}$, symmetry is preserved. This ensures that one can continue to utilise algorithms tailored to work with PSD matrices. In algorithm **2**, we outline a conjugate gradient method based on this new formulation.

3.2.4 Real data experiments

In this subsection, we evaluate our GSR method using a dataset consisting of daily new SARS-CoV-2 cases reported in 372 lower-tier local authorities across the United Kingdom from February 5, 2020, to March 18, 2023 (1138 days) taken from the UK government website[1]. Specifically, we focused on the "newCasesBySpecimenDateRollingRate"

[1]See https://coronavirus.data.gov.uk/details/download

Algorithm 2 Conjugate gradient method with graph-spectral preconditioner

Input: Observation matrix $\mathbf{Y} \in \mathbb{R}^{N \times T}$
Input: Sensing matrix $\mathbf{S} \in \{0,1\}^{N \times T}$
Input: Space-like graph Laplacian $\mathbf{L}_N \in \mathbb{R}^{N \times N}$
Input: Time-like graph Laplacian $\mathbf{L}_T \in \mathbb{R}^{T \times T}$
Input: Regularisation parameter $\gamma \in \mathbb{R}$
Input: Graph filter function $g(\,\cdot\,;\boldsymbol{\beta})$

Decompose $\mathbf{L}_N$ into $\mathbf{U}_N \boldsymbol{\Lambda}_L \mathbf{U}_N^\top$ and $\mathbf{L}_T$ into $\mathbf{U}_T \boldsymbol{\Lambda}_T \mathbf{U}_T^\top$

Compute $\mathbf{G} \in \mathbb{R}^{N \times T}$ as $\mathbf{G}_{nt} = g\left(\lambda_a^{(A)}, \lambda_b^{(B)};\, \beta_a, \beta_b\right)$

Initialise $\mathbf{Z} \in \mathbb{R}^{N \times T}$ randomly

$\mathbf{R} \leftarrow \mathbf{G} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T) - \gamma \mathbf{Z} - \mathbf{G} \circ \left(\mathbf{U}_N^\top \big(\mathbf{S} \circ (\mathbf{U}_N \,(\mathbf{G} \circ \mathbf{Z})\, \mathbf{U}_T^\top)\big)\, \mathbf{U}_T \right)$

$\mathbf{D} \leftarrow \mathbf{R}$

while $|\Delta \mathbf{R}| > \text{tol}$ **do**

$\mathbf{A}_D \leftarrow \gamma \mathbf{D} + \mathbf{G} \circ \left(\mathbf{U}_N^\top \big(\mathbf{S} \circ (\mathbf{U}_N \,(\mathbf{G} \circ \mathbf{D})\, \mathbf{U}_T^\top)\big)\, \mathbf{U}_T \right)$

$\alpha \leftarrow \operatorname{tr}\left(\mathbf{R}^\top \mathbf{R}\right) \,/\, \operatorname{tr}\left(\mathbf{R}^\top \mathbf{A}_D \mathbf{R}\right)$

$\mathbf{Z} \leftarrow \mathbf{Z} + \alpha \mathbf{D}$

$\mathbf{R} \leftarrow \mathbf{R} - \alpha \mathbf{A}_D$

$\delta \leftarrow \operatorname{tr}\left(\mathbf{R}^\top \mathbf{R}\right) \,/\, \operatorname{tr}\left((\mathbf{R} + \alpha \mathbf{A}_D)^\top (\mathbf{R} + \alpha \mathbf{A}_D)\right)$

$\mathbf{D} \leftarrow \mathbf{R} + \delta \mathbf{D}$

end while

Output: $\mathbf{U}_N \,(\mathbf{G} \circ \mathbf{Z})\, \mathbf{U}_T^\top$

metric, which represents the daily number of cases reported per 100,000 residents in each local reporting authority over a 7-day rolling period. To create a graph, we used boundary data [2], setting adjacency matrix entries as $\mathbf{A}_{ij} = 1$ if districts i and j share a border, and 0 otherwise. Figure 3.5 illustrates a snapshot of this dataset on December 1, 2020.

Before beginning the experiments, we performed two preprocessing steps on the raw signal data. First, we took the logarithm of 10 plus the original case rate. This was to eliminate the long tail in the case rate histogram, transforming it to be closer to a Gaussian. We then normalised by subtracting the overall mean and dividing by the standard deviation. The resultant signal, of shape 372×1138, we refer to as $\mathbf{Y}_0$. Note that 4.8% of the entries in $\mathbf{Y}_0$ were already missing from the original dataset (mostly occurring in the earlier stages of the pandemic).

[2] Data from the Office for National Statistics [ONS, 2019]

The experiment was conducted as follows. First, we removed data from $\mathbf{Y}_0$ such that a total fraction m was no longer present. This created two matrices: $\mathbf{Y}$, the partially observed signal with missing values filled with zeros; and $\mathbf{S}$, the corresponding binary sensing matrix. Data was removed in four distinct ways. First, node/times were selected uniformly at random for removal ('uniform'). Second, strings of 100 days, beginning at a random date, were removed for individual randomly selected districts ('strings'). Third, the entire time series for randomly chosen districts was removed ('districts'). Finally, the signal across every node at randomly selected dates was removed ('dates'). These four techniques for data removal are depicted for clarity in fig. 3.6. For each technique, we solved the signal reconstruction problem using the GSR model described in this chapter and compared its performance to other baseline reconstruction strategies. In particular, for uniform and string removal, we compared it to linear interpolation in time and longitudinal averaging across all districts. For date removal, we compared it to interpolation in time only, since longitudinal averaging is not possible in this case. Finally, for district removal, we compared it to longitudinal averaging, since linear interpolation in time is not possible in this case. For each model, for each method of data removal, we

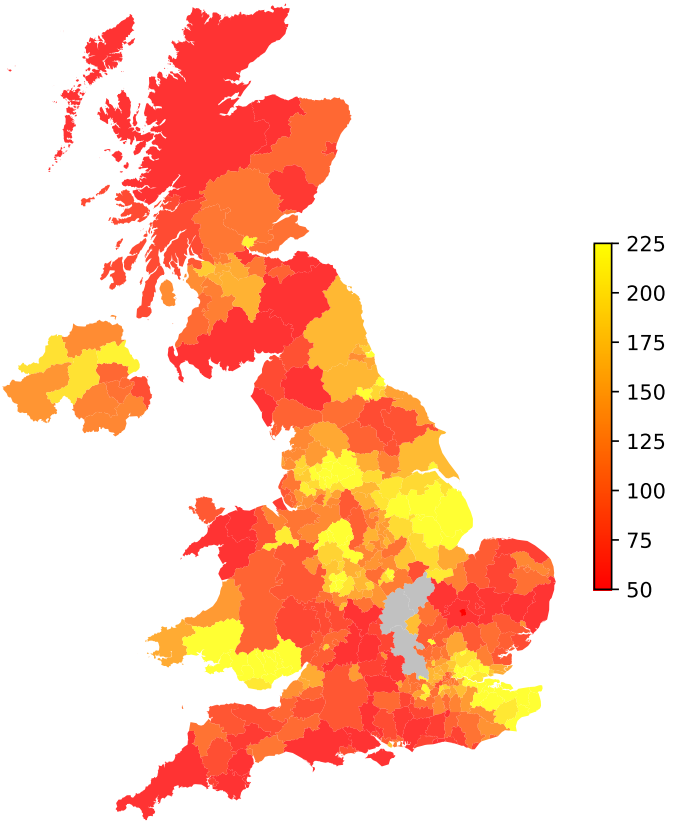


FIGURE 3.5: The seven day rolling rate of new covid cases reported per 100,000 residents in each lower tier local reporting authority in the United Kingdom on the 1st of December 2020. Missing data is indicated in gray.

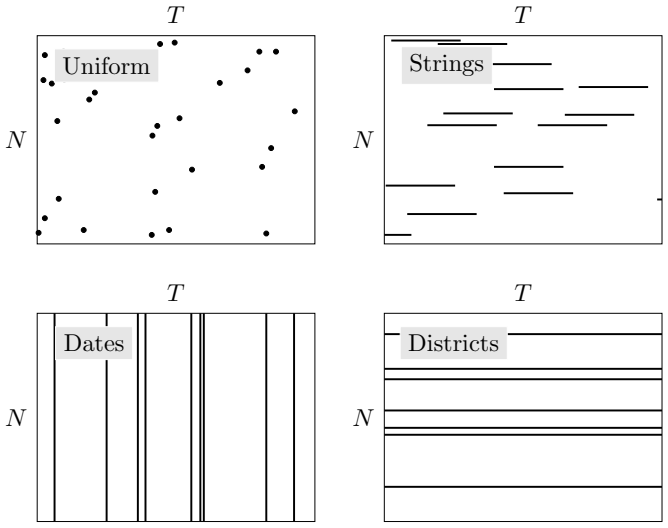


FIGURE 3.6: A visual depiction of the four ways we removed data. Black lines/dots indicate data that was removed.

	Uniform				Strings			
m	0.1	0.3	0.5	0.7	0.1	0.3	0.5	0.7
GSR	0.037	0.040	0.048	0.081	0.230	0.249	0.247	0.243
Linear interp	0.034	0.039	0.048	0.070	0.540	0.531	0.520	0.526
Longitudinal avrg	0.350	0.348	0.347	0.348	0.341	0.350	0.356	0.347
	Dates				Districts			
m	0.1	0.3	0.5	0.7	0.1	0.3	0.5	0.7
GSR	0.038	0.055	0.058	0.075	0.202	0.249	0.271	0.273
Linear interp	0.035	0.038	0.046	0.066	NA	NA	NA	NA
Longitudinal avrg	NA	NA	NA	NA	0.318	0.333	0.347	0.345

TABLE 3.4: The RMSE for different reconstruction models and data removal techniques on the UK SARS-CoV-2 case rate dataset. For each value of m, the best performing model is highlighted in green. Entries where the model is not applicable is indicated by NA.

measured the Root Mean Square Error (RMSE) across the reconstructed entries, over four increasing values of m. The results are shown in table 3.4.

The findings reveal that the GSR method demonstrates superior performance by a noticeable margin when substantial segments of time series data are removed from a specific district. This is evidenced by the low RMSE across all values of m for the 'strings' and 'districts' removal techniques. Notice also that GSR significantly outperforms longitudinal averaging in both instances, suggesting that the incorporation of topological relations between districts has yielded substantial advantages. On the other hand, when individual dates are removed, meaning only brief sections of the time series are likely missing within any given district, GSR remains competitive, but linear interpolation tends to exhibit slightly better performance. This observation is logical for this dataset, as the metric is calculated using a 7-day rolling average, resulting in a highly smooth signal over time. GSR, however, remains the most versatile technique, as it can be applied across all patterns of missing data.

3.3 Convergence properties

In this section, we conduct a thorough theoretical analysis of the convergence properties of both the SIM and the CGM. As we will demonstrate, their respective convergence rates are heavily influenced by the values of the hyperparameters β, which describes the strength of the graph filter, γ, which determines the regularization strength, and $m = |\mathcal{S}'|/NT$, which represents the fraction of values missing from the original graph signal. This helps provide an explanation for the empirical convergence behaviour, and offers insight into trade-offs when it comes to hyperparameter selection. Furthermore, it allows users to make an informed decision with regard to selecting an appropriate method, considering the inherent characteristics of their specific problem.

It is well-known that the worst-case number of iterations required to reduce the error below some specific tolerance level for matrix splitting methods is inversely proportional to $-\log \rho(\mathbf{M}^{-1}\mathbf{N})$, where $\rho(\cdot)$ denotes the spectral radius (absolute value of the maximum eigenvalue) of a matrix [Demmel, 1997]. For completeness, we provide a brief proof of this in theorem A.3. In the specific context of our graph signal reconstruction algorithm as outlined in section 3.2.2, $\mathbf{M}$ and $\mathbf{N}$ have the following values.

$$\mathbf{M} = \left(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\right)^{-1}, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\mathbf{S}'}$$

where $\mathbf{U} = \mathbf{U}_T \otimes \mathbf{U}_N$, $\mathbf{D}_{\mathbf{J}} = \operatorname{diag}(\operatorname{vec}(\mathbf{J}))$, and $\mathbf{D}_{\mathbf{S}'} = \operatorname{diag}(\operatorname{vec}(\mathbf{S}'))$. Therefore, the number of iterations required for convergence of the SIM scales as

$$n_{\text{SIM}} \propto -\frac{1}{\log \rho \left(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'} \right)} \tag{3.44}$$

Note that the matrix $\mathbf{J}$ [see eq. (3.26) for its definition] has entries that depend on both the the regularisation parameter γ and the spectral scaling matrix $\mathbf{G}$, which is itself a function of the graph filter parameter(s) β [see eqs. (3.8) and (3.9)]. The matrix $\mathbf{D}_{\mathbf{S}'}$ has entries that depend on the structure of the missing data in the graph signal. Therefore should we expect that the spectral radius, ρ, and consequently the number of steps required for convergence, n_{SIM}, can be affected by all three.

Similarly, in the conjugate gradient method, the worst-case number of steps required to achieve a specific termination criterion is well-known to be proportional to $\sqrt{\kappa}$, where κ represents the condition number of the coefficient matrix, i.e. the ratio between the largest and smallest eigenvalue Kelley [1995]. In our particular scenario, the coefficient matrix is provided in eq. (3.40). Therefore, we should expect that the number of iterations required for convergence of the CGM will scale as

$$n_{\text{CGM}} \propto \sqrt{\kappa \left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma \mathbf{I}_{NT} \right)} \tag{3.45}$$

where $\mathbf{D}_{\mathbf{G}} = \text{diag}\left(\text{vec}\left(\mathbf{G}\right)\right)$. Once again, this expression contains the matrix $\mathbf{G}$, which depends on the strength of the graph filter function parameter β, the matrix $\mathbf{D}_{\mathbf{S}}$, which depends on the structure of this missing data, and the precision parameter γ. Consequently, we should expect that, in general, convergence of the CGM is affected by all three of these variables.

Whilst it is not possible in general to obtain an analytic expression for ρ or κ as a function of γ, β and m, we can nonetheless gain useful insight into how each of these variables can be expected to affect convergence. We achieve this by considering two distinct limits: one in which the graph filter is very strong (i.e β is very large) and one in which the graph filter is very weak (i.e. β is very small).

3.3.1 Upper bound on convergence: the weak filter limit

Consider the limiting case of a weak filter, where all spectral components are allowed to pass through unaffected. In this case, a graph filter $\mathbf{H}$ [see eq. (3.7)], which appears in the prior distribution for $\mathbf{F}$ [see eq. (3.13)], becomes the identity matrix $\mathbf{I}_{NT}$. This means no topological information is included in the prior for $\mathbf{F}$ at all. Given the definitions of the graph filters in **??** and table 3.3, we can conceptualise this as the limit where the parameter characterising the graph filter $\beta \to 0$ (or, more generally, the limit as

$\boldsymbol{\beta} \to [0, 0]$ for an anisotropic graph filter). Since all spectral components are maintained, every element of the spectral scaling matrix $\mathbf{G}$ will be equal to one. Given eq. (3.26), this further implies the every entry in the matrix $\mathbf{J}$ becomes $1/(1+\gamma)$.

$$\lim_{\beta \to 0} \mathbf{D}_{\mathbf{G}} = \mathbf{I}_{NT}, \quad \text{and} \quad \lim_{\beta \to 0} \mathbf{D}_{\mathbf{J}} = \frac{1}{1+\gamma} \mathbf{I}_{NT}$$

Now consider the spectral radius ρ of the update matrix in the SIM. Given this limiting value of $\mathbf{D}_{\mathbf{J}}$, it can be directly evaluated as

$$\lim_{\beta \to 0} \rho\Big(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}\Big) = \frac{1}{1+\gamma}\rho\Big(\mathbf{D}_{\mathbf{S}'}\Big) = \frac{1}{1+\gamma} \tag{3.46}$$

Next, consider the condition number κ of the coefficient matrix in the CGM. Again, since in this limit $\mathbf{D}_{\mathbf{G}} = \mathbf{I}$, it can be directly evaluated as

$$\lim_{\beta \to 0} \kappa\left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I}\right) = \kappa\left(\mathbf{U}^{\top}\left(\mathbf{D}_{\mathbf{S}} + \gamma\mathbf{I}\right)\mathbf{U}\right) = \frac{1+\gamma}{\gamma} \tag{3.47}$$

Given eqs. (3.44) and (3.45), we can characterise the number of iterations required to reach some convergence criterion in the weak filter limit for the SIM and CGM respectively as

$$\lim_{\beta \to 0} n_{\text{SIM}} \propto \frac{1}{\log(1+\gamma)}, \quad \text{and} \quad \lim_{\beta \to 0} n_{\text{CGM}} \propto \sqrt{\frac{1}{\gamma} + 1} \tag{3.48}$$

These expressions imply that when γ is large, both methods converge quickly. However, they both see the number of iterations increase to infinity as $\gamma \to 0$. To characterise this more precisely, consider the Taylor expansion of each expression around $\gamma = 0$.

$$\lim_{\beta \to 0} n_{\text{SIM}} \propto \gamma^{-1} + O(\gamma), \quad \text{and} \quad \lim_{\beta \to 0} n_{\text{CGM}} \propto \gamma^{-1/2} + O\left(\gamma^{1/2}\right) \tag{3.49}$$

As visible, dominant behaviour for small γ follows $O(\gamma^{-1})$ for the SIM and $O(\gamma^{-1/2})$ for the CGM.

3.3.2 Lower bound on convergence: the strong filter limit

Consider now the limiting case of a strong filter as applied to a signal on a fully connected Cartesian product graph. In this case, every spectral component is filtered out except

the the first Laplacian frequency component $\mathbf{u}_1^{(T)} \otimes \mathbf{u}_1^{(N)} \propto \mathbf{1}$ (also known as the bias), with eigenvalue $\lambda_1^{(T)} + \lambda_1^{(N)} = 0$, which passes through the filter unaffected. When a filter of this kind is used in the prior for $\mathbf{F}$, it effectively forces predictions that are constant across all nodes. Given the definitions of the graph filter functions given in **??** and table 3.3, we can associate this with the limit as $\beta \to \infty$. Here, the effect of applying the graph filter to a generic graph signal $\mathrm{vec}\,(\mathbf{Y})$ is to extract the mean, that is

$$\mathbf{H}\mathrm{vec}\,(\mathbf{Y}) = \frac{1}{NT}\left(\sum_{n,t} \mathbf{Y}_{nt}\right)\mathbf{1}$$

In this case, the spectral scaling matrix $\mathbf{G}$ has entries that are zero for all elements except (1, 1) which has the value one. Similarly, the matrix $\mathbf{J}$ has the value $1/(1+\gamma)$ at element (1, 1) and zeros elsewhere. This implies that

$$\lim_{\beta\to\infty} \mathbf{D}_{\mathbf{G}} = \mathbf{\Delta}, \quad \text{and} \quad \lim_{\beta\to\infty} \mathbf{D}_{\mathbf{J}} = \frac{1}{1+\gamma}\mathbf{\Delta},$$

where $\mathbf{\Delta}$ is an $NT \times NT$ matrix given by

$$\mathbf{\Delta} = \begin{bmatrix} 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & \\ \vdots & & & \ddots \end{bmatrix}$$

In the case of the SIM, the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$ in this limit is therefore given by

$$\lim_{\beta\to\infty} \rho\Big(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}'}\Big) = \frac{1}{1+\gamma}\rho\Big(\mathbf{U}\mathbf{\Delta}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}'}\Big)$$

Note that

$$\mathbf{U}\mathbf{\Delta}\mathbf{U}^\top = \mathbf{u}_1\mathbf{u}_1^\top = \frac{1}{NT}\mathbf{O}_{NT}$$

where $\mathbf{O}_{NT}$ is an $NT \times NT$ matrix of ones. Therefore the spectral radius is given by

$$\lim_{\beta\to\infty} \rho(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}'}) = \frac{1}{NT(1+\gamma)}\,\rho\left(\begin{bmatrix}\mathrm{vec}\,(\mathbf{S}')^\top\\ \mathrm{vec}\,(\mathbf{S}')^\top\\ \dots\\ \mathrm{vec}\,(\mathbf{S}')^\top\end{bmatrix}\right)$$

Since the matrix in brackets is just the vector $\mathrm{vec}\,(\mathbf{S}')^\top$ repeated in every row it is surely of rank one, and therefore must have an eigenvalue of 0 with multiplicity $NT-1$. This implies the the only non-zero eigenvalue (and therefore the spectral radius ρ) is given by its trace, which is $\sum_{n,t}\mathbf{S}'_{nt} = |\mathcal{S}'|$. Denoting $m = |\mathcal{S}'|/NT$, this can be expressed as

$$\lim_{\beta\to\infty} \rho(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}'}) = \frac{1}{1+\gamma}\frac{|\mathcal{S}'|}{NT} = \frac{m}{1+\gamma} \tag{3.50}$$

Now consider the condition number κ of the CGM coefficient matrix. In the strong filter limit, this is given by

$$\begin{aligned}\lim_{\beta\to\infty} \quad \kappa\left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I}\right) &= \kappa\left(\mathbf{\Delta}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{\Delta} + \gamma\mathbf{I}\right)\\ &= \kappa\left(\begin{bmatrix}\mathbf{u}_1^\top\\ \mathbf{0}^\top\\ \vdots\\ \mathbf{0}^\top\end{bmatrix}\mathbf{D}_{\mathbf{S}}\begin{bmatrix}\mathbf{u}_1, \mathbf{0}, \dots, \mathbf{0}\end{bmatrix} + \gamma\mathbf{I}\right)\\ &= \kappa\left(\frac{1}{NT}\begin{bmatrix}|\mathcal{S}| & 0 & 0 & \dots\\ 0 & 0 & 0 & \\ \vdots & & & \ddots\end{bmatrix} + \gamma\mathbf{I}\right)\\ &= \frac{1-m+\gamma}{\gamma}\end{aligned} \tag{3.51}$$

Given eqs. (3.44) and (3.45), we can write the scaling rate for the number of iterations in the SIM and CGM respectively.

$$\lim_{\beta\to\infty} n_{\mathrm{SIM}} \propto \frac{1}{\log(1+\gamma) - \log m}, \quad \text{and} \quad \lim_{\beta\to\infty} n_{\mathrm{CGM}} \propto \sqrt{\frac{1-m+\gamma}{\gamma}} \tag{3.52}$$

	n_{SIM}		n_{CGM}	
	All γ	Small γ	All γ	Small γ
$\beta \to 0$	$\dfrac{1}{\log(1+\gamma)}$	γ^{-1}	$\sqrt{\dfrac{1+\gamma}{\gamma}}$	$\gamma^{-1/2}$
$\beta \to \infty$	$\dfrac{1}{\log(1+\gamma) - \log m}$	$-\dfrac{1}{\log m}$	$\sqrt{\dfrac{1-m+\gamma}{\gamma}}$	$\left(\dfrac{\gamma}{1-m}\right)^{-1/2}$

TABLE 3.5: The scaling behaviour of the number of steps required for convergence is shown as a function of γ and m. The upper row gives the behaviour in the limit of a weak filter, and the lower row gives the behaviour in the limit of a strong filter. We also show the dominant term in the taylor expansion about $\gamma = 0$ ("small γ" columns) which give a clearer picture of the asymptotic behaviour as $\gamma \to 0$.

A key feature of these expressions is that increasing the fraction of missing data m will increase n_{SIM}, whereas decrease n_{CGM} Note also that, in the case of a strong filter, the number of iterations required for convergence of the CGM, n_{CGM}, still goes to infinity as $\gamma \to 0$. However, this behaviour is no longer present for n_{SIM}, which tends towards a constant value of $-1/\log m$. Taking a Taylor series expansion of both expressions about $\gamma = 0$ demonstrates the asymptotic behaviour in terms of γ more clearly.

$$\lim_{\beta\to\infty} n_{\text{SIM}} \propto -\frac{1}{\log m} + O(\gamma), \qquad \text{and} \qquad \lim_{\beta\to\infty} n_{\text{CGM}} \propto \left(\frac{\gamma}{1-m}\right)^{-1/2} + O\left(\gamma^{1/2}\right)$$

In particular, at small γ, the CGM still runs with complexity proportional to $\gamma^{-1/2}$ whereas the SIM does not involve γ to a negative power at all. Note that m cannot scale arbitrarily close to zero or one, since it will surely be between $1/NT$ and $1 - 1/NT$.

3.3.3 Practical implications and method selection

In the preceding two sections, we have derived several formulae that characterise the convergence behaviour of both the CGM and the SIM in the limiting case of a strong and weak filter, where $\beta \to \infty$ and $\beta \to 0$ respectively. For the sake of clarity, we have consolidated the critical expressions in table 3.5. In this section, we examine these expressions more closely and distil out the key features that are relevant for implementing these methods in practice.

First consider the value of γ, which enters the model via eq. (3.13), and acts as a strictly positive regularisation parameter. For both the SIM and CGM, smaller values of γ

will universally increase the total number of iterations. This can be proven by taking the partial derivative of each expression with respect to γ, and demonstrating that the resulting expressions are negative for all valid values of γ and m. For completeness, we perform this explicitly in the proof of theorem A.4. Furthermore, the number of iterations will tend towards infinity as $\gamma \to 0$ in all cases except an asymptotically strong filter with the SIM. When the filter is weak, we should expect that the number of iterations grows faster for the SIM than the CGM as γ approaches zero. This can be seen from the Taylor series expansion about $\gamma = 0$, which has a dominant term of γ^{-1} for the SIM and $\gamma^{-1/2}$ for the CGM. When γ is large we should expect fast convergence for both the SIM and CGM regardless of the value of the other hyperparameters.

Consider now the parameter β, which characterises the strength of the graph filter, and also enters the model in the prior for $\mathbf{F}$ in eq. (3.13). For both the SIM and CGM, higher values of β, which more aggressively filter out high-frequency spectral components, are associated with faster convergence. In other words, like γ, increasing β will decrease the number of iterations required to reach a fixed termination condition. This is evidently true since the expressions for n_{SIM} and n_{CGM} are lower when $\beta \to \infty$ than they are when $\beta \to 0$, for any valid value of $0 \leq m \leq 1$. It is also reasonable to expect that the number of iterations will decrease monotonically as β is increased, however, we provide no formal proof of this. In contrast to the behaviour of γ, however, these two bounds are still both finite meaning that, the number of steps required for convergence remains finite as $\beta \to 0$.

Whilst the convergence rates of both the SIM and CGM have the same directional response to changes in γ and β, they display the opposite behaviour when m is varied. In particular, a higher proportion of missing value in $\mathbf{Y}$, corresponding to an increasing value of m, causes the number of iterations to rise for the SIM but fall for the CGM, at least in the limit of a strong filter as $\beta \to \infty$. This can be shown explicitly by taking the partial derivative of n_{SIM} and n_{CGM} with respect to m and demonstrating that it is universally positive for the former and negative for the latter (which we show in theorem A.5). While m has no effect for either the SIM or CGM in the limit as $\beta \to 0$, we can reasonably expect that it will have *some* effect for intermediate values of β. This insight is especially important since, unlike γ and β, m is a feature of the data rather than an adjustable hyperparameter.

3.3.4 Experimental validation

In order to verify aspects of this theoretically predicted behaviour, we ran several experiments using synthetic data. In particular, we generated two random fully connected

graphs, each with 50 nodes, and take their Cartesian product to create a product graph with 2500 nodes. Next, we generated a random 50×50 matrix of i.i.d. Gaussian noise with unit variance, and chose a fraction m to be removed uniformly at random to generate an observed graph signal $\mathbf{Y} \in \mathbb{R}^{50\times 50}$ and corresponding binary sensing matrix $\mathbf{S} \in \{0,1\}^{50\times 50}$. Next, we set a prior for the underlying smooth signal $\mathbf{F}$ with precision γ using an isotropic diffusion filter (see **??**) with parameter β. We then solved the graph signal reconstruction problem using both the SIM and CGM and counted the number of iterations required to reach a termination condition. Specifically, for the SIM, we terminate when $\Delta\mathbf{F}$ has a root mean square value of 10^{-8}, and, for the CGM, when the residual has a root mean square value of 10^{-5}, which we empirically determined resulted in similar final precision. In the following experiments, we completed this procedure for various values of m, β and γ, and compared the empirical number of iterations required for convergence against the theoretical predictions determined in section 3.3.

3.3.4.1 Experiment 1: testing the strong and weak filter limits

In the first experiment, we fix β at zero and ∞, and measure the number of iterations required for convergence over a range of values of m and γ. Since these correspond to the weak and strong filter limits respectively, we expect that the convergence rate should be bounded by a function proportional to those given in table 3.5. First, we set $\beta = 0$. Note that, in this case, m has no effect on the convergence rate of the SIM or CGM (which we also find empirically to be the case). We therefore varied γ alone in 50 logarithmically spaced increments from 10^{-4} to 10^{2}. The results are shown in fig. 3.7. As is visible, the functions broadly follow the expected convergence rate given by the theoretical prediction, performing slightly better in practice in both cases. Note that this is the expected behaviour, since the limits give a *worst case* scaling rate. In both cases, we also scaled the theoretical prediction (corresponding to a vertical shift in the log-log plot) to fit the experimental data. This is also valid, since the theoretical predictions give a function proportional to the number of iterations rather than the number of iterations itself. For the SIM, this factor was approximately 10, and for the CGM this factor was approximately 4.5.

For the second part of experiment 1, we set $\beta = \infty$, corresponding to the strong filter limit. In this case, we expect the number of iterations to be a function of both m and γ as specified in table 3.5. Therefore, we varied γ in 50 logarithmically spaced increments from 10^{-6} to 10^{2} and m in 50 linearly spaced increments from 0.01 to 0.99. For each unique pair of m and γ, we then counted the number of iterations required for convergence and compared this to the theoretical predictions. The results are shown in fig. 3.8. Again, the empirical results broadly follow the theoretical functions which,

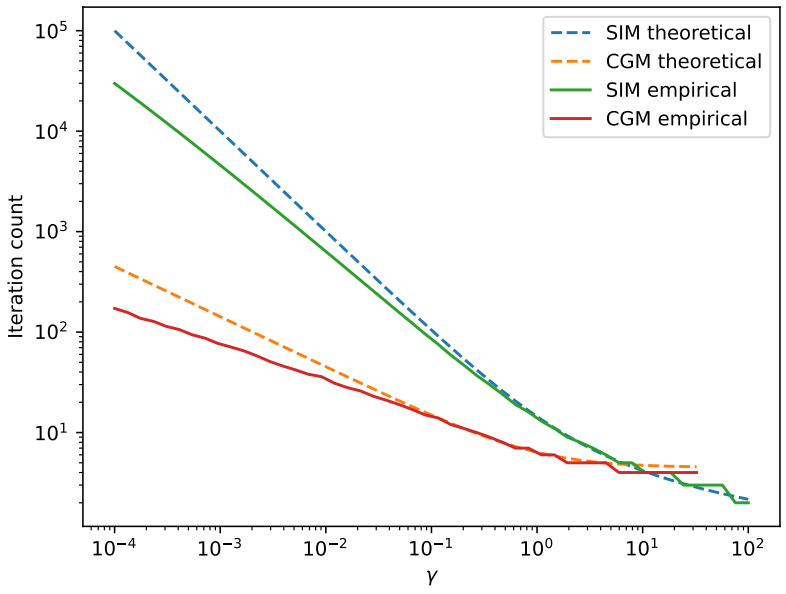


FIGURE 3.7: The number of iterations required for convergence in the weak filter limit for the SIM an CGM both theoretically and empirically, as a function of γ.

as before, are scaled to fit the data. As with the weak filter limit, we again see that empirical scaling rate is slightly better than the worst case theoretical prediction.

3.3.4.2 Experiment 2: Testing intermediate values of β

In the second experiment, we test the number of iterations required for convergence for intermediate values of β. Note that the analysis carried out in section 3.3 applies only for extremal values of β, meaning the intermediate behaviour is not clearly defined, and will depend on the filter chosen in practice. This was carried out for three values of m: 0.05, 0.5 and 0.95, representing low, medium and high prevalence of missing data respectively. For each value of m, we compute the solution using both algorithms over a grid of 50 β values and 50 γ values which were logarithmically spaced between 10^{-1}-10^2 and 10^{-4}-10^1 respectively. The results are shown in fig. 3.9.

This experiment validates several key aspects of the theoretical convergence analysis. Firstly, notice that the basic behaviour that the number of iterations rises in all cases as β and γ get closer to zero. Furthermore, in all cases, regardless of the value of m or β, convergence is very fast when γ is large. We also observe the predicted behaviour that the number of iterations plateaus as a function of decreasing γ when β is large for the SIM while it continues to increase for the CGM. However, for this data, the CGM level seems to remain below that of the SIM in this limit.

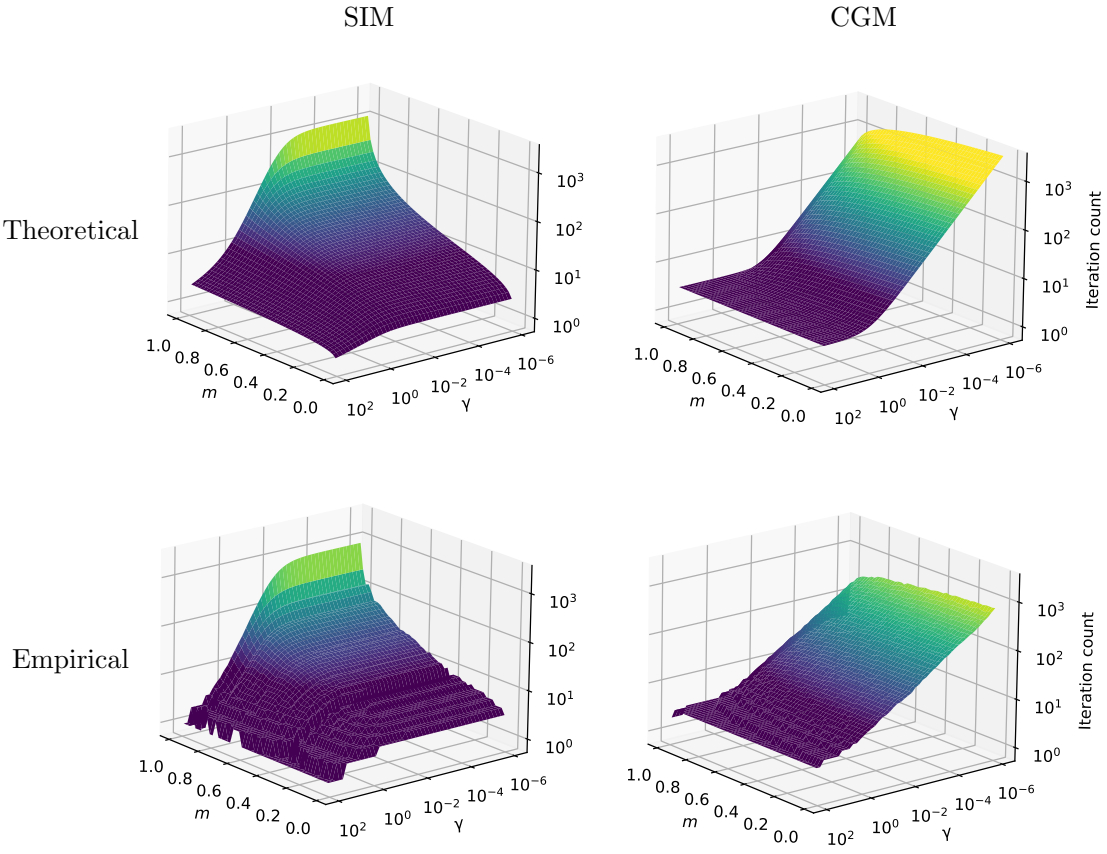


FIGURE 3.8: The number of iterations required for convergence is shown both theoretically and empirically for the SIM and CGM in the strong filter limit, where $\beta = \infty$. In this case, each plot is a function of both m and γ. The colormap corresponds to vertical height and is nomalised across each plot to aid comparison.

One key result of this experiment that aligns with the theoretical prediction is that convergence is accelerated as m rises for the CGM but convergence slows as m rises for the SIM. In particular, for this dataset, $n_{\text{SIM}} \leq n_{\text{CGM}}$ for all values of γ and β when $m = 0.05$, but $n_{\text{CGM}} \leq n_{\text{SIMM}}$ for all values of γ and β when $m = 0.95$. This is particularly impactful as it strongly indicates the CGM should be preferable when data is sparsely observed and the SIM should be preferable when the data is densely observed.

3.4 Conclusions

In this chapter we have introduced a Bayesian model for the reconstruction of signals defined over the nodes of a Cartesian product graph. In particular, we show that the posterior mean of the smooth underlying signal, $\mathbf{F}$, is obtained by solving a linear system of size $NT \times NT$, given in eq. (3.17). While a naive approach to computing this would have a time complexity of $O(N^3T^3)$, we can utilise the classic methods of matrix splitting

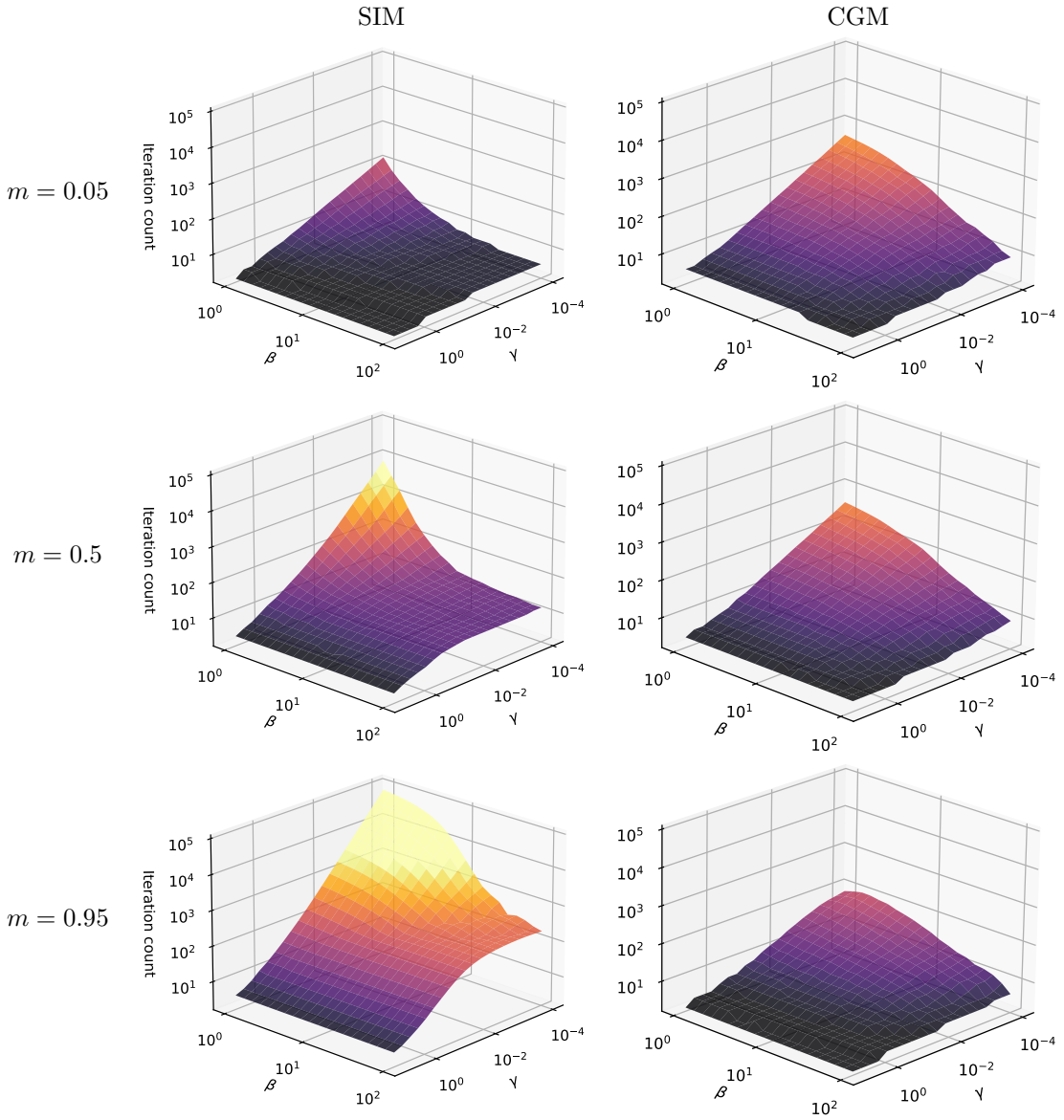


FIGURE 3.9: Six 3D surfaces are shown representing the number of iterations required for convergence for three values of m for both the SIM and CGM algorithms. In each plot, the x-axis represents the filter parameter β, and the y-axis represents the regularisation parameter γ. The colormap, which is normalsied equally across all plots, tracks the vertical height.

and conjugate gradients, customised for the context of graph signal reconstruction, to compute a solution iteratively. When used in conjunction with the properties of the Kronecker product, the linear system can be solved with complexity $O(N^2T + NT^2)$ per iterative step. Furthermore, we show that this can be reduced to $O(N^2T + NT \log T)$ when considering time-vertex problems, and to $O(NT \log NT)$ when operating on a grid by making use of the Fast Cosine Transform (FCT).

The key output of this chapter is two algorithms, namely the Stationary Iterative Method

	Small γ			Medium γ			Large γ		
	S m	M m	L m	S m	M m	L m	S m	M m	L m
S β	SIM	CGM	CGM	SIM	CGM	CGM	SIM	CGM	CGM
M β	SIM	SIM	CGM	SIM	CGM	CGM	SIM	CGM	CGM
L β	SIM	SIM	CGM	SIM	SIM	CGM	SIM	CGM	CGM

TABLE 3.6: This table gives a rough rule of thumb for which iterative method should be chosen under various hyperparameter settings. S, M and L denotes small, medium and large. Note that small and large m mean close to zero or one respectively, small and large γ mean around 10^{-6} and around one respectively, and small and large β mean values which cause the filter function to approach the weak and strong filter limits respectively.

(SIM) and the Conjugate Gradient Method (CGM), tailored for the task of graph signal reconstruction. These were designed by making use of domain knowledge to produce a matrix splitting for the SIM and a preconditioner for the CGM that both guarantee bounded convergence. By analysing the spectral properties of the matrices involved in each algorithm, we provided a detailed analysis of the expected convergence rate in both cases. The important results are summarised in table 3.5, which gives a factor proportional to the number of iterations required for convergence in terms of the hyperparameters β, γ and m. Given these results, we have provided some rules-of-thumb for making a choice of iterative method in practice, which is summarised in table 3.6. This decision is of less importance when γ is large, as both methods converge quickly in this domain, however it is of particular significance when γ is small.

Chapter 4

Multivariate Regression Models for Time-Varying Graph Signals

In this chapter, we focus on time-series regression models for graph signals. Specifically, our interest lies in the analysis of repeated measurements of a graph signal, where each sample is associated with a set of explanatory variables. The primary objective is to construct a predictive model capable of estimating these signals as a function of the aforementioned explanatory variables. Although regression has traditionally garnered less attention within the Graph Signal Processing (GSP) community compared to reconstruction, this framework holds significant relevance for numerous real-world applications. Consequently, it warrants further exploration and in-depth examination.

In the subsequent discussion, we emphasise two distinct data scenarios that may emerge within the context of graph signal regression. In the first scenario, the explanatory variables are exogenous or 'global' with respect to the graph signals. In this setting, each graph signal $\mathbf{y}_t$ is accompanied by a feature vector $\mathbf{x}_t$, which is global in the sense that it is associated with the entire signal rather than a specific node. For example, consider predicting the quarterly earnings growth for a network of firms based on economic factors such as interest rates, inflation, and unemployment. A visual representation of the data present in such a scenario can be found in fig. 4.1.

In the second scenario under consideration, the explanatory variables are 'local' to the nodes of the graph signal. This implies that for each graph signal $\mathbf{y}_t$, every node possesses an individual vector of explanatory variables $\mathbf{x}_{nt}$. For instance, consider a network of air quality monitoring stations with the objective of predicting the concentration of a specific airborne pollutant. Each station may simultaneously track various local factors, including temperature, pressure, precipitation, and so on. Figure 4.2 offers a visual depiction of the data encountered in this particular scenario.

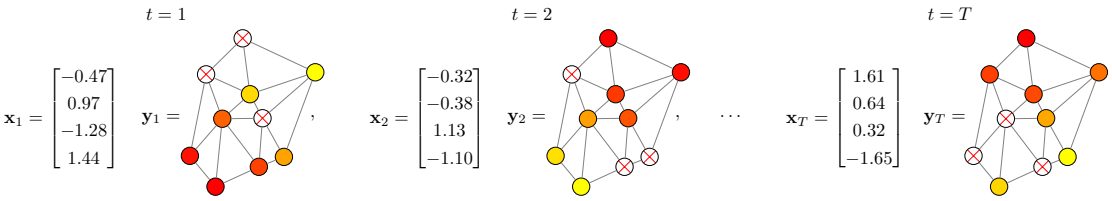


FIGURE 4.1: A visual representation of a time-series graph signal regression problem with exogenous variables. Here, there are T graph signals measured over a static graph, each with independent missing data (indicated in white with a red cross). Each signal $\mathbf{y}_t$ is accompanied by a unique vector of global explanatory variables $\mathbf{x}_t$.

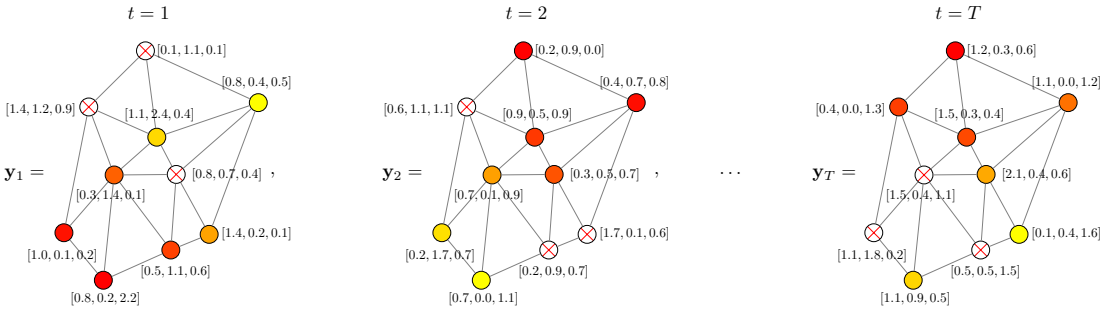


FIGURE 4.2: A visual representation of a time-series graph signal regression problem with local explanatory variables. As before, there are T graph signals measured over a static graph, each with independent missing data (indicated in white with a red cross). In this case, each signal $\mathbf{y}_t$ has a vector of explanatory variables $\mathbf{x}_{nt}$ for every node in the signal.

The core goal of this chapter is to derive some useful extensions to existing methods for graph signal regression in both the local and global variable context. In the case of exogenous explanatory variables, we focus on Kernel Graph Regression (KGR) [Elias et al., 2022, Venkitaraman et al., 2019] and the closely related topic of Gaussian Processes over Graphs (GPoG) [Venkitaraman et al., 2020]. A notable issue in practice with existing KGR-type models is the assumption that the input signals $\mathbf{y}_t$ are always fully observed, meaning there is no missing data. As discussed in chapter 3, a prevalent characteristic of graph signals in real-world applications is the high occurrence of missing data. This situation forces the end user to either eliminate entire rows (i.e., the complete time series for a specific node) if any elements in the series are not recorded, which may result in the loss of valuable topological information, or alternatively employ interpolation methods, which could prove inadequate for extended strings of missing data. Our proposed model accommodates fully general patterns of missing data in the input signals, enabling users to optimally utilise all available data.

For the scenario involving local explanatory variables, we enhance a model known as Regression with Network Cohesion (RNC) [Le and Li, 2022, Li et al., 2019] by incorporating several beneficial extensions. Specifically, the original RNC model was designed for a single graph signal rather than multiple repeated samples. Moreover, the original

specification did not account for the presence of missing data. Consequently, we advance the model in at least two significant directions.

4.1 Kernel Graph Regression with Unrestricted Missing Data Patterns

4.1.1 Model description

Consider a sequence of T real-valued graph signals measured over a static N-node graph, for which an arbitrary subset of the elements of each signal may be missing, including potentially the entire signal at certain time points. At each time t, there also exists a vector of variables $\mathbf{x}_t \in \mathbb{R}^M$ encompassing M distinct explanatory features. Each graph signal may have unique and arbitrary missing data, as indicated by a binary vector where ones represent successfully collected data and zeros signify missing data. Any absent data in $\mathbf{y}_t$ should be filled with zeros. Hence, the input data for this problem can be concisely described as follows.

$$\text{input data} = \left\{ \mathbf{X} \in \mathbb{R}^{T\times M},\ \ \mathbf{Y} \in \mathbb{R}^{N\times T},\ \ \mathbf{S} \in \{0,1\}^{N\times T},\ \ \mathbf{A} \in \mathbb{R}^{NT\times NT} \right\}$$

In the following, we assume that the explanatory variables do not contain missing values. However, if any missing values are present, conventional methods such as those described in Little and Rubin [2019] can be employed to fill them. It should be noted that, with this model specification, there is no rigid distinction between in-sample and out-of-sample prediction. To indicate a particular value of $\mathbf{x}_t$ for which a comprehensive prediction should be generated, one can set the corresponding value of $\mathbf{y}_t = \mathbf{s}_t = \mathbf{0}$.

As in section 3.2.1, the graph signals can be stacked together into a matrix $\mathbf{Y}$ of shape $(N \times T)$. Note that this in contrast to the typical shape found in multivariate regression, which is most commonly $(T \times N)$ with the index referring to each sample varying first, however, we adopt the opposite convention here for the reasons outlined section 3.1.3.

Consider now a P-dimensional basis function representation of each explanatory vector $\mathbf{x}_t$, denoted as $\boldsymbol{\phi}(\mathbf{x}_t) \in \mathbb{R}^P$.

$$\boldsymbol{\phi}(\mathbf{x}_t) = \begin{bmatrix} \phi_1(\mathbf{x}_t) & \phi_2(\mathbf{x}_t) & \dots & \phi_P(\mathbf{x}_t) \end{bmatrix}^\top \tag{4.1}$$

Let us assume that each element of $\mathbf{y}_t$ can be modelled as a noisy linear combination of these basis functions, which may or may not have been observed. This is summarised in the following statistical model.

$$\mathbf{y}_t = \mathbf{s}_t \circ \big(\mathbf{W}\boldsymbol{\phi}(\mathbf{x}_t) + \mathbf{e}_t\big), \quad \text{for } t = 1, 2, ..., T \tag{4.2}$$

Here, $\mathbf{W} \in \mathbb{R}^{N \times P}$ represents the model coefficients defining the linear combination, and $\mathbf{e}_t \in \mathbb{R}^N$ is a vector of i.i.d. Gaussian noise with zero mean and unit variance. The basis function vectors can be horizontally stacked together to form a design matrix $\boldsymbol{\Phi}$.

$$\boldsymbol{\Phi} \in \mathbb{R}^{P \times T} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \dots & \phi_1(\mathbf{x}_T) \\ \phi_2(\mathbf{x}_1) & \phi_2(\mathbf{x}_2) & \dots & \phi_2(\mathbf{x}_T) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_P(\mathbf{x}_1) & \phi_P(\mathbf{x}_2) & \dots & \phi_P(\mathbf{x}_T) \end{bmatrix} \tag{4.3}$$

Therefore, the statistical model can be written in matrix form as

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{W}\boldsymbol{\Phi} + \mathbf{E}\big) \tag{4.4}$$

where $\mathbf{S} \in \{0, 1\}^{N \times T}$ is the binary sensing matrix as defined in section 3.2.1, which is also each $\mathbf{s}_t$ stacked horizontally, and $\mathbf{E} \in \mathbb{R}^{N \times T}$ is a matrix of i.i.d. Gaussian noise with zero mean and unit variance. This implies that the probability distribution for $\text{vec}\,(\mathbf{Y}) \mid \mathbf{W}$ is given by

$$\text{vec}\,(\mathbf{Y}) \mid \mathbf{W} \sim \mathcal{N}\big(\text{vec}\,(\mathbf{S} \circ (\mathbf{W}\boldsymbol{\Phi}))\,,\ \mathbf{I}_{NT}\big) \tag{4.5}$$

To determine the posterior distribution of the model coefficients $\mathbf{W}$, it is necessary to specify a prior that reflects the assumption that predicted signals are expected to be smooth with respect to the graph's topology. We assert that an appropriate prior for $\mathbf{W}$ is given by

$$\text{vec}\,(\mathbf{W}) \sim \mathcal{N}\left(\mathbf{0},\ \gamma^{-1}\mathbf{I}_P \otimes \mathbf{H}_N^2\right) \tag{4.6}$$

Here, $\mathbf{H}_N \in \mathbb{R}^N$ represents a graph filter constructed in accordance with one of the univariate filter functions defined in **??**, while γ serves as a hyperparameter denoting the prior precision. The rationale for this prior, as explicated in Venkitaraman et al.

[2020], is as follows. Consider a random graph signal formulated as $\mathbf{W}\boldsymbol{\phi}$, where both $\mathbf{W}$ and $\boldsymbol{\phi}$ have Gaussian i.i.d. entries with zero mean and unit variance. The probability distribution of their product will also be an i.i.d. multivariate Gaussian with zero mean. By applying a graph filter $\mathbf{H}_N$ to smooth this signal, the resulting signal will exhibit the same probability distribution as $\mathbf{W}\boldsymbol{\phi}$, assuming $\mathbf{W}$ was drawn from the distribution specified in eq. (4.6). Consequently, this prior serves to enhance the likelihood of smooth signals.

Consider now a transformed variable $\mathbf{F}$ defined by $\mathbf{F} = \mathbf{W}\boldsymbol{\Phi}$. Given that $\mathbf{W}$ has a prior distribution given in eq. (4.6), we can ask what the implied prior for $\mathbf{F} \,|\, \boldsymbol{\Phi}$ is. Clearly, since the expected value of $\mathbf{W}$ is zero for all entries, the expected value of $\mathbf{F}$ should also be zero for all entries regardless of the value of $\boldsymbol{\Phi}$. The covariance of $\mathbf{F}$ can also be computed easily.

$$\begin{aligned}
\mathrm{Cov}\big[\mathrm{vec}\,(\mathbf{F})\,\big] &= \mathrm{Cov}\big[\mathrm{vec}\,(\mathbf{W}\boldsymbol{\Phi})\,\big] \\
&= \mathrm{Cov}\Big[\big(\boldsymbol{\Phi}^\top \otimes \mathbf{I}\big)\,\mathrm{vec}\,(\mathbf{W})\Big] \\
&= \big(\boldsymbol{\Phi}^\top \otimes \mathbf{I}\big)\,\mathrm{Cov}\big[\mathrm{vec}\,(\mathbf{W})\,\big]\,\big(\boldsymbol{\Phi} \otimes \mathbf{I}\big) \\
&= \gamma^{-1}\big(\boldsymbol{\Phi}^\top \otimes \mathbf{I}\big)\,\big(\mathbf{I} \otimes \mathbf{H}_N^2\big)\,\big(\boldsymbol{\Phi} \otimes \mathbf{I}\big) \\
&= \gamma^{-1}\big(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\big) \otimes \mathbf{H}_N^2
\end{aligned}$$

Therefore, we can rewrite the model in terms of the new transformed variable as follows.

$$\mathrm{vec}\,(\mathbf{Y})\;|\,\mathbf{F} \sim \mathcal{N}\big(\,\mathrm{vec}\,(\mathbf{S} \circ \mathbf{F})\,,\,\mathbf{I}_{NT}\,\big) \tag{4.7}$$

$$\mathrm{vec}\,(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\;\gamma^{-1}\big(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\big) \otimes \mathbf{H}_N^2\Big) \tag{4.8}$$

The final step to convert this into a non-parametric regression model is to apply the 'kernel-trick' [Scholkopf and Smola, 2018]. Consider the PSD matrix $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$. Entry (i, j) will be the inner product between the basis function expansion of $\mathbf{x}_i$ and $\mathbf{x}_j$.

$$\boldsymbol{\Phi}^\top\boldsymbol{\Phi} \in \mathbb{R}^{T\times T} = \begin{bmatrix}
\boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\
\boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\
\vdots & \vdots & \ddots & \vdots \\
\boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_T)
\end{bmatrix} \tag{4.9}$$

The trick is to characterise each inner product in terms of a predefined function such that $\boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. This means that instead of mapping the explanatory variables via $\boldsymbol{\phi}$ and computing the inner product directly, it is done in a single operation, leaving the mapping implicit. This means we can replace the matrix $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ with a so-called kernel (or *Gram*) matrix $\mathbf{K} \in \mathbb{R}^{T \times T}$, which has entries defined by

$$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{4.10}$$

where $\kappa(\cdot, \cdot)$ is any valid Mercer kernel [Rasmussen and Williams, 2005]. A common example is the Gaussian kernel given below.

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{4.11}$$

Therefore, the prior distribution over $\mathbf{F}$ is given in terms of $\mathbf{K}$ by

$$\text{vec}\,(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\ \gamma^{-1}\mathbf{K} \otimes \mathbf{H}_N^2\Big) \tag{4.12}$$

In a similar manner to section 3.2.1, the posterior distribution for $\mathbf{F}|\mathbf{Y}$ is given by

$$\text{vec}\,(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\left(\bar{\mathbf{P}}^{-1}\text{vec}\,(\mathbf{Y}),\ \bar{\mathbf{P}}^{-1}\right) \tag{4.13}$$

where $\bar{\mathbf{P}}$ is the posterior precision matrix for $\text{vec}\,(\mathbf{F})$, given by

$$\bar{\mathbf{P}} = \mathbf{D}_\mathbf{S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} \tag{4.14}$$

As before, $\mathbf{D}_\mathbf{S} = \text{diag}\,(\text{vec}\,(\mathbf{S}))$.

4.1.2 Relation to graph signal reconstruction

Despite arising from a different set of modelling assumptions, Kernel Graph Regression as described in section 4.1.1 bares a stark mathematical resemblance to Graph Signal Reconstruction. The only key difference between the specification of these two models is that in GSR, the prior distribution over $\text{vec}\,(\mathbf{F})$ has covariance $\mathbf{H}^2$ [see eq. (3.13)], and in KGR it has covariance $\mathbf{K} \otimes \mathbf{H}_N^2$ [see eq. (4.12)]. Intuitively speaking, the prior used in the GSR model encodes the belief that the underlying graph signal should be smooth with respect to the topology of the 2D Cartesian product graph. On the other hand,

the prior used in the KGR model encodes the belief that the predicted signal should be smooth with respect to the topology of the 1D graph and the explanatory variables, i.e. graph signals $\mathbf{y}_i$ and $\mathbf{y}_j$ are expected to be similar if the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar.

As a result of the differences in their respective priors, the posterior precision matrix for $\mathrm{vec}\,(\mathbf{F})$ is given by $\mathbf{D_S} + \gamma\mathbf{H}^{-2}$ in the case of GSR and $\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}$ in the case of KGR. In both cases, it is the sum of $\mathbf{D_S}$ and a Hermitian matrix. To make this correspondence clearer, we can expand the second term of each expression in terms of its respective eigendecomposition. For GSR, this is

$$\begin{aligned}\mathbf{D_S} + \gamma\mathbf{H}^{-2} &= \mathbf{D_S} + \gamma\big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\,\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{G}))^{-2}\,\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big)\\ &= \mathbf{D_S} + \mathbf{U}\,\mathbf{D}_{\mathbf{G}}^{-2}\,\mathbf{U}^\top\end{aligned}$$

where $\mathbf{U} = \mathbf{U}_T \otimes \mathbf{U}_N$, and $\mathbf{D_G} = \mathrm{diag}\,(\mathrm{vec}\,(\mathbf{G}))$. For KGR this is

$$\begin{aligned}\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} &= \mathbf{D_S} + \gamma\big(\mathbf{V} \otimes \mathbf{U}_N\big)\,\mathrm{diag}\,\big(\mathrm{vec}\,\big(\bar{\mathbf{G}}\big)\big)^{-2}\,\big(\mathbf{V}^\top \otimes \mathbf{U}_N^\top\big)\\ &= \mathbf{D_S} + \bar{\mathbf{U}}\,\mathbf{D}_{\bar{\mathbf{G}}}^{-2}\,\bar{\mathbf{U}}^\top\end{aligned}$$

where $\mathbf{K}$ and $\mathbf{H}_N$ have eigendecompositions given by

$$\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top, \quad \text{and} \quad \mathbf{H}_N = \mathbf{U}_N\,g\,(\boldsymbol{\Lambda}_N)\,\mathbf{U}_N^\top \tag{4.15}$$

with $\boldsymbol{\Lambda}_K = \mathrm{diag}\left(\left[\lambda_1^{(K)},\ \lambda_2^{(K)},\ \ldots,\ \lambda_T^{(K)}\right]\right)$, $\bar{\mathbf{U}} = \mathbf{V} \otimes \mathbf{U}_N$ and $\mathbf{D}_{\bar{\mathbf{G}}} = \mathrm{diag}\,\big(\mathrm{vec}\,\big(\bar{\mathbf{G}}\big)\big)$. $\bar{\mathbf{G}} \in \mathbb{R}^{N\times T}$ has elements given by

$$\bar{\mathbf{G}}_{nt} = g\left(\lambda_n^{(N)}\right)\sqrt{\lambda_t^{(K)}} \quad \Longleftrightarrow \quad \mathbf{D}_{\bar{\mathbf{G}}} = \boldsymbol{\Lambda}_K^{1/2} \otimes g\,(\boldsymbol{\Lambda}_N) \tag{4.16}$$

Therefore, KGR is algebraically equivalent to GSR under the following change of variables:

$$\mathbf{U} \to \bar{\mathbf{U}}, \quad \text{and} \quad \mathbf{G} \to \bar{\mathbf{G}}$$

As such, the iterative algorithms developed in chapter 3 can be largely recycled with only minor modifications. However, it is important to bear in mind that the value of

the maximum entry in $\mathbf{G}$ is one, whereas the maximum value in $\bar{\mathbf{G}}$ is $\sqrt{\rho(\mathbf{K})}$. This has some implications for the SIM and CGM convergence rate, as explored in the following sections.

4.1.3 Solving for the posterior mean

We now briefly restate the SIM and CGM algorithms to highlight the small changes necessary to accommodate KGR with arbitrary missing data. The goal is to solve the following linear system

$$\operatorname{vec}(\mathbf{F}) = \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right)^{-1} \operatorname{vec}(\mathbf{Y}) \tag{4.17}$$

First, let us revisit the SIM. Recall that the strategy is to split the coefficient matrix into $\mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is easy to invert. In this case, we can put

$$\mathbf{M} = \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\mathbf{S}'}. \tag{4.18}$$

where, as before, $\mathbf{D}_{\mathbf{S}'} = \operatorname{diag}(\mathbf{1} - \operatorname{vec}(\mathbf{S}))$. Consider the matrix $\mathbf{M}^{-1}$.

$$\begin{aligned}
\mathbf{M}^{-1} &= \left(\gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\gamma \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{G}}}^{-2}\, \bar{\mathbf{U}}^{\top} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right) \bar{\mathbf{U}}^{\top}\right)^{-1} \\
&= \bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right)^{-1} \bar{\mathbf{U}}^{\top} \\
&= \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{J}}}\, \bar{\mathbf{U}}^{\top}
\end{aligned} \tag{4.19}$$

where $\mathbf{D}_{\bar{\mathbf{J}}} = \operatorname{diag}\left(\operatorname{vec}\left(\bar{\mathbf{J}}\right)\right)$, and $\bar{\mathbf{J}} \in \mathbb{R}^{N \times T}$ has entries given by

$$\bar{\mathbf{J}}_{nt} = \frac{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right)}{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right) + \gamma} = \frac{\bar{\mathbf{G}}_{nt}^2}{\bar{\mathbf{G}}_{nt}^2 + \gamma} \tag{4.20}$$

The SIM algorithm then proceeds in much the same way as described in section 3.2.2. As before, it is clear to see that convergence will be achieved, since the spectral radius of

$\mathbf{M}^{-1}\mathbf{N}$ will surely be less than one for any positive γ. In this case, the update formula is given by

$$\mathbf{F}_{k+1} = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k\right) \mathbf{V}\right)\right) \mathbf{V}^\top + \mathbf{F}_0 \tag{4.21}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{V}\right)\right) \mathbf{V}^\top \tag{4.22}$$

Note that each update step can be performed with $O(N^2T + NT^2)$ multiplications.

Next, let us return to the CGM. Recall that the strategy for solving the linear system in eq. (4.17) is to utilise a symmetric a preconditioner $\boldsymbol{\Psi}$ such that the new system is given by

$$\left(\bar{\boldsymbol{\Psi}}^\top \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right) \bar{\boldsymbol{\Psi}}\right)\left(\bar{\boldsymbol{\Psi}}^{-1} \operatorname{vec}\left(\mathbf{F}\right)\right) = \bar{\boldsymbol{\Psi}}^\top \operatorname{vec}\left(\mathbf{Y}\right), \tag{4.23}$$

$\bar{\boldsymbol{\Psi}}$ should be chosen such that new coefficient matrix $\bar{\boldsymbol{\Psi}}^\top \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right) \bar{\boldsymbol{\Psi}}$ has a reduced condition number. In the present case, we assert that an effective preconditioner is given by

$$\bar{\boldsymbol{\Psi}} = \left(\mathbf{V} \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}\left(\bar{\mathbf{G}}\right)\right) = \bar{\mathbf{U}} \mathbf{D}_{\bar{\mathbf{G}}}. \tag{4.24}$$

This preconditioner transforms the coefficient matrix into

$$\begin{aligned}
\bar{\boldsymbol{\Psi}}^\top \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right) \bar{\boldsymbol{\Psi}} &= \mathbf{D}_{\bar{\mathbf{G}}} \left(\mathbf{V}^\top \otimes \mathbf{U}_N^\top\right) \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right) \left(\mathbf{V} \otimes \mathbf{U}_N\right) \mathbf{D}_{\bar{\mathbf{G}}} \\
&= \mathbf{D}_{\bar{\mathbf{G}}} \left(\mathbf{V}^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_{\mathbf{S}} \left(\mathbf{V} \otimes \mathbf{U}_N\right) \mathbf{D}_{\bar{\mathbf{G}}} + \gamma \mathbf{I} \\
&= \mathbf{D}_{\bar{\mathbf{G}}} \bar{\mathbf{U}}\, \mathbf{D}_{\mathbf{S}}\, \bar{\mathbf{U}}^\top \mathbf{D}_{\bar{\mathbf{G}}} + \gamma \mathbf{I}
\end{aligned}$$

Note that we can efficiently compute the action of multiplying this preconditioned matrix onto an arbitrary vector $\operatorname{vec}\left(\mathbf{R}\right) \in \mathbb{R}^{NT}$ as follows.

$$\operatorname{mat}\left(\left(\mathbf{D}_{\bar{\mathbf{G}}} \bar{\mathbf{U}}\, \mathbf{D}_{\mathbf{S}}\, \bar{\mathbf{U}}^\top \mathbf{D}_{\bar{\mathbf{G}}} + \gamma \mathbf{I}\right) \operatorname{vec}\left(\mathbf{R}\right)\right) = \gamma \mathbf{R} + \bar{\mathbf{G}} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S} \circ \left(\mathbf{U}_N (\bar{\mathbf{G}} \circ \mathbf{R}) \mathbf{V}^\top\right)\right) \mathbf{V}\right)$$

As with the SIM, this action can be performed with $O(N^2T + NT^2)$ multiplications.

4.2 Regression with Network Cohesion

4.2.1 Model description

In this model, we consider a sequence of T real-valued signals, which are regularly sampled over a static N-node graph, and may contain arbitrary missing values. As with KGR and GSR, this is represented using an $(N \times T)$ matrix $\mathbf{Y}$, with missing values set to zero and further clarified with the binary sensing matrix $\mathbf{S} \in \{0,1\}^{N \times T}$ which holds zeros at entries where the corresponding element of Y is missing data, and ones elsewhere. In this model each node, n, at each time instant, t, under consideration, has an associated length-M vector of explanatory variables $\mathbf{x}_{nt} \in \mathbb{R}^M$. The objective is to make use of both the node-level explanatory variables and the network topology to estimate the missing values in the graph signal. A visual depiction of the data available in this kind of scenario is given in fig. 4.2.

$$\text{input data} = \left\{ \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times M},\ \ \mathbf{Y} \in \mathbb{R}^{N \times T},\ \ \mathbf{S} \in \{0,1\}^{N \times T},\ \ \mathbf{L} \in \mathbb{R}^{NT \times NT} \right\}$$

Here, we have introduced the notation $\boldsymbol{\mathcal{X}}$, which is an order-3 tensor, which has three independent axes signifying node number n, time instant t, and covariate number m, respectively. In the following sections, we index/slice this tensor of explanatory variables, $\boldsymbol{\mathcal{X}}$, in various ways, therefore, it is beneficial to establish some notational standards for this purpose. In general, we adhere to the conventions outlined in Kolda and Bader [2009], which are also similar to those found in Kiers [2000].

To refer to an individual element of an order-3 tensor (i.e. a scalar entry), we use the notation $\boldsymbol{\mathcal{X}}_{ntm}$. The object generated by fixing two indices and letting a third vary (resulting in a vector) is referred to in this context as a fibre. Using tensor notation, the vector of covariates $\mathbf{x}_{nt} \in \mathbb{R}^M$, which exists at every node, at every time point, can alternatively be written as $\boldsymbol{\mathcal{X}}_{nt:}$. A two-dimensional section of a tensor (i.e. a matrix), where two indices can vary and a single index is fixed, is known as a slice. For example, we could consider the slice given by a specific covariate measured across the whole graph, at every time instant, for which we would use the notation $\boldsymbol{\mathcal{X}}_{::m}$. Figure 4.3 gives a visual representation of several ways of indexing/slicing an order-3 tensor.

For the purposes of this section, we also define the special matrix $\mathbf{X}$, which is constructed by horizontally stacking the M vectorized slices $\boldsymbol{\mathcal{X}}_{::m}$, resulting in an object of shape $(NT \times M)$.

$$\mathbf{X} \in \mathbb{R}^{NT \times M} = \begin{bmatrix} \text{vec}\left(\boldsymbol{\mathcal{X}}_{::1}\right) & \text{vec}\left(\boldsymbol{\mathcal{X}}_{::2}\right) & \dots & \text{vec}\left(\boldsymbol{\mathcal{X}}_{::M}\right) \end{bmatrix} \tag{4.25}$$

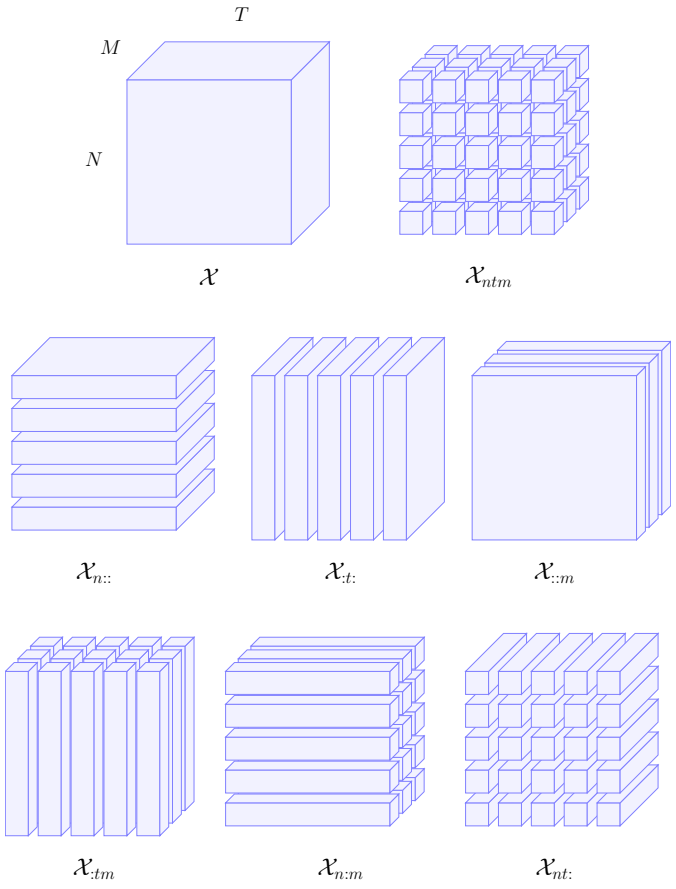


FIGURE 4.3: A visual representation of a several different ways of indexing into an order-3 tensor, with the corresponding notation indicated.

In this section we consider several extensions to a model known as Regression with Network Cohesion (RNC) [Li et al., 2019]. In our version of the model, we assume that every element of the observed graph signal $\mathbf{Y}$ can be represented as the sum of an intercept term, a linear combination of the node-level covariates, and some Gaussian noise. However, the intercept term is flexible in the sense that each node/time can have a distinct value. To avoid underspecification, the model postulates that the intercepts are smooth with respect to the graph's topology. This is represented as follows.

$$\text{vec}\,(\mathbf{Y}) = \text{vec}\,(\mathbf{S}) \circ \big(\text{vec}\,(\mathbf{C}) + \mathbf{X}\mathbf{w} + \text{vec}\,(\mathbf{E})\big) \tag{4.26}$$

In this context, $\mathbf{C} \in \mathbb{R}^{N \times T}$ represents the flexible intercept term, $\mathbf{w} \in \mathbb{R}^{M}$ the vector of regression coefficients, and $\mathbf{E} \in \mathbb{R}^{N \times T}$ a matrix of independent and identically distributed Gaussian noise with unit variance. The key assumption of the RNC model is that the intercept term $\mathbf{C}$ is smooth with respect to the topology of the entire $T - V$ Cartesian product graph. Furthermore, the above expression can be restated in terms

of a single model coefficient vector $\boldsymbol{\theta}$, by making use of the definition of $\mathbf{X} \in \mathbb{R}^{NT \times M}$ given in eq. (4.25), as follows.

$$\operatorname{vec}(\mathbf{Y}) = \operatorname{vec}(\mathbf{S}) \circ \Big(\begin{bmatrix} \mathbf{I}_{NT} & \mathbf{X} \end{bmatrix} \boldsymbol{\theta} + \operatorname{vec}(\mathbf{E}) \Big) \tag{4.27}$$

where $\begin{bmatrix} \mathbf{I}_{NT} & \mathbf{X} \end{bmatrix} \in \mathbb{R}^{NT \times (NT+M)}$ is a block matrix consisting of the $(NT \times NT)$ identity matrix alongside $\mathbf{X}$, and $\boldsymbol{\theta}$ is the block vector consisting of $\operatorname{vec}(\mathbf{C})$ stacked on top of $\mathbf{w}$, that is,

$$\boldsymbol{\theta} \in \mathbb{R}^{NT+M} = \begin{bmatrix} \operatorname{vec}(\mathbf{C}) \\ \mathbf{w} \end{bmatrix} \tag{4.28}$$

Given eq. (4.27), we can write the probability distribution for $\mathbf{Y} \,|\, \boldsymbol{\theta}$ as follows.

$$\operatorname{vec}(\mathbf{Y}) \,|\, \boldsymbol{\theta} \sim \mathcal{N}\Big(\operatorname{vec}(\mathbf{S}) \circ \Big(\begin{bmatrix} \mathbf{I}_{NT} & \mathbf{X} \end{bmatrix} \boldsymbol{\theta} \Big),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S})) \Big) \tag{4.29}$$

In order to complete the specification of the Bayesian model, we need a prior distribution for $\boldsymbol{\theta}$. In this case, we can independently combine both a graph-spectral prior for the $\operatorname{vec}(\mathbf{C})$ section and an L2 prior for the $\mathbf{w}$ section. This can be written as follows.

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0},\ \begin{bmatrix} \gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M \end{bmatrix} \right) \tag{4.30}$$

Here, $\mathbf{H} \in \mathbb{R}^{NT \times NT}$ is a graph filter defined to act over the entire T-V product graph. This block matrix including $\mathbf{H}$ and $\mathbf{I}_M$ both encodes the smoothness assumption for the flexible intercept term, and provides regularisation for the regression coefficients $\mathbf{w}$. Given this, the posterior distribution over $\boldsymbol{\theta}$ is given by

$$\boldsymbol{\theta} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1} \begin{bmatrix} \operatorname{vec}(\mathbf{Y}) \\ \mathbf{X}^\top \operatorname{vec}(\mathbf{Y}) \end{bmatrix},\ \widetilde{\mathbf{P}}^{-1} \right) \tag{4.31}$$

where

$$\widetilde{\mathbf{P}} \in \mathbb{R}^{(NT+M) \times (NT+M)} = \begin{bmatrix} \mathbf{D}_\mathbf{S} + \gamma \mathbf{H}^{-2} & \mathbf{D}_\mathbf{S}\mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_\mathbf{S} & \mathbf{X}^\top \mathbf{D}_\mathbf{S} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix} \tag{4.32}$$

A proof of this is given in theorem A.6. As before, $\mathbf{D}_\mathbf{S} = \operatorname{diag}(\operatorname{vec}(\mathbf{S}))$.

4.2.2 Solving for the posterior mean

In the case of RNC, there is no simple or convenient way to reuse the SIM method to solve for the posterior mean. To see this, consider splitting the matrix $\widetilde{\mathbf{P}}$ into $\mathbf{M} - \mathbf{N}$ where

$$\mathbf{M} = \begin{bmatrix} \mathbf{I}_{NT} + \gamma \mathbf{H}^{-2} & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_M \end{bmatrix}, \quad \text{and} \quad \mathbf{N} = \begin{bmatrix} \mathbf{D}_{\mathbf{S}'} & -\mathbf{D}_{\mathbf{S}}\mathbf{X} \\ -\mathbf{X}^\top \mathbf{D}_{\mathbf{S}} & -\mathbf{X}^\top \mathbf{D}_{\mathbf{S}} \mathbf{X} \end{bmatrix}$$

Recall that each iterative step in the SIM requires multiplication of a vector by $\mathbf{M}^{-1}\mathbf{N}$. Although $\mathbf{M}$ remains easy to invert, the spectral radius, $\rho(\mathbf{M}^{-1}\mathbf{N})$ is no longer guaranteed to be less than one, resulting in a potential lack of convergence. This issue is not unique to a specific permutation of $\mathbf{M}$ and $\mathbf{N}$, but rather persists across multiple arrangements.

On the other hand, we can find an effective preconditioner $\widetilde{\mathbf{\Psi}} \in \mathbb{R}^{(NT+M)\times(NT+M)}$ for use with the CGM, which transforms the linear system as follows.

$$\left(\widetilde{\mathbf{\Psi}}^\top \begin{bmatrix} \mathbf{D}_{\mathbf{S}} + \gamma \mathbf{H}^{-2} & \mathbf{D}_{\mathbf{S}}\mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_{\mathbf{S}} & \mathbf{X}^\top \mathbf{D}_{\mathbf{S}} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix} \widetilde{\mathbf{\Psi}} \right) \left(\widetilde{\mathbf{\Psi}}^{-1} \boldsymbol{\theta} \right) = \widetilde{\mathbf{\Psi}}^\top \begin{bmatrix} \operatorname{vec}(\mathbf{Y}) \\ \mathbf{X}^\top \operatorname{vec}(\mathbf{Y}) \end{bmatrix} \tag{4.33}$$

To obtain a suitable value for $\widetilde{\mathbf{\Psi}}$, first let us define the eigendecomposition of $\mathbf{X}^\top \mathbf{D}_{\mathbf{S}} \mathbf{X}$.

$$\mathbf{X}^\top \mathbf{D}_{\mathbf{S}} \mathbf{X} = \mathbf{U}_M \mathbf{\Lambda}_M \mathbf{U}_M^\top \tag{4.34}$$

Since $\mathbf{X}^\top \mathbf{D}_{\mathbf{S}} \mathbf{X}$ is positive semi-definite, $\mathbf{U}_M$ can be chosen to be orthonormal. Next, let us also introduce the diagonal matrix $\mathbf{D}_M$, which has the following definition

$$\mathbf{D}_M = \left(\mathbf{\Lambda}_M + \lambda \mathbf{I}_M \right)^{-1/2} \tag{4.35}$$

We propose that an effective preconditioner is given by

$$\widetilde{\mathbf{\Psi}} = \begin{bmatrix} \mathbf{U}\mathbf{D}_{\mathbf{G}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M \mathbf{D}_M \end{bmatrix} \tag{4.36}$$

where, as before, $\mathbf{U} = \mathbf{U}_T \otimes \mathbf{U}_N$. We choose this preconditioner because it transforms the block-diagonal elements of the original coefficient matrix $\widetilde{\mathbf{P}}$ into matrices with eigenvalues bounded between $1 + \gamma$ and γ. In particular, the new coefficient matrix is given by

$$\widetilde{\mathbf{\Psi}}^\top \widetilde{\mathbf{P}} \widetilde{\mathbf{\Psi}} = \begin{bmatrix} \mathbf{D_G U}^\top \mathbf{D_S U D_G} + \gamma \mathbf{I}_{NT} & \mathbf{D_G U}^\top \mathbf{D_S X U}_M \mathbf{D}_M \\ \mathbf{D}_M \mathbf{U}_M^\top \mathbf{X}^\top \mathbf{D_S U D_G} & \mathbf{I}_M \end{bmatrix}$$

Note that the upper left block can be multiplied onto a length NT vector with complexity $O(N^2T + NT \log T)$, by leveraging the properties of the Kronecker product and making use of the Fast Cosine Transform (FCT). The upper right block of this matrix consists of M length NT column vectors, which can be computed with complexity $O(MN^2T + MNT \log T)$ and multiplied onto a length M vector with complexity $O(NTM)$. This also applies to the lower left block, which is the transpose of the upper right, and can be multiplied onto a length M vector in the same number of operations. Finally, the lower right block can be multiplied onto a length M vector with M operations. Therefore, the overall complexity of multiplying the entire matrix onto a vector of length $NT + M$ is $O(N^2T + NT \log T)$.

4.2.3 Discussion

Limits as $\gamma, \lambda \to \infty$.

4.2.4

4.3 Network regression with both global and local explanatory variables

In section 4.1, we introduced Kernel Graph Regression (KGR) with arbitrary missing values as a non-parametric regression technique, applicable for modelling scenarios where a series of graph signals need to be predicted based on exogenous or 'global' explanatory variables. This concept is visually represented in fig. 4.1. On the other hand, Regression with Network Cohesion (RNC), discussed in section 4.2, is suitable for situations where 'local' explanatory variables are associated with individual nodes over time, as depicted in fig. 4.2.

In certain cases, both global and local explanatory variables may coexist. For example, consider a network of businesses where firms are connected based on industry similarity

or supply chain integration. In this scenario, the objective may be to predict quarterly revenue growth using variables affecting all firms, such as inflation and interest rates, while also considering firm-level local variables like employee turnover or debt ratio.

The input data in such a scenario can be summarised as follows

$$\text{input data} = \left\{ \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times M},\ \mathbf{X}' \in \mathbb{R}^{T \times M'}\ \mathbf{Y} \in \mathbb{R}^{N \times T},\ \mathbf{S} \in \{0,1\}^{N \times T},\ \mathbf{A} \in \mathbb{R}^{NT \times NT} \right\}$$

Once again, $\mathbf{Y}$ is a sequence of partially observed graph signals and $\mathbf{S}$ is the corresponding binary sensing matrix defining which values are missing. $\boldsymbol{\mathcal{X}}$ represents the tensor of local explanatory variables, where each node at each time has an associated length-M feature vector. In the example, this would represent the firm-level variables such as employee turnover and debt ratio. $\mathbf{X}'$ is the matrix of global explanatory variables, where the t-th row represents the global length-M' feature vector at each time t. In this section we also continue to use the notation $\mathbf{X}$ (without the prime) to denote the matrix which is constructed by horizontally stacking the M vectorized slices $\boldsymbol{\mathcal{X}}_{::m}$, resulting in an object of shape $(NT \times M)$, as defined in eq. (4.25).

Leveraging the methods developed for KGR and RNC in sections 4.1 and 4.2, we can naturally integrate both $\boldsymbol{\mathcal{X}}$ and $\mathbf{X}'$ into the problem. The RNC model supposes that the observed graph signal is a noisy partial observation of a smooth underlying signal $\mathbf{C}$ added to a regularised linear combination of node-level covariates. Therefore, RNC combines aspects of graph signal reconstruction and ridge regression. As discussed in section 4.1.2, KGR can be understood as a small modification of the graph signal reconstruction problem, where the underlying signal $\mathbf{F}$ is instead assumed to be smooth with respect to both the 1D network and the explanatory variables, rather than a 2D Cartesian product graph. This involved altering the prior distribution over $\mathbf{F}$ to include the kernel matrix $\mathbf{K}$ which is a function of the global variables $\mathbf{X}'$, as defined in eq. (4.10).

Similarly, we can modify the prior over the flexible intercept term $\mathbf{C}$ in the RNC model. Instead of using a prior covariance of $\gamma\mathbf{H}^2$, we can employ a covariance matrix $\gamma\mathbf{K} \otimes \mathbf{H}_N^2$, where $\mathbf{K}$ is the kernel matrix and $\mathbf{H}_N$ is a graph filter acting upon the 1D network, as in the KGR model. Since the RNC model is defined in terms of the aggregate parameter vector $\boldsymbol{\theta}$, we can adjust its prior distribution accordingly.

$$\hat{\boldsymbol{\theta}} \sim \mathcal{N}\left(\mathbf{0},\ \begin{bmatrix} \gamma^{-1}\mathbf{K} \otimes \mathbf{H}_N^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M \end{bmatrix}\right) \tag{4.37}$$

Just as with KGR, this new model, which we refer to as Kernel Graph Regression with Network Cohesion (KG-RNC), is algebraically equivalent to the RNC model under the transformation

$$\mathbf{U} \to \bar{\mathbf{U}}, \quad \text{and} \quad \mathbf{G} \to \bar{\mathbf{G}}$$

where $\mathbf{K}$ has an eigendecomposition $\mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top$ and $\bar{\mathbf{U}}$ and $\bar{\mathbf{G}}$ are defined by

$$\bar{\mathbf{U}} = \mathbf{V} \otimes \mathbf{U}_N, \quad \text{and} \quad \bar{\mathbf{G}}_{nt} = g\left(\lambda_n^{(N)}\right)\sqrt{\lambda_t^{(K)}} \quad \to \quad \mathbf{D}_{\bar{\mathbf{G}}} = \boldsymbol{\Lambda}_K^{1/2} \otimes g\left(\boldsymbol{\Lambda}_N\right)$$

Given the prior for $\hat{\boldsymbol{\theta}}$ in eq. (4.37), and the likelihood which is identical to that of the RNC model given in eq. (4.29), the posterior distribution is given by

$$\hat{\boldsymbol{\theta}} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\hat{\mathbf{P}}^{-1}\begin{bmatrix}\text{vec}\left(\mathbf{Y}\right)\\ \mathbf{X}^\top\text{vec}\left(\mathbf{Y}\right)\end{bmatrix}, \hat{\mathbf{P}}^{-1}\right) \tag{4.38}$$

where

$$\hat{\mathbf{P}} \in \mathbb{R}^{(NT+M)\times(NT+M)} = \begin{bmatrix}\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} & \mathbf{D_S}\mathbf{X}\\ \mathbf{X}^\top\mathbf{D_S} & \mathbf{X}^\top\mathbf{D_S}\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix} \tag{4.39}$$

Once again, we can solve the linear system for the mean using the CGM, with a symmetric preconditioner $\hat{\boldsymbol{\Psi}}$.

$$\left(\hat{\boldsymbol{\Psi}}^\top\begin{bmatrix}\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} & \mathbf{D_S}\mathbf{X}\\ \mathbf{X}^\top\mathbf{D_S} & \mathbf{X}^\top\mathbf{D_S}\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix}\hat{\boldsymbol{\Psi}}\right)\left(\hat{\boldsymbol{\Psi}}^{-1}\hat{\boldsymbol{\theta}}\right) = \hat{\boldsymbol{\Psi}}^\top\begin{bmatrix}\text{vec}\left(\mathbf{Y}\right)\\ \mathbf{X}^\top\text{vec}\left(\mathbf{Y}\right)\end{bmatrix} \tag{4.40}$$

where

$$\hat{\boldsymbol{\Psi}} = \begin{bmatrix}\bar{\mathbf{U}}\mathbf{D}_{\bar{\mathbf{G}}} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M\end{bmatrix} \tag{4.41}$$

The new preconditioned coefficient matrix in this case is given by

$$\hat{\boldsymbol{\Psi}}^\top\hat{\mathbf{P}}\hat{\boldsymbol{\Psi}} = \begin{bmatrix}\mathbf{D}_{\bar{\mathbf{G}}}\bar{\mathbf{U}}^\top\mathbf{D_S}\bar{\mathbf{U}}\mathbf{D}_{\bar{\mathbf{G}}} + \gamma\mathbf{I}_{NT} & \mathbf{D}_{\bar{\mathbf{G}}}\bar{\mathbf{U}}^\top\mathbf{D_S}\mathbf{X}\mathbf{U}_M\mathbf{D}_M\\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D_S}\bar{\mathbf{U}}\mathbf{D}_{\bar{\mathbf{G}}} & \mathbf{I}_M\end{bmatrix}$$

which can be efficiently multiplied onto a length $NT + M$ vector by leveraging the properties of the Kronecker product. Note that, in this case, the complexity of this multiplication is no longer $O(N^2T + NT \log T)$, but is now $O(N^2T + NT^2)$, since we cannot make use of the Fast Cosine Transform to multiply the matrix $\mathbf{V} \otimes \mathbf{U}$ onto a vector.

4.4 Network regression for pollutant monitoring

In this section, we analyse the performance of the regression models developed in this chapter by applying them to an environmental modeling task. Our focus is on predicting the concentration of $PM_{2.5}$, a common airborne pollutant which consists of particulate matter with a diameter of 2.5 micrometers or less. We apply our models over a network of monitoring stations located in California, USA, using a dataset obtained from the U.S. Environmental Protection Agency's Air Data program. This program is responsible for generating daily readings of several airborne pollutants, including Ozone, Sulfur Dioxide (SO_2), Carbon Monoxide (CO), Nitrous Dioxide (NO_2), $PM_{2.5}$ and PM_{10} at hundreds of locations across the country [EPA, 2023].

In total, we consider $N = 571$ monitoring stations across a period of $T = 3957$ days from the 1^{st} of January 2012 to the 30^{th} of September 2022. Our method for graph construction was as follows. First, we took the latitude and longitude coordinates of each monitoring station and created a two-dimensional Voronoi diagram bounded by the California border [Fortune, 1986]. This tessellated the state into 571 distinct regions; one per station. Next, we connected each pair of stations i and j if their respective Voronoi regions shared a boundary, and weight that connection by $1/(1 + kd_{ij})$, where k is a scaling factor and d_{ij} is their geodesic separation. This creates a sparse graph, which is visually represented in fig. 4.4. The total product graph used was then the Cartesian product between the size-T path graph (i.e. T nodes connected together in a chian representing time), and the cross-sectional network represented in fig. 4.4.

In our experiments, we predicted $\log(1 + c)$, where c was the concentration of $PM_{2.5}$ in parts per million across the network using all three regression algorithms outlined in this chapter. In the case of Kernel Graph Regression, we used a set of global explanatory variables which were not associated with conditions at the individual monitoring stations, and for Regression with Network Cohesion, we used local explanatory variables that were associated with conditions at the individual monitoring stations. For KGR, we considered the number of active wildfires (which is known to be a strong contributor to $PM_{2.5}$ concentration [Jaffe et al., 2020]) and meterological conditions across the state including temperature, pressure and humidity. In particular, for the wildfires, we

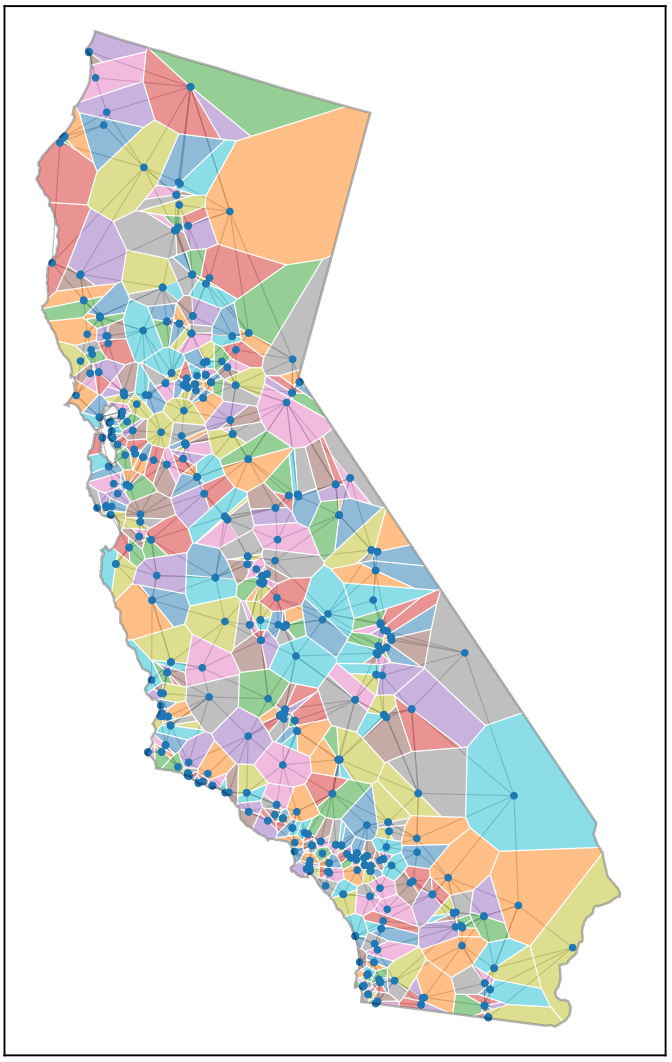


FIGURE 4.4: The network of monitoring stations, created via Voronoi tessellation. The geographic coordinates of the stations are used to create closed the regions, which are connected by a weighted edge to neighbouring regions.

calculated the number of acres actively burning on each day, within nine different regions using data from the CAL FIRE service from the Department of Forestry and Fire Protection [CALFIRE, 2023]. For the meterological variables, we took readings also published by the Air Data program from locations across the state and took the first five principal components.

For RNC, we used the other pollutants measured over the network, i.e. Ozone, SO_2, CO, NO_2, and PM_{10}. Since, in general, a considerable proportion of this data was missing, we first used Graph Signal Reconstruction. as described in chapter 3, to interpolate the missing values. In addition, we used information local to each monitoring station such as land use and elevation. We also tested the KG-RNC framework outlined in section 4.3 by combining both the local and global explanatory variables. Further information about the explanatory variables used is given in tables 4.1 and 4.2.

Global explanatory variables	
Variable	**Description**
Time	A simple linear variable.
Season	Two additional variables tracking season were created by taking the sin and cosine of $2\pi t/365$, where t is the day of the year.
Active wildfires	The original dataset comprised a list of wildfire events in California, containing information including the geographical coordinates, the start date, the end date and the total number of acres burned. Using the coordinates, we assigned each fire into one of nine regions. Then, by dividing the total acres burned by the incident duration, we calculated the average number of acres burning each day for each fire. By summing this for every region, we had an estimate of the total number of acres burning on each day for each region.
Temperature	Using data from the Air Data program, we obtained readings for temperature at 90 locations across California. The readings were selected such that no missing data was present. Then, the first five principal components were taken.
Pressure	As above, but for pressure. Readings from 48 locations were transformed into five PCA components.
Humidity	As above, for humidity. Readings from 51 locations were transformed into five PCA components

TABLE 4.1: The global explanatory variables used in this experiment. Each feature is not associated with any node in particular, but covaries with the graph signal over time. This resulted in a feature matrix $\mathbf{X}$ of shape 3957×27. All columns were normalised to have a mean of zero and standard deviation of one.

As a baseline approach, we also tested univariate and multivariate ridge regression using the local and global explanatory variables respectively. However, since multivariate ridge regression has no standard approach for the situation where values are missing from the multivariate targets, we first filled in these values according to the mean reading at each station. Where no readings were available at all at a particular station, we used the mean reading across time.

In order to

Cleland et al. [2020].

Local explanatory variables	
Variable	**Description**
Ozone	Ozone readings from the Air Data program were taken at all 571 monitoring stations. Where data was missing, graph signal reconstruction was used to interpolate the missing values. 72% of the original data was missing.
Sulfur Dioxide	As above. 95% of the original data was missing.
Carbon Monoxide	As above. 89% of the original data was missing.
Nitrous Dioxide	As above. 83% of the original data was missing.
PM_{10}	As above. 85% of the original data was missing.
Elevation	The elevation of the station in meters. This variable remained constant over time
Land use	Each monitoring station was had a land use flag, which was one of 'Commercial', 'Residential', 'Agricultural', 'Forest', 'Industrial', 'Desert', or 'Military Reservation'. This was turned into a length-7 vector for each station using one-hot encoding which remained constant over time.
Location setting	Each station also had a location flag, which was one of 'Urban And Center City', 'Suburban' or 'Rural. In the same fashion as above, we created a one-hot encoding for each station, which remained constant over time.

TABLE 4.2: The local explanatory variables used in this experiment. Each variable is associated with a monitoring station, and also varies over time. For elevation, land use and location setting which are fixed for each station, the values are simply repeated over time. This creates an explanatory tensor $\boldsymbol{\mathcal{X}}$ of shape $571 \times 3957 \times 16$.

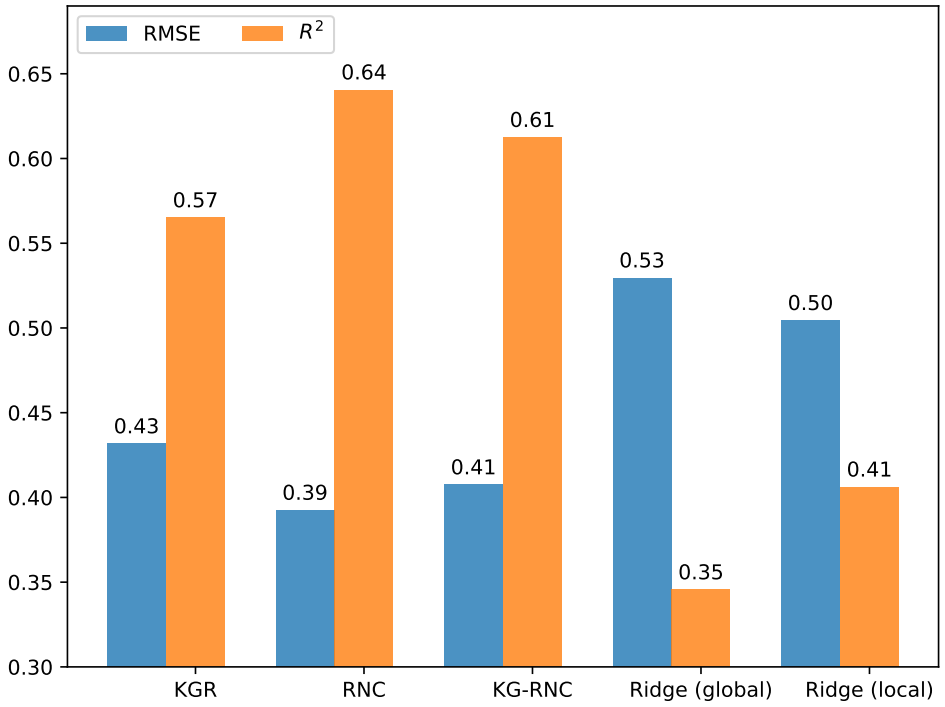


FIGURE 4.5: A visual representation of a time-series graph signal regression problem with exogenous variables. Here, there are T graph signals measured over a static graph, each with independent missing data (indicated in white). Each signal $\mathbf{y}_t$ is accompanied by a unique vector of global explanatory variables $\mathbf{x}_t$.

Chapter 5

Regression and Reconstruction with Tensor-Valued Multiway Graph Signals

Multiway Graph Signal Processing (MWGSP) is an emerging framework for analysing signals with multiple distinct axes (or 'ways'), where the relation between elements within each axis is described by a graph topology [Stanley et al., 2020]. For example, consider an fMRI experiment where cerebral blood flow is measured at a set of 3D voxels across time, for multiple subjects, in response to various stimuli. This dataset could be modelled as a six-way graph signal with the three spatial coordinates and the one time coordinate forming a four-way hypergrid graph, the subjects forming a graph based on characteristics, and the stimuli forming a graph based on similarity [Cichocki et al., 2015]. A visual depiction of this is given in fig. 5.1.

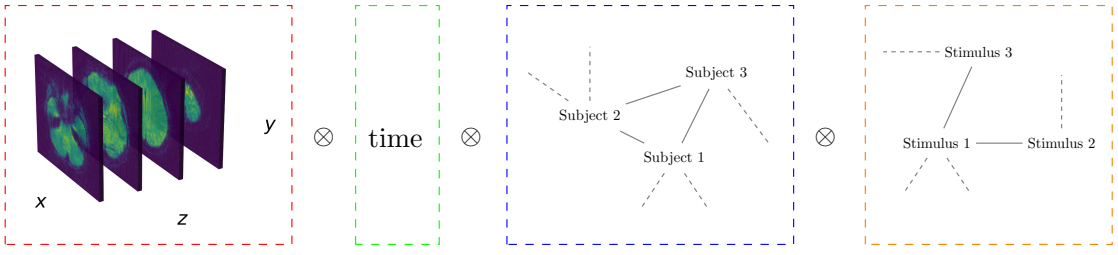


FIGURE 5.1: Graphical depiction of a six-way graph signal originating from a hypothetical fMRI experiment.

The goal of this chapter is to extend the methods developed in chapters 3 and 4 such that they can accommodate multiway graph signals. In particular, Graph Signal Reconstruction (GSR), Kernel Graph Regression (KGR) and Regression with Network Cohesion (RNC) can all be understood as a special two-dimensional case of their more general MWGSP counterpart. In this chapter, we translate all three models into a their d-dimensional form, and demonstrate how to solve efficiently for the posterior mean.

Multiway signals are described using tensors, which can be represented as d-dimensional arrays. Tensor algebra is well-established in fields such as physics and mechanics [Renteln, 2013], however it is less widespread in the graph signal processing community. As such, the first section of this chapter sets out some conceptual and notational standards regarding tensors, which are core to the present and proceeding chapters.

We begin section 5.1 by defining the Cartesian product of more than two graphs, and discuss the algebraic and spectral structure of the resultant objects. Next, we cover the tensor representation of multiway graph signals and give the general definition of a graph-spectral operators in d-dimensions. We also discuss issues surrounding computational efficiency and how the so called 'vec-trick', utilised in prior chapters, can be generalised to the tensor setting. In section 5.2, we begin the core contributions of this chapter by generalising Bayesian GSR as defined in chapter 3 to the MWGSP setting. This necessitates an updated version of the SIM and CGM to accommodate tensor-valued data in arbitrary dimensions, which we give in sections 5.2.1 and 5.2.2. Next, in section 5.3, we generalise KGR for the non-parametric prediction of multiway graph signals as a function of exogenous variables. Finally, in section 5.4, we the generalise RNC as defined in section 4.2 in much the same way.

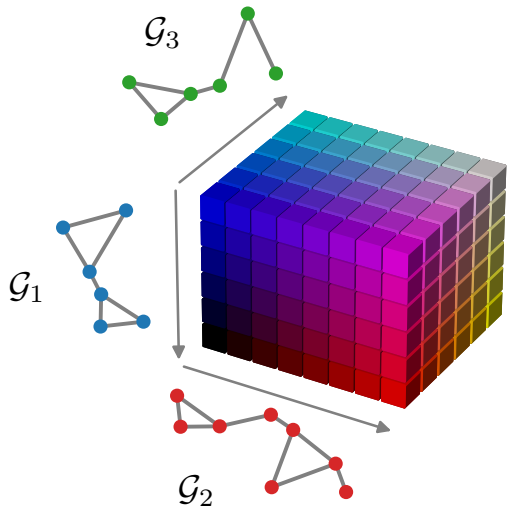


FIGURE 5.2: Graphical depiction of an order-3 tensor signal with graphs underlying each axis, which are linked together to form a Cartesian product graph.

5.1 Multiway Graph Signal Processing

5.1.1 The Cartesian product of more than two graphs

In section 3.1.1 we gave the general definition of a product between two graphs and highlighted four standard examples, namely the Cartesian, direct, strong and lexicographic products. Each of these product types can be straightforwardly extended to more than two factor graphs by applying their respective definition recursively. For example, consider the Cartesian product between graphs $\mathcal{G}_A = \{\mathcal{V}_A, \mathcal{E}_A\}$, $\mathcal{G}_B = \{\mathcal{V}_B, \mathcal{E}_B\}$ and $\mathcal{G}_C = \{\mathcal{V}_C, \mathcal{E}_C\}$ where $|\mathcal{V}_A| = A$, $|\mathcal{V}_B| = B$ and $|\mathcal{V}_C| = C$. This can be written as

$$\mathcal{G} \;=\; \mathcal{G}_A \,\square\, \mathcal{G}_B \,\square\, \mathcal{G}_C \;=\; \{\mathcal{V},\, \mathcal{E}\} \tag{5.1}$$

The new vertex set, $\mathcal{V}$, is given by the Cartesian product of the individual vertex sets, arranged in lexicographic order.

$$\mathcal{V} = \mathcal{V}_A \times \mathcal{V}_B \times \mathcal{V}_C = \{(a,\, b,\, c) \in \mathbb{N}^3 \,|\, a \leq A,\; b \leq B, \text{and } c \leq C\} \tag{5.2}$$

The new edge set, $\mathcal{E}$, is given by recursively applying conditions 1 and 7 from, section 3.1.1 to the new node set. In particular, any two nodes $(a,\, b,\, c)$ and (a', b', c') are connected in $\mathcal{E}$ if they satisfy any of the following three conditions.

1.	$[a,\, a'] \in \mathcal{E}_A$	and	$b = b'$	and	$c = c'$
2.	$a = a'$	and	$[b,\, b'] \in \mathcal{E}_B$	and	$c = c'$
3.	$a = a'$	and	$b = b'$	and	$[c,\, c'] \in \mathcal{E}_C$

Figure 5.3 gives a visual representation of a Cartesian product graph formed from three simple factor graphs. Notice that the size of the new vertex and edge set both grow very quickly. In particular,

$$|\mathcal{V}| = |\mathcal{V}_A||\mathcal{V}_B||\mathcal{V}_C| \quad \text{and} \quad |\mathcal{E}| = |\mathcal{E}_A||\mathcal{V}_B||\mathcal{V}_C| + |\mathcal{V}_A||\mathcal{E}_B||\mathcal{V}_C| + |\mathcal{V}_A||\mathcal{V}_B||\mathcal{E}_C|$$

Happily, the adjacency matrix of a Cartesian product graph $\mathbf{A}$ has a straightforward representation in terms of the factor adjacency matrices (here $\mathbf{A}_A$, $\mathbf{A}_B$ and $\mathbf{A}_C$). Specifically, it is given by their Kronecker sum.

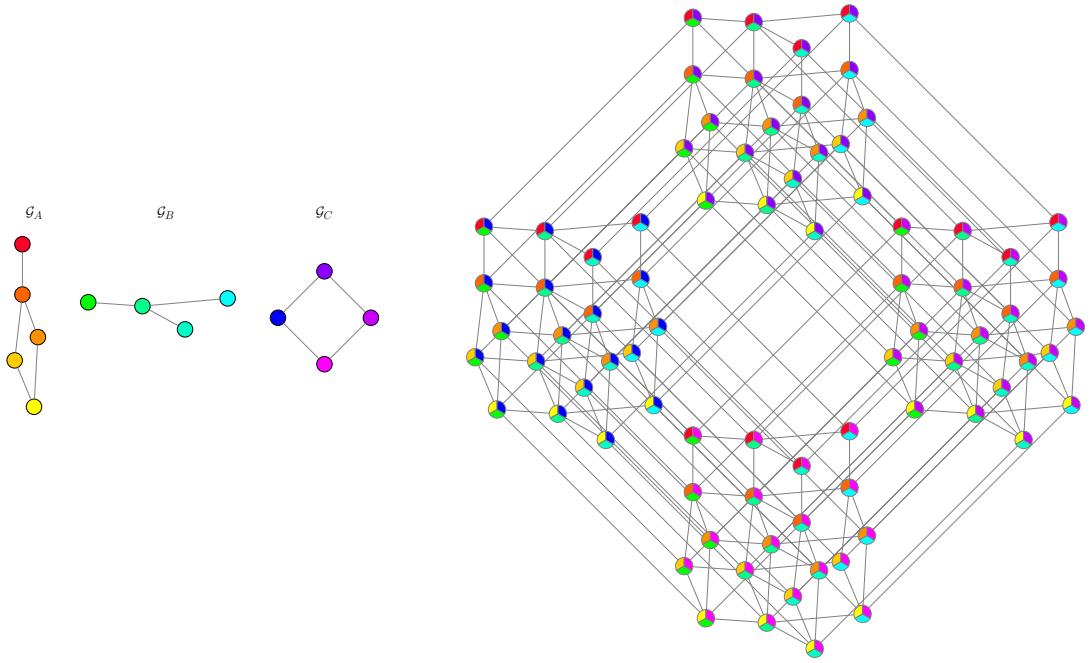


FIGURE 5.3: Graphical depiction of a 3D Cartesian product graph

$$\begin{aligned}\mathbf{A} &= \mathbf{A}_A \oplus \mathbf{A}_B \oplus \mathbf{A}_C \\ &= \mathbf{A}_A \otimes \mathbf{I}_B \otimes \mathbf{I}_C + \mathbf{I}_A \otimes \mathbf{A}_B \otimes \mathbf{I}_C + \mathbf{I}_A \otimes \mathbf{I}_B \otimes \mathbf{A}_C \end{aligned} \tag{5.3}$$

In general, we can consider the Cartesian product of d factor graphs with adjacency matrices denoted as $\mathbf{A}^{(1)} \in \mathbb{R}^{N_1 \times N_2}, \mathbf{A}^{(2)} \in \mathbb{R}^{N_2 \times N_2}, \dots \mathbf{A}^{(d)} \in \mathbb{R}^{N_d \times N_d}$. The full adjacency matrix will have size $N \times N$, where $N = \prod N_i$, and is given by

$$\begin{aligned}\mathbf{A} = \mathbf{A}^{(1)} &\oplus \mathbf{A}^{(2)} \oplus \ \dots \ \oplus \mathbf{A}^{(d)} \\ = \mathbf{A}^{(1)} &\otimes \mathbf{I}_{N_2} \otimes \ \dots \ \otimes \mathbf{I}_{N_d} + \\ \mathbf{I}_{N_1} &\otimes \mathbf{A}^{(2)} \otimes \ \dots \ \otimes \mathbf{I}_{N_d} + \ \dots \ + \\ \mathbf{I}_{N_1} &\otimes \mathbf{I}_{N_2} \otimes \ \dots \ \otimes \mathbf{A}^{(d)} \end{aligned} \tag{5.4}$$

This can be written compactly as

$$\mathbf{A} = \bigoplus_{i=1}^{d} \mathbf{A}^{(i)} \tag{5.5}$$

Similarly, the Laplacian of the product graph, $\mathbf{L}$, can be written as the Kronecker sum of the individual factor graph Laplacians $\mathbf{L}^{(i)}$.

$$\mathbf{L} = \bigoplus_{i=1}^{d} \mathbf{L}^{(i)} \tag{5.6}$$

We can perform eigendecomposition on each of the individual graph Laplacians as follows.

$$\mathbf{L}^{(i)} = \mathbf{U}^{(i)} \mathbf{\Lambda}^{(i)} (\mathbf{U}^{(i)})^{\top} \tag{5.7}$$

where $\mathbf{U}^{(i)}$ is an orthogonal matrix such that each column is an eigenvector of $\mathbf{L}^{(i)}$, and $\mathbf{\Lambda}^{(i)}$ is a diagonal matrix containing the corresponding eigenvalues, which are typically listed in ascending order.

$$\mathbf{\Lambda}^{(i)} = \begin{bmatrix} \lambda_1^{(i)}, & & & \\ & \lambda_2^{(i)} & & \\ & & \ddots & \\ & & & \lambda_{N_i}^{(i)} \end{bmatrix}$$

Given this, the Laplacian of the product graph can be decomposed as follows.

$$\begin{aligned} \mathbf{L} &= \bigoplus_{i=1}^{d} \mathbf{U}^{(i)} \mathbf{\Lambda}^{(i)} (\mathbf{U}^{(i)})^{\top} \\ &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right) \left(\bigoplus_{i=1}^{d} \mathbf{\Lambda}^{(i)} \right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right)^{\top} \\ &= \mathbf{U} \mathbf{\Lambda} \mathbf{U}^{\top} \end{aligned} \tag{5.8}$$

where

$$\mathbf{U} = \bigotimes_{i=1}^{d} \mathbf{U}^{(i)}, \quad \text{and} \quad \mathbf{\Lambda} = \bigoplus_{i=1}^{d} \mathbf{\Lambda}^{(i)} \tag{5.9}$$

As with the Kronecker sum, here we have used the notation $\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}$ to denote the chained Kronecker product of matrices $\{\mathbf{U}^{(i)}\}$.

5.1.2 Representing d-dimensional graph signals

Since each node in a d-dimensional product graph is specified by d independent indices, a signal $\boldsymbol{\mathcal{Y}}$ existing over the nodes has a natural representation as a tensor of order d. One way to conceptualise a d-dimensional tensor signal is as a multi-dimensional array with d independent axes. If the i-th factor graph has N_i vertices, then $\boldsymbol{\mathcal{Y}}$ will be of shape $(N_1, N_2, ...N_d)$. An individual element of this tensor signal can be specified via a vector index $\mathbf{n} = [n_1, n_2, ..., n_d]$, where $1 \leq n_i \leq N_i$.

Alternatively, signals can be represented as a vector of length $N = \prod N_i$. This is essential if we are to interpret the $\otimes$ symbol strictly as a Kronecker product, rather than a tensor or outer product. Under the Kronecker interpretation, the chained use of $\otimes$ used in expressions such as eq. (5.9) results in matrices of shape $N \times N$, providing a linear map from $\mathbb{R}^N \to \mathbb{R}^N$. Therefore, for an operator to act on a tensor graph signal $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times ... \times N_d}$, we need a method of mapping tensors with shape $(N_1, N_2, ...N_d)$ to vectors of length N. In order for this vectorisation process to be consistent with the operators, it should result in a vectors whose elements are arranged in lexicographic order. In some fields, this is referred to as *row-major* vectorisation since, in the case of an order-2 tensor, the index representing the row varies before the column index. In the following, we symbolise this operation mathematically as $\text{vec}_{\text{RM}}(\cdot) : \mathbb{R}^{N_1 \times N_2 \times ... \times N_d} \to \mathbb{R}^N$, and its reverse operation as $\text{ten}_{\text{RM}}(\cdot) : \mathbb{R}^N \to \mathbb{R}^{N_1 \times N_2 \times ... \times N_d}$. We use the 'RM' subscript to indicate explicitly that this process is occurring in row-major order, since the standard $\text{vec}(\cdot)$ function defined for matrices is most commonly assumed to act in column-major order.

In the following, we use bold lower-case symbols (e.g. $\mathbf{y}$) to indicate graph signals existing in their vector form, and bold upper-case calligraphic symbols (e.g. $\boldsymbol{\mathcal{Y}}$) to indicate graph signals in their multi-dimensional array form. That is,

$$\mathbf{y} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{Y}}) \quad \Longleftrightarrow \quad \boldsymbol{\mathcal{Y}} = \text{ten}_{\text{RM}}(\mathbf{y})$$

Figure 5.4 shows gives a visual summary of the process of converting between these two representations for an order-3 tensor.

To calculate the vector index k which a tensor element with index $\mathbf{n} = [n_1, n_2, ..., n_d]$ is mapped to in row-major order, we can apply the following formula.

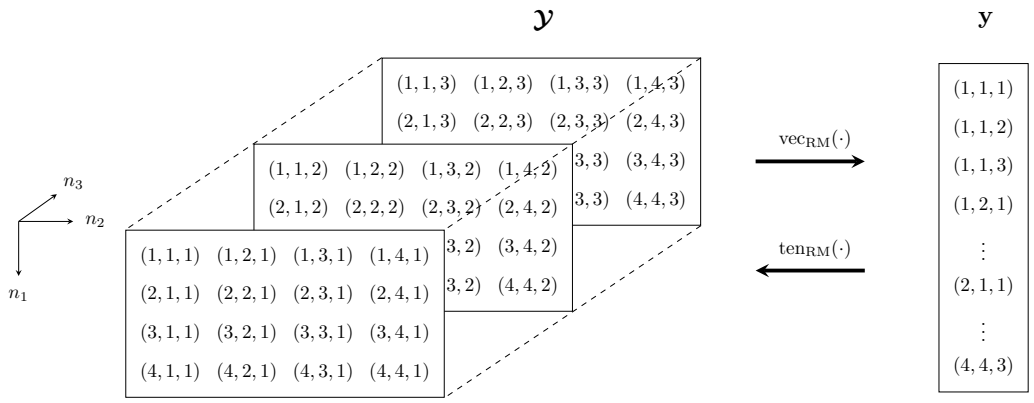


FIGURE 5.4: A graphical depiction of the process of converting an order-3 tensor between its multidimensional array and vector form in row-major order. Note that the elements in the vectorised signal are lexicographically ordered.

$$k = 1 + \sum_{i=1}^{d} \Big(\prod_{j=i+1}^{d} N_j \Big) (n_i - 1) \tag{5.10}$$

(Note the ± 1 disappear when indexing begins from zero). The reverse operation, i.e. mapping a vector element index k to a tensor index $\mathbf{n}$ can be achieved by running the algorithm **3**.

Given these two operations, two arrays of any consistent shape can be mapped between one another by first vectorising according to eq. (5.10), and then converting to a tensor using the given algorithm.

Algorithm 3 Mapping a vector element to a tensor element in row major order

Input: The target vector element k
Input: The shape of the output tensor $(N_1, N_2, ..., N_d)$

$k \leftarrow k - 1$

for i **from** d **to** 1 **do**

$n_i \leftarrow k \mod N_i$
$k \leftarrow \lfloor k / N_i \rfloor$

end for

Output: $(n_1 + 1, n_2 + 1, ..., n_d + 1)$

5.1.3 GSP in d-dimensions

Consider a tensor graph signal $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$ represented in its multi-dimensional array form. In direct analogy to the two dimensional case given in eqs. (3.5) and (3.6), we can define the Graph Fourier Transform (GFT) and its corresponding inverse (IGFT) of this signal as follows.

$$\text{GFT}(\boldsymbol{\mathcal{Y}}) = \text{ten}_{\text{RM}}\left(\mathbf{U}^\top \mathbf{y}\right) = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right)^\top \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)\right) \tag{5.11}$$

$$\text{IGFT}(\boldsymbol{\mathcal{Y}}) = \text{ten}_{\text{RM}}\left(\mathbf{U}\mathbf{y}\right) \quad = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)\right) \tag{5.12}$$

The concept of a graph filter for signals defined on a Cartesian product graph follows naturally from this definition. Just as in the one and two-dimensional case, a graph filter is constructed by first taking the GFT of a signal, then applying some scaling function to each spectral component, then transforming back into the vertex domain via the IGFT. In the simplest case, we can consider an isotropic graph filter function $g(\lambda, \beta)$, such as one of those defined in **??**. A filter $\mathbf{H}$, defined to act on a vectorised graph signal, can be represented as an $N \times N$ matrix, constructed as follows.

$$\begin{aligned}\mathbf{H} &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) g\left(\bigoplus_{i=1}^{d} \boldsymbol{\Lambda}^{(i)}; \beta\right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right)^\top \\ &= \mathbf{U}\,\text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right)\,\mathbf{U}^\top \end{aligned} \tag{5.13}$$

Here, $\boldsymbol{\mathcal{G}}$ represents the spectral scaling tensor, which has element $\mathbf{n} = [n_1, n_2, ..., n_d]$ given by

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\left(\sum_{i=1}^{d} \lambda_{n_i}^{(i)};\, \beta\right) \tag{5.14}$$

In certain special filter types, such as the diffusion filter, this function may be multiplicatively separable. In this case, we have that

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = \prod_{i=1}^{d} g\left(\lambda_{n_i}^{(i)};\, \beta\right) \tag{5.15}$$

Filter	$g(\boldsymbol{\lambda};\, \boldsymbol{\beta})$
1-hop random walk	$(1 + \boldsymbol{\beta}^\top \boldsymbol{\lambda})^{-1}$
Diffusion	$\exp(-\boldsymbol{\beta}^\top \boldsymbol{\lambda})$
ReLu	$\max(1 - \boldsymbol{\beta}^\top \boldsymbol{\lambda}, 0)$
Sigmoid	$2\big(1 + \exp(\boldsymbol{\beta}^\top \boldsymbol{\lambda})\big)^{-1}$
Bandlimited	1, if $\boldsymbol{\beta}^\top \boldsymbol{\lambda} \leq 1$ else 0

TABLE 5.1: Anisotropic graph filter functions in an arbitrary number of dimensions

This further implies that the graph filter $\mathbf{H}$ can be decomposed as

$$\mathbf{H} = \bigotimes_{i=1}^{d} \mathbf{H}^{(i)}, \quad \text{where} \quad \mathbf{H}^{(i)} = \mathbf{U}^{(i)} g\big(\boldsymbol{\Lambda}^{(i)}\big) \left(\mathbf{U}^{(i)}\right)^\top \tag{5.16}$$

However, in the following, we typically don't consider this special case and assume that the graph filter function is non-separable in general.

The concept of a multi-way graph filter can be further generalised to include anisotropic filter functions, where the intensity of the filtering operation is not restricted to be equal in each dimension. Table 5.1 gives some examples of anisotropic graph filter functions defined to act in an arbitrary number of dimensions.

In this case, the spectral scaling tensor is given by

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n});\, \boldsymbol{\beta}\big) \tag{5.17}$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$ is the parameter vector characterising the filter intensity in each dimension, and $\boldsymbol{\lambda}(\mathbf{n}) \in \mathbb{R}^d$ is a vector holding the n_i-th eigenvalue of each graph Laplacian in the Cartesian product.

$$\boldsymbol{\lambda}(\mathbf{n}) = \begin{bmatrix} \lambda_{n_1}^{(1)} & \lambda_{n_2}^{(2)} & \dots & \lambda_{n_d}^{(d)} \end{bmatrix}^\top \tag{5.18}$$

In general, we can define any spectral transformation in tensor terms as follows.

$$\boldsymbol{\mathcal{Y}}' = \mathrm{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \mathrm{GFT}(\boldsymbol{\mathcal{Y}})\right) \tag{5.19}$$

5.1.4 Fast computation of the d-dimensional GFT and IGFT

Consider the definition of the GFT and IGFT of a d-dimensional graph signal given in eqs. (5.11) and (5.12). In both cases, we are required to compute the result of a chained Kronecker product matrix acting on a length-N vector. Whilst the obvious approach to computing this product would have time and memory complexity of $O(N^2)$, a much more efficient implementation can be achieved by taking advantage of the Kronecker structure of the matrix. Specifically, the memory and time complexity of this operation can be reduced to $O(N)$ and $O(N\sum N_i)$ respectively. The importance of this fact cannot be understated, as it enables scaling to much larger product graphs that would otherwise be possible.

This general algorithm for achieving this is well-known, and can be summarised as follows. Consider the application of a chained Kronecker product matrix acting on a vector $\mathbf{y}$.

$$\mathbf{y} = \left(\mathbf{U}^{(1)} \otimes \mathbf{U}^{(2)} \otimes ... \otimes \mathbf{U}^{(d)}\right) \mathbf{z}$$

This can be factorised as follows

$$\mathbf{y} = \left(\mathbf{U}^{(1)} \otimes \mathbf{I} \otimes ... \otimes \mathbf{I}\right) \left(\mathbf{I} \otimes \mathbf{U}^{(2)} \otimes ... \otimes \mathbf{I}\right) ... \left(\mathbf{I} \otimes \mathbf{I} \otimes ... \otimes \mathbf{U}^{(d)}\right) \mathbf{z}$$

As is visible, the original multiplication has now been broken into d stages. However, the i-th stage can be completed with $N \times N_i$ multiplications by reshaping the vector in the appropriate way and leveraging the properties of the Kronecker product. The reshaping operation can be completed using strided permutation matrices which can be applied in practice for virtually zero computational cost [Granata et al., 1992]. This idea is also key to the FFT and related algorithms, which can be understood as finding a recursive Kronecker structure in the Fourier matrix [Tolimieri et al., 2013].

Work on efficient computational procedures for this operation can be traced back to Roth [1934] who formulated the original 2-dimensional "vec trick" algorithm. The d-dimensional generalisation was proposed in Pereyra and Scherer [1973] and improved in DeBoor [1979]. More recent work, such as Fackler [2019], has focused on further optimisations such as minimising data transit times and parallel processing.

Furthermore, if the i-th factor graph in the Cartesian product is a path or ring graph, then the corresponding matrix transformation can be completed with only $N \log N_i$ multiplications by making use of the FCT/FST/FFT algorithms. In the extreme case, where

every factor graph has this special structure, the computational complexity reaches parity with the multidimensional FFT algorithm, and will have a runtime complexity of $O(N \log N)$ [Smith and Smith, 1995].

In order to execute computations of this nature in a way that is maximally efficient, we have developed the Python library *PyKronecker*, which is described in detail in Antonian et al. [2023]. This library offers a high-level API for constructing Kronecker-based operators and applying them to either vectors or tensors, whilst optimising the underlying computation using parallel GPU processing and Just In Time (JIT) compilation using the Jax library [Bradbury et al., 2018]. In the following, it will be assumed that all chained Kronecker product matrices are applied to vectors/tensors using an efficient implementation. This is essential for computing the d-dimensional GFT and IGFT, the basic pseudocode for which is shown in algorithm 4. Note that the 'reshape' operation should always be applied using the row-major convention. This is the standard convention in languages such as C and Python's NumPy library [Harris et al., 2020], but not in languages such as Matlab and Fortran.

A note on tensor notation

The use of tensor algebra is well established in fields such as phyics and mechanics [Abraham et al., 1988, Renteln, 2013], however, it is less widespread in the signal processing community. For this reason, we choose to adopt a notation that leans more on standard linear algebra, however, all the equations and algorithms discussed in the following chapters could be alternatively written in a purer form of tensor notation. For example, consider the IGFT of a tensor signal $\boldsymbol{\mathcal{Z}}$. In our notation, this is written as

$$\boldsymbol{\mathcal{Y}} = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Z}}\right)\right) \tag{5.20}$$

As is visible, this describes the process in terms of regular matrix-vector multiplication, but requires the additional definition of the $\text{vec}_{\text{RM}}(\cdot)$ and $\text{ten}_{\text{RM}}(\cdot)$ operations. Alternatively, this expression could be written using tensor indexing and Einstein summation notation as follows.

$$\boldsymbol{\mathcal{Y}}^{i_1,i_2,\ldots,i_d} = \left(\mathbf{U}^{(1)}\right)^{i_1}_{j_1} \left(\mathbf{U}^{(2)}\right)^{i_2}_{j_2} \ldots \left(\mathbf{U}^{(d)}\right)^{i_d}_{j_d} \boldsymbol{\mathcal{Z}}^{j_1,j_2,\ldots,j_d} \tag{5.21}$$

Note that here the indices $j_1, j_2, \ldots, j_d$ on the right hand side are implicitly summed over. This eliminates the need to consider vectorisation at all and is perhaps a more elegant way of describing the d-dimensional GFT/IGFT. However, there

Algorithm 4 Efficient GFT and IGFT in d-dimensions

Input: List of Laplacian eigenvector matrices $\left\{\mathbf{U}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$

function GFT($\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$)
  for i **from** 1 **to** d **do**
    $\boldsymbol{\mathcal{Y}} \leftarrow \text{reshape}\Big(\boldsymbol{\mathcal{Y}},\ \big(N_i,\ N/N_i\big)\Big)$
    $\boldsymbol{\mathcal{Y}} \leftarrow \Big(\big(\mathbf{U}^{(i)}\big)^\top\ \boldsymbol{\mathcal{Y}}\Big)^\top$
  end for
  return $\text{reshape}\Big(\boldsymbol{\mathcal{Y}},\ \big(N_1,\ N_2,\ \ldots,\ N_d\big)\Big)$
end function

function IGFT($\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$)
  for i **from** 1 **to** d **do**
    $\boldsymbol{\mathcal{Z}} \leftarrow \text{reshape}\Big(\boldsymbol{\mathcal{Z}},\ \big(N_i,\ N/N_i\big)\Big)$
    $\boldsymbol{\mathcal{Z}} \leftarrow \big(\mathbf{U}^{(i)}\ \boldsymbol{\mathcal{Z}}\big)^\top$
  end for
  return $\text{reshape}\Big(\boldsymbol{\mathcal{Z}},\ \big(N_1,\ N_2,\ \ldots,\ N_d\big)\Big)$
end function

are multiple indices to keep track of which becomes somewhat unaesthetic in a variable number of dimensions.

Both forms offer different trade-offs, however, there is no practical difference when it comes to executing the signal precessing algorithms themselves. Note that, as described in section 5.1.4, the full $N \times N$ matrix implied by eq. (5.20) is never actually instantiated in memory (see algorithm 4).

5.2 Multiway Graph Signal reconstruction

The model we use to describe graph signal reconstruction in the multi-dimensional setting is as follows. Consider a tensor signal $\boldsymbol{\mathcal{Y}}$ of shape $\big(N_1,\ N_2,\ \ldots N_d\big)$ with elements interpreted as existing on the nodes of a d-dimensional Cartesian product graph. Only

a partial set $\mathcal{S} = \{\mathbf{n}_1, \mathbf{n}_2, ...\}$ of the vector elements of $\boldsymbol{\mathcal{Y}}$ are available at observation time, with unobserved values set to zero. The goal is to estimate the signal value at these unobserved entries.

To aid with the model description, we also introduce a binary sensing tensor $\boldsymbol{\mathcal{S}}$, of the same shape as $\boldsymbol{\mathcal{Y}}$, which is used to indicate which elements of $\boldsymbol{\mathcal{Y}}$ were observed. This is defined as follows.

$$\boldsymbol{\mathcal{S}}_{\mathbf{n}} = \begin{cases} 1 & \text{if } \mathbf{n} \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{5.22}$$

The input data for signal reconstruction on a d-dimensional Cartesian product graph can therefore be summarised as follows.

$$\text{input data} = \left\{ \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times ... \times N_d}, \;\; \boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times ... \times N_d}, \;\; \mathbf{L} \in \mathbb{R}^{N \times N} \right\}$$

In analogy with the two dimensional case, (see section 3.2.1), we assume that $\boldsymbol{\mathcal{Y}}$ is a noisy partial observation of an underlying tensor, $\boldsymbol{\mathcal{F}}$, which is smooth with respect to the topology of the Cartesian product graph. This is represented by the following statistical model.

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{S}} \circ \left(\boldsymbol{\mathcal{F}} + \boldsymbol{\mathcal{E}} \right) \tag{5.23}$$

where, here, the $\circ$ symbol represents the generalised tensor Hadamard product, i.e. element-wise multiplication of two tensors. $\boldsymbol{\mathcal{E}}$ is a random tensor where each element has an independent normal distribution with with unit variance. That is,

$$\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{E}}\right) \;\sim\; \mathcal{N}\left(\mathbf{0}, \mathbf{I}_N\right) \tag{5.24}$$

Given this distribution over the model noise, the conditional distribution of $\boldsymbol{\mathcal{Y}} | \boldsymbol{\mathcal{F}}$ is given by

$$\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right) \,|\, \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{F}}\right) \sim \mathcal{N}\Big(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}} \circ \boldsymbol{\mathcal{F}}\right), \; \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)\right) \Big) \tag{5.25}$$

Or, more concisely,

$$\mathbf{y} \,|\, \mathbf{f} \sim \mathcal{N}\big(\mathbf{s} \circ \mathbf{f}, \; \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big) \tag{5.26}$$

where $\mathbf{f} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{F}})$, $\mathbf{y} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{Y}})$, $\mathbf{s} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{S}})$ and $\mathbf{D}_{\boldsymbol{\mathcal{S}}} = \text{diag}(\mathbf{s})$. In order to encode the belief that the underlying tensor $\boldsymbol{\mathcal{F}}$ is smooth with respect to the topology of the graph, we can make use of the following prior distribution.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma^{-1}\mathbf{H}^2\right) \tag{5.27}$$

where $\mathbf{H}$ is constructed from a graph filter function in the manner specified in eq. (5.13), i.e.

$$\mathbf{H} = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top$$

where $\mathbf{D}_{\boldsymbol{\mathcal{G}}} = \text{diag}\left(\text{vec}_{\text{RM}}(\boldsymbol{\mathcal{G}})\right)$, and $\boldsymbol{\mathcal{G}}$ is the spectral scaling matrix obtained by applying a graph filter to the graph Laplacian eigenvalues. The intuition for this prior can be obtained from a direct generalisation of the one and two dimensional cases. In essence, tensor signals drawn from this prior will have the same probability density function as iid noise filtered by $\mathbf{H}$, so samples will be naturally smooth with respect to the topology of the underlying Cartesian product graph. By applying Bayes theorem, we obtain the posterior distribution for $\mathbf{f}$ conditioned on $\mathbf{y}$.

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right) \tag{5.28}$$

where

$$\mathbf{P} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{H}^{-2} \tag{5.29}$$

Therefore, the mean of this posterior is obtained by solving the following linear system.

$$\mathbf{f}^\star = \left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{H}^{-2}\right)^{-1}\mathbf{y} \tag{5.30}$$

Once again, we are faced with a similar set of problems to those outlined in section 3.2.1. Namely, the coefficient matrix is very large and potentially ill-defined. This can be solved by turing to iterative methods such as the SIM and CGM, which are repeated here in their form adapted for the general tensor case.

5.2.1 Tensor SIM

Generalising the SIM, which we describe in section 3.2.2, to the tensor setting is straightforward. In particular, eq. (5.30) can be solved by splitting the coefficient matrix $\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2}\right)$ into $\mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ and $\mathbf{N}$ take on the following values

$$\mathbf{M} = \gamma \mathbf{H}^{-2} + \mathbf{I}_N, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\boldsymbol{\mathcal{S}}'} \tag{5.31}$$

where $\mathbf{D}_{\boldsymbol{\mathcal{S}}'} = \operatorname{diag}(\mathbf{1} - \mathbf{s})$. In this case, the inverse of $\mathbf{M}$ has the form

$$\begin{aligned} \mathbf{M}^{-1} &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right) \operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{J}})\right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right)^{\top} \\ &= \mathbf{U}\, \mathbf{D}_{\boldsymbol{\mathcal{J}}}\, \mathbf{U}^{\top} \end{aligned} \tag{5.32}$$

where the tensor $\boldsymbol{\mathcal{J}}$ has entries with the vector index $\mathbf{n} = [n_1, n_2, ..., n_d]$ given by

$$\boldsymbol{\mathcal{J}}_{\mathbf{n}} = \frac{\boldsymbol{\mathcal{G}}_{\mathbf{n}}^2}{\boldsymbol{\mathcal{G}}_{\mathbf{n}}^2 + \gamma}. \tag{5.33}$$

Here, the entries of $\boldsymbol{\mathcal{G}}$ are given by either eq. (5.14) or eq. (5.17), which correspond to an isotropic or anisotropic filter function respectively. In a very similar manner to eq. (3.21), the SIM update equation is given by

$$\mathbf{f}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{f}_k + \mathbf{M}^{-1}\mathbf{y} \tag{5.34}$$

Note that each step can be achieved with time complexity $O(N \sum N_i)$ by making use of the fast Kronecker algorithm for computing the d-dimensional GFT/IFGT highlighted in section 5.1.4. To be explicit, this update formula can be computed efficiently as

$$\begin{aligned} \boldsymbol{\mathcal{F}}_{k+1} &= \operatorname{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \operatorname{GFT}\left(\boldsymbol{\mathcal{S}}' \circ \boldsymbol{\mathcal{F}}_k\right)\Big) + \boldsymbol{\mathcal{F}}_0 \\ \text{where} \quad \boldsymbol{\mathcal{F}}_0 &= \operatorname{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \operatorname{GFT}\left(\boldsymbol{\mathcal{Y}}\right)\Big) \end{aligned}$$

or, equivalently,

Algorithm 5 The tensor SIM for GSR

Input: Observed tensor graph signal $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$

Input: Sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times N_2 \times \ldots \times N_d}$

Input: Factor graph Laplacians $\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\}_{i=1}^{d}$

Input: Regularisation parameter $\gamma \in \mathbb{R}^{+}$

Input: Graph filter function $g(\,\cdot\,; \boldsymbol{\beta} \in \mathbb{R}^{d})$

For i from 1 to d, decompose $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

Compute $\boldsymbol{\mathcal{G}}$ by applying eq. (5.17)

Compute $\boldsymbol{\mathcal{J}}$ as $\boldsymbol{\mathcal{J}}_{\mathbf{n}} = \boldsymbol{\mathcal{G}}_{\mathbf{n}}^2 / (\boldsymbol{\mathcal{G}}_{\mathbf{n}}^2 + \gamma)$

$\Delta\boldsymbol{\mathcal{F}} \leftarrow \mathrm{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \mathrm{GFT}\big(\boldsymbol{\mathcal{Y}}\big)\Big)$

$\boldsymbol{\mathcal{F}} \leftarrow \Delta\boldsymbol{\mathcal{F}}$

while $|\Delta\boldsymbol{\mathcal{F}}| > \mathrm{tol}$ **do**

$\Delta\boldsymbol{\mathcal{F}} \leftarrow \mathrm{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \mathrm{GFT}\big(\boldsymbol{\mathcal{S}}' \circ \Delta\boldsymbol{\mathcal{F}}_k\big)\Big)$

$\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} + \Delta\boldsymbol{\mathcal{F}}$

end while

Output: $\boldsymbol{\mathcal{F}}$

$$\Delta\boldsymbol{\mathcal{F}}_{k+1} = \mathrm{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \mathrm{GFT}\big(\boldsymbol{\mathcal{S}}' \circ \Delta\boldsymbol{\mathcal{F}}_k\big)\Big)$$

$$\text{where} \quad \Delta\boldsymbol{\mathcal{F}}_0 = \mathrm{IGFT}\Big(\boldsymbol{\mathcal{J}} \circ \mathrm{GFT}\big(\boldsymbol{\mathcal{Y}}\big)\Big)$$

using the fast GFT/IGFT algorithms described in 4. For clarity, the full SIM algorithm is given in algorithm 5.

Once again, the worst-case scaling rate of the number of steps required for convergence, n_{SIM}, is bounded by

$$\frac{1}{\log(1+\gamma) - \log m} \;\leq\; n_{\mathrm{SIM}} \;\leq\; \frac{1}{\log(1+\gamma)}$$

where m is the fraction of data that is missing in the input tensor $\boldsymbol{\mathcal{Y}}$ (see section 3.3). As before, the true scaling rate will depend on the strength of the graph filter.

5.2.2 Tensor CGM

The tensor version of the CGM also follows naturally from the two-dimensional case outlined in section 3.2.3. In particular, eq. (5.30) can be transformed into the following equivalent preconditioned linear system.

$$\mathbf{f}^{\star} = \boldsymbol{\Psi}\left(\boldsymbol{\Psi}^{\top}\left(\mathbf{D}_{\mathcal{S}} + \gamma\mathbf{H}^{-2}\right)\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^{\top}\mathbf{y} \tag{5.35}$$

where, in the tensor case,

$$\boldsymbol{\Psi} = \left(\bigotimes_{i=1}^{d}\mathbf{U}^{(i)}\right)\operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right) = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \tag{5.36}$$

This means eq. (5.35) can be expressed as

$$\mathbf{f}^{\star} = \boldsymbol{\Psi}\left(\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{N}\right)^{-1}\boldsymbol{\Psi}^{\top}\mathbf{y} \tag{5.37}$$

Note that, once again, this preconditioned coefficient matrix can be multiplied onto any appropriate tensor $\boldsymbol{\mathcal{Z}}$ efficiently by making use of the chained Kronecker multiplication procedure given in algorithm 4. As with the SIM, this can be performed with $O(N\sum N_i)$ multiplications. In particular,

$$\begin{aligned}\boldsymbol{\mathcal{Z}}' &= \operatorname{ten}_{\mathrm{RM}}\left(\left(\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{N}\right)\operatorname{vec}_{\mathrm{RM}}\left(\boldsymbol{\mathcal{Z}}\right)\right)\\ &= \boldsymbol{\mathcal{G}} \circ \operatorname{GFT}\Big(\boldsymbol{\mathcal{S}} \circ \operatorname{IGFT}\big(\boldsymbol{\mathcal{G}} \circ \boldsymbol{\mathcal{Z}}\big)\Big) + \gamma\boldsymbol{\mathcal{Z}}\end{aligned}$$

Just as with the two-dimensional case, we can bound the condition number of the preconditioned coefficient matrix to find the worst case scaling rates in the limit of a weak and strong filter. As before, this falls between

$$\sqrt{\frac{1-m+\gamma}{\gamma}} \;\leq\; n_{\mathrm{CGM}} \;\leq\; \sqrt{\frac{1+\gamma}{\gamma}}$$

Algorithm 6 The tensor CGM for GSR

Input: Observed tensor graph signal $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$

Input: Sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times N_2 \times \ldots \times N_d}$

Input: Factor graph Laplacians $\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\}_{i=1}^{d}$

Input: Regularisation parameter $\gamma \in \mathbb{R}^{+}$

Input: Graph filter function $g(\,\cdot\,; \boldsymbol{\beta} \in \mathbb{R}^{d})$

For i from 1 to d, decompose $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

Compute $\boldsymbol{\mathcal{G}}$ by applying eq. (5.17)

Initialise $\boldsymbol{\mathcal{Z}}$

$\boldsymbol{\mathcal{R}} \leftarrow \boldsymbol{\mathcal{G}} \circ \mathrm{GFT}\left(\boldsymbol{\mathcal{Y}}\right)$

$\boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{R}}$

while $|\Delta \mathbf{R}| > \mathrm{tol}$ **do**

$\boldsymbol{\mathcal{A}} \leftarrow \boldsymbol{\mathcal{G}} \circ \mathrm{GFT}\Big(\boldsymbol{\mathcal{S}} \circ \mathrm{IGFT}\big(\boldsymbol{\mathcal{G}} \circ \boldsymbol{\mathcal{D}}\big)\Big) + \gamma \boldsymbol{\mathcal{D}}$

$\alpha \leftarrow \sum_{\mathbf{n}} \boldsymbol{\mathcal{R}}_{\mathbf{n}}^2 \,/\, \sum_{\mathbf{n}} \boldsymbol{\mathcal{R}}_{\mathbf{n}} \boldsymbol{\mathcal{A}}_{\mathbf{n}}$

$\boldsymbol{\mathcal{Z}} \leftarrow \boldsymbol{\mathcal{Z}} + \alpha \boldsymbol{\mathcal{D}}$

$\boldsymbol{\mathcal{R}} \leftarrow \boldsymbol{\mathcal{R}} - \alpha \boldsymbol{\mathcal{A}}$

$\delta \leftarrow \sum_{\mathbf{n}} \boldsymbol{\mathcal{R}}_{\mathbf{n}}^2 \,/\, \sum_{\mathbf{n}} (\boldsymbol{\mathcal{R}} + \alpha \boldsymbol{\mathcal{A}})^2$

$\boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{R}} + \delta \boldsymbol{\mathcal{D}}$

end while

Output: $\mathrm{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \boldsymbol{\mathcal{Z}}\right)$

5.3 Multiway Kernel Graph Regression

In this section, we take the Kernel Graph Regression (KGR) algorithm, developed in section 4.1, and generalise it to the multiway/tensor case. Consider a series of T product graph signals, each denoted as $\boldsymbol{\mathcal{Y}}_t$, with dimensions $(N_1 \times \ldots \times N_d)$. In this scenario, each signal is associated with an explanatory vector, $\mathbf{x}_t \in \mathbb{R}^{M}$. Therefore, the available data consist of labeled pairs $\left\{\mathbf{x}_t, \boldsymbol{\mathcal{Y}}_t\right\}_{t=1}^{T}$. This data can be consolidated into a a matrix, $\mathbf{X}$, of dimensions $(T \times M)$, and a tensor, $\boldsymbol{\mathcal{Y}}$, with shape $(T \times N_1 \times \ldots \times N_d)$.

The observed tensor $\boldsymbol{\mathcal{Y}}$ may also contain an arbitrary amount of missing data, which is described by the binary sensing tensor $\boldsymbol{\mathcal{S}}$. In accordance with previous sections, $\boldsymbol{\mathcal{S}}$ shares its shape with $\boldsymbol{\mathcal{Y}}$ and holds ones at elements where successful observations were made and zeros elsewhere. As a result, the input data for multiway KGR can be summarised as follows.

$$\text{input data} = \left\{ \mathbf{X} \in \mathbb{R}^{T \times M}, \;\; \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{T \times N_1 \times \ldots \times N_d}, \;\; \boldsymbol{\mathcal{S}} \in \{0,1\}^{T \times N_1 \times \ldots \times N_d}, \;\; \mathbf{L} \in \mathbb{R}^{N \times N} \right\}$$

The goal of KGR is to estimate all the missing values within $\boldsymbol{\mathcal{Y}}$, accounting for both the topology of the graph and the explanatory variables. As with the two-dimensional KGR model elaborated in section 4.1, no distinction is necessary between "training" and "testing" data in this context. To make a completely new prediction for some explanatory vector $\mathbf{x}_t$, we merely assign zero to the corresponding values in $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{S}}$.

The fundamental logic underpinning the tensor-valued version of KGR remains consistent with the two-dimensional scenario. We continue to assume that $\boldsymbol{\mathcal{Y}}$ is a noisy, partial observation of an underlying latent signal $\boldsymbol{\mathcal{F}}$, which is captured in the following statistical model.

$$\mathbf{y} \,|\, \mathbf{f} \sim \mathcal{N}\left(\mathbf{s} \circ \mathbf{f}, \mathbf{I}_{NT}\right) \tag{5.38}$$

In this section, we omit a detailed derivation of the prior distribution over $\mathbf{f}$ from first principles, as the process closely resembles the steps provided in section 4.1. For the purposes of this section, we justify it by stating that the signal $\boldsymbol{\mathcal{F}}$ should display smooth variations across both the space of explanatory variables and the topology of the Cartesian product graph. This can be encoded in the following prior distribution for $\mathbf{f}$.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma \mathbf{K} \otimes \mathbf{H}^2\right) \tag{5.39}$$

In this expression, $\mathbf{K}$ denotes a $(T \times T)$ kernel matrix, which is created by applying a valid Mercer kernel, κ, to pairs of explanatory variables such that $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Multiple options for κ are available, including popular choices such as the Gaussian kernel (see eq. (4.11)). By applying Bayes' rule to eqs. (5.38) and (5.39), the posterior distribution over the latent signal $\mathbf{f}$ can be established. This is given by

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\bar{\mathbf{P}}^{-1}\mathbf{y}, \bar{\mathbf{P}}^{-1}\right) \tag{5.40}$$

where, in this case, the posterior precision matrix $\bar{\mathbf{P}}$ is an $NT \times NT$ matrix given by

$$\bar{\mathbf{P}} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}^{-2} \tag{5.41}$$

5.3.1 Computation of the posterior mean

In order to find the posterior mean, we must solve the linear system $\bar{\mathbf{P}}^{-1}\mathbf{y}$. Again, this can be achieved by utilising the tensor versions of the SIM or CGM. In close analogy with the two dimensional case, the process is the same as that of multiway GSR, under the following change of variables.

$$\mathbf{U} \to \bar{\mathbf{U}}, \quad \text{and} \quad \boldsymbol{\mathcal{G}} \to \bar{\boldsymbol{\mathcal{G}}} \tag{5.42}$$

To define $\bar{\mathbf{U}}$ and $\bar{\boldsymbol{\mathcal{G}}}$, first we must define the eigendecomposition of the kernel matrix $\mathbf{K}$.

$$\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top \tag{5.43}$$

where $\mathbf{V}$ is the matrix with columns representing the normalised eigenvectors of $\mathbf{K}$, and $\boldsymbol{\Lambda}_K$ is the diagonal matrix holding its eigenvalues in ascending order, such that

$$\boldsymbol{\Lambda}_K = \operatorname{diag}\left(\left[\lambda_1^{(K)},\ \lambda_2^{(K)},\ \ldots,\ \lambda_T^{(K)}\right]\right) \tag{5.44}$$

Then, $\bar{\mathbf{U}}$ and $\bar{\boldsymbol{\mathcal{G}}}$ are defined as follows.

$$\bar{\mathbf{U}} \in \mathbb{R}^{NT\times NT} = \mathbf{V} \otimes \mathbf{U} \tag{5.45}$$

and

$$\bar{\boldsymbol{\mathcal{G}}}_{t,\mathbf{n}} = g(\boldsymbol{\lambda}(\mathbf{n}), \boldsymbol{\beta})\sqrt{\lambda_t^{(K)}} \quad \Longleftrightarrow \quad \mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}} = \boldsymbol{\Lambda}_K^{1/2} \otimes \mathbf{D}_{\boldsymbol{\mathcal{G}}} \tag{5.46}$$

From here, either the SIM or CGM, as given in line 9 and algorithm 6, can be applied. The only changes necessary are the swap in $\bar{\mathbf{U}}$ for $\mathbf{U}$ and $\bar{\boldsymbol{\mathcal{G}}}$ for $\boldsymbol{\mathcal{G}}$. As such, this means that whenever $\mathrm{GFT}(\cdot)$ is used, it instead signifies the following operation.

$$\mathrm{GFT}(\boldsymbol{\mathcal{Y}}) = \operatorname{ten}_{\mathrm{RM}}\left(\bar{\mathbf{U}}^\top\mathbf{y}\right) = \operatorname{ten}_{\mathrm{RM}}\left(\mathbf{V}^\top \otimes \left(\bigotimes_{i=1}^{d}\mathbf{U}^{(i)}\right)^\top \operatorname{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{Y}})\right) \tag{5.47}$$

$$\mathrm{IGFT}(\boldsymbol{\mathcal{Y}}) = \operatorname{ten}_{\mathrm{RM}}\left(\bar{\mathbf{U}}\mathbf{y}\right) = \operatorname{ten}_{\mathrm{RM}}\left(\mathbf{V} \otimes \left(\bigotimes_{i=1}^{d}\mathbf{U}^{(i)}\right) \operatorname{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{Y}})\right) \tag{5.48}$$

Note that these no longer technically represent the GFT and IGFT, but a modified Kronecker operation that now includes an additional dimension described by $\mathbf{V}$.

5.4 Multiway Regression with Network Cohesion

In this section, we extend the Regression with Network Cohesion (RNC) and Kernel Graph Regression with Network Cohesion (KGRNC) algorithms, initially discussed in section 4.2, to the multiway GSP framework. Our starting point will be multiway RNC. In this context, we are concerned with a graph signal residing on the nodes of a known d-dimensional Cartesian product graph, where an arbitrary subset of the elements could be corrupted or absent. Furthermore, each of the $N = N_1 \times ... \times N_d$ nodes is accompanied by a length-M vector of explanatory variables. Consequently, the available data can be summarised as follows.

$$\text{input data} = \left\{ \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times ... \times N_d \times M},\ \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{N_1 \times ... \times N_d},\ \boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times ... \times N_d},\ \mathbf{L} \in \mathbb{R}^{N \times N} \right\}$$

As before, the binary sensing tensor $\boldsymbol{\mathcal{S}}$ describes which elements of $\boldsymbol{\mathcal{Y}}$ were available at observation time. We also define the matrix $\mathbf{X}$ which is a reshaping of the tensor of explanatory variables $\boldsymbol{\mathcal{X}}$ such that

$$\mathbf{X} \in \mathbb{R}^{N \times M} = \begin{bmatrix} \text{vec}\left(\boldsymbol{\mathcal{X}}_{:,1}\right) & \text{vec}\left(\boldsymbol{\mathcal{X}}_{:,2}\right) & \dots & \text{vec}\left(\boldsymbol{\mathcal{X}}_{:,M}\right) \end{bmatrix} \tag{5.49}$$

In this model, we assume that the signal $\boldsymbol{\mathcal{Y}}$ is a noisy partial observation of the sum of a smooth tensor signal $\boldsymbol{\mathcal{C}}$ and a linear combination of the explanatory variables at each node. This is summarised in the following model.

$$\mathbf{y} = \mathbf{s} \circ \left(\mathbf{c} + \mathbf{X}\mathbf{w} + \mathbf{e}\right) \tag{5.50}$$

Here, $\mathbf{c} \in \mathbb{R}^N = \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{C}}\right)$ is the flexible intercept term, which is assumed to vary smoothly across the d-dimensional product graph. $\mathbf{w} \in \mathbb{R}^M$ is the learned coefficient vector which specifies the contribution of the explanatory variables to the prediction at each node. $\mathbf{e} \in \mathbb{R}^N$ is a random vector of iid Gaussian noise representing the model error. As in the two dimensional case, we can stack $\mathbf{c}$ and $\mathbf{w}$ into a single parameter vector $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta} \in \mathbb{R}^{N+M} = \begin{bmatrix} \mathbf{c} \\ \mathbf{w} \end{bmatrix} \tag{5.51}$$

Then, the probability distribution over $\mathbf{y}$ given $\boldsymbol{\theta}$ can be expressed as follows.

$$\mathbf{y} \,|\, \boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{s} \circ \left(\begin{bmatrix} \mathbf{I}_N & \mathbf{X} \end{bmatrix} \boldsymbol{\theta}\right), \mathbf{D}_{\boldsymbol{\mathcal{S}}}\right) \tag{5.52}$$

To create a prior over $\boldsymbol{\theta}$, we can combine the assumption that $\boldsymbol{\mathcal{C}}$ should vary smoothly over the graph with an L2 prior over the coefficient vector $\mathbf{w}$. This is expressed as

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M \end{bmatrix}\right) \tag{5.53}$$

The resultant posterior distribution over $\boldsymbol{\theta}$ is therefore given by

$$\boldsymbol{\theta} \,|\, \boldsymbol{\mathcal{Y}} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1} \begin{bmatrix} \mathbf{y} \\ \mathbf{X}^\top \mathbf{y} \end{bmatrix}, \widetilde{\mathbf{P}}^{-1}\right) \tag{5.54}$$

where

$$\widetilde{\mathbf{P}} \in \mathbb{R}^{(N+M)\times(N+M)} = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix} \tag{5.55}$$

As in section 4.2, we can solve the linear system $\widetilde{\mathbf{P}}^{-1} \begin{bmatrix} \mathbf{y} \\ \mathbf{X}^\top \mathbf{y} \end{bmatrix}$ using the CGM. In particular, we first define the symmetric preconditioner $\widetilde{\boldsymbol{\Psi}}$ as follows.

$$\widetilde{\boldsymbol{\Psi}} \in \mathbb{R}^{(N+M)\times(N+M)} = \begin{bmatrix} \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M \mathbf{D}_M \end{bmatrix} \tag{5.56}$$

where $\mathbf{U}_M$ and $\mathbf{D}_M$ are generated form the eigendecomposition of $\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X}$ as follows.

$$\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} = \mathbf{U}_M \boldsymbol{\Lambda}_M \mathbf{U}_M^\top, \quad \text{and} \quad \mathbf{D}_M = (\boldsymbol{\Lambda}_M + \lambda \mathbf{I}_M)^{-1/2} \tag{5.57}$$

This allows us to generate a preconditioned coefficient matrix as follows.

$$\widetilde{\boldsymbol{\Psi}}^\top \widetilde{\mathbf{P}} \widetilde{\boldsymbol{\Psi}} = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{G}}} \mathbf{U}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{U} \mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma \mathbf{I}_{NT} & \mathbf{D}_{\boldsymbol{\mathcal{G}}} \mathbf{U}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} \mathbf{U}_M \mathbf{D}_M \\ \mathbf{D}_M \mathbf{U}_M^\top \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{U} \mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{I}_M \end{bmatrix}$$

Then we instead solve the modified linear system as follows.

$$\boldsymbol{\theta}^\star = \widetilde{\mathbf{P}}^{-1} \begin{bmatrix} \mathbf{y} \\ \mathbf{X}^\top \mathbf{y} \end{bmatrix} \quad \Longrightarrow \quad \boldsymbol{\theta}^\star = \widetilde{\boldsymbol{\Psi}} \left(\widetilde{\boldsymbol{\Psi}}^\top \widetilde{\mathbf{P}} \widetilde{\boldsymbol{\Psi}} \right)^{-1} \widetilde{\boldsymbol{\Psi}}^\top \begin{bmatrix} \mathbf{y} \\ \mathbf{X}^\top \mathbf{y} \end{bmatrix}$$

5.5 GSR and KGR for green bond yield prediction

In this section, we undertake an analysis of the multiway Graph Signal Reconstruction and Kernel Graph Regression algorithms, developed in sections 5.2 and 5.3, applying them to the problem of predicting the yield of a set of green bonds over time. Green finance has recently emerged as an important asset class for funding the transition to a lower carbon economy [Peters et al., 2022]. Green bonds provide a mechanism for governments and municipalities to raise capital for large projects with specific climate or environmental sustainability objectives. In this case study, we utilise the yield time series for a number of US municipal green bonds, collected from Bloomberg, and attempt to estimate the missing values. This approach presents a novel method for estimating the theoretical price of bonds with various characteristics and for analysing the relationship between external factors, such as the federal funding rate or inflation, and yield.

The yield (to maturity) of a bond is the single discount rate that, when applied to all cash flows, results in a present value equal to the current market price [Hull, 2009]. Broadly speaking, yields represent return on investment and therefore reflect the perceived risk of a particular security, with higher yields necessary to justify riskier assets. External macroeconomic factors such as inflation, central bank interest rates, and economic uncertainty also play a key role in determining yields.

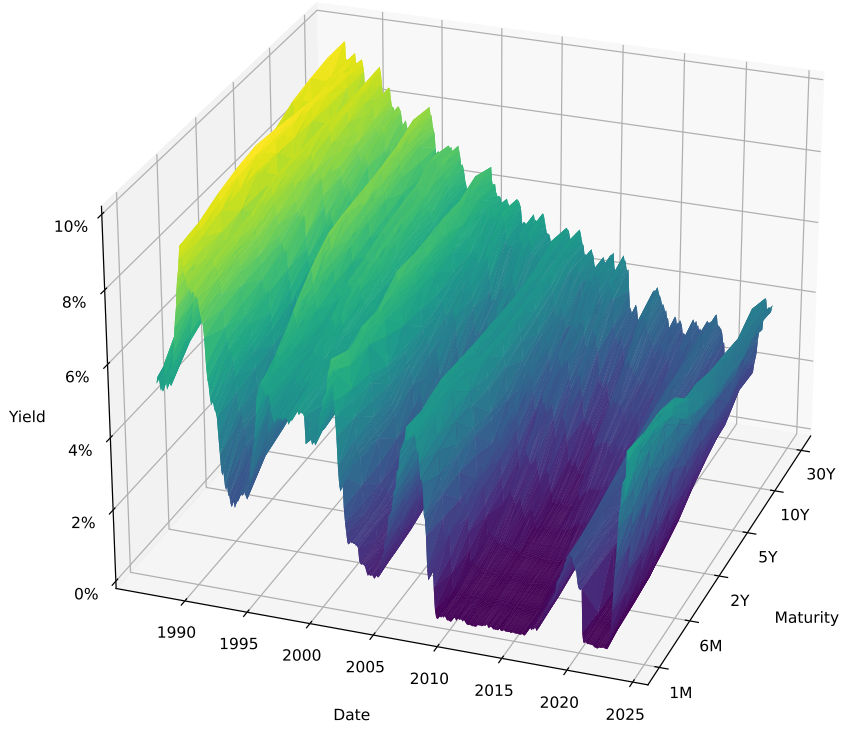


FIGURE 5.5: A 3D plot of the yield on US treasuries of various maturities over time.

One key intrinsic factor that can influence the yield of a bond is its maturity, representing the length of time before the principal must be repaid. Under normal market conditions, long-term fixed-income securities, such as 20-year bonds, typically provide higher returns than their short-term counterparts, such as 2-year bonds. This is primarily due to the greater uncertainty associated with longer-term securities, especially concerning potential fluctuations in future interest rates. For a set of equivalent bonds differing only based on their maturity length, the yield is expected to interpolate smoothly at any given time, forming the so-called yield curve. Figure 5.5 demonstrates how the yield curve for US treasuries has varied over the past 35 years.

For bonds issued by other entities, such as state authorities, numerous intrinsic factors other than maturity may affect the yield. These include the credit rating of the issuer, the tax exemption status, and, particularly relevant to green bonds, the Environmental, Social and Governance (ESG) impact of the funded project. In this case study, we frame the problem of estimating unknown yields as a multiway graph signal processing task. Specifically, we categorise the bonds based on several factors likely to influence the yield, such that each bond can be associated with a node on a four-dimensional Cartesian product graph. In particular, we take into account the following factors, each of which forms the basis of one of the factor graphs.

1. *Use of funds.* Each bond in our dataset was associated with a project category (such as water, pollution or power) and sub-category (such as waste management, habitat conservation, or solar farms). We used this information to construct a tree graph, beginning with a root node, and branching into category and sub-category nodes. The resulting tree, comprising a total of 161 nodes, is visualised in fig. 5.6. Each bond can then be associated with a specific leaf node based on their particular project category.

2. *Credit rating.* Each bond also had an assigned credit rating associated with the issuer, designated by a ratings agency. These ratings range from the highest possible, "AAA", to the lowest in our dataset, "BBB+", giving a total of eight categories. Since these ratings have a natural order, they can be represented using a simple chain graph.

3. *Maturity.* Each bond has a fixed period from its issue date to the date of maturity. Unlike treasury bonds, the green bonds we considered did not necessarily have standard maturity lengths. We thus categorised each bond by the length of this period, sorted into eight different bins, with edges corresponding to 0, 1, 2, 5, 10, 20, 30, 40 and 50 years. As with credit rating, the natural order enables us to represent this with a chain graph.

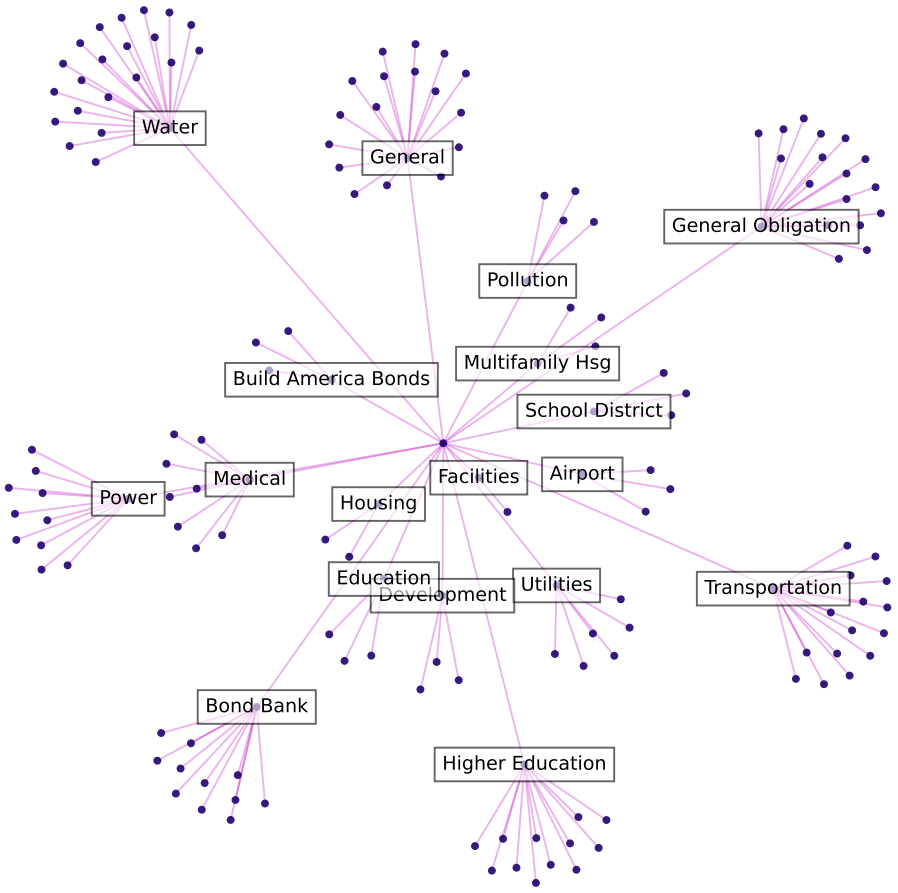


FIGURE 5.6: A representation of the graph used for categorising municipal green bonds based on the sector of the project.

4. *Tax status.* Each bond also has an associated tax status. This includes information about whether the bond was taxed at the federal and state level, as well as certain other provisions such as special exemptions for specific institutions. We represented tax status as a network, where any two codes were connected if they shared at least one property in common. Further information on the possible green bond tax statuses is presented in table 5.2, and the graph we used to connect them is depicted in fig. 5.7.

Tax Status	**Description**
Fed & St Tax-Exempt	Exempt from tax at both the federal and state level.
Fed Taxable/St Tax-Exempt	Taxable at the federal level, exempt at the state level.
Fed Tax-Exempt/St Taxable	Exempt at the federal level, taxable at the state level.
Fed BQ/St Tax-Exempt	"Bank Qualified" (special provisions for large banks) at the federal level, exempt at the state level.
Fed Tax-Exempt	Exempt at the federal level, no information about the state level.
AMT/St Tax-Exempt	"Alternate Minimum Tax" (special provision removing certain tax deductions) at the federal level, exempt at the state level.
Fed Taxable/St Taxable	Taxable at both the federal and state level.

TABLE 5.2: Further information on the possible tax status of the green bonds used in this case study.

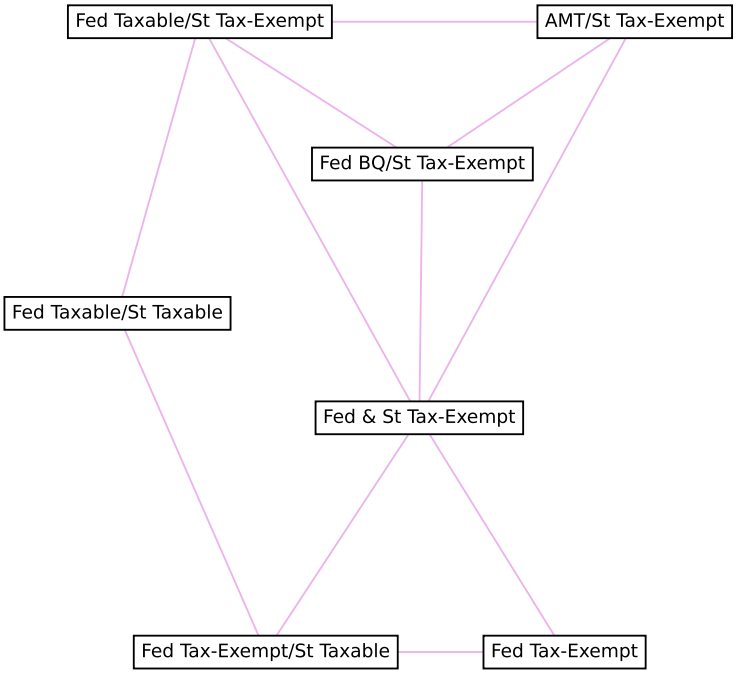


FIGURE 5.7: A representation of the graph used for categorising municipal green bonds based on the associated tax status. Two different statuses are linked via an edge if they share at least one property in common.

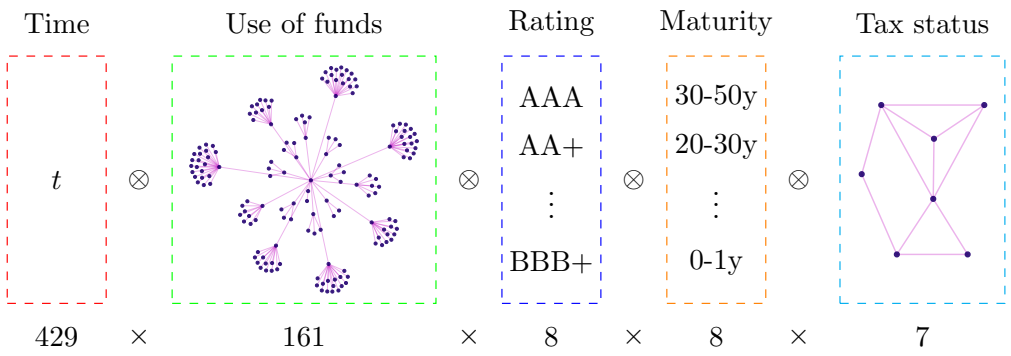


FIGURE 5.8: A visualisation of the tensor coordinates used in this application. Rating, maturity and coupon are all represented by a chain graph, while category is a tree graph as described above.

Given these factors, each bond could be associated with a particular four-dimensional coordinate: use of funds, credit rating, maturity, and tax status. With the inclusion of a time coordinate, encompassing weekly data from June 2018 to March 2023, the resulting input tensor $\mathcal{Y}$ had five distinct axes, with an overall shape of $(429, 161, 8, 8, 7)$. For the KGR model, this can be interpreted as $T = 429$ repeated measurements of a 4-dimensional multiway graph signal, and for the GSR model, it can be regarded as a single measurement of a 5-dimensional graph signal, with time also represented by a chain graph. Figure 5.8 provides a visual representation of the tensor signal, $\mathcal{Y}$, utilised in this application.

For many specific tensor coordinates, no bond yield existed, and as such the tensor $\mathcal{Y}$ was very sparse. This could be due to the fact that the bond was not trading at this particular time, or because no bond ever existed with these particular characteristics. Moreover, since the parent nodes in the graph representing the use of funds are in a sense "artificial" - serving solely to create the tree structure - no realised yield could possibly exist there. Under the framework developed in this thesis, this does not present a problem, since we can simply allocate these coordinates as containing missing data, as specified by the binary sensing tensor $\mathcal{S}$.

Additionally, there were certain bonds that shared identical characteristics, i.e., the same use of funds, rating, maturity, and tax status. In these instances, we simply selected the bond with the longest available history and randomly broke ties. In total, we used 829 bonds (out of a theoretically possible $161 \times 8 \times 8 \times 7 = 72,128$). As many bonds did not span the total 429 weeks, this left a total missingness of $m = 99.27\%$.

As previously noted, one model we employ in this case study to predict the yields is Kernel Graph Regression. This necessitates, in addition to the input tensor $\mathcal{Y}$ and the sensing tensor $\mathcal{S}$, a matrix of explanatory variables $\mathbf{X}$ with shape (T, M). Here, we use

	Training		Validation		Test	
	MSE	R^2	MSE	R^2	MSE	R^2
GSR	0.263	0.846	0.332	0.819	0.319	0.823
KGR	0.266	0.845	0.333	0.820	0.317	0.824
Ridge	0.387	0.775	0.501	0.729	0.497	0.725
Lasso	0.424	0.753	0.482	0.739	0.506	0.720

TABLE 5.3: The Mean Square Error (MSE) and R^2 statistic from the bond yield experiment are shown for both GSR and KGR. Results are reported on the training set, the validation set, and the test set.

the treasury yields over 11 different maturities, along with core inflation, the Federal Reserve fund rate, and a simple linear variable t ranging from -1 to 1, totalling $M = 14$ explanatory variables. This data was obtained from the FRED API [Federal Reserve Bank of St. Louis, 2023].

Our experiment proceeded as follows. Firstly, we randomly divided the 829 bonds into a training, validation, and test set in an 80:10:10 ratio. We then trained the GSR and KGR models on the training set, using the validation set to select the optimal hyperparameters, identified using an optimisation algorithm based on the Nelder-Mead method [Gao and Han, 2012]. In addition, we evaluated two standard linear regression techniques, namely Ridge and Lasso [Murphy, 2012]. In these scenarios, the yield at each moment in time for each bond was modelled as a linear combination of a constant term, the features comprising the matrix $\mathbf{X}$, and a one-hot encoding of each of the bond descriptors (use of funds, rating, etc). This resulted in $1 + 14 + 159 + 8 + 8 + 7 = 197$ explanatory variables for each yield. The regularisation parameter for the Ridge and Lasso models was also selected on the validation set using a straightforward line search.

The results from the experiment can be found in table 5.3. As can be observed, the GSR and KGR methods attain similar performance to each other, consistently outperforming the Ridge and Lasso techniques. Both also display a Mean Square Error (MSE) and R^2 statistic that remains relatively constant across the training, validation, and test sets, suggesting that overfitting has largely been avoided. Figures 5.9 and 5.10 showcases the output of the GSR and KGR models respectively compared with the ground truth, for nine randomly representative bonds from the test set. The shaded regions represent two

FIGURE 5.9: The predicted yield is shown from the GSR model, as compared to the ground truth, on a set of green bonds taken from the test set.

standard deviations of uncertainty, which is calculated by taking 20 samples from the posterior. (The method for producing these samples is explained in chapter 6). We can observe that both GSR and KGR make broadly similar predictions, which often correlate closely with the ground truth. However, there are instances where the predictions do systematically under or overestimate the yield, for example in the upper middle, middle left, and lower middle plots. This may indicate there was hidden information about this bond not accounted for in our explanatory variables. Further investigation into the reasons for these deviations would be valuable.

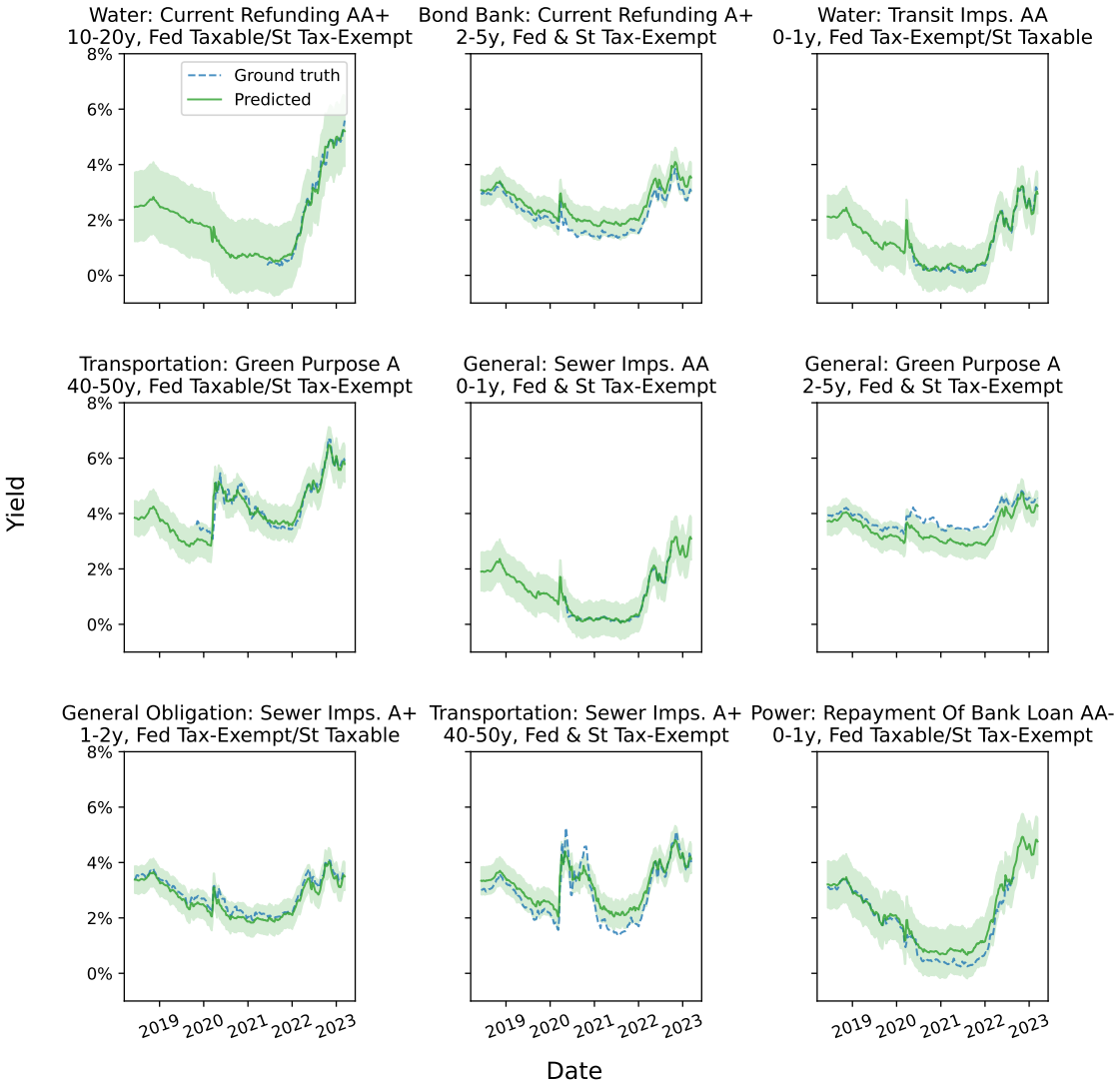


FIGURE 5.10: The predicted yield is shown from the KGR model as compared to the ground truth on a set of green bonds taken from the test set.

5.6 Conclusions

In this chapter we have been concerned with generalising several graph signal processing methods to the d-dimensional tensor setting. In section 5.1, we discussed how the Cartesian product can be generalised to include more than two factor graphs, resulting in tensor-valued multiway graph signals. In sections 5.1.3 and 5.1.4, we reviewed how GSP techniques such as the GFT can be understood for d-dimensional signals, and introduced the concept of a general anisotropic graph filter. We also discussed how d-dimensional Kronecker products can be efficiently computed and presented the PyKronecker software library for this purpose [Antonian et al., 2023].

In section 5.2, we reexamined the Graph Signal Reconstruction model developed in

chapter 3, generalising it to d dimensions. This necessitated a fresh look at the SIM and CGM methods previously developed to make them applicable for this setting. Next, in sections 5.3 and 5.4 respectively we took the Kernel Graph Regression and Regression with Network Cohesion techniques, which were developed in chapter 4, and generalised them for the multiway case. Finally, in section 5.5, we showed how the multiway KGR and GSR methods could be applied to the problem of predicting the yield of novel green bonds in a time series application. In particular, we showed that both GSR and KGR could achieve good performance, significantly out-performing the standard techniques of Ridge and Lasso regression.

Chapter 6

Signal Uncertainty: Estimation and Sampling

So far in this thesis we have introduced several Bayesian GSP models and focused on tractable methods for finding mean of the associated Gaussian posterior. In this chapter, we turn our attention to the posterior *covariance*, which presents several new interesting challenges. These challenges stem from the dimensionality the associated covariance matrix which, in each case, is the inverse of a large Kronecker-structured operator. For example, consider the two-dimensional graph signal reconstruction problem defined in section 3.2. The posterior mean has shape $(N \times T)$ and the posterior covariance matrix has shape $(NT \times NT)$. For a modest-sized problem comprising a 200-node graph measured over 365 time points, the (known) precision matrix will have over 5×10^9 elements, corresponding to over 20GB of memory with 32-bit floating-point numbers. Even if this could be held in RAM, inverting a matrix of this size would be intractable on consumer-grade hardware. This problem is only compounded when considering the tensor-valued models introduced in chapter 5, where the covariance matrices have $O(N^2)$ elements, where $N = \prod N_i$. Table 6.1 lists the posterior covariance matrices that appear in these models, along with their associated dimensionality.

Given these challenges, the goal of this chapter is to to gain insight into the uncertainty about the predicted posterior mean, whilst circumventing the need to instantiate, invert or decompose large matrices directly in memory. To this end, we specify two separate but related tasks of interest:

1. *To estimate the posterior marginal variance.* Given a large, known precision matrix with a Kronecker structure, the task is to estimate the diagonal of the associated covariance matrix (its inverse). Whilst the most straightforward approach would

Model	Covariance matrix	Element count
GSR	$\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2}\right)^{-1}$	$\prod N_i^2$
KGR	$\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}^{-2}\right)^{-1}$	$T^2 \prod N_i^2$
RNC	$\begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix}^{-1}$	$\left(\prod N_i + M\right)^2$
KG-RNC	$\begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix}^{-1}$	$\left(T \prod N_i + M\right)^2$

TABLE 6.1: The posterior covariance matrix appearing in the tensor GSR, KGR, RNC and KR-RNC models.

be simply to invert the precision matrix directly and take the diagonal, this will be too memory-intensive for most practical applications. Computing the marginal variance alone disregards information about the correlation between elements, but still gives valuable insight into prediction uncertainty whilst remaining tractable to store in memory. Therefore, the first task is to produce a scalable algorithm for estimating the diagonal of the posterior covariance matrix.

2. *To sample directly from the posterior.* For many applications, the ability to draw independent samples directly from the posterior is of significant interest. However, due to the size of the relevant matrices, direct Cholesky decomposition approaches will be intractable for most realistic use cases. Therefore, the second task is to produce an algorithm that can efficiently draw independent random samples from the posterior, whilst avoiding the memory and computational difficulties associated with the most straightforward approaches.

The chapter is structured as follows. First, in section 6.1, we consider different approaches for estimating the posterior marginal variance. In particular, we compare a recent stochastic technique for estimating a matrix diagonal [Bekas et al., 2007, Tang and Saad, 2012] with several custom regression models which leverage intuition about the network structure of the problem. We show that, by including these network effects, it is possible to significantly increase prediction accuracy compared with the aforementioned stochastic technique on a test dataset. Next, in section 6.2, we make use of a recently proposed technique for direct sampling known as Perturbation Optimisation (PO) [Vono

et al., 2022]. PO is relevant when the precision matrix can be split into a sum of matrices with a known eigenvalue decomposition which, as we will show, is always the case in our models. This allows us to draw independent samples from the posterior distribution for the same computational cost as is required to calculate the posterior mean which, as established in chapter 5, can be achieved efficiently using iterative methods.

The methods derived in this chapter are designed to be applicable to tensor-valued GSP problems covered in chapter 5. However, since the two-dimensional methods covered in chapters 3 and 4 can be considered special cases of the more general MWGSP format, these too are included under the same framework.

6.1 Estimating the posterior marginal variance

Consider the multiway Graph Signal Reconstruction problem as defined in chapter 5. Here, the goal is to estimate a smooth underlying signal, $\boldsymbol{\mathcal{F}}$, with shape $(N_1, N_2, ..., N_d)$, given a partially observed graph signal, $\boldsymbol{\mathcal{Y}}$, and binary sensing tensor, $\boldsymbol{\mathcal{S}}$, both of which have the same shape as $\boldsymbol{\mathcal{F}}$. In section 5.2, we demonstrate that the posterior distribution over $\mathbf{f} \in \mathbb{R}^N = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{F}})$, given $\mathbf{y} \in \mathbb{R}^N = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{Y}})$, is

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right) \tag{6.1}$$

where

$$\mathbf{P} \in \mathbb{R}^{N \times N} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2} \tag{6.2}$$

The goal of this section is to estimate the posterior marginal variance, i.e. diagonal of the covariance matrix $\mathbf{P}^{-1}$, without directly evaluating the matrix in full. For the sake of brevity, in this section we concentrate on accomplishing this task for the GSR model only. However, the methods discussed can easily be generalised to KGR, RNC and KG-RNC by making suitable modifications.

6.1.1 A baseline approach

As previously established, it is possible to efficiently solve a linear system of the form $\mathbf{P}^{-1}\mathbf{z}$, where $\mathbf{z}$ is an arbitrary length-N vector, using the SIM or CGM. Therefore, a straightforward approach to computing the marginal variance in full is immediately available: to compute the n-th column of $\mathbf{P}^{-1}$, we can solve the linear system $\mathbf{P}^{-1}\mathbf{e}_n$,

where $\mathbf{e}_n$ is the n-th unit basis vector. Consequently, to compute the entire marginal variance in full, we solve this system $N = \prod N_i$ times, once for each $n \in [1, 2, ..., N]$, retaining only the n-th element of the resultant solution each time.

Whilst this strategy avoids holding any $(N \times N)$ matrix directly in memory, and has lower time complexity than performing direct inversion of $\mathbf{P}$, it still presents significant computational challenges. Since obtaining each element of the marginal variance necessitates solving a linear system, the whole process takes N times longer than the computation of the mean, which itself can be intensive for large problems. It is clear that this approach is still not scalable. A natural question, then, is whether the full marginal variance can be *estimated* for a lower computational cost.

6.1.2 Matrix diagonal estimation

One notable approach for estimating the diagonal of an arbitrary matrix comes from Bekas et al. [2007], which was subsequently updated in Tang and Saad [2012]. Here, the authors propose a stochastic technique, based on the earlier work of Hutchinson [1990], which was designed to estimate the trace of a matrix. The authors define the estimator for $\text{diag}\left(\mathbf{P}^{-1}\right)$ as follows.

$$\text{diag}\left(\mathbf{P}^{-1}\right) \approx \left(\sum_{r=1}^{R} \mathbf{v}_r \circ \mathbf{P}^{-1}\mathbf{v}_r\right) \oslash \left(\sum_{k=1}^{s} \mathbf{v}_r \circ \mathbf{v}_r\right) \tag{6.3}$$

Here, $\oslash$ denotes element-wise division, and $\circ$ is the Hadamard product (element-wise multiplication) as before. In the original specification, the vectors $\{\mathbf{v}_r\}$ have entries of ± 1 which are drawn uniformly at random. This estimator converges to the true diagonal of $\mathbf{P}^{-1}$ as $R \to \infty$ [Bekas et al., 2007]. For this technique to be scalable, it is necessary for the system $\mathbf{P}^{-1}\mathbf{v}$ to be solved efficiently for an arbitrary vector $\mathbf{v}$ (which it can in our case, using the SIM or CGM). Furthermore, the algorithm requires $\mathbf{P}^{-1}\mathbf{v}$ to be computed for R different values of $\mathbf{v}_r$. Therefore the overall complexity of this approach is proportional to R. For this estimator to be better than the baseline approach outlined in section 6.1.1, we must reach an acceptable level of accuracy for $R \ll N$.

6.1.3 A supervised learning approach

In this subsection we introduce the key contribution of the first half of this chapter. The basic idea is as follows. Using the approach outlined in section 6.1.1, we can compute the 'true' variance at a limited number, Q, of the elements by solving the linear system

$\mathbf{P}^{-1}\mathbf{e}_n$, and retaining only the n-th element of the resultant vector, for a set of elements $\mathcal{Q}$. Next, we *predict* the the marginal variance at all the other elements not in the set $\mathcal{Q}$. If we can construct a set of artificial explanatory variables associated with each element, then this becomes a standard supervised learning problem.

Since variance is strictly positive, we choose to predict the *log*-variance rather than the variance itself. This transformation changes the prediction range from the positive values of $[0, \infty]$ to the full range of $[-\infty, \infty]$, which is more suitable for standard regression techniques. We define the target variable, denoted as $\boldsymbol{\Omega}$, as follows.

$$\boldsymbol{\Omega} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d} = \text{ten}_{\text{RM}}\left(\log\left(\text{diag}^{-1}\left(\mathbf{P}^{-1}\right)\right)\right) \tag{6.4}$$

Note that here we have introduced the notation $\text{diag}^{-1}(\cdot)$, which denotes a function mapping the diagonal of an $(N \times N)$ matrix into a length-N vector. Therefore, the whole expression can be understood as taking the diagonal of the covariance matrix $\mathbf{P}^{-1}$, performing an element-wise logarithm, and transforming this into a tensor, $\boldsymbol{\Omega}$, of shape $(N_1, N_2, ..., N_d)$ in row major order. The objective of the regression algorithms discussed in this section is to predict the tensor $\boldsymbol{\Omega}$.

To make this prediction, we first compute a subset of the elements of $\boldsymbol{\Omega}$. Since $\boldsymbol{\Omega}$ is an order-d tensor, its elements are indexed by a length-d integer vector, $\mathbf{n}$. Therefore we can describe the indices that we choose to compute by the set $\mathcal{Q} = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_Q\}$. Additionally, we define the binary tensor $\boldsymbol{\mathcal{Q}}$, which also indicates which elements of $\boldsymbol{\Omega}$ have been computed.

$$\boldsymbol{\mathcal{Q}}_{\mathbf{n}} = \begin{cases} 1 & \text{if } \mathbf{n} \in \mathcal{Q} \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

For each index $\mathbf{n}$, the corresponding element of $\boldsymbol{\Omega}$ can be computed as

$$\boldsymbol{\Omega}_{\mathbf{n}} = \text{ten}_{\text{RM}}\left(\log\left(\mathbf{P}^{-1}\mathbf{e}_n\right)\right)_{\mathbf{n}} \tag{6.6}$$

For a vector index $\mathbf{n} = [n_1, n_2, ..., n_d]$, the corresponding unit vector $\mathbf{e}_n$ is

$$\mathbf{e}_n = \mathbf{e}_{n_1} \otimes \mathbf{e}_{n_2} \otimes \ldots \otimes \mathbf{e}_{n_d} = \bigotimes_{i=1}^{d} \mathbf{e}_{n_i} \tag{6.7}$$

	Description	**Representation**
$\boldsymbol{\mathcal{X}}_{:,1}$	Constant	$\mathbf{1}_{(N_1,N_2,\ldots,N_d)}$
$\boldsymbol{\mathcal{X}}_{:,2}$	Missing observations	$\boldsymbol{\mathcal{S}}'$
$\boldsymbol{\mathcal{X}}_{:,3}$	Missing obs filtered	$\text{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \text{GFT}\left(\boldsymbol{\mathcal{X}}_{:,2}\right)\right)$
$\boldsymbol{\mathcal{X}}_{:,4}$	Diagonal of $\mathbf{H}$	$\text{ten}_{\text{RM}}\left(\text{diag}^{-1}\left(\mathbf{H}\right)\right)$
$\boldsymbol{\mathcal{X}}_{:,5}$	Diagonal of $\mathbf{H}^2$	$\text{ten}_{\text{RM}}\left(\text{diag}^{-1}\left(\mathbf{H}^2\right)\right)$
$\boldsymbol{\mathcal{X}}_{:,6}$	N-neighbours	$\text{ten}_{\text{RM}}\left(\mathbf{A}\mathbf{1}_N\right)$
$\boldsymbol{\mathcal{X}}_{:,7}$	N-neighbours filtered	$\text{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \text{GFT}\left(\boldsymbol{\mathcal{X}}_{:,6}\right)\right)$

TABLE 6.2: The explanatory variables used to predict the posterior log-variance. Note that $\boldsymbol{\mathcal{X}}_{:,4}$ and $\boldsymbol{\mathcal{X}}_{:,5}$ have an efficient computation - see theorem A.7

The linear system $\mathbf{P}^{-1}\mathbf{e}_n$ can then be solved efficiently using either the tensor SIM or CGM as discussed in sections 5.2.1 and 5.2.2. We refer to the process of computing these elements of $\boldsymbol{\Omega}$ as *querying*. The result, after Q queries, is a partial observation of the matrix $\boldsymbol{\Omega}$, where Q of its entries have been filled in, and the rest are set to zero. We refer to this tensor as $\boldsymbol{\Omega}_Q$.

$$\boldsymbol{\Omega}_Q = \boldsymbol{\mathcal{Q}} \circ \boldsymbol{\Omega} \tag{6.8}$$

As mentioned, we also need a set of explanatory variables to help predict the unknown elements of $\boldsymbol{\Omega}$. In particular, every element, $\mathbf{n}$, should have an associated feature vector, $\mathbf{x}_\mathbf{n}$, which should have some explanatory power over $\boldsymbol{\Omega}_\mathbf{n}$. In this way, the problem becomes a regular supervised learning task where the goal is to predict $\boldsymbol{\Omega}$ given the labelled training pairs $\{\boldsymbol{\Omega}_\mathbf{n}, \mathbf{x}_\mathbf{n}\}_{\mathbf{n}\in\mathcal{Q}}$. If each feature vector has length k, the explanatory variables as a whole can be described by the tensor $\boldsymbol{\mathcal{X}}$ with shape $(N_1, ..., N_d, k)$. The question then becomes how to construct a tensor $\boldsymbol{\mathcal{X}}$ which is likely to explain the posterior marginal variance well.

By inspecting eq. (6.2), it is clear that the marginal variance can only depend on $\boldsymbol{\mathcal{S}}$, and $\mathbf{H}$, since these are the only matrices appearing in the posterior covariance (γ can be considered a constant for the purpose of this problem). $\mathbf{H}$ itself also depends on the graph filter used and the structure of the graph. Therefore, we should choose features which capture different aspects of these objects.

In the following, we use $k = 7$ different descriptors which are summarised in table 6.2. For the features that include a filtering operation ($\boldsymbol{\mathcal{X}}_{:,3}$ and $\boldsymbol{\mathcal{X}}_{:,7}$), we use the same filter

parameters that were used in the original GSR specification. As in the RNC model of section 5.4, we can combine these tensor variables together into a single matrix $\mathbf{X}$ as follows.

$$\mathbf{X} \in \mathbb{R}^{N\times 7} = \left[\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,1}\right) \quad \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,2}\right) \quad \dots \quad \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,7}\right)\right] \tag{6.9}$$

6.1.3.1 Solving with Ridge Regression

The first strategy we use to predict the posterior log-variance is simply ridge regression. Here, we model the posterior log-variance $\boldsymbol{\Omega}$ as a linear combination of the features given in table 6.2, with the optimal weighting, $\mathbf{w}$, learned from the observed data. In particular, the predicted value of $\boldsymbol{\Omega}$, which we label $\boldsymbol{\Omega}^\star$ is given simply by

$$\boldsymbol{\Omega}^\star = \text{ten}_{\text{RM}}\left(\mathbf{X}\mathbf{w}^\star\right)$$

where

$$\mathbf{w}^\star = \left(\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{Q}}} \mathbf{X} + \lambda \mathbf{I}_N\right)^{-1} \mathbf{X}^\top \text{vec}_{\text{RM}}\left(\boldsymbol{\Omega}_Q\right)$$

Note that since $\mathbf{D}_{\boldsymbol{\mathcal{Q}}} = \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Q}}\right)\right)$ is a sparse diagonal matrix, the product $\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{Q}}} \mathbf{X}$ can be computed efficiently with $O(k^2 Q)$ operations. This model includes a regularisation parameter λ which must be set ahead of time (or potentially learned via cross validation).

6.1.3.2 Solving with RNC

The next strategy we use to predict the marginal variance is tensor Regression with Network Cohesion (RNC), as set out in section 5.4. This extends ridge regression by including a flexible intercept term, $\boldsymbol{\mathcal{C}}$, which is assumed to be smooth with respect to the graph topology. Since we expect that posterior uncertainty should propagate via closely connected nodes, it is reasonable to assume that $\boldsymbol{\Omega}$ varies smoothly over the network. In this case, the predicted value for $\boldsymbol{\Omega}$ is given by

$$\boldsymbol{\Omega}^\star = \boldsymbol{\mathcal{C}}^\star + \text{ten}_{\text{RM}}\left(\mathbf{X}\mathbf{w}^\star\right) \tag{6.10}$$

The optimal values for $\boldsymbol{\mathcal{C}}^\star$ and $\mathbf{w}^\star$ can be computed using the same methodology as described in section 5.4. As such, they are given by

$$\begin{bmatrix} \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{C}}^\star\right) \\ \mathbf{w}^\star \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{Q}}} + \gamma \mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mathcal{Q}}}\mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{Q}}} & \mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{Q}}} \mathbf{X} + \lambda \mathbf{I}_7 \end{bmatrix}^{-1} \begin{bmatrix} \text{vec}_{\text{RM}}\left(\boldsymbol{\Omega}_Q\right) \\ \mathbf{X}^\top \text{vec}_{\text{RM}}\left(\boldsymbol{\Omega}_Q\right) \end{bmatrix} \tag{6.11}$$

Once again, this linear system can be solved efficiently using the SIM or CGM.

6.1.3.3 Ridge regression with learned filter parameters

The final novel method we consider for prediction of the posterior log-variance is based on the empirical observation that the predictions made by ridge regression can be improved by adjusting the parameters that characterise the graph filter used to construct features $\boldsymbol{\mathcal{X}}_{:,3}$ and $\boldsymbol{\mathcal{X}}_{:,7}$. However, it is difficult to tell ahead of time the direction and magnitude of the optimal adjustment. For this reason, we propose a new estimator for $\boldsymbol{\Omega}$ that simultaneously learns the coefficients multiplying each feature in table 6.2, as well as the filter parameters characterising $\boldsymbol{\mathcal{X}}_{:,3}$ and $\boldsymbol{\mathcal{X}}_{:,7}$, which can vary independently. This can be expressed as follows.

$$\boldsymbol{\Omega}^\star = \text{ten}_{\text{RM}}\left(\mathbf{X}(\boldsymbol{\beta}^\star)\mathbf{w}^\star\right) \tag{6.12}$$

Note that now $\mathbf{X}$ is a function of the filter parameters $\boldsymbol{\beta}$. In order to find the optimal values for both $\mathbf{w}^\star$ and $\boldsymbol{\beta}^\star$, we use the following loss function.

$$\xi(\mathbf{w}, \boldsymbol{\beta}) = ||\boldsymbol{\Omega}_Q - \boldsymbol{\mathcal{Q}} \circ \text{ten}_{\text{RM}}\left(\mathbf{X}(\boldsymbol{\beta})\mathbf{w}\right)||_F^2 + \lambda \left(||\mathbf{w}||^2 + ||\boldsymbol{\beta} - \boldsymbol{\beta}_0||^2\right) \tag{6.13}$$

where $\boldsymbol{\beta}_0$ represents the parameters characterising the original graph filter in the base GSR problem. Minimising the expression in eq. (6.13) is in general a non-convex optimisation problem. However, a reasonable initial estimate for $\mathbf{w}$ is easily attainable by solving the ridge regression problem. From there, only minor adjustments are necessary for significant improvement in predictive performance. Therefore, off-the-shelf nonlinear optimisation algorithms such as quasi-Newton methods like BFGS can be readily applied [Fletcher, 2013].

6.1.4 Query strategies

So far in this discussion we have not addressed the question of how to select which elements of $\boldsymbol{\Omega}$ to query. The simplest approach is to uniformly select samples at random. However, although unbiased, this can lead to a high variance estimator, particularly when the query size is very small. A preferable approach is to select elements in a principled way, such that the variance of the estimator is reduced for a small sample size. This is likely to be achieved when the associated features of the samples are representative of the population at large.

Here we make use of a simple but effective query strategy. First, cluster the elements $\{\mathbf{n}\}$ into $K \leq Q$ groups based on the data matrix $\mathbf{X}$. Any clustering algorithm can be used here: a simple and fast option is K-means [Hartigan and Wong, 1979, MacQueen, 1967]. Next, generate a query set $\mathcal{Q}$ by randomly choosing a new data point from each cluster in order. This simple approach is made explicit in algorithm 7.

Algorithm 7 Querying based on representative samples

Input: Number of clusters K
Input: Number of queries Q
Input: Data matrix $\mathbf{X} \in \mathbb{R}^{N \times 7}$

Group data into K randomly ordered cluster sets $\{\mathcal{C}_k\}_{k=1}^{K}$ using K-means algorithm
Create empty query set $\mathcal{Q}$
for q from 1 to Q **do**
    Select next non-empty cluster set $\mathcal{C}$
    Choose member n_q from cluster set $\mathcal{C}$ at random
    Add member n_q to query set $\mathcal{Q}$
    Remove member n_q from cluster set $\mathcal{C}$
end for

Output: Query set $\mathcal{Q}$

6.1.5 Comparison and analysis

In order to compare the performance of the various techniques discussed in this section, we ran the following simple experiment which was setup as follows. First, a product graph was generated by taking the Cartesian product of three random connected graphs, with 10, 15 and 20 nodes respectively, resulting in a graph with $10 \times 15 \times 20 = 3000$ total nodes. Next, we generated a random binary sensing tensor $\boldsymbol{\mathcal{S}}$ of shape $(10, 15, 20)$ by setting elements to zero or one uniformly at random. Finally, we chose a anisotropic diffusion graph filter with $\boldsymbol{\beta} = [0.5, 0.7, 0.6]$, and set γ to one. The task, then, was to

predict the diagonal of $(\mathbf{D}_{\boldsymbol{\mathcal{S}}}+\gamma\mathbf{H}^{-2})^{-1}$ given the values as specified. Given the relatively small size of the problem, we could compute the true value of $\boldsymbol{\Omega}$ by running the baseline algorithm as discussed in section 6.1.1 over 3000 elements.

Next, we compared the performance of each of the algorithms discussed in this section. They were the traditional diagonal estimation algorithm of Bekas et al. [2007] (Bekas) outlined in section 6.1.2, Ridge Regression (RR), as outlined in section 6.1.3.1, Regression with Network Cohesion (RNC) as outlined in section 6.1.3.2, and finally ridge regression with Learned Filter Parameters (LFP), as outlined in section 6.1.3.3. For each strategy, we measured the performance of the estimator by computing the total square error of the predicted variance (not the log-variance), and the R^2 statistic of the predicted variance. To be explicit, these were calculated as follows.

$$\text{Square error} = ||\exp(\boldsymbol{\Omega}) - \exp(\boldsymbol{\Omega}^\star)||_F^2 \tag{6.14}$$

$$R^2 = 1 - \frac{||\exp(\boldsymbol{\Omega}) - \exp(\boldsymbol{\Omega}^\star)||_F^2}{||\exp(\boldsymbol{\Omega}) - \overline{\exp(\boldsymbol{\Omega})}||_F^2} \tag{6.15}$$

where the exponential is understood as element-wise, and $\overline{\exp(\boldsymbol{\Omega})}$ means the mean of $\exp(\boldsymbol{\Omega})$ across all elements. For the case of the Bekas estimator, there was no need to exponentiate the prediction, since it estimates the posterior variance directly, not the log-variance.

For each of these different estimators, we measured the performance via these two metrics across a range of values of Q, i.e. for different numbers of queries. In the case of the Bekas estimator, we measured the performance for different values of R [see eq. (6.3)]. Since each query requires solving the linear system $\mathbf{P}^{-1}\mathbf{e}_n$, and each iteration of the Bekas algorithm requires solving the linear system $\mathbf{P}^{-1}\mathbf{v}_r$, these two measures have a direct correspondence in terms of total compute time.

For the case of RR, RNC and LFP, we also compared the performance when using a random query strategy, and the representative sampling query strategy as outlined in algorithm 7. The results are shows in fig. 6.1. On the x-axis, we show the number of queries (number of Bekas iterations) as a percentage of the total number of elements, i.e. Q/N (or R/N for Bekas).

The results show that the three supervised learning-based strategies (RR, RNC and LFP) all significantly out-perform Bekas on this synthetic dataset. Moreover, when coupled with the representative sampling strategy of algorithm 7, all three achieve an R^2 statistic of around 0.99 when only 1% of the elements of $\boldsymbol{\Omega}$ are queried. These results clearly

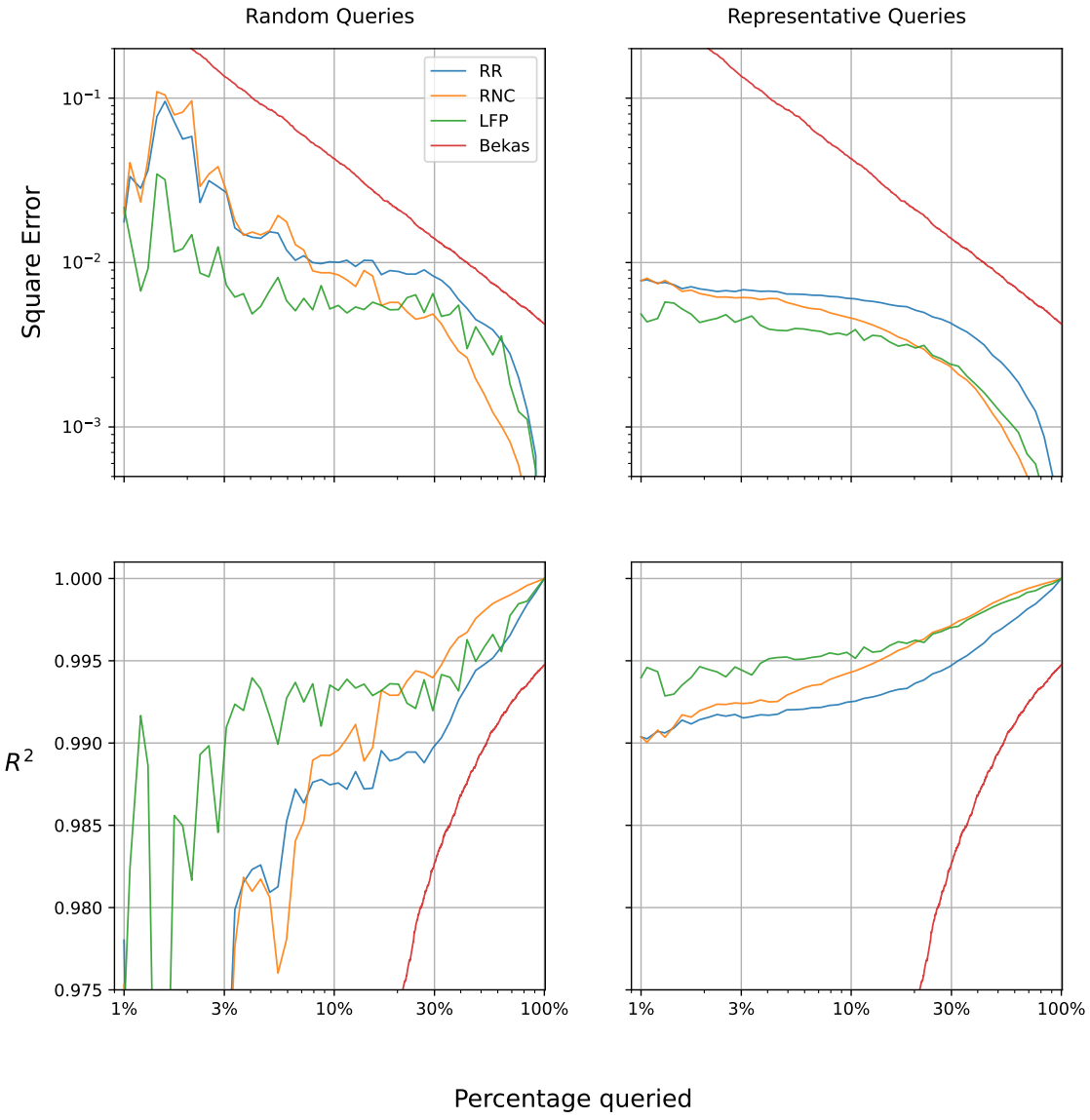


FIGURE 6.1: The performance of the various algorithms for predicting the posterior log-variance is shown as a function of the percentage of $\boldsymbol{\Omega}$ that has been queried. The top row shows the total square error of the posterior variance, and the bottom row shows the R^2 statistic. The left column shows results for random queries and the right column show the results when using representative queries as outlined in algorithm 7. Since the Bekas technique does make use of the active learning strategy, it is repeated in the left and right columns for reference.

demonstrate the value of this sampling strategy, especially when the number of queries is low. On this dataset, algorithm 7 has a significant positive impact on performance up to a query percentage of around 3%, beyond which the difference is less pronounced.

Of the three supervised learning-based techniques, LFP tends to display the strongest performance at low query percentages, for both random and representative query strategies. RR and RNC have similar performance in this domain, with RNC improving more as the query percentage increases.

Overall, these results demonstrate that high accuracy predictions (R^2 of roughly 0.99) can be achieved for a very low query number relative to the total number of nodes N, especially when combined with algorithm 7. While limited to our graph signal processing models, the supervised learning strategy is clearly more effective than the more general technique of Bekas et al. [2007] on this dataset.

6.2 Posterior Sampling

In this section we move on to the second aim of this chapter, which is to produce an algorithm capable of drawing independent samples directly from the posterior. Let us again begin with the tensor-valued Graph Signal Reconstruction problem. Recall that the posterior distribution is

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right) \tag{6.16}$$

where

$$\mathbf{P} \in \mathbb{R}^{N\times N} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{H}^{-2} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-2}\mathbf{U}^\top \tag{6.17}$$

The most straightforward approach would be to perform Cholesky decomposition directly on the covariance matrix $\mathbf{P}^{-1}$, such that $\mathbf{P}^{-1} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix. Every positive-definite matrix (and thus also every valid covariance matrix) has a unique Cholesky decomposition [Horn and Johnson, 2012]. Independent samples can then be drawn according to

$$\mathbf{f}_i = \mathbf{P}^{-1}\mathbf{y} + \mathbf{L}\mathbf{z}_i \tag{6.18}$$

where $\mathbf{z}_i \in \mathbb{R}^N$ is a random vector with components that are independent standard normal variates which can be generated, for example, by using the Box-Muller transform [Box and Muller, 1958]. Alternatively, if we can perform eigen-decomposition on the covariance matrix into $\mathbf{P}^{-1} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$, then samples can be drawn according to

$$\mathbf{f}_i = \mathbf{P}^{-1}\mathbf{y} + \mathbf{U}\boldsymbol{\Lambda}^{1/2}\mathbf{z}_i \tag{6.19}$$

Both these techniques assume that the covariance matrix $\mathbf{P}^{-1}$ can be directly decomposed in a reasonable amount of time. However, in the case of the GSR problem, $\mathbf{P}^{-1}$ is

not directly available. We instead have a representation of $\mathbf{P}$ in terms of a diagonal and Kronecker structured operator, as given in eq. (6.18). For even a modestly-sized GSR problem, this makes these approaches numerically infeasible due to both computational cost and memory footprint.

6.2.1 Perturbation optimization (PO)

Perturbation Optimisation (PO) is a technique for sampling from high-dimensional Gaussian distributions when the precision matrix $\mathbf{P}$ is of the form

$$\mathbf{P} = \sum_{k=1}^{K} \mathbf{M}_k^\top \mathbf{R}_k^{-1} \mathbf{M}_k \tag{6.20}$$

where $\{\mathbf{R}_k\}$ are a set of alternative covariance matrices from which it is easier to draw samples from, and $\{\mathbf{M}_k\}$ are a set of arbitrary full-rank square matrices [Orieux et al., 2012, Papandreou and Yuille, 2010]. The PO algorithm gives a method for drawing samples from the distribution $\mathcal{N}\left(\mathbf{0}, \mathbf{P}^{-1}\right)$, which is achieved by completing the following steps.

1. Step P (Perturbation): For k from 1 to K, draw samples $\boldsymbol{\eta}_k$ from $\mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right)$
2. Step O (Optimisation): Compute $\mathbf{f}$ as the minimiser of the criterion

$$J(\mathbf{f}) = \sum_{k=1}^{K} \left(\boldsymbol{\eta}_k - \mathbf{M}_k \mathbf{f}\right)^\top \mathbf{R}_k^{-1} \left(\boldsymbol{\eta}_k - \mathbf{M}_k \mathbf{f}\right)$$

Once the optimisation step has been completed, the resultant vector, $\mathbf{f}$, will be distributed according to $\mathcal{N}\left(\mathbf{0}, \mathbf{P}^{-1}\right)$. This can be trivially extended to non-centered distributions by adding the mean to the each sample. It's important to note that for PO to be an effective algorithm, it necessitates efficient resolution of the optimisation step. As we will see, in our case we can exploit the Kronecker structure of the relevant matrices to accomplish this step for the same computational cost as computing the mean.

6.2.2 PO for Graph Signal Reconstruction

In the case of graph signal reconstruction, we have that

$$\mathbf{P} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{U} \mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-2} \mathbf{U}^\top$$

Comparing this to the form required for PO given in eq. (6.20), we can substitute the following values. Note that, since $\mathbf{D}_{\boldsymbol{\mathcal{S}}}$ is diagonal and binary, $\mathbf{D}_{\boldsymbol{\mathcal{S}}}^2 = \mathbf{D}_{\boldsymbol{\mathcal{S}}}$.

$$\mathbf{M}_1 = \mathbf{D}_{\boldsymbol{\mathcal{S}}}, \quad \mathbf{R}_1 = \mathbf{I}_N, \quad \mathbf{M}_2 = \sqrt{\gamma}\,\mathbf{U}^\top, \quad \mathbf{R}_2 = \mathbf{D}_{\boldsymbol{\mathcal{G}}}^2$$

The random samples $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$ can be generated by setting $\boldsymbol{\eta}_1 = \mathbf{z}_1$ and $\boldsymbol{\eta}_2 = \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{z}_2$ where $\mathbf{z}_1$ and $\mathbf{z}_2$ are independently drawn from a multivariate standard normal distribution. This gives the following optimisation objective $J(\mathbf{f})$.

$$J(\mathbf{f}) = (\mathbf{z}_1 - \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{f})^\top (\mathbf{z}_1 - \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{f}) + \left(\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{z}_2 - \sqrt{\gamma}\,\mathbf{U}^\top\mathbf{f}\right)^\top \mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-2}\left(\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{z}_2 - \sqrt{\gamma}\,\mathbf{U}^\top\mathbf{f}\right) \quad (6.21)$$

Taking the derivative with respect to $\mathbf{f}$, setting the result equal to zero, and solving for $\mathbf{f}$ gives

$$\mathbf{f} = \mathbf{P}^{-1}\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-1}\mathbf{z}_2\right) \quad (6.22)$$

This generates random vectors $\mathbf{f}$ distributed according to $\mathcal{N}\left(\mathbf{0}, \mathbf{P}^{-1}\right)$. To transform this into a distribution with mean $\mathbf{P}^{-1}\mathbf{y}$, we can simply add this term on.

$$\mathbf{f} = \mathbf{P}^{-1}\left(\mathbf{y} + \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-1}\mathbf{z}_2\right) \quad (6.23)$$

However, in its current form, the solution presents two issues. Firstly, the multiplication by the matrix $\mathbf{P}^{-1}$ may result in an ill-conditioned linear system. Secondly, $\mathbf{z}_2$ is multiplied by the matrix $\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-1}$, which, for certain filters like the bandlimited filter that maps some values to zero, may map some values to infinity. Both problems can be addressed by introducing the symmetric preconditioner $\boldsymbol{\Psi}$ as discussed in section 5.2.2. As previously stated, this is given by

$$\boldsymbol{\Psi} = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \quad (6.24)$$

This transforms the system into

$$\begin{aligned}\mathbf{f} &= \boldsymbol{\Psi}\left(\boldsymbol{\Psi}^\top \mathbf{P}\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^\top\left(\mathbf{y} + \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-1}\mathbf{z}_2\right)\\ &= \boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^\top\mathbf{y} + \boldsymbol{\Psi}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)\end{aligned} \tag{6.25}$$

where

$$\mathbf{Q} = \boldsymbol{\Psi}^\top\mathbf{P}\boldsymbol{\Psi} = \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_N$$

With this preconditioner in place, the linear system $\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^\top\mathbf{y} + \boldsymbol{\Psi}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)$ can then be solved efficiently using the CGM. For the sake of completeness, we now provide a brief proof that the expression given in eq. (6.25) generates samples from the correct distribution.

Theorem 6.1. *Consider the following expression for the vector* $\mathbf{f} \in \mathbb{R}^N$*.*

$$\mathbf{f} = \boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^\top\mathbf{y} + \boldsymbol{\Psi}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)$$

If $z_1, z_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ *, then* $\mathbf{f}$ *is distributed according to*

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right)$$

Proof. To prove this statement it suffices to show that the expected value of $\mathbf{f}$ is $\mathbf{P}^{-1}\mathbf{y}$, and that the covariance is $\mathbf{P}^{-1}$. Note that the following identities hold.

$$\begin{aligned}&(a)\ \ \mathbf{Q} = \boldsymbol{\Psi}^\top\mathbf{P}\boldsymbol{\Psi}, && (b)\ \ \mathbf{P} = \boldsymbol{\Psi}^{-\top}\mathbf{Q}\boldsymbol{\Psi}^{-1},\\ &(c)\ \ \mathbf{Q}^{-1} = \boldsymbol{\Psi}^{-1}\mathbf{P}^{-1}\boldsymbol{\Psi}^{-\top}, && (d)\ \ \mathbf{P}^{-1} = \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^\top\end{aligned}$$

Let us begin with the expected value.

$$\mathrm{E}\left[\mathbf{f}\right] = \mathrm{E}\left[\boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^\top\mathbf{y} + \boldsymbol{\Psi}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)\right]$$

Since $\mathrm{E}\left[\mathbf{z}_1\right] = \mathrm{E}\left[\mathbf{z}_2\right] = \mathbf{0}$, these two terms can be dropped from the expression, leaving

$$\mathrm{E}\left[\mathbf{f}\right] = \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top}\mathbf{y}$$

Making use of identity (d), this is then

$$\mathrm{E}\left[\mathbf{f}\right] = \mathbf{P}^{-1}\mathbf{y}$$

Next, let us compute the covariance.

$$\mathrm{Cov}\left[\mathbf{f}\right] = \mathrm{Cov}\left[\boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^{\top}\mathbf{y} + \boldsymbol{\Psi}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)\right]$$

Consider the three terms within the inner brackets. Since the first has no stochastic component, it does not contribute to the covariance and can therefore dropped.

$$\begin{aligned}
\mathrm{Cov}\left[\mathbf{f}\right] &= \mathrm{Cov}\left[\boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1 + \sqrt{\gamma}\,\mathbf{z}_2\right)\right] \\
&= \boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\mathrm{Cov}\left[\boldsymbol{\Psi}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1\right] + \mathrm{Cov}\left[\sqrt{\gamma}\,\mathbf{z}_2\right]\right)\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top} \\
&= \boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\,\mathrm{Cov}\left[\mathbf{z}_1\right]\mathbf{D}_{\boldsymbol{\mathcal{S}}}\boldsymbol{\Psi} + \gamma\,\mathrm{Cov}\left[\mathbf{z}_2\right]\right)\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top} \\
&= \boldsymbol{\Psi}\mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\boldsymbol{\Psi} + \gamma\,\mathbf{I}_N\right)\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top} \\
&= \boldsymbol{\Psi}\mathbf{Q}^{-1}\mathbf{Q}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top} \\
&= \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top} \\
&= \mathbf{P}^{-1}
\end{aligned}$$

We have therefore shown that $\mathrm{E}\left[\mathbf{f}\right] = \mathbf{P}^{-1}\mathbf{y}$ and that $\mathrm{Cov}\left[\mathbf{f}\right] = \mathbf{P}^{-1}$, completing the proof.

□

Given the expression for $\mathbf{f}$ outlined in eq. (6.25), we have a straightforward procedure for sampling from the posterior distribution of the GSR model, as detailed in algorithm 8. It's important to note that the primary computational load of this algorithm stems from solving the linear system using the CGM. Consequently, the computational complexity of drawing a single sample from the posterior distribution is equivalent to solving one linear system via the CGM.

Algorithm 8 Draw one sample from the GSR posterior distribution

Sample $\mathbf{z}_1$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_N)$
Sample $\mathbf{z}_2$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_N)$
$\mathbf{a} \leftarrow \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{z}_1$
$\mathbf{b} \leftarrow \sqrt{\gamma}\,\mathbf{z}_2$
$\mathbf{c} \leftarrow \mathbf{a} + \mathbf{b}$
$\boldsymbol{\Psi} \leftarrow \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}$
$\mathbf{Q} \leftarrow \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_N$
$\mathbf{f} \leftarrow \mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^\top\mathbf{y} + \mathbf{c}\right)$ (solve with the CGM)
$\mathbf{f} \leftarrow \boldsymbol{\Psi}\mathbf{f}$

Output: $\mathbf{f}$

6.2.3 Drawing samples for KGR, RNC and KG-RNC

So far in this section, we have shown how perturbation optimisation can be used to draw samples from the posterior distribution of the GSR model. Using the same basic strategy, we can also derive algorithms for the KGR, RNC and KG-RNC models.

6.2.3.1 PO with KGR

In the case of Kernel Graph Regression (KGR), there is little change from the GSR algorithm. Recall how, as discussed in section 4.1.2, KGR bares a strong algebraic correspondence to GSR. In particular, the posterior distribution over the underlying graph signal $\mathbf{f}$ is given by

$$\mathbf{f} \sim \mathcal{N}\left(\bar{\mathbf{P}}^{-1}\mathbf{y}, \bar{\mathbf{P}}^{-1}\right) \tag{6.26}$$

where

$$\bar{\mathbf{P}} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\bar{\mathbf{U}}\mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}}^{-2}\bar{\mathbf{U}}^\top \tag{6.27}$$

The values for $\bar{\mathbf{U}}$ and $\mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}}$ are modified from the GSR model as follows.

$$\bar{\mathbf{U}} = \mathbf{V} \otimes \mathbf{U}, \quad \text{and} \quad \mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}} = \operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}\left(\bar{\boldsymbol{\mathcal{G}}}\right)\right) \tag{6.28}$$

with the elements of $\boldsymbol{\mathcal{G}}$ given by

$$\boldsymbol{\mathcal{G}}_{[t,\mathbf{n}]} = g\big(\boldsymbol{\lambda}(\mathbf{n});\, \boldsymbol{\beta}\big)\sqrt{\lambda_t^{(K)}} \tag{6.29}$$

With these modifications in place, we can make use of the same algorithm developed in section 6.2.2, since all the derivations otherwise remain the same. Therefore, in order to sample from the posterior distribution of the KGR model, we can simply reuse algorithm 8, replacing all instances of $\mathbf{U}$ with $\bar{\mathbf{U}}$, and all instances of $\mathbf{D}_{\boldsymbol{\mathcal{G}}}$ with $\mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}}$.

6.2.3.2 PO with RNC

In the case of Regression with Network Cohesion (RNC), the algorithm must be adjusted. Here, the relevant posterior distribution is over the parameter vector $\boldsymbol{\theta}$, rather than the predicted signal $\mathbf{f}$. This posterior distribution is given by

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^\top\mathbf{y}\end{bmatrix}, \widetilde{\mathbf{P}}^{-1}\right) \tag{6.30}$$

where

$$\widetilde{\mathbf{P}} \in \mathbb{R}^{(N+M)\times(N+M)} = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}^{-2}\mathbf{U}^\top & \mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X}\\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix} \tag{6.31}$$

Recall that we can perform eigendecomposition on the matrix $\mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X}$ as

$$\mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X} = \mathbf{U}_M\boldsymbol{\Lambda}_M\mathbf{U}_M^\top$$

and additionally define the diagonal matrix $\mathbf{D}_M$ as

$$\mathbf{D}_M = \big(\boldsymbol{\Lambda}_M + \lambda\mathbf{I}_M\big)^{-1/2}$$

Recall also, that the preconditioner used in the case of RNC is

$$\widetilde{\boldsymbol{\Psi}} = \begin{bmatrix}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M\end{bmatrix}$$

We now present the expression that can be used to sample from the posterior distribution of $\boldsymbol{\theta}$ in theorem 6.2.

Theorem 6.2. *Consider the following expression for the vector* $\boldsymbol{\theta} \in \mathbb{R}^{N+M}$

$$\boldsymbol{\theta} = \widetilde{\boldsymbol{\Psi}}\widetilde{\mathbf{Q}}^{-1}\left(\widetilde{\boldsymbol{\Psi}}^{\top}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^{\top}\mathbf{y}\end{bmatrix} + \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{0}\\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}} & \mathbf{0}\end{bmatrix}\mathbf{z}_1 + \begin{bmatrix}\sqrt{\gamma}\,\mathbf{I}_N & \mathbf{0}\\ \mathbf{0} & \sqrt{\lambda}\mathbf{D}_M\end{bmatrix}\mathbf{z}_2\right) \tag{6.32}$$

where the matrix $\widetilde{\mathbf{Q}}$ *is given by*

$$\widetilde{\mathbf{Q}} = \widetilde{\boldsymbol{\Psi}}^{\top}\widetilde{\mathbf{P}}\widetilde{\boldsymbol{\Psi}} = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_N & \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M\\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{I}_M\end{bmatrix}$$

If $z_1, z_2 \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{(N+M)}\right)$ *, then*

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^{\top}\mathbf{y}\end{bmatrix}, \widetilde{\mathbf{P}}^{-1}\right)$$

Proof. As in theorem 6.1, it suffices to show that

$$\mathbf{E}\left[\boldsymbol{\theta}\right] = \widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^{\top}\mathbf{y}\end{bmatrix}, \quad \text{and} \quad \mathrm{Cov}\left[\boldsymbol{\theta}\right] = \widetilde{\mathbf{P}}^{-1}$$

Note that the following identities hold.

$$(a)\ \widetilde{\mathbf{Q}} = \widetilde{\boldsymbol{\Psi}}^{\top}\widetilde{\mathbf{P}}\widetilde{\boldsymbol{\Psi}}, \qquad (b)\ \widetilde{\mathbf{P}} = \widetilde{\boldsymbol{\Psi}}^{-\top}\widetilde{\mathbf{Q}}\widetilde{\boldsymbol{\Psi}}^{-1},$$
$$(c)\ \widetilde{\mathbf{Q}}^{-1} = \widetilde{\boldsymbol{\Psi}}^{-1}\widetilde{\mathbf{P}}^{-1}\widetilde{\boldsymbol{\Psi}}^{-\top}, \qquad (d)\ \widetilde{\mathbf{P}}^{-1} = \widetilde{\boldsymbol{\Psi}}\widetilde{\mathbf{Q}}^{-1}\widetilde{\boldsymbol{\Psi}}^{\top}$$

First, let us consider the expected value. Since $\mathbf{E}\left[\mathbf{z}_1\right] = \mathbf{E}\left[\mathbf{z}_2\right] = \mathbf{0}$, these terms do not contribute to the expected value of $\boldsymbol{\theta}$. Therefore, $\mathbf{E}\left[\boldsymbol{\theta}\right]$ is given by

$$\begin{aligned}\mathbf{E}\left[\boldsymbol{\theta}\right] &= \mathbf{E}\left[\widetilde{\boldsymbol{\Psi}}\widetilde{\mathbf{Q}}^{-1}\widetilde{\boldsymbol{\Psi}}^{\top}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^{\top}\mathbf{y}\end{bmatrix}\right]\\ &= \widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathbf{y}\\ \mathbf{X}^{\top}\mathbf{y}\end{bmatrix}\end{aligned}$$

For the covariance, let us consider each of the three terms within the brackets of eq. (6.32) in isolation. Since the first term has no stochastic component, the covariance of this is zero. For the second term, the covariance is given by

$$\begin{aligned}
\operatorname{Cov}\left[\begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \end{bmatrix}\mathbf{z}_1\right] &= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \end{bmatrix}\operatorname{Cov}[\mathbf{z}_1]\begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \end{bmatrix}^\top \\
&= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_M\mathbf{\Lambda}_M\mathbf{D}_M \end{bmatrix}
\end{aligned}$$

Now consider the covariance of the third term.

$$\begin{aligned}
\operatorname{Cov}\left[\begin{bmatrix} \sqrt{\gamma}\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \sqrt{\lambda}\mathbf{D}_M \end{bmatrix}\mathbf{z}_2\right] &= \begin{bmatrix} \sqrt{\gamma}\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \sqrt{\lambda}\mathbf{D}_M \end{bmatrix}\operatorname{Cov}[\mathbf{z}_2]\begin{bmatrix} \sqrt{\gamma}\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \sqrt{\lambda}\mathbf{D}_M \end{bmatrix} \\
&= \begin{bmatrix} \gamma\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{D}_M^2 \end{bmatrix}
\end{aligned}$$

Note that these two matrices summed together give

$$
\begin{aligned}
&\begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_M\mathbf{\Lambda}_M\mathbf{D}_M \end{bmatrix} + \begin{bmatrix} \gamma\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{D}_M^2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} + \gamma\,\mathbf{I}_N & \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{D}_M\mathbf{\Lambda}_M\mathbf{D}_M + \lambda\mathbf{D}_M^2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} + \gamma\,\mathbf{I}_N & \mathbf{D}_{\mathcal{G}}\mathbf{U}^\top\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^\top\mathbf{X}^\top\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{I}_M \end{bmatrix} \\
&= \widetilde{\mathbf{Q}}
\end{aligned}
$$

Note that here we have used the fact that $\mathbf{D}_M\mathbf{\Lambda}_M\mathbf{D}_M + \lambda\mathbf{D}_M^2 = \mathbf{I}_M$. This is true since

$$
\begin{aligned}
\mathbf{D}_M\mathbf{\Lambda}_M\mathbf{D}_M + \lambda\mathbf{D}_M^2 &= \mathbf{\Lambda}_M\mathbf{D}_M^2 + \lambda\mathbf{D}_M^2 \\
&= (\mathbf{\Lambda}_M + \lambda\mathbf{I}_M)\mathbf{D}_M^2 \\
&= \mathbf{D}_M^{-2}\mathbf{D}_M^2 \\
&= \mathbf{I}_M
\end{aligned}
$$

Therefore, the covariance of the entire expression is given by

$$
\begin{aligned}
\operatorname{Cov}[\boldsymbol{\theta}] &= \widetilde{\mathbf{\Psi}}\widetilde{\mathbf{Q}}^{-1}\widetilde{\mathbf{Q}}\widetilde{\mathbf{Q}}^{-1}\widetilde{\mathbf{\Psi}}^\top \\
&= \widetilde{\mathbf{\Psi}}\widetilde{\mathbf{Q}}^{-1}\widetilde{\mathbf{\Psi}}^\top \\
&= \widetilde{\mathbf{P}}^{-1}
\end{aligned}
$$

We have therefore shown that

$$
\mathbf{E}[\boldsymbol{\theta}] = \widetilde{\mathbf{P}}^{-1}\begin{bmatrix} \mathbf{y} \\ \mathbf{X}^\top\mathbf{y} \end{bmatrix}, \quad \text{and} \quad \operatorname{Cov}[\boldsymbol{\theta}] = \widetilde{\mathbf{P}}^{-1}
$$

completing the proof.

□

Given this expression for $\boldsymbol{\theta}$ stated in eq. (6.32), a simple procedure for sampling from the posterior distribution of the RNC model is available. This is given in algorithm 9.

Algorithm 9 Draw one sample from the RNC posterior distribution

Sample $\mathbf{z}_1$ from $\mathcal{N}\left(\mathbf{0}, \mathbf{I}_{(N+M)}\right)$

Sample $\mathbf{z}_2$ from $\mathcal{N}\left(\mathbf{0}, \mathbf{I}_{(N+M)}\right)$

$$\mathbf{a} \leftarrow \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathcal{S}} & \mathbf{0} \\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{D}_{\mathcal{S}} & \mathbf{0} \end{bmatrix} \mathbf{z}_1$$

$$\mathbf{b} \leftarrow \begin{bmatrix} \sqrt{\gamma}\,\mathbf{I}_N & \mathbf{0} \\ \mathbf{0} & \sqrt{\lambda}\mathbf{D}_M \end{bmatrix} \mathbf{z}_2$$

$$\mathbf{c} \leftarrow \mathbf{a} + \mathbf{b}$$

$$\boldsymbol{\Psi} \leftarrow \begin{bmatrix} \mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M \end{bmatrix}$$

$$\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{D}_{\mathcal{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} + \gamma\,\mathbf{I}_N & \mathbf{D}_{\mathcal{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{U}\mathbf{D}_{\mathcal{G}} & \mathbf{I}_M \end{bmatrix}$$

$$\boldsymbol{\theta} \leftarrow \mathbf{Q}^{-1}\left(\boldsymbol{\Psi}^{\top}\begin{bmatrix} \mathbf{y} \\ \mathbf{X}^{\top}\mathbf{y} \end{bmatrix} + \mathbf{c}\right) \quad \text{(solve with the CGM)}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\Psi}\boldsymbol{\theta}$$

Output: $\boldsymbol{\theta}$

6.2.3.3 PO with KG-RNC

For the case of Kernel Graph Regression with Network Cohesion (KG-RNC), the situation is algebraically very similar to that of RNC. Just as the algorithm for sampling from the KGR model posterior can be obtained by making a small modification to the GSR case as discussed in section 6.2.3.1, the same is true of KG-RNC, with only a small modification of the RNC algorithm. In particular, the posterior distribution over the parameter vector $\hat{\boldsymbol{\theta}}$ is given by

$$\hat{\boldsymbol{\theta}} \sim \mathcal{N}\left(\hat{\mathbf{P}}^{-1}\begin{bmatrix} \mathbf{y} \\ \mathbf{X}^{\top}\mathbf{y} \end{bmatrix}, \hat{\mathbf{P}}^{-1}\right) \tag{6.33}$$

where

$$\hat{\mathbf{P}} \in \mathbb{R}^{(N+M)\times(N+M)} = \begin{bmatrix} \mathbf{D}_{\mathcal{S}} + \gamma\bar{\mathbf{U}}\mathbf{D}_{\bar{\mathcal{G}}}^{-2}\bar{\mathbf{U}}^{\top} & \mathbf{D}_{\mathcal{S}}\mathbf{X} \\ \mathbf{X}^{\top}\mathbf{D}_{\mathcal{S}} & \mathbf{X}^{\top}\mathbf{D}_{\mathcal{S}}\mathbf{X} + \lambda\mathbf{I}_M \end{bmatrix} \tag{6.34}$$

Note that this is the same posterior distribution as in the RNC model, with the objects $\mathbf{U}$ and $\boldsymbol{\mathcal{G}}$ replaced by $\bar{\mathbf{U}}$ and $\bar{\boldsymbol{\mathcal{G}}}$. These have the same values as in section 6.2.3.1. In particular

$$\bar{\mathbf{U}} = \mathbf{V} \otimes \mathbf{U}, \quad \text{and} \quad \mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}} = \operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}\left(\bar{\boldsymbol{\mathcal{G}}}\right)\right) \tag{6.35}$$

with the elements of $\boldsymbol{\mathcal{G}}$ given by

$$\boldsymbol{\mathcal{G}}_{[t,\mathbf{n}]} = g\big(\boldsymbol{\lambda}(\mathbf{n});\, \boldsymbol{\beta}\big)\sqrt{\lambda_t^{(K)}} \tag{6.36}$$

Therefore, in order to draw samples from the distribution given in eq. (6.33), we can simply reuse algorithm 9, replacing every instance of $\mathbf{U}$ with $\bar{\mathbf{U}}$ and every instance of $\boldsymbol{\mathcal{G}}$ with $\bar{\boldsymbol{\mathcal{G}}}$.

6.3 Variance Estimation: Prediction vs Sampling

So far in this chapter we have addressed estimation of the posterior marginal variance and direct sampling from the posterior as separate objectives. In section 6.1, we proposed several methods based on a supervised learning approach, aiming to predict the entire marginal variance (i.e., the diagonal of the posterior covariance matrix), given our ability to "query" a subset of these matrix elements. On the other hand, in section 6.2, we developed methods for direct sampling from the posterior. However, it's evident that a valid approach to estimating the marginal variance would be to simply draw samples, using the techniques discussed in section 6.2, and compute the sample variance as an estimator for the population variance. This naturally raises the question of whether this approach is superior to the supervised learning approaches discussed in section 6.1.

Let us take the Graph Signal Reconstruction model as the primary example. If we have already computed the posterior mean, $\mathbf{f}^\star = \mathbf{P}^{-1}\mathbf{y}$, then an unbiased estimator for the posterior marginal variance based on K samples $\{\mathbf{f}_k\}_{k=1}^K$ is clearly

$$\boldsymbol{\sigma}_K^2 = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{f}_k - \mathbf{f}^\star)^2 \tag{6.37}$$

where the exponent is understood as element-wise. In order not to hold all K samples in memory simultaneously, we can write this in recursive form as

$$\boldsymbol{\sigma}_{K+1}^2 = \frac{1}{K+1}\left(K\boldsymbol{\sigma}_K^2 + (\mathbf{f}_{k+1} - \mathbf{f}^\star)^2\right) \tag{6.38}$$

The relative error of this estimator follows from the properties of the χ^2 distribution, and is thus given by

$$r = \sqrt{\frac{2}{K}} \tag{6.39}$$

meaning $K = 2/r^2$ samples are necessary to reduce the relative error to a fraction r. For example, 50 samples would be required to reduce the relative error to 20%. However, one attractive feature of this estimator is that its accuracy is independent of the size of the graph. Whether this is also the case for the supervised learning estimators is not immediately clear.

6.3.1 Experiments

In order to investigate the properties of these two approaches to estimating the posterior marginal variance, that is, the supervised learning approaches discussed in section 6.1, and the sample variance approach highlighted in section 6.3, we ran the following experiment. Graphs, comprising the Cartesian product of three random fully-connected sub-graphs, were created in increasing size. For each, we generated a random pattern of missingness $\boldsymbol{\mathcal{S}}$ by uniformly sampling each element from $\{0, 1\}$, and tested the three supervised learning approaches (with active learning), the Bekas diagonal estimator, and the sample-based approach for predicting the posterior marginal variance. As with the experiments in section 6.1.5, we used a diffusion filter with $\boldsymbol{\beta} = [0.5, 0.7, 0.6]$ and set $\gamma = 1$. Each estimator was given a budget of 50 linear system solves, and its prediction performance was measured by the R^2 score. The result are show in fig. 6.2.

As visible, RR, RNC and LFP all have similar performance over the range of graph sizes, with R^2 never dropping below 0.9. Of the three, LFP once again achieves the highest accuracy, although the difference compared to RR and RNC is fairly small. The sample-based estimator achieves an R^2 of around 0.85-0.9 across the range of graph sizes, with performance dropping as low as 0.7 but mostly remaining within this bound. Its performance is also between that of the Bekas diagonal estimator and the three supervised learning techniques over the whole range.

While the supervised learning estimators are fairly consistent, there is some evidence that their performance is dropping as node number increases. Unfortunately, it is difficult to test this hypothesis since, in order to tets the accuracy scores, it is necessary to solve

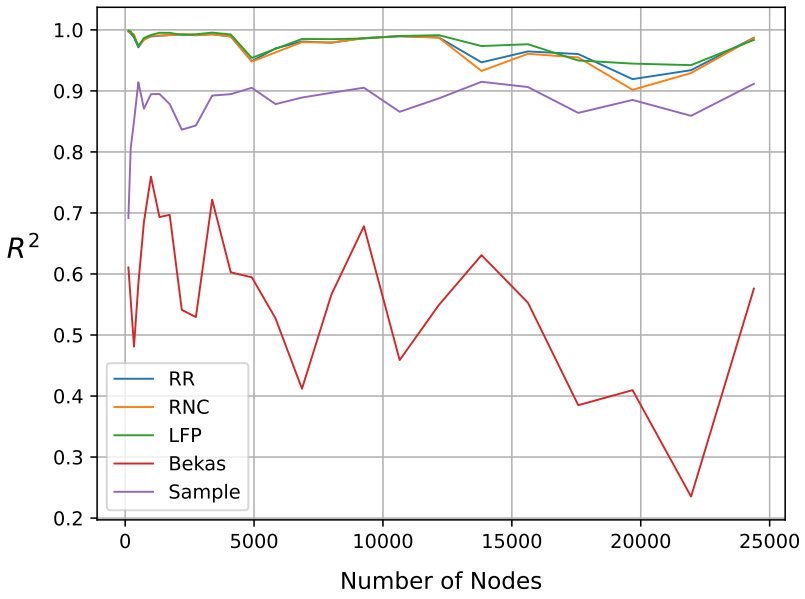


FIGURE 6.2: The predictive performance, as measured by R^2, is shown for the three supervised learning strategies detailed in section 6.1.3, the Bekas diagonal estimator of section 6.1.2, and the sampling strategy of section 6.3.

for the entire posterior variance meaning the number of linear system solves required to run this experiment increases rapidly as the number of nodes increases.

6.4 Conclusions

In this chapter, we have presented a detailed examination of various methods related to the posterior covariance of the tensor GSP models introduced in chapter 5. In the case of GSR, KGR, RNC and KG-RNC, the posterior distribution is a high-dimensional Gaussian with a precision matrix, $\mathbf{P}$, characterised by Kronecker-structured operators. The implicit dimensionality of $\mathbf{P}$ makes direct decomposition and inversion operations challenging, necessitating the use of computationally efficient techniques that avoid the time and memory costs associated with more straightforward approaches.

We concentrated on two distinct tasks. First, in section 6.1, we aimed to estimate the posterior marginal variance, i.e. the diagonal of $\boldsymbol{\Sigma} = \mathbf{P}^{-1}$. In particular, we compared a recent stochastic algorithm devised by Bekas et al. [2007] against three supervised learning-based techniques. This was achieved by first noting that it is possible to compute (or "query") a subset of the elements of $\mathbf{P}^{-1}$ directly by solving the linear system $\mathbf{P}^{-1}\mathbf{e}_n$, where $\mathbf{e}_n$ is a unit basis vector. Next, by creating a set of artificial explanatory variables associated with each element, the task of estimating the non-queried elements

was framed in terms of a traditional supervised learning task. We proposed three separate estimators, namely Ridge Regression (RR), Regression with Network Cohesion (RNC), and Learned Filter Parameters (LFP), for prediction. On a synthetic dataset, all three were capable of achieving high accuracy ($R^2 > 0.99$) for a small fraction of the computational cost of computing the marginal variance in full. We also combined this with an active learning strategy which increased prediction accuracy further, especially at low query numbers.

In section 6.2, we focused on the task of drawing samples directly from the posterior distribution. To achieve this, we made use of a technique known as Perturbation Optimisation (PO) [Orieux et al., 2012], which takes advantage of the special structure of the posterior precision matrix to reduce the computational cost to one linear system solve per sample. By coupling this with the symmetric preconditioner $\mathbf{\Psi}$, we were able to remedy the numerical difficulties associated with the algorithm in its basic form to produce an efficient procedure using the CGM. We demonstrated that this procedure yielded the correct posterior distribution for the GSR, RNC, KGR, and KG-RNC algorithms in their general tensor form.

Finally, in section 6.3, we explored whether the sampling algorithm developed in section 6.2 could serve as an alternative method for estimating the posterior marginal variance. By assessing the prediction accuracy over product graphs of increasing size, we found that the supervised learning techniques typically outperform the sampling strategy. However, for very large graphs, sampling may be preferable, as the relative error is independent of the graph's size, while the supervised learning techniques show some signs of performance degradation as the number of nodes grows. Further research into this question would be valuable.

Chapter 7

Reconstruction and Regression with Binary-Valued Graph Signals

Up to this point in this thesis, our attention has been focused on reconstruction and regression models for real-valued graph signals, as discussed in chapters 3 to 6. In this chapter, we shift our attention to scenarios where the signal of interest is binary-valued, which can be employed to describe node classification tasks conducted over networks. For example, consider the task of predicting whether each user in a social network will engage with a specific online advertisement. Assuming it is shown to a subset of the user population, we can represent the outcome (click/no click) as a partially observed binary graph signal across the network. It is reasonable to assume that closely connected users will exhibit a more similar propensity to click than distantly connected users; or in other words, we might expect that the click probability varies smoothly with respect to the topology of the graph. Hence, incorporating this relational information into a predictive task could potentially improve its accuracy.

This example represents a binary classification task over the network, since there are only two possible outcomes. However, there may also be situations in which each node must be classified into one of $C > 2$ groups. Again, it may be reasonable to assume that the relative class probabilities vary smoothly over the graph. Figures 7.1 and 7.2 give visual representations of binary and multiclass classification tasks over a network, respectively.

Within the graph signal processing community, a significant amount of research has been dedicated to the study of real-valued signals, since they are naturally well-suited

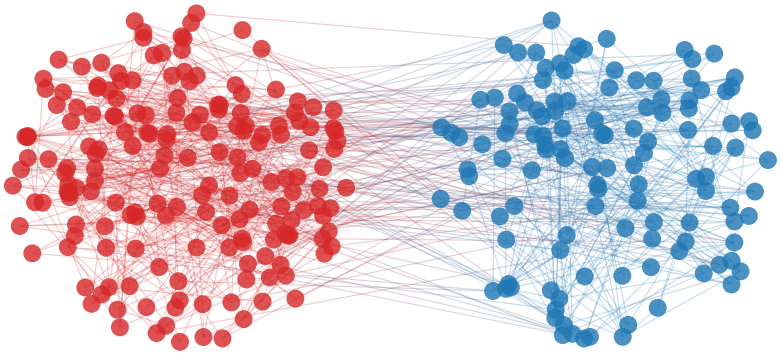


FIGURE 7.1: An example visualisation of a binary classification task over a network, with red and blue representing the true binary labels of each node.

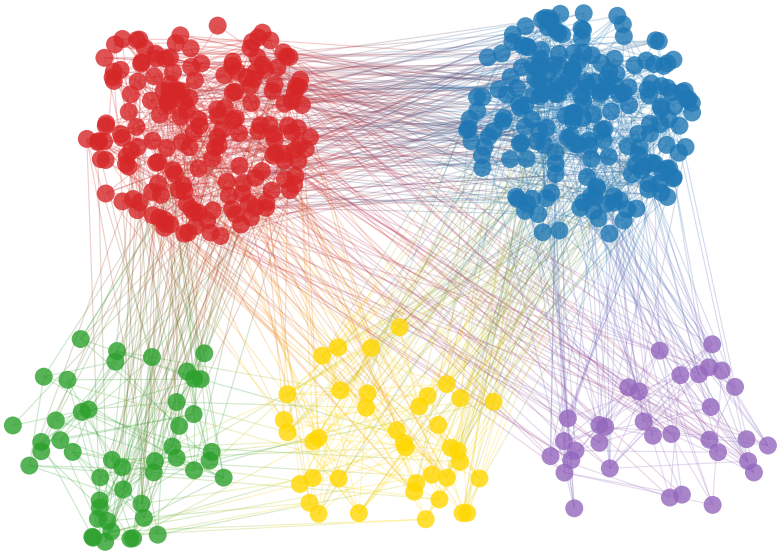


FIGURE 7.2: An example visualisation of a multiclass classification task over a network. Here, the five distinct colours represent the true class label of each node.

to spectral analysis via the GFT and representation in terms of Gaussian random fields. By contrast, binary-valued graph signals, as well as other discrete or non-Gaussian distributions, have received notably less attention. In the machine learning community, binary graph signals have been studied in the context of semi-supervised learning, see for example Kondor and Lafferty [2002], Zhu et al. [2003]. However, there, the graph is usually constructed by considering a distance metric in a feature space, and the focus is on maximising the utility of unlabelled data using algorithms such as label propagation (see, for example Zhang et al. [2017]). The GSP community, by contrast, focuses on statistical processes that occur over intrinsically graph-structured objects using spectral methods. While some work on binary reconstruction and regression from this perspective has occurred (see, for example Tran et al. [2020]), many of the prominent GSP algorithms and frameworks are yet to be generalised to binary data. In this chapter, we build on some of the models already developed in this thesis (GSR, KGR and RNC)

to define associated "logistic" reconstruction and regression algorithms (L-GSR, L-KGR and L-RNC). These models are targeted towards applications of interest to the GSP community and are applicable to multiway graph signals in an arbitrary number of dimensions. This means they are particularly suited to multivariate applications such as hyperspectral image processing, graph time series etc.

This chapter is structured as follows. First, in section 7.1, we define a model for Logistic Graph Signal Reconstruction (L-GSR) on a Cartesian product graph. Here, we describe how the real-valued multiway GSR framework we developed in chapter 5 can be modified to produce a statistical model to describe d-dimensional binary graph signals. By making use of the Conjugate Gradient Method (CGM) in conjunction with the Iteratively Reweighted Least Squares (IRLS) algorithm, we demonstrate how to efficiently estimate the underlying class probabilities at each node. Next, in section 7.2, we adapt this algorithm to accommodate multiclass classification problems. This necessitates the introduction of a new class of Kronecker operator, which appears in the preconditioned coefficient matrix. Using Gerschgorin's circle theorem we are able provide guarantees for the CGM convergence by ascertaining an upper bound for its condition number. In section 7.3, adapt the Kernel Graph Regression (KGR) and Regression with Network Cohesion (RNC) models developed in chapter 4, and adapt them for the task of binary and multiclass node classification, given additional explanatory variables. Finally, in section 7.4, we explore the behaviour of the L-GSR and L-RNC models on an image segmentation task. In particular, we analyse the interpretation of the hyperparameters γ and β in the context of the logistic models, and provide intuition for how these can be set in practice. We also discuss some of the convergence issues associated with the IRLS algorithm and provide practical solutions for mediating them.

7.1 Logistic Graph Signal Reconstruction (L-GSR)

In this section we define a model for the reconstruction of binary signals existing on the nodes of a d-dimensional Cartesian product graph, which we term Logistic Graph Signal Reconstruction (L-GSR). As before, the graph is characterised by d graph Laplacians $\left\{\mathbf{L}^{(i)}\right\}_{i=1}^{d}$, with the total Laplacian given by their Kronecker sum (see section 5.1). The partially observed graph signal, $\boldsymbol{\mathcal{Y}}$, is an order-d tensor of shape $(N_1, N_2, .., N_d)$, with binary-valued elements representing the two classes. This is accompanied by another binary tensor, $\boldsymbol{\mathcal{S}}$, of the same shape which, as in previous chapters, contains the information about which elements of $\boldsymbol{\mathcal{Y}}$ were observed by holding ones where successful observations were made and zeros elsewhere. Where no observation was made (i.e. $\boldsymbol{\mathcal{S}}_{\mathbf{n}} = 0$), the corresponding element $\boldsymbol{\mathcal{Y}}_{\mathbf{n}}$ is set to zero by default. The goal is to predict

the value of the graph signal at elements where no observation was made using solely the topology of the graph. As such, the input data for this problem can be summarised as follows.

$$\text{input data} = \left\{ \boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d}, \;\; \boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d}, \;\; \left\{ \mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i} \right\}_{i=1}^{d} \right\}$$

In the following, we assume each observed element of the tensor $\boldsymbol{\mathcal{Y}}$ follows a Bernoulli distribution, according to some underlying latent probability tensor $\boldsymbol{\mathcal{M}}$, which specifies the expected value of the outcome at each node. All other elements of $\boldsymbol{\mathcal{Y}}$, which were not in the set of observed elements $\mathcal{S}$, are set to zero with probability one. This can be summarised by the following statistical model.

$$\boldsymbol{\mathcal{Y}}_{\mathbf{n}} \sim \begin{cases} \text{Bern}\left(\boldsymbol{\mathcal{M}}_{\mathbf{n}}\right) & \text{if } \; \mathbf{n} \in \mathcal{S} \\ \text{Bern}\left(0\right) & \text{otherwise} \end{cases} \tag{7.1}$$

We refer to $\boldsymbol{\mathcal{M}} \in [0,1]^{N_1,\ldots,N_d}$ as the mean tensor. It has the same shape as $\boldsymbol{\mathcal{Y}}$ and elements contained within the interval $[0,1]$ describing the probability that the corresponding entry of $\boldsymbol{\mathcal{Y}}$ is one. For a given mean tensor, $\boldsymbol{\mathcal{M}}$, the probability of observing a binary signal $\boldsymbol{\mathcal{Y}}$ is given by the following expression.

$$p(\boldsymbol{\mathcal{Y}} \,|\, \boldsymbol{\mathcal{M}}) = \prod_{\mathbf{n} \in \mathcal{S}} \boldsymbol{\mathcal{M}}_{\mathbf{n}}^{\boldsymbol{\mathcal{Y}}_{\mathbf{n}}} \left(1 - \boldsymbol{\mathcal{M}}_{\mathbf{n}}\right)^{1 - \boldsymbol{\mathcal{Y}}_{\mathbf{n}}} \tag{7.2}$$

As such, the log-likelihood of observing a signal $\boldsymbol{\mathcal{Y}}$, which is simply the natural log of this expression, is given by

$$\begin{aligned} \log p(\boldsymbol{\mathcal{Y}} \,|\, \boldsymbol{\mathcal{M}}) &= \sum_{\mathbf{n}} \boldsymbol{\mathcal{S}}_{\mathbf{n}} \Big(\boldsymbol{\mathcal{Y}}_{\mathbf{n}} \log \boldsymbol{\mathcal{M}}_{\mathbf{n}} + (1 - \boldsymbol{\mathcal{Y}}_{\mathbf{n}}) \log \left(1 - \boldsymbol{\mathcal{M}}_{\mathbf{n}}\right) \Big) \\ &= \mathbf{s}^{\top} \left(\mathbf{y} \circ \log \boldsymbol{\mu} + (\mathbf{1} - \mathbf{y}) \circ \log \left(\mathbf{1} - \boldsymbol{\mu}\right) \right) \end{aligned} \tag{7.3}$$

where $\mathbf{s} = \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)$, $\mathbf{y} = \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)$ and $\boldsymbol{\mu} = \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{M}}\right)$. Here, the logarithm function is understood as being applied element-wise.

The goal of our L-GSR model is to estimate the value of the latent probability tensor $\boldsymbol{\mathcal{M}}$ given the partially observed binary graph signal $\boldsymbol{\mathcal{Y}}$. The key assumption that anchors the model to the graph, and avoids underspecification, is that the probability tensor

varies smoothly with respect to the topology of the Cartesian product graph. In previous chapters, when working with real-valued signals, this was achieved by setting a Gaussian prior over the latent signal $\boldsymbol{\mathcal{F}}$. However, since the elements of $\boldsymbol{\mathcal{M}}$ fall within the interval $[0, 1]$, we need a distribution that naturally supports this range. To achieve this, we follow the approach of standard logistic regression (see, for example, Murphy [2012]), by describing the probability in terms of a logistic link function. In particular, we assume that the mean tensor, $\boldsymbol{\mathcal{M}}$, is generated by applying the logistic function to a real-valued tensor graph signal $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ as follows.

$$\boldsymbol{\mathcal{M}}(\boldsymbol{\mathcal{F}}) = \frac{\mathbf{1}}{\mathbf{1} + \exp(-\boldsymbol{\mathcal{F}})} \quad \Longleftrightarrow \quad \boldsymbol{\mu}(\mathbf{f}) = \frac{\mathbf{1}}{\mathbf{1} + \exp(-\mathbf{f})} \tag{7.4}$$

Here, the exponential and division should be interpreted as element-wise. Next, we make the assumption that $\mathbf{f} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{F}})$ is smooth with respect to the topology of the graph by assigning it the following Gaussian prior.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma^{-1}\mathbf{H}^2\right) \tag{7.5}$$

As in previous chapters, the covariance matrix $\mathbf{H}^2$ is the square of a graph filter operator, derived by applying a filter function $g(\cdot)$ to the product graph Laplacian. In particular,

$$\mathbf{H} = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top$$

where $\mathbf{U}$ is the Kronecker product of each of the eigenvector matrices of the individual factor graph Laplacians, and $\mathbf{D}_{\mathbf{G}} = \text{diag}\left(\text{vec}_{\text{RM}}(\boldsymbol{\mathcal{G}})\right)$ is the diagonalised spectral scaling tensor, created by applying an isotropic or anisotropic graph filter to the corresponding eigenvalues (see sections 5.1 and 5.1.3 for details).

By applying this prior we encode the belief that $\boldsymbol{\mathcal{F}}$ is smooth with respect to the topology of the graph, which is then applied by proxy to the mean tensor $\boldsymbol{\mathcal{M}}$. In particular, $\boldsymbol{\mu}$ is distributed on $[0, 1]^N$ according to a multivariate logit-normal distribution. Note that this is distinct from the multivariate logistic-normal distribution, which is produced by applying a softmax function to a random Gaussian vector to produce vectors on a simplex [Atchinson and Shen, 1980]. Figure 7.3 gives some visual intuition for this by showing a colour-map of several smooth signals on a 2D lattice graph, along with the corresponding mean tensor $\boldsymbol{\mathcal{M}}$. As visible, the qualitative smoothness properties of $\boldsymbol{\mathcal{F}}$ broadly translate under the transformation of the logistic link function. That is, increasingly smooth real-valued signals correspond to increasingly smooth signals on the unit interval.

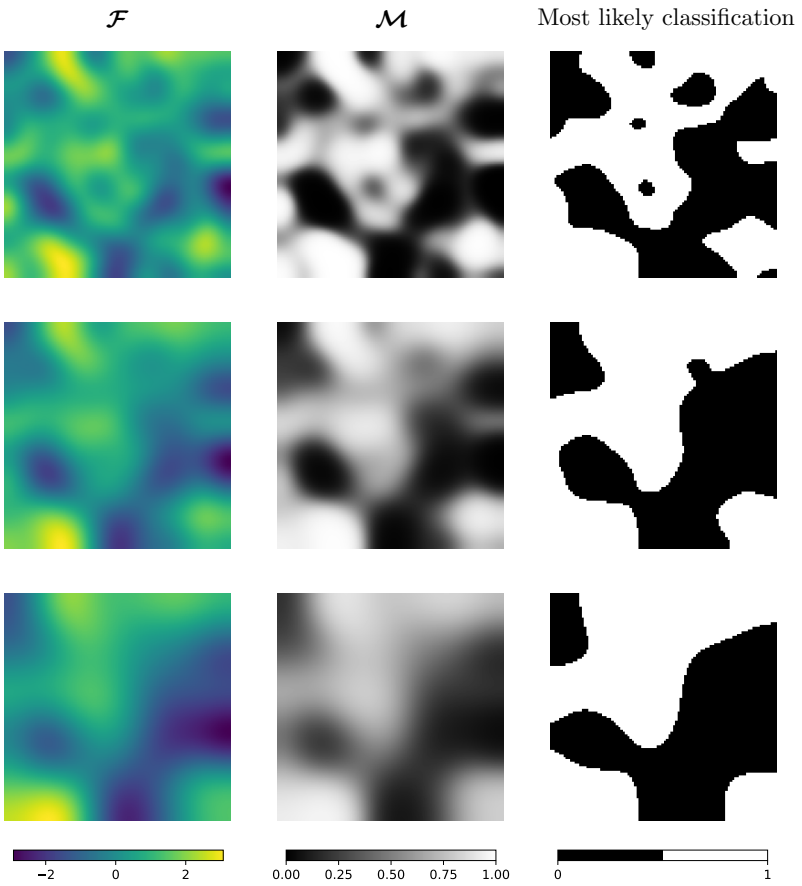


FIGURE 7.3: An example visualisation of transforming a smooth signal via the logistic link function, on a 100×100 grid of pixels ($N_1 = 100, N_2 = 100$). Left: a random smooth graph signal $\boldsymbol{\mathcal{F}}$ drawn from the distribution of eq. (7.5) with a diffusion filter. Middle: the value of the tensor $\boldsymbol{\mathcal{M}}$ found by applying the logistic link function of eq. (7.4). Right: the resultant most likely classification given by $\boldsymbol{\mathcal{M}}_{\mathbf{n}} > 0.5$ each pixel, $\mathbf{n}$. Each row shows an increasingly smooth signal by increasing the value of β characterising the graph filter.

As previously stated, the goal of the L-GSR model is to find the most likely value of $\boldsymbol{\mathcal{M}}$ given $\boldsymbol{\mathcal{Y}}$. Since we have a prior distribution over $\boldsymbol{\mathcal{F}}$, we can derive an equation for its posterior distribution given $\boldsymbol{\mathcal{Y}}$ through the application of Bayes' rule. The Maximum A Posteriori (MAP) estimator for $\boldsymbol{\mathcal{F}}$ corresponds to the value that minimises the negative log-likelihood of $\boldsymbol{\mathcal{F}} \,|\, \boldsymbol{\mathcal{Y}}$. Given the MAP estimator for $\boldsymbol{\mathcal{F}}$, we can compute the most likely mean tensor $\boldsymbol{\mathcal{M}}$ by applying eq. (7.4). To this end, we define the objective function $\xi(\mathbf{f})$, which is equal to $-\log p(\mathbf{f} \,|\, \mathbf{y})$ up to an additive constant.

$$\xi(\mathbf{f}) = -\mathbf{s}^\top \big(\mathbf{y} \circ \log \boldsymbol{\mu}(\mathbf{f}) + (\mathbf{1} - \mathbf{y}) \circ \log\left(\mathbf{1} - \boldsymbol{\mu}(\mathbf{f})\right)\big) + \frac{\gamma}{2} \mathbf{f}^\top \mathbf{H}^{-2} \mathbf{f} \tag{7.6}$$

The goal of the next section is to find an algorithm for minimising this expression with respect to $\mathbf{f}$, i.e. to find the MAP estimator for $\boldsymbol{\mathcal{F}}$, and subsequently compute the most likely value of the mean tensor $\boldsymbol{\mathcal{M}}$.

7.1.1 Solving for the MAP estimator with the IRLS algorithm

Unlike the normal models of the previous sections, no closed-form solution exists to minimise eq. (7.6), meaning we must resort to numerical methods. One standard approach is the Iteratively Reweighted Least Squares (IRLS) algorithm. The IRLS algorithm is a numerical method used to solve certain non-linear optimisation problems. It essentially works by iteratively solving a sequence of weighted least squares sub-problems until a satisfactory solution is found. As we will show, in our case, this is particularly attractive, since it is possible to leverage the properties of the Kronecker product to solve these simpler sub-problems efficiently.

In the context of maximum likelihood estimation for generalised linear models, the IRLS algorithm is equivalent to Fisher's scoring algorithm, which is a variant of the Newton-Raphson method [Nelder and Wedderburn, 1972]. Instead of using the actual Hessian matrix (second derivatives of the log-likelihood), Fisher's scoring algorithm uses its expected value, called the Fisher Information matrix. In this setting, each iteration of the IRLS algorithm involves re-weighting the data based on the current estimate of the parameters, then solving a weighted least squares problem to update the parameters, akin to the update step in Fisher Scoring [Fan et al., 2020].

Fisher's scoring algorithm begins with an initial estimate $\mathbf{f}_0$, which is then iteratively refined to reach the global minimum according to the following formula.

$$\mathbf{f}_{k+1} = \mathbf{f}_k - \mathbf{P}^{-1}(\mathbf{f}_k)\,\mathbf{g}(\mathbf{f}_k) \tag{7.7}$$

where $\mathbf{g}(\mathbf{f}) \in \mathbb{R}^N$ is the gradient of the optimisation objective $\xi(\mathbf{f})$ with respect to the vector $\mathbf{f}$, and $\mathbf{P}(\mathbf{f}) \in \mathbb{R}^{N\times N}$ is the Hessian, which is the matrix of second derivatives.

$$\mathbf{g}_n = \frac{\partial \xi(\mathbf{f})}{\partial \mathbf{f}_n}, \quad \text{and} \quad \mathbf{P}_{nm} = \frac{\partial^2 \xi(\mathbf{f})}{\partial \mathbf{f}_n \partial \mathbf{f}_m}$$

In our case, the gradient is given by

$$\mathbf{g}(\mathbf{f}) = \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}\big) + \gamma \mathbf{H}^{-2}\mathbf{f} \tag{7.8}$$

and the Hessian is given by

$$\mathbf{P}(\mathbf{f}) = \frac{\partial \mathbf{g}(\mathbf{f})}{\partial \mathbf{f}} = \mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}) + \gamma \mathbf{H}^{-2} \tag{7.9}$$

where $\mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f})$ is a diagonal matrix with the following definition.

$$\mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}) = \operatorname{diag}\big(\mathbf{s} \circ \boldsymbol{\mu}(\mathbf{f}) \circ (\mathbf{1} - \boldsymbol{\mu}(\mathbf{f}))\big) \tag{7.10}$$

A derivation of the expressions for both gradient and the Hessian in this context can be found in theorem A.8. Note that they are both evaluated at a given $\mathbf{f}$.

Consider the update formula given in eq. (7.7). Substituting in the derived values of $\mathbf{g}(\mathbf{f})$ and $\mathbf{P}(\mathbf{f})$, and using the shorthands

$$\mathbf{P}(\mathbf{f}_k) = \mathbf{P}_k, \quad \mathbf{g}(\mathbf{f}_k) = \mathbf{g}_k, \quad \boldsymbol{\mu}(\mathbf{f}_k) = \boldsymbol{\mu}_k, \quad \text{and} \quad \mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}_k) = \mathbf{D}_{\boldsymbol{\mu}}^k$$

gives

$$\begin{aligned}
\mathbf{f}_{k+1} &= \mathbf{f}_k - \mathbf{P}_k^{-1}\,\mathbf{g}_k \\
&= \mathbf{f}_k - \mathbf{P}_k^{-1}\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\boldsymbol{\mu}_k - \mathbf{y}\big) + \gamma \mathbf{H}^{-2}\mathbf{f}_k\right) \\
&= \mathbf{f}_k + \mathbf{P}_k^{-1}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}_k\big) - \mathbf{P}_k^{-1}(\gamma \mathbf{H}^{-2}\mathbf{f}_k) \\
&= \mathbf{P}_k^{-1}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{P}_k^{-1}\left(\mathbf{P}_k\mathbf{f}_k - \gamma \mathbf{H}^{-2}\mathbf{f}_k\right) \\
&= \mathbf{P}_k^{-1}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{P}_k^{-1}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{f}_k \\
&= \mathbf{P}_k^{-1}\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{f}_k\right) \\
&= \mathbf{P}_k^{-1}\mathbf{t}_k
\end{aligned}$$

where

$$\mathbf{t}_k = \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{f}_k \tag{7.11}$$

As visible, each iteration of the IRLS algorithm reduces to solving the linear system $\mathbf{P}_k^{-1}\mathbf{t}_k$, for some vector $\mathbf{t}_k$. As such, we must be able to compute a solution to this problem in a way that is maximally efficient.

7.1.2 Completing IRLS iterations with the CGM

When it comes to solving the linear system $\mathbf{P}_k^{-1}\mathbf{t}_k$, we encounter familiar challenges relating to dimensionality and conditioning. Specifically, the implicit size of $\mathbf{P}_k$ can be substantial for tensor-valued graph signals, and the inverse-squared filter matrix $\mathbf{H}^{-2}$ appearing in the definition of $\mathbf{P}_k$ [see eq. (7.9)] may suffer from severe ill-conditioning. In a similar manner to previous chapters, these issues can be overcome by employing the SIM or CGM techniques introduced in sections 3.2.2 and 3.2.3.

In this chapter, our focus will be on the CGM for two primary reasons. Firstly, as the dimensionality of tensor-valued binary graph signals increases, it is likely that only a small fraction of nodes will have valid observed data in most practical applications. As demonstrated in section 3.3.3, the CGM exhibits superior scaling properties when the input graph signal $\boldsymbol{\mathcal{Y}}$ is sparsely observed. Secondly, as discussed in section 4.2.2, the CGM is applicable to GSR, KGR and RNC problems while the SIM is not suitable for RNC models. Therefore, for simplicity, and for the sake of brevity we focus on the CGM.

Recall that the CGM seeks to solve a linear system by introducing the symmetric preconditioner $\boldsymbol{\Psi}$, for the purpose of reducing the condition number of the coefficient matrix. In the case of logistic GSR, we can obtain an alternative expression for the update formula with a reduced condition number as follows.

$$\mathbf{f}_k = \mathbf{P}_k^{-1}\mathbf{t}_k \quad \Longrightarrow \quad \mathbf{f}_k = \boldsymbol{\Psi}\mathbf{Q}_k^{-1}\boldsymbol{\Psi}^\top\mathbf{t}_k$$

where, as in the cass of standard real-valued GSR,

$$\boldsymbol{\Psi} = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}$$

By instead solving the linear system $\mathbf{Q}_k^{-1}(\boldsymbol{\Psi}^\top\mathbf{t}_k)$ using the conjugate gradient method, and then left-multiplying the result by $\boldsymbol{\Psi}$, we can obtain $\mathbf{f}_{k+1}$ with typically far fewer iterative steps than solving $\mathbf{P}^{-1}\mathbf{t}_k$ directly. In this case, $\mathbf{Q}_k$ is given by the following expression.

$$\mathbf{Q}_k = \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^\top\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_N \tag{7.12}$$

Note that this is similar to the preconditioned coefficient matrix appearing in standard GSR, with the modification that $\mathbf{D}_{\boldsymbol{\mathcal{S}}}$ has been replaced by $\mathbf{D}_{\boldsymbol{\mu}}^k$.

The conditioning of the new coefficient matrix $\mathbf{Q}_k$ is significantly improved from the original problem. While the condition number, κ, of $\mathbf{P}_k$ was potentially unbounded, $\kappa(\mathbf{Q}_k)$ is guaranteed to have a maximum value of $(0.25 + \gamma)/\gamma$. This is shown formally in theorem 7.1.

Theorem 7.1. *The condition number, κ, of the preconditioned coefficient matrix $\mathbf{Q}_k$, defined in eq. (7.12), is bounded as follows.*

$$\kappa(\mathbf{Q}_k) \leq \frac{0.25 + \gamma}{\gamma} \tag{7.13}$$

Proof. As established in section 3.3.1, the worst-case convergence rate is achieved in the limit of a weak filter, where $\mathbf{D}_{\mathcal{G}} = \mathbf{I}_N$. In this case, the condition number of $\mathbf{Q}_k$ is given by

$$\begin{aligned}
\kappa(\mathbf{Q}_k) &= \kappa\left(\mathbf{U}^\top \mathbf{D}_{\boldsymbol{\mu}}^k \mathbf{U} + \gamma \mathbf{I}_N\right) \\
&= \kappa\left(\mathbf{U}^\top \left(\mathbf{D}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N\right) \mathbf{U}\right) \\
&= \kappa\left(\mathbf{D}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N\right)
\end{aligned}$$

Since $\mathbf{D}_{\boldsymbol{\mu}}^k = \operatorname{diag}\big(\mathbf{s} \circ \boldsymbol{\mu}_k \circ (\mathbf{1} - \boldsymbol{\mu}_k)\big)$, the maximum possible value along the diagonal of $\mathbf{D}_{\boldsymbol{\mu}}^k$ will be 0.25, occurring when the corresponding element of $\boldsymbol{\mu}_k$ is $1/2$. Furthermore, since $\mathbf{s}$ is a binary vector, the smallest possible value along the diagonal is 0. Therefore, the ratio between the largest and smallest eigenvalues of $\mathbf{D}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N$ must be less than or equal to $(0.25 + \gamma)/\gamma$. □

For completeness, we now give the full algorithm for logistic graph signal reconstruction in algorithm 10. Note that, by making use of the fast Kronecker product algorithm described in section 5.1.4, the runtime complexity of the CGM step is bounded by

$$O\left(\frac{0.25 + \gamma}{\gamma} N \sum_{i=1}^{d} N_i\right)$$

The run time complexity of the overall algorithm of course depends on the convergence rate of the IRLS iterations. This is known to be super-linear, and approximately quadratic when a sufficiently accurate starting value is used [Burden and Faires, 2010]. Whilst an in-depth theoretical exploration of the IRLS algorithm for this particular application is beyond the scope of this chapter, in practice, the IRLS algorithm converges very quickly, usually taking on the order of 10 steps to achieve negligible error.

Algorithm 10 Logistic Graph Signal Reconstruction

Input: Observed binary tensor $\boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$
Input: Binary sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$
Input: Cartesian product graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$
Input: Regularisation parameter $\gamma \in \mathbb{R}^{+}$
Input: Graph filter function $g(\,\cdot\,;\boldsymbol{\beta})$

$\mathbf{y} \leftarrow \mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{Y}})$
$\mathbf{s} \leftarrow \mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{S}})$
Decompose each $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$
$\mathbf{U} \leftarrow \bigotimes \mathbf{U}^{(i)}$
Compute $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ as $\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n});\, \boldsymbol{\beta}\big)$ (see eqs. (5.17) and (5.18))
$\mathbf{D}_{\boldsymbol{\mathcal{G}}} \leftarrow \mathrm{diag}\left(\mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{G}})\right)$
$\mathbf{D}_{\boldsymbol{\mathcal{S}}} \leftarrow \mathrm{diag}\left(\mathbf{s}\right)$
$\boldsymbol{\Psi} \leftarrow \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}$
Initialise $\mathbf{f} \in \mathbb{R}^{N}$ randomly
while $|\Delta \mathbf{f}| > \mathrm{tol}$ **do**
  $\boldsymbol{\mu} \leftarrow \mathbf{1}/\big(\mathbf{1} + \exp(-\mathbf{f})\big)$
  $\mathbf{D}_{\boldsymbol{\mu}} \leftarrow \mathrm{diag}\big(\mathrm{vec}_{\mathrm{RM}}\left(\mathbf{s} \circ \boldsymbol{\mu} \circ (1 - \boldsymbol{\mu})\right)\big)$
  $\mathbf{t} \leftarrow \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\mathbf{y} - \boldsymbol{\mu}\big) + \mathbf{D}_{\boldsymbol{\mu}}\mathbf{f}$
  $\mathbf{Q} \leftarrow \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mu}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_N$
  $\mathbf{f} \leftarrow \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top}\mathbf{t}$ (solve with the CGM, leveraging the structure of $\mathbf{Q}$)
end while
$\boldsymbol{\mu} \leftarrow \mathbf{1}/\big(\mathbf{1} + \exp(-\mathbf{f})\big)$

Output: $\mathrm{reshape}\big(\boldsymbol{\mu},\, (N_1, \ldots N_d)\big)$

7.2 Multiclass Logistic Graph Signal Reconstruction

So far we have been focused on binary-valued graph signals, which can be used to represent two-class classification tasks over a multiway network. However, in many practical applications, the task may involve classifying each node into one of multiple distinct groups. Therefore, the objective of this section is to extend and generalise the methods we have developed thus far to encompass multiclass classification problems.

Consider the task of multiclass graph signal reconstruction. Here, the goal is to estimate the probability that each node, $\mathbf{n}$, in a d-dimensional Cartesian product graph belongs to each of $C > 2$ distinct classes. To do this, we have access to the factor graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$, and the true class label at a subset of nodes, $\mathcal{S}$. The partially observed class labels can be represented as an order-$(d+1)$ tensor graph signal, where

the final dimension contains a "one-hot" encoding of the class label. As such, the input data for multiclass L-GSR can be summarised as follows.

$$\text{input data} = \Big\{ \boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d \times C},\ \ \boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d},\ \ \Big\{ \mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i} \Big\}_{i=1}^{d} \Big\}$$

As in previous sections, the tensor $\boldsymbol{\mathcal{S}}$, of shape $(N_1 \times \ldots \times N_d)$, indicates which nodes in the product graph have been successfully observed by holding a one in the corresponding entry, and zeros elsewhere. Note that the observed graph signal, $\boldsymbol{\mathcal{Y}}$, has an additional dummy dimension of length C to represent the class label, which is not present in $\boldsymbol{\mathcal{S}}$. Where no observation was made (i.e. $\boldsymbol{\mathcal{S}}_{\mathbf{n}} = 0$), all values along the corresponding fibre $\boldsymbol{\mathcal{Y}}_{\mathbf{n},:}$ can be safely set to zero by default. Note that $\boldsymbol{\mathcal{Y}}_{\mathbf{n},:}$ can have a maximum of one non-zero entry.

We describe the probability that node $\mathbf{n}$ has class c by introducing another tensor, $\boldsymbol{\mathcal{M}}$. Like $\boldsymbol{\mathcal{Y}}$, this tensor has shape $(N_1 \times \ldots \times N_d \times C)$. Each fiber $\boldsymbol{\mathcal{M}}_{\mathbf{n},:}$ is interpreted as the probability mass function over the C mutually exclusive classes at node $\mathbf{n}$. In particular,

$$p(\text{node } \mathbf{n} \text{ is of class } c) = \boldsymbol{\mathcal{M}}_{\mathbf{n},c}, \quad \sum_{c=1}^{C} \boldsymbol{\mathcal{M}}_{\mathbf{n},c} = 1$$

Since every length-C fibre $\boldsymbol{\mathcal{M}}_{\mathbf{n},:}$ must sum to one, the tensor $\boldsymbol{\mathcal{M}}$ exists in the space of N independent C-dimensional simplexes with $C-1$ degrees of freedom.

This implies that, for a given value of $\boldsymbol{\mathcal{M}}$, the probability of observing a signal $\boldsymbol{\mathcal{Y}}$ is given by

$$p(\boldsymbol{\mathcal{Y}} \,|\, \boldsymbol{\mathcal{M}}) = \prod_{c,\mathbf{n} \in \mathcal{S}} \boldsymbol{\mathcal{M}}_{\mathbf{n},c}^{\boldsymbol{\mathcal{Y}}_{\mathbf{n},c}} \tag{7.14}$$

As such, the log-likelihood of observing a signal $\boldsymbol{\mathcal{Y}}$, for a given value of Mt, is

$$\begin{aligned} \log p(\boldsymbol{\mathcal{Y}} \,|\, \boldsymbol{\mathcal{M}}) &= \sum_{\mathbf{n},c} \boldsymbol{\mathcal{S}}_{\mathbf{n}} \, \boldsymbol{\mathcal{Y}}_{\mathbf{n},c} \log \boldsymbol{\mathcal{M}}_{\mathbf{n},c} \\ &= (\mathbf{s} \otimes \mathbf{1}_C)^{\top} \big(\mathbf{y} \circ \log \boldsymbol{\mu} \big) \end{aligned} \tag{7.15}$$

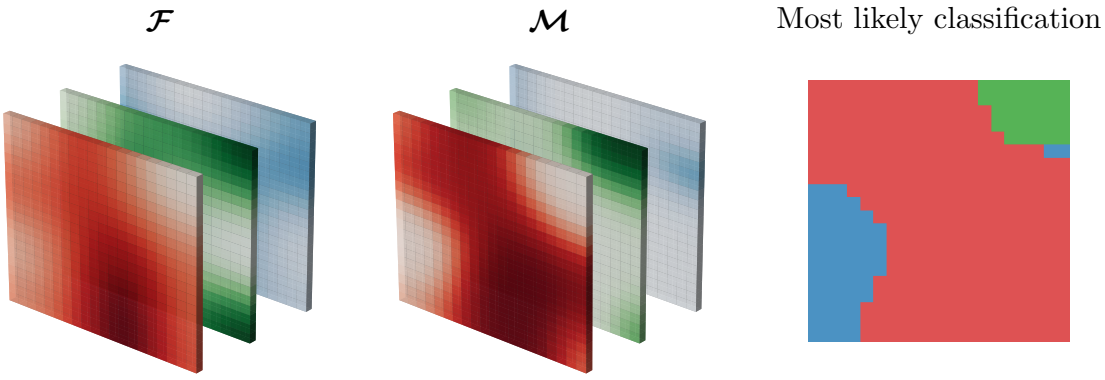


FIGURE 7.4: An example visualisation of transforming a smooth tensor signal via the softmax function on a 20×20 grid of pixels ($N_1 = 20, N_2 = 20, C = 3$). Left: a random smooth tensor signal $\boldsymbol{\mathcal{F}}$, drawn from the Gaussian distribution specified in eq. (7.17), split across three channels. Middle: the value of the tensor $\boldsymbol{\mathcal{M}}$ found by applying the softmax function of eq. (7.16) in the class dimension. Right: the resultant most likely classification given by the maximum probability class label for each pixel.

where, as before, $\mathbf{s} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{S}})$, $\mathbf{y} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{Y}})$ and $\boldsymbol{\mu} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{M}})$. Furthermore, to enforce the restriction that all probabilities must sum to one, we can generate $\boldsymbol{\mathcal{M}}$ by applying a softmax function to each length-C fibre of a tensor $\boldsymbol{\mathcal{F}}$.

$$\boldsymbol{\mathcal{M}}_{\mathbf{n},c} = \frac{\exp(\boldsymbol{\mathcal{F}}_{\mathbf{n},c})}{\sum_{c'=1}^{C} \exp(\boldsymbol{\mathcal{F}}_{\mathbf{n},c'})} \quad \Longleftrightarrow \quad \boldsymbol{\mu}(\mathbf{f}) = \frac{\exp(\mathbf{f})}{\big((\mathbf{I}_N \otimes \mathbf{1}_C)\exp(\mathbf{f})\big) \otimes \mathbf{1}_C} \tag{7.16}$$

Here, $\boldsymbol{\mathcal{F}}$ is a real-valued tensor signal which, like $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{Y}}$, has shape $(N_1 \times ... \times N_d \times C)$. A graphical representation of the relation between $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{M}}$ and the most likely classification is shown in fig. 7.4. The goal, then, of multiclass logistic graph signal reconstruction is to find the most likely value for the tensor $\boldsymbol{\mathcal{F}}$, which allows us to make a prediction for the probability of each class by applying eq. (7.16).

Assuming no particular relational structure between the C classes, we can encode the assumption that class probabilities should vary smoothly over the graph by assigning the following prior to $\boldsymbol{\mathcal{F}}$.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma^{-1}\mathbf{H}^2 \otimes \mathbf{I}_C\right) \tag{7.17}$$

By applying Bayes' rule, we can therefore derive an expression $\xi(\mathbf{f})$, representing the negative log-likelihood of observing an underlying signal $\mathbf{f}$ given an observed signal $\mathbf{y}$.

$$\xi(\mathbf{f}) = -(\mathbf{s} \otimes \mathbf{1}_C)^\top \big(\mathbf{y} \circ \log \boldsymbol{\mu}(\mathbf{f})\big) + \frac{\gamma}{2}\mathbf{f}^\top \left(\mathbf{H}^{-2} \otimes \mathbf{I}_C\right)\mathbf{f} \tag{7.18}$$

Once again, we can solve for the minimising value of $\mathbf{f}$ using the IRLS algorithm. In this case, the gradient and Hessian are respectively given by

$$\mathbf{g}(\mathbf{f}) = \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}\big) + \gamma\left(\mathbf{H}^{-2} \otimes \mathbf{I}_C\right)\mathbf{f} \tag{7.19}$$

and

$$\mathbf{P}(\mathbf{f}) = \mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f}) + \gamma\mathbf{H}^{-2} \otimes \mathbf{I}_C \tag{7.20}$$

Unlike the binary model where the Hessian was the sum of a diagonal matrix $\mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f})$ and $\gamma\mathbf{H}^{-2}$, the matrix $\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})$ is no longer diagonal. In this case, $\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})$ is given by

$$\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f}) = \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\left(\operatorname{diag}\big(\boldsymbol{\mu}(\mathbf{f})\big) - \sum_{n=1}^{N} \boldsymbol{\Delta}_n \otimes \mathbf{m}_n(\mathbf{f})\mathbf{m}_n^{\top}(\mathbf{f})\right) \tag{7.21}$$

where $\mathbf{m}_n \in \mathbb{R}^C$ is the probability mass function at node n (i.e. the n-th lexicographically ordered fiber within the tensor $\boldsymbol{\mathcal{M}}$), and $\boldsymbol{\Delta}_n$ is a matrix of zeros, with a single one at element (n, n). Given the definition of the Kronecker product, we can see that $\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})$ is a block-diagonal matrix with the following structure.

$$\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f}) = \begin{bmatrix} \mathbf{B}_1 & & & \\ & \mathbf{B}_2 & & \\ & & \ddots & \\ & & & \mathbf{B}_N \end{bmatrix}, \quad \text{where} \quad \mathbf{B}_n = \mathbf{s}_n\Big(\operatorname{diag}\left(\mathbf{m}_n\right) - \mathbf{m}_n\mathbf{m}_n^{\top}\Big)$$

Despite no longer being diagonal, we can still leverage the block structure of this matrix to multiply $\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})$ onto an arbitrary vector, $\mathbf{v}$, efficiently. This can be achieve as follows.

$$\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})\mathbf{v} = (\mathbf{s} \otimes \mathbf{1}_C) \circ \boldsymbol{\mu}(\mathbf{f}) \circ \mathbf{v} - \Big(\big((\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{1}_C)(\boldsymbol{\mu}(\mathbf{f}) \circ \mathbf{v})\big) \otimes \mathbf{1}_C\Big) \circ \boldsymbol{\mu}(\mathbf{f}) \tag{7.22}$$

While a naive implementation of the product $\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f})\mathbf{v}$ would have memory time and memory complexity $O\left(C^2N^2\right)$, by following the formula given in eq. (7.22), it can instead be achieved with time complexity $O(NC^2)$ and memory complexity $O(NC)$.

Now consider the IRLS update formula. As before, we will use the following shorthands.

$$\mathbf{P}(\mathbf{f}_k) = \mathbf{P}_k, \quad \mathbf{g}(\mathbf{f}_k) = \mathbf{g}_k, \quad \boldsymbol{\mu}(\mathbf{f}_k) = \boldsymbol{\mu}_k, \quad \text{and} \quad \mathbf{R}_{\boldsymbol{\mu}}(\mathbf{f}_k) = \mathbf{R}_{\boldsymbol{\mu}}^k$$

Applying the update formula gives

$$\begin{aligned}
\mathbf{f}_{k+1} &= \mathbf{f}_k - \mathbf{P}_k^{-1}\,\mathbf{g}_k \\
&= \mathbf{f}_k - \mathbf{P}_k^{-1}\Big(\big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\boldsymbol{\mu}_k - \mathbf{y}\big) + \gamma\left(\mathbf{H}^{-2} \otimes \mathbf{I}_C\right)\mathbf{f}_k\Big) \\
&= \mathbf{f}_k + \mathbf{P}_k^{-1}\big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}_k\big) - \mathbf{P}_k^{-1}\Big(\gamma\left(\mathbf{H}^{-2} \otimes \mathbf{I}_C\right)\mathbf{f}_k\Big) \\
&= \mathbf{P}_k^{-1}\big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{P}_k^{-1}\Big(\mathbf{P}_k\mathbf{f}_k - \gamma\left(\mathbf{H}^{-2} \otimes \mathbf{I}_C\right)\mathbf{f}_k\Big) \\
&= \mathbf{P}_k^{-1}\big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{P}_k^{-1}\mathbf{R}_{\boldsymbol{\mu}}^k\mathbf{f}_k \\
&= \mathbf{P}_k^{-1}\Big(\big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{R}_{\boldsymbol{\mu}}^k\mathbf{f}_k\Big) \\
&= \mathbf{P}_k^{-1}\mathbf{t}_k
\end{aligned}$$

where

$$\mathbf{t}_k = \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}_k\big) + \mathbf{R}_{\boldsymbol{\mu}}^k\mathbf{f}_k$$

As before, we can solve the linear system $\mathbf{P}_k^{-1}\mathbf{t}_k$ by introducing a symmetric preconditioner $\boldsymbol{\Psi}$. The only modification is that it must be of dimension of $(NC \times NC)$, rather than $(N \times N)$. This can be achieved as follows.

$$\boldsymbol{\Psi} = (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}}) \otimes \mathbf{I}_C$$

Using this, we can transform the linear system into

$$\mathbf{f}_k = \mathbf{P}_k^{-1}\mathbf{t}_k \quad \Longrightarrow \quad \mathbf{f}_k = \boldsymbol{\Psi}\mathbf{Q}_k^{-1}\boldsymbol{\Psi}^\top\mathbf{t}_k$$

where

$$\mathbf{Q}_k = \big(\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C\big)^\top\mathbf{R}_{\boldsymbol{\mu}}^k\big(\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C\big) + \gamma\mathbf{I}_{NC} \tag{7.23}$$

Once again, the effect of introducing this preconditioner is to radically improve the conditioning of the coefficient matrix. Again, while the condition number, κ, of $\mathbf{P}_k$ is potentially unbounded, $\kappa(\mathbf{Q}_k)$ can be strongly bounded. In particular, it is guaranteed to have a maximum value of $(0.5 + \gamma)/\gamma$. The proof, which is somewhat more involved than the binary case, is given formally in theorem 7.2.

Theorem 7.2. *The condition number of* $\mathbf{Q}_k$ *is bounded by a maximum value of*

$$\kappa(\mathbf{Q}_k) \leq \frac{0.5 + \gamma}{\gamma} \tag{7.24}$$

Proof. As established in section 3.3.1, the worst-case convergence rate is achieved in the limit of a weak filter, where $\mathbf{D}_{\mathcal{G}} = \mathbf{I}_N$. In this case, the condition number of $\mathbf{Q}_k$ is given by

$$\begin{aligned}
\kappa(\mathbf{Q}_k) &= \kappa\left((\mathbf{U} \otimes \mathbf{I}_C)^\top \mathbf{R}_{\boldsymbol{\mu}}^k (\mathbf{U} \otimes \mathbf{I}_C) + \gamma \mathbf{I}_{NC}\right) \\
&= \kappa\left((\mathbf{U} \otimes \mathbf{I}_C)^\top \left(\mathbf{R}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N\right)(\mathbf{U} \otimes \mathbf{I}_C)\right) \\
&= \kappa\left(\mathbf{R}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N\right)
\end{aligned}$$

Since $\mathbf{R}_{\boldsymbol{\mu}}^k$ is block-diagonal, the set of associated eigenvalues is equal to the union of the eigenvalues of each individual block. Furthermore, each block within $\mathbf{R}_{\boldsymbol{\mu}}^k$ is of the form

$$\mathbf{B} = s\left(\text{diag}\,(\mathbf{m}) - \mathbf{m}\mathbf{m}^\top\right)$$

where $\mathbf{m} \in \mathbb{R}^C$ is a probability mass vector, and s is either zero or one. For the case where $s \neq 0$, we can see that $\mathbf{B}$ must have at least one zero eigenvalue, with eigenvector $\mathbf{1}$, since

$$\begin{aligned}
\mathbf{B}\mathbf{1} &= \left(\text{diag}\,(\mathbf{m}) - \mathbf{m}\mathbf{m}^\top\right)\mathbf{1} \\
&= \text{diag}\,(\mathbf{m})\,\mathbf{1} - (\mathbf{m}^\top \mathbf{1})\mathbf{m} \\
&= \mathbf{m} - \mathbf{m} = \mathbf{0}
\end{aligned}$$

Note, we have used the fact that $\mathbf{m}^\top \mathbf{1} = 1$, since $\mathbf{m}$ represents a probability mass function and must therefore sum to one. Furthermore, we can show that all eigenvalues of $\mathbf{B}$ must be within the interval $[0, 1/2]$, no matter the value of $\mathbf{m}$. To achieve this, we can make use of the Gerschgorin circle theorem [Gerschgorin, 1931]. This states that all eigenvalues of a square matrix must fall within at least one of the closed disks $D(a_{ii}, R_i) \subset \mathbb{C}$ in the complex plane. Here, a_{ii} are the centres of the disks given by the diagonal elements of the matrix, and R_i are the radii of the disks, given by

$$R_i = \sum_{j \neq i} |a_{ij}|$$

i.e. the sum of the absolute value of the off-diagonal elements in row i. Consider the structure of $\mathbf{B}$ when $s \neq 0$.

$$\mathbf{B} = \operatorname{diag}(\mathbf{m}) - \mathbf{m}\mathbf{m}^\top = \begin{bmatrix} m_1 - m_1^2 & -m_1 m_2 & \dots & -m_1 m_C \\ -m_2 m_1 & m_2 - m_2^2 & \dots & -m_2 m_C \\ \vdots & \vdots & \ddots & \vdots \\ -m_C m_1 & -m_C m_2 & \dots & m_C - m_C^2 \end{bmatrix}$$

where $m = [m_1, m_2, ..., m_C]$. The disk associated with row c will be given by

$$D\left(m_c - m_c^2,\ \sum_{c' \neq c} | - m_c m_{c'}|\right) = D\Big(m_c\,(1 - m_c),\ m_c\,(1 - m_c)\Big)$$

In our particular scenario, the matrix $\mathbf{B}$ is symmetric-real, guaranteeing that its eigenvalues are real. This implies that, instead of complex regions, each of these disks represents a real interval. As such, every eigenvalue is constrained to fall within the interval $m_c\,(1 - m_c) \pm m_c\,(1 - m_c) = \big[0,\ 2m_c\,(1 - m_c)\big]$. The maximum width of this interval occurs when $m_c = 0.5$, resulting in an interval of $[0, 0.5]$. Consequently, the largest possible eigenvalue of matrix $\mathbf{B}$ is 0.5.

Finally, since $\mathbf{R}_{\boldsymbol{\mu}}^k$ is composed of blocks $\mathbf{B}_n$, this means the largest possible eigenvalue of $\mathbf{R}_{\boldsymbol{\mu}}^k$ is 0.5, and the smallest is zero. Therefore

$$\kappa\left(\mathbf{R}_{\boldsymbol{\mu}}^k + \gamma \mathbf{I}_N\right) \leq \frac{0.5 + \gamma}{\gamma}$$

completing the proof.

□

We now present the full algorithm for logistic graph signal reconstruction, which is given in algorithm 10. Note that, by making use of the fast Kronecker product algorithm described in section 5.1.4, the runtime complexity of the CGM step is bounded by

$$O\left(\frac{0.5 + \gamma}{\gamma} NC\Big(C + \sum_{i=1}^{d} N_i\Big)\right)$$

Algorithm 11 Multiclass Logistic Graph Signal Reconstruction

Input: Observed binary tensor $\boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d \times C}$
Input: Binary sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$
Input: Cartesian product graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$
Input: Regularisation parameter $\gamma \in \mathbb{R}^{+}$
Input: Graph filter function $g(\,\cdot\,;\boldsymbol{\beta})$

$\mathbf{y} \leftarrow \mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{Y}})$
$\mathbf{s} \leftarrow \mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{S}})$
Decompose each $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$
$\mathbf{U} \leftarrow \bigotimes \mathbf{U}^{(i)}$
Compute $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ as $\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n});\,\boldsymbol{\beta}\big)$ (see eqs. (5.17) and (5.18))
$\mathbf{D}_{\boldsymbol{\mathcal{G}}} \leftarrow \mathrm{diag}\left(\mathrm{vec}_{\mathrm{RM}}(\boldsymbol{\mathcal{G}})\right)$
$\mathbf{D}_{\boldsymbol{\mathcal{S}}} \leftarrow \mathrm{diag}\left(\mathbf{s}\right)$
$\boldsymbol{\Psi} \leftarrow \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C$
Initialise $\mathbf{f} \in \mathbb{R}^{NC}$ randomly
while $|\Delta \mathbf{f}| > \mathrm{tol}$ **do**
  $\boldsymbol{\mu} \leftarrow \exp(\mathbf{f}) / \big((\mathbf{I}_N \otimes \mathbf{1}_C)\exp(\mathbf{f})\big) \otimes \mathbf{1}_C$
  $\mathbf{M} \leftarrow \mathrm{reshape}\big(\boldsymbol{\mu}, (N, C)\big)$
  $\mathbf{R}_{\boldsymbol{\mu}} \leftarrow \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\left(\mathrm{diag}\big(\boldsymbol{\mu}\big) - \sum_{n=1}^{N} \boldsymbol{\Delta}_n \otimes \mathbf{m}_n\mathbf{m}_n^{\top}\right)$ ($\mathbf{m}_n$ is the n-th row of $\mathbf{M}$)
  $\mathbf{t} \leftarrow \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big)\big(\mathbf{y} - \boldsymbol{\mu}\big) + \mathbf{R}_{\boldsymbol{\mu}}\mathbf{f}$
  $\mathbf{Q} \leftarrow \big(\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C\big)^{\top}\mathbf{R}_{\boldsymbol{\mu}}\big(\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C\big) + \gamma\mathbf{I}_{NC}$
  $\mathbf{f} \leftarrow \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top}\mathbf{t}$ (solve with the CGM, leveraging the structure of $\mathbf{Q}$)
end while
$\boldsymbol{\mu} \leftarrow \exp(\mathbf{f}) / \big((\mathbf{I}_N \otimes \mathbf{1}_C)\exp(\mathbf{f})\big) \otimes \mathbf{1}_C$
Output: $\mathrm{reshape}\big(\boldsymbol{\mu}, (N_1, \ldots, N_d, C)\big)$

7.3 Logistic Graph Signal Regression

So far in this chapter we have developed binary and multiclass multiway graph signal reconstruction algorithms, where no additional data exists to help predict the value of the graph signal at the unlabelled nodes. In this section, we explore several circumstances where explanatory variables are at our disposal, and propose models to leverage this information to estimate the underlying class label probabilities.

7.3.1 Logistic Kernel Graph Regression (L-KGR)

The first regression models we consider are binary and multiclass Logistic Kernel Graph Regression (L-KGR). The goal of L-KGR is to predict the underlying class probabilities at the nodes of a sequence of multiway graph signals, given that each signal is paired with an associated vector of explanatory variables. In this respect, it parallels the real-valued KGR model introduced in section 4.1 (and extended to general tensor signals in section 5.3), except here we are concerned with classification rather than real-valued regression. As we will demonstrate, mirroring the pattern of previous sections, the L-KGR model can be interpreted as a relatively straightforward extension of logistic graph signal reconstruction.

7.3.1.1 Binary L-KGR

First, let us consider binary L-KGR. In this case, we have a sequence of T partially observed binary tensor graph signals, $\boldsymbol{\mathcal{Y}}_t$, each with a shape of $(N_1, ..., N_d)$. As before, each signal is interpreted as existing on the nodes of a d-dimensional Cartesian product graph, with factor graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$. Where no class outcome was recorded, the relevant element in the signal $\boldsymbol{\mathcal{Y}}_t$ can default to zero. Furthermore, at each time instant, we have a length-M vector of explanatory variables $\mathbf{x}_t$, which is anticipated to have some non-linear explanatory relation to each $\boldsymbol{\mathcal{Y}}_t$.

The sequence of partially observed signals can be assembled into a single tensor $\boldsymbol{\mathcal{Y}}$ of shape $(T, N_1, ..., N_d)$. This is accompanied by a binary sensing tensor, $\boldsymbol{\mathcal{S}}$, of the same shape, which indicates which values were observed and which were absent. The explanation variables can also be aggregated into a single object, in this case a matrix $\mathbf{X}$ with a shape of (T, M). The goal is to leverage the explanatory variables, in conjunction with the topology of the underlying graph, to estimate the class probability at each node, at each time instant. The relevant input data can therefore be described as follows.

$$\text{input data} = \left\{ \mathbf{X} \in \mathbb{R}^{T \times M}, \;\; \boldsymbol{\mathcal{Y}} \in \{0,1\}^{T \times N_1 \times ... \times N_d}, \right. \\ \left. \boldsymbol{\mathcal{S}} \in \{0,1\}^{T \times N_1 \times ... \times N_d}, \;\; \left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d} \right\}$$

Following the same pattern as the KGR algorithms of previous sections (see section 4.1.2 for details and further explanation), we can conveniently derive a logistic kernel graph

regression model by making small modifications to the L-GSR model derived in section 7.1.

As with L-GSR, we first assume each element of $\boldsymbol{\mathcal{Y}}$ is distributed according to a Bernoulli distribution, with a mean given by a latent mean tensor $\boldsymbol{\mathcal{M}}$, which has the same shape as $\boldsymbol{\mathcal{Y}}$ (see eqs. (7.1) to (7.3)). However, in this case, we assume that $\boldsymbol{\mathcal{M}}$ is smooth with respect to both the topology of the graph, *and* with with respect to the feature space. (This is admittedly a somewhat vague statement on its own - see section 4.1 for a more detailed model derivation and section 4.1.2 for further intuition). This belief can be encoded by proxy via a tensor $\boldsymbol{\mathcal{F}}$, which is again related to $\boldsymbol{\mathcal{M}}$ via the logistic link function of eq. (7.4). We implement these assumptions by establishing the following prior distribution for $\mathbf{f} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{F}})$.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma\mathbf{K} \otimes \mathbf{H}^2\right) \tag{7.25}$$

As before, $\mathbf{K}$ is the $T \times T$ kernel matrix created by applying a symmetric function, such as the Gaussian kernel, $\kappa(\cdot, \cdot)$ to pairs of feature vectors such that $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Next, we compute the eigendecomposition of $\mathbf{K}$ as $\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top$. Finally, we define the transformed variables $\bar{\mathbf{U}}$ and $\bar{\boldsymbol{\mathcal{G}}}$ as follows.

$$\bar{\mathbf{U}} = \mathbf{V} \otimes \mathbf{U}, \quad \text{and} \quad \bar{\boldsymbol{\mathcal{G}}}_{t,\mathbf{n}} = \boldsymbol{\mathcal{G}}_{\mathbf{n}}\sqrt{\lambda_t^{(K)}} \tag{7.26}$$

where $\lambda_t^{(K)}$ is the t-th eigenvalue of $\mathbf{K}$. This implies that

$$\mathbf{K} \otimes \mathbf{H}^2 = \bar{\mathbf{U}}\mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}}\bar{\mathbf{U}}^\top \tag{7.27}$$

where $\mathbf{D}_{\bar{\boldsymbol{\mathcal{G}}}} = \text{diag}\left(\text{vec}_{\text{RM}}\left(\bar{\boldsymbol{\mathcal{G}}}\right)\right)$. The L-KGR algorithm then follows by modifying algorithm 10 such that every instance of $\mathbf{U}$ and $\boldsymbol{\mathcal{G}}$ is substituted for $\bar{\mathbf{U}}$ and $\bar{\boldsymbol{\mathcal{G}}}$ respectively. Note tha, as with the KGR models of previous sections, this has an impact on the convergence rate of the CGM step, since the condition number of the preconditioned coefficient matrix $\mathbf{Q}_k$ is increased. As a result, the upper bound on the run time complexity for the whole algorithm is given by

$$O\left(\rho(\mathbf{K})\frac{0.25+\gamma}{\gamma}TN\Big(T + \sum_{i=1}^{d} N_i\Big)\right) \approx O\left(\frac{0.25+\gamma}{\gamma}T^2N\Big(T + \sum_{i=1}^{d} N_i\Big)\right)$$

7.3.1.2 Multiclass L-KGR

The same principles can be used to generate a multiclass logistic kernel graph regression algorithm. In this case, the input data is a sequence of T partially observed multiway graph signals, where each observed node has been classified into one of $C > 2$ classes. This can be represented by the binary tensor $\boldsymbol{\mathcal{Y}}$ with shape $(T \times N_1 \times ... \times N_d \times C)$, where an extra dimension has been added containing a one-hot encoding of the class label. Where no data was recorded, the full length-C fibre at time t and node $\mathbf{n}$ can be set to zero. As such, the data is described as follows.

$$\text{input data} = \left\{ \begin{array}{l} \mathbf{X} \in \mathbb{R}^{T \times M}, \;\; \boldsymbol{\mathcal{Y}} \in \{0,1\}^{T \times N_1 \times ... \times N_d \times C}, \\ \qquad\qquad \boldsymbol{\mathcal{S}} \in \{0,1\}^{T \times N_1 \times ... \times N_d}, \;\; \left\{ \mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i} \right\}_{i=1}^{d} \end{array} \right\}$$

Just as with the multiclass L-GSR model derived in section 7.2, we introduce the tensor $\boldsymbol{\mathcal{M}}$, with the same shape as $\boldsymbol{\mathcal{Y}}$, where each length-C fibre $\boldsymbol{\mathcal{M}}_{t,\mathbf{n},:}$ describes the probability mass function over the C classes for node $\mathbf{n}$ at time t. As before, $\boldsymbol{\mathcal{M}}$ is constructed by applying a softmax function to each fibre of another tensor $\boldsymbol{\mathcal{F}}$.

$$\boldsymbol{\mathcal{M}}_{t,\mathbf{n},c} = \frac{\exp(\boldsymbol{\mathcal{F}}_{t,\mathbf{n},c})}{\sum_{c'=1}^{C} \exp(\boldsymbol{\mathcal{F}}_{t,\mathbf{n},c'})} \quad \Longleftrightarrow \quad \boldsymbol{\mu}(\mathbf{f}) = \frac{\exp(\mathbf{f})}{\big((\mathbf{I}_{NT} \otimes \mathbf{1}_C)\exp(\mathbf{f})\big) \otimes \mathbf{1}_C} \tag{7.28}$$

We then place a prior over the signal $\mathbf{f} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{F}}) \in \mathbb{R}^{TNC}$ of

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \gamma \mathbf{K} \otimes \mathbf{H}^2 \otimes \mathbf{I}_C\right) \tag{7.29}$$

This encodes the belief that each of the c class probabilities should vary smoothly (and independently) with respect to the topology of the graph and the feature space. once again, the full procedure for outputting the tensor $\boldsymbol{\mathcal{M}}$ can be achieved by running algorithm 11 after substituting the variables $\mathbf{U} \to \bar{\mathbf{U}}$ and $\boldsymbol{\mathcal{G}} \to \bar{\boldsymbol{\mathcal{G}}}$, as described in eq. (7.26). Given the effect this substitution has on the the preconditioned coefficient matrix $\mathbf{Q}_k$ appearing in each iteration of the IRLS algorithm, it is simple to show that the run time complexity of multiclass L-KGR becomes bounded by the following.

$$O\left(\rho(\mathbf{K})\frac{0.5+\gamma}{\gamma}TNC\Big(T+C+\sum_{i=1}^{d}N_i\Big)\right)\approx O\left(\frac{0.5+\gamma}{\gamma}T^2NC\Big(T+C+\sum_{i=1}^{d}N_i\Big)\right)$$

7.3.2 Logistic Regression with Network Cohesion (L-RNC)

In this section we present models for binary and multiclass Logistic Regression with Network Cohesion (L-RNC). These models are relevant when there is a d-dimensional Cartesian product graph, for which a subset of nodes have a known class label. In addition, each node has a length-M vector of explanatory variables. The goal is to utilise both the topology of the graph, and the explanatory variables to estimate the class probabilities at the unlabelled nodes.

7.3.2.1 Binary L-RNC

First, let us consider binary L-RNC. In this scenario, we encounter a partially observed binary multiway graph signal $\boldsymbol{\mathcal{Y}}$, with shape $(N_1 \times ... \times N_d)$, where a subset of the nodes have been labelled with the true class. As before, $\boldsymbol{\mathcal{Y}}$ is interpreted as existing on the nodes of a known Cartesian product graph, with factor Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$. In addition, each node $\mathbf{n}$ in the product graph has an associated length-M vector of explanatory variables, collected into the tensor $\boldsymbol{\mathcal{X}}$ with shape $(N_1 \times ... \times N_d \times M)$. The goal is to predict the class probabilities at each node $\mathbf{n}$. Once again, we also have a binary tensor $\boldsymbol{\mathcal{S}}$ describing which nodes have been observed. As such, the data for the binary L-RNC algorithm can be summarised as follows.

$$\text{input data} = \left\{ \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times ... \times N_d \times M},\ \boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times ... \times N_d},\right.$$
$$\left.\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times ... \times N_d},\ \left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d} \right\}$$

As with the L-GSR model, we suppose that each observed element of $\boldsymbol{\mathcal{Y}}$ is a Bernoulli random variable, with the probability of success given by the tensor $\boldsymbol{\mathcal{M}} \in [0,1]^{N_1 \times ... \times N_d}$. However, in the L-RNC model, we assume that the probability at each node is given by the logistic function applied to the sum of an intercept term and a linear combination of the explanatory variables at that node. As with the real-valued RNC model introduced in section 4.2, the intercept term is flexible and assumed to vary smoothly with respect

to the topology of the product graph. Denoting $\boldsymbol{\mu} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{M}})$, this is summarised by the following expression.

$$\boldsymbol{\mu}(\mathbf{c}, \mathbf{w}) = \frac{\mathbf{1}}{\mathbf{1} + \exp\big(-(\mathbf{c} + \mathbf{X}\mathbf{w})\big)} \tag{7.30}$$

Here, $\mathbf{c} = \text{vec}_{\text{RM}}(\boldsymbol{\mathcal{C}})$ the a real-valued vector form of a tensor $\boldsymbol{\mathcal{C}}$, which is assumed to vary smoothly with respect to the topology of the multiway graph, $\mathbf{X}$ is the tensor of explanatory variables reshaped into a matrix of dimensions $(N \times M)$ (see eq. (5.49)), and $\mathbf{w} \in \mathbb{R}^M$ is the length-M vector of coefficients specifying the contribution of each of the explanatory variables. As in previous sections on RNC, we can stack $\mathbf{c}$ and $\mathbf{w}$ into a single parameter vector $\boldsymbol{\theta}$, such that

$$\boldsymbol{\mu}(\boldsymbol{\theta}) = \frac{\mathbf{1}}{\mathbf{1} + \exp\big(-\begin{bmatrix}\mathbf{I}_N & \mathbf{X}\end{bmatrix}\boldsymbol{\theta}\big)} \tag{7.31}$$

Furthermore, we can place a prior over $\boldsymbol{\theta}$ that encodes the belief that $\boldsymbol{\mathcal{C}}$ should be smooth with respect to the topology of the Cartesian product graph, as well as preventing overfitting of $\mathbf{w}$ with an L2 penalty.

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}\right) \tag{7.32}$$

As such, the MAP estimator for $\boldsymbol{\theta}$ is given by the minimiser of the following expression.

$$\xi(\boldsymbol{\theta}) = -\mathbf{s}^\top\big(\mathbf{y} \circ \log\boldsymbol{\mu}(\boldsymbol{\theta}) + (\mathbf{1} - \mathbf{y}) \circ \log\left(\mathbf{1} - \boldsymbol{\mu}(\boldsymbol{\theta})\right)\big) + \frac{1}{2}\boldsymbol{\theta}^\top\begin{bmatrix}\gamma^1\mathbf{H}^{-2} & \mathbf{0} \\ \mathbf{0} & \lambda^1\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta} \tag{7.33}$$

Proceeding in the same way as L-GSR, we must compute both the gradient, $\widetilde{\mathbf{g}}(\boldsymbol{\theta})$, and the Hessian, $\widetilde{\mathbf{P}}(\boldsymbol{\theta})$, of this expression with respect to $\boldsymbol{\theta}$. These are given by

$$\widetilde{\mathbf{g}}(\boldsymbol{\theta}) \in \mathbb{R}^{N+M} = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}\big(\boldsymbol{\mu}(\boldsymbol{\theta}) - \mathbf{y}\big) + \begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta} \tag{7.34}$$

and

$$\widetilde{\mathbf{P}}(\boldsymbol{\theta}) \in \mathbb{R}^{(N+M)\times(N+M)} = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) + \gamma\mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta})\mathbf{X} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) & \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta})\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix} \tag{7.35}$$

where, as with L-GSR, $\mathbf{D}_{\boldsymbol{\mu}}$ is the diagonal $N \times N$ matrix given by

$$\mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) = \operatorname{diag}\big(\mathbf{s} \circ \boldsymbol{\mu}(\boldsymbol{\theta}) \circ (\mathbf{1} - \boldsymbol{\mu}(\boldsymbol{\theta}))\big) \tag{7.36}$$

Then, using the shorthands

$$\widetilde{\mathbf{P}}(\boldsymbol{\theta}_k) = \widetilde{\mathbf{P}}_k, \quad \widetilde{\mathbf{g}}(\boldsymbol{\theta}_k) = \widetilde{\mathbf{g}}_k, \quad \boldsymbol{\mu}(\boldsymbol{\theta}_k) = \boldsymbol{\mu}_k, \quad \text{and} \quad \mathbf{D}_{\boldsymbol{\mu}}(\boldsymbol{\theta}_k) = \mathbf{D}_{\boldsymbol{\mu}}^k$$

The IRLS update formula can be derived as follows.

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \widetilde{\mathbf{P}}_k^{-1}\,\widetilde{\mathbf{g}}_k \\
&= \boldsymbol{\theta}_k - \widetilde{\mathbf{P}}_k^{-1}\left(\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}(\boldsymbol{\mu}_k - \mathbf{y}) + \begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta}_k\right) \\
&= \boldsymbol{\theta}_k + \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}(\mathbf{y} - \boldsymbol{\mu}_k) - \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta}_k \\
&= \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}(\mathbf{y} - \boldsymbol{\mu}_k) + \widetilde{\mathbf{P}}_k^{-1}\left(\widetilde{\mathbf{P}}_k\boldsymbol{\theta}_k - \begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta}_k\right) \\
&= \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}(\mathbf{y} - \boldsymbol{\mu}_k) + \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mu}}^k & \mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}^k & \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X}\end{bmatrix}\boldsymbol{\theta}_k \\
&= \widetilde{\mathbf{P}}_k^{-1}\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}\Big(\mathbf{y} - \boldsymbol{\mu}_k + \mathbf{D}_{\boldsymbol{\mu}}^k\big[\mathbf{I}_N \;\; \mathbf{X}\big]\boldsymbol{\theta}_k\Big) \\
&= \widetilde{\mathbf{P}}_k^{-1}\widetilde{\mathbf{t}}_k
\end{aligned}$$

where

$$\widetilde{\mathbf{t}}_k = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}\Big(\mathbf{y} - \boldsymbol{\mu}_k + \mathbf{D}_{\boldsymbol{\mu}}^k\big[\mathbf{I}_N \;\; \mathbf{X}\big]\boldsymbol{\theta}_k\Big) \tag{7.37}$$

Note that, in the derivation above, we have used the fact that $\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{D}_{\boldsymbol{\mu}}^k = \mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{D}_{\boldsymbol{\mathcal{S}}} = \mathbf{D}_{\boldsymbol{\mu}}^k$. This implies that

$$\begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}\mathbf{D}_{\boldsymbol{\mu}}^k\big[\mathbf{I}_N \;\; \mathbf{X}\big] = \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mu}}^k & \mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X} \\ \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}^k & \mathbf{X}^\top\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X}\end{bmatrix}$$

As before, the linear system $\widetilde{\mathbf{P}}_k^{-1}\widetilde{\mathbf{t}}_k$ cannot easily be solved in this form due to ill-conditioning. To overcome this issue, we can again use the symmetrically preconditioned CGM. In particular, we can transform the system into

$$\boldsymbol{\theta}_k = \widetilde{\mathbf{P}}_k^{-1}\widetilde{\mathbf{t}}_k \quad \Longrightarrow \quad \boldsymbol{\theta}_k = \widetilde{\boldsymbol{\Psi}}_k\widetilde{\mathbf{Q}}_k^{-1}\widetilde{\boldsymbol{\Psi}}_k^{\top}\widetilde{\mathbf{t}}_k$$

In order to define $\widetilde{\boldsymbol{\Psi}}_k$ and $\widetilde{\mathbf{Q}}_k$, first we must compute the eigendecomposition of $\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X}$.

$$\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X} = \mathbf{U}_M^k\boldsymbol{\Lambda}_M^k(\mathbf{U}_M^k)^{\top} \tag{7.38}$$

Next, we define the matrix $\mathbf{D}_M^k$ as follows.

$$\mathbf{D}_M^k = \left(\boldsymbol{\Lambda}_M^k + \lambda\mathbf{I}_M\right)^{-1/2} \tag{7.39}$$

Then, the operators $\widetilde{\boldsymbol{\Psi}}_k$ and $\widetilde{\mathbf{Q}}_k$ are defined as follows.

$$\widetilde{\boldsymbol{\Psi}}_k = \begin{bmatrix} \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M^k\mathbf{D}_M^k \end{bmatrix} \tag{7.40}$$

and

$$\widetilde{\mathbf{Q}}_k = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{NT} & \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{X}\mathbf{U}_M^k\mathbf{D}_M^k \\ \mathbf{D}_M^k\left(\mathbf{U}_M^k\right)^{\top}\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mu}}^k\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{I}_M \end{bmatrix} \tag{7.41}$$

Note that, in contrast to L-GSR, the preconditioning matrix $\widetilde{\boldsymbol{\Psi}}_k$ is changing on each iteration of the IRLS algorithm. We now have all the mechanics in place to compute the value of $\boldsymbol{\theta}$ that minimised eq. (7.33). This, in turn, can be used to compute $\boldsymbol{\mathcal{M}}$, according to eq. (7.31). For clarity, we now present the complete L-RNC algorithm in algorithm 12. Once again, we expect the IRLS iterations to quickly converge. As such, the main computational bottleneck occurs in solving the CGM, which can be achieved efficiently by leveraging the Kronecker and block structure of the matrix $\mathbf{Q}_k$.

Algorithm 12 Logistic Regression with Network Cohesion

Input: Explanatory variables $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times \ldots \times N_d \times M}$
Input: Observed binary tensor $\boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$
Input: Binary sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$
Input: Cartesian product graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$
Input: Regularisation parameter $\gamma \in \mathbb{R}^{+}$
Input: Graph filter function $g(\,\cdot\,;\boldsymbol{\beta})$
Input: Regularisation parameter $\lambda \in \mathbb{R}^{+}$

$\mathbf{X} \leftarrow \text{reshape}\big(\boldsymbol{\mathcal{X}}, (N, M)\big)$

$\mathbf{y} \leftarrow \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)$

$\mathbf{s} \leftarrow \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)$

Decompose each $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

$\mathbf{U} \leftarrow \bigotimes \mathbf{U}^{(i)}$

Compute $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ as $\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n}); \boldsymbol{\beta}\big)$ (see eqs. (5.17) and (5.18))

$\mathbf{D}_{\boldsymbol{\mathcal{G}}} \leftarrow \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right)$

$\mathbf{D}_{\boldsymbol{\mathcal{S}}} \leftarrow \text{diag}\left(\mathbf{s}\right)$

Initialise $\boldsymbol{\theta} \in \mathbb{R}^{N+M}$ randomly

while $|\Delta\boldsymbol{\theta}| > \text{tol}$ **do**

$\boldsymbol{\mu} \leftarrow \mathbf{1}/\mathbf{1} + \exp\big(-\begin{bmatrix}\mathbf{I}_N & \mathbf{X}\end{bmatrix}\boldsymbol{\theta}\big)$

$\mathbf{D}_{\boldsymbol{\mu}} \leftarrow \text{diag}\big(\mathbf{s} \circ \boldsymbol{\mu} \circ (1 - \boldsymbol{\mu})\big)$

Decompose $\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mu}}\mathbf{X}$ into $\mathbf{U}_M\boldsymbol{\Lambda}_M\mathbf{U}_M^{\top}$

$\mathbf{D}_M \leftarrow \left(\boldsymbol{\Lambda}_M + \lambda\mathbf{I}_M\right)^{-1/2}$

$\boldsymbol{\Psi} \leftarrow \begin{bmatrix}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M\end{bmatrix}$

$\mathbf{t} \leftarrow \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\end{bmatrix}\big(\mathbf{y} - \boldsymbol{\mu} + \mathbf{D}_{\boldsymbol{\mu}}\begin{bmatrix}\mathbf{I}_N & \mathbf{X}\end{bmatrix}\boldsymbol{\theta}\big)$

$\mathbf{Q} \leftarrow \begin{bmatrix}\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mu}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{NT} & \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mu}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mu}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{I}_M\end{bmatrix}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top}\mathbf{t}$ (solve with the CGM, leveraging the structure of $\mathbf{Q}$)

end while

$\boldsymbol{\mu} \leftarrow \mathbf{1}/\mathbf{1} + \exp\big(-\begin{bmatrix}\mathbf{I}_N & \mathbf{X}\end{bmatrix}\boldsymbol{\theta}\big)$

Output: $\text{reshape}\big(\boldsymbol{\mu}, (N_1, \ldots, N_d)\big)$

7.3.2.2 Multiclass L-RNC

In this section, we consider the multiclass generalisation of the L-RNC model. The data available for this model is much the same as the binary version outlined in the previous section, except here we have a partially labelled graph signal where each observed node has been classified into one of $C > 2$ groups. As with the L-GSR model developed in section 7.2, this input data can be described a multiway tensor signal $\boldsymbol{\mathcal{Y}}$, with shape $(N_1, ..., N_d, C)$, where the final dimension represents a "one-hot" encoding of the class label. Each element, $\mathbf{n}$, is interpreted as existing on a Cartesian product graph with factor Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$, and is accompanied by a binary sensing tensor $\boldsymbol{\mathcal{S}}$, with shape $(N_1, ..., N_d)$, indicating which nodes were observed and which were not. Once again, we also have a length-M vector of explanatory variables at each node in the product graph, which can be collected into a single tensor $\boldsymbol{\mathcal{X}}$.

$$\text{input data} = \left\{ \begin{array}{l} \boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times ... \times N_d \times M}, \;\; \boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times ... \times N_d \times C}, \\ \qquad\qquad\qquad\qquad \boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times ... \times N_d}, \;\; \left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d} \end{array} \right\}$$

Just as in section 7.2, we describe the probability distribution over each of the C classes at node $\mathbf{n}$ with the tensor $\boldsymbol{\mathcal{M}}$, which has the same shape as $\boldsymbol{\mathcal{Y}}$, where the probability mass function is given by the length-C fibre $\boldsymbol{\mathcal{M}}_{\mathbf{n},:}$. Given this, the log-likelihood of observing a signal $\boldsymbol{\mathcal{Y}}$ is given by

$$\begin{aligned} \log p(\boldsymbol{\mathcal{Y}} \,|\, \boldsymbol{\mathcal{M}}) &= \sum_{\mathbf{n},c} \boldsymbol{\mathcal{S}}_{\mathbf{n}} \, \boldsymbol{\mathcal{Y}}_{\mathbf{n},c} \log \boldsymbol{\mathcal{M}}_{\mathbf{n},c} \\ &= (\mathbf{s} \otimes \mathbf{1}_C)^\top \big(\mathbf{y} \circ \log \boldsymbol{\mu}\big) \end{aligned} \tag{7.42}$$

Furthermore, we assume that $\boldsymbol{\mathcal{M}}$ is generated by applying a softmax function to each length-C fibre of another tensor $\boldsymbol{\mathcal{F}}$.

$$\boldsymbol{\mathcal{M}}_{\mathbf{n},c} = \frac{\exp(\boldsymbol{\mathcal{F}}_{\mathbf{n},c})}{\sum_{c'=1}^{C} \exp(\boldsymbol{\mathcal{F}}_{\mathbf{n},c'})} \quad \Longleftrightarrow \quad \boldsymbol{\mu}(\mathbf{f}) = \frac{\exp(\mathbf{f})}{\big((\mathbf{I}_N \otimes \mathbf{1}_C)\exp(\mathbf{f})\big) \otimes \mathbf{1}_C} \tag{7.43}$$

The key feature of the multiclass L-RNC model is that each of the C slices of the tensor $\boldsymbol{\mathcal{F}}$, denoted as $\boldsymbol{\mathcal{F}}_{:,c}$, are given by the sum of a flexible intercept $\boldsymbol{\mathcal{C}}_t$, and a weighted linear

combination of the features at each node. Both the intercept and the weighting are unique to each slice, c. This means, in order to describe the tensor $\boldsymbol{\mathcal{F}}$, we need a set of C intercepts and weights $\{\boldsymbol{\mathcal{C}}_c, \mathbf{w}_c\}_{c=1}^{C}$. These can both be aggregated into singe objects: $\boldsymbol{\mathcal{C}}$, a tensor with shape $(N_1, ..., N_d, C)$ and $\mathbf{w}$, a vector with length MC. Then, $\boldsymbol{\mathcal{F}}$ is defined as follows.

$$\boldsymbol{\mathcal{F}}_{:,c} = \boldsymbol{\mathcal{C}}_c + \mathbf{X}\mathbf{w}_c \quad \Longleftrightarrow \quad \boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{C}} + \text{ten}_{\text{RM}}\left(\left(\mathbf{X} \otimes \mathbf{I}_C\right)\mathbf{w}\right) \tag{7.44}$$

As before, $\mathbf{X}$ is a matrix view of the tensor $\boldsymbol{\mathcal{X}}$ reshaped to have dimensions (N, M). In vectorised form, this can be written as

$$\mathbf{f} = \mathbf{c} + \left(\mathbf{X} \otimes \mathbf{I}_C\right)\mathbf{w} \tag{7.45}$$

The goal of L-RNC will be to find the most likely values for $\boldsymbol{\mathcal{C}}$ and $\mathbf{w}$ which, in turn, will allow us to compute $\boldsymbol{\mathcal{M}}$, generating the predicted class probabilities across the product graph. Following the pattern of previous RNC models, we can stack $\mathbf{c}$ and $\mathbf{w}$ into a single vector $\boldsymbol{\theta}$ as follows.

$$\boldsymbol{\theta} \in \mathbb{R}^{NC+MC} = \begin{bmatrix} \mathbf{c} \\ \mathbf{w} \end{bmatrix} \tag{7.46}$$

Then, $\mathbf{f}$ can be rewritten as

$$\mathbf{f} = \begin{bmatrix} \mathbf{I}_{NC} & \mathbf{X} \otimes \mathbf{I}_C \end{bmatrix} \boldsymbol{\theta} \tag{7.47}$$

As such, we can rewrite the formula for $\boldsymbol{\mu}$ in terms of $\boldsymbol{\theta}$.

$$\boldsymbol{\mu}(\boldsymbol{\theta}) = \frac{\exp\left(\begin{bmatrix} \mathbf{I}_N & \mathbf{X} \otimes \mathbf{I}_C \end{bmatrix}\boldsymbol{\theta}\right)}{\left(\left(\mathbf{I}_N \otimes \mathbf{1}_C\right)\exp\left(\begin{bmatrix} \mathbf{I}_N & \mathbf{X} \otimes \mathbf{I}_C \end{bmatrix}\boldsymbol{\theta}\right)\right) \otimes \mathbf{1}_C} \tag{7.48}$$

Next, we place the following prior over $\boldsymbol{\theta}$, to encode the assumption that each of the flexible intercept terms should vary smoothly with respect to the topology of the graph, and to regularise the coefficient vector $\mathbf{w}$.

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \gamma^{-1}\mathbf{H}^2 \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_{MC} \end{bmatrix}\right) \tag{7.49}$$

This leads to the following negative log-likelihood.

$$\xi(\boldsymbol{\theta}) = -(\mathbf{s} \otimes \mathbf{1}_C)^\top \big(\mathbf{y} \circ \log \boldsymbol{\mu}(\boldsymbol{\theta})\big) + \frac{1}{2}\boldsymbol{\theta}^\top \begin{bmatrix} \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_{MC} \end{bmatrix} \boldsymbol{\theta} \tag{7.50}$$

The gradient of $\xi(\boldsymbol{\theta})$ is the vector of length $NC + MC$ given by

$$\widetilde{\mathbf{g}}(\boldsymbol{\theta}) \in \mathbb{R}^{NC+MC} = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} \big(\boldsymbol{\mu}(\boldsymbol{\theta}) - \mathbf{y}\big) + \begin{bmatrix} \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_{MC} \end{bmatrix} \boldsymbol{\theta} \tag{7.51}$$

The Hessian $\widetilde{\mathbf{P}}(\boldsymbol{\theta})$ with shape $(NC + MC) \times (NC + MC)$ is given by

$$\widetilde{\mathbf{P}}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) + \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta})(\mathbf{X} \otimes \mathbf{I}_C) \\ (\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) & (\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta})(\mathbf{X} \otimes \mathbf{I}_C) + \lambda \mathbf{I}_{MC} \end{bmatrix} \tag{7.52}$$

where, as in section 7.2,

$$\mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta}) = \big(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\big) \left(\operatorname{diag}\big(\boldsymbol{\mu}(\boldsymbol{\theta})\big) - \sum_{n=1}^{N} \boldsymbol{\Delta}_n \otimes \mathbf{m}_n(\boldsymbol{\theta}) \mathbf{m}_n^\top(\boldsymbol{\theta}) \right) \tag{7.53}$$

Using the shorthands

$$\widetilde{\mathbf{P}}(\boldsymbol{\theta}_k) = \widetilde{\mathbf{P}}_k, \quad \widetilde{\mathbf{g}}(\boldsymbol{\theta}_k) = \widetilde{\mathbf{g}}_k, \quad \boldsymbol{\mu}(\boldsymbol{\theta}_k) = \boldsymbol{\mu}_k, \quad \text{and} \quad \mathbf{R}_{\boldsymbol{\mu}}(\boldsymbol{\theta}_k) = \mathbf{R}_{\boldsymbol{\mu}}^k$$

the IRLS algorithm then proceeds according to the following update formula.

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \widetilde{\mathbf{P}}_k^{-1} \widetilde{\mathbf{g}}_k \\
&= \boldsymbol{\theta}_k - \widetilde{\mathbf{P}}_k^{-1} \left(\begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} (\boldsymbol{\mu}_k - \mathbf{y}) + \begin{bmatrix} \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_{MC} \end{bmatrix} \boldsymbol{\theta}_k \right) \\
&= \boldsymbol{\theta}_k + \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} (\mathbf{y} - \boldsymbol{\mu}_k) - \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_{MC} \end{bmatrix} \boldsymbol{\theta}_k \\
&= \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} (\mathbf{y} - \boldsymbol{\mu}_k) + \widetilde{\mathbf{P}}_k^{-1} \Big(\widetilde{\mathbf{P}}_k \boldsymbol{\theta}_k - \begin{bmatrix} \gamma \mathbf{H}^{-2} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I}_{MC} \end{bmatrix} \boldsymbol{\theta}_k \Big)
\end{aligned}$$

$$
\begin{aligned}
&= \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} (\mathbf{y} - \boldsymbol{\mu}_k) + \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \mathbf{R}_{\boldsymbol{\mu}}^k & \mathbf{R}_{\boldsymbol{\mu}}^k(\mathbf{X} \otimes \mathbf{I}_C) \\ (\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}^k & (\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}^k(\mathbf{X} \otimes \mathbf{I}_C) \end{bmatrix} \boldsymbol{\theta}_k \\
&= \widetilde{\mathbf{P}}_k^{-1} \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} \Big(\mathbf{y} - \boldsymbol{\mu}_k + \mathbf{R}_{\boldsymbol{\mu}}^k \big[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C \big] \boldsymbol{\theta}_k \Big) \\
&= \widetilde{\mathbf{P}}_k^{-1} \mathbf{t}_k
\end{aligned}
$$

where

$$
\mathbf{t}_k = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C \\ (\mathbf{X}^\top \mathbf{D}_{\boldsymbol{\mathcal{S}}}) \otimes \mathbf{I}_C \end{bmatrix} \Big(\mathbf{y} - \boldsymbol{\mu}_k + \mathbf{R}_{\boldsymbol{\mu}}^k \big[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C \big] \boldsymbol{\theta}_k \Big) \tag{7.54}
$$

As before, to overcome the ill-conditioning present in the linear system $\widetilde{\mathbf{P}}_k^{-1}\widetilde{\mathbf{t}}_k$, we can use the symmetrically preconditioned CGM. In particular, we can transform the system into

$$
\boldsymbol{\theta}_k = \widetilde{\mathbf{P}}_k^{-1}\widetilde{\mathbf{t}}_k \quad \Longrightarrow \quad \boldsymbol{\theta}_k = \widetilde{\boldsymbol{\Psi}}_k \widetilde{\mathbf{Q}}_k^{-1} \widetilde{\boldsymbol{\Psi}}_k^\top \widetilde{\mathbf{t}}_k
$$

where $\widetilde{\mathbf{Q}}_k$ has a lower condition number than the matrix $\widetilde{\mathbf{P}}_k$. In order to define $\widetilde{\boldsymbol{\Psi}}_k$ and $\widetilde{\mathbf{Q}}_k$, first we must compute the eigendecomposition of $(\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}^k(\mathbf{X} \otimes \mathbf{I}_C)$.

$$
(\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}^k(\mathbf{X} \otimes \mathbf{I}_C) = \mathbf{U}_M^k \boldsymbol{\Lambda}_M^k (\mathbf{U}_M^k)^\top \tag{7.55}
$$

Next, we define the matrix $\mathbf{D}_M^k$ as follows.

$$
\mathbf{D}_M^k = \Big(\boldsymbol{\Lambda}_M^k + \lambda \mathbf{I}_M \Big)^{-1/2} \tag{7.56}
$$

Then, the operators $\widetilde{\boldsymbol{\Psi}}_k$ and $\widetilde{\mathbf{Q}}_k$ are defined as follows.

$$
\widetilde{\boldsymbol{\Psi}}_k = \begin{bmatrix} \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M^k \mathbf{D}_M^k \end{bmatrix} \tag{7.57}
$$

and

$$\widetilde{\mathbf{Q}}_k = \begin{bmatrix} (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C)^\top \mathbf{R}^k_{\boldsymbol{\mu}} (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C) + \gamma \mathbf{I}_{NC} & (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C)^\top \mathbf{R}^k_{\boldsymbol{\mu}} (\mathbf{X} \otimes \mathbf{I}_C) \mathbf{U}^k_M \mathbf{D}^k_M \\ \mathbf{D}^k_M \left(\mathbf{U}^k_M\right)^\top (\mathbf{X}^\top \otimes \mathbf{I}_C) \mathbf{R}^k_{\boldsymbol{\mu}} (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C) & \mathbf{I}_{MC} \end{bmatrix} \tag{7.58}$$

This leads to the complete algorithm for computing the class probabilities, given in algorithm 13.

Algorithm 13 Multiclass Logistic Regression with Network Cohesion

Input: Explanatory variables $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N_1 \times \ldots \times N_d \times M}$

Input: Observed binary tensor $\boldsymbol{\mathcal{Y}} \in \{0,1\}^{N_1 \times \ldots \times N_d \times C}$

Input: Binary sensing tensor $\boldsymbol{\mathcal{S}} \in \{0,1\}^{N_1 \times \ldots \times N_d}$

Input: Cartesian product graph Laplacians $\left\{\mathbf{L}^{(i)} \in \mathbb{R}^{N_i \times N_i}\right\}_{i=1}^{d}$

Input: Graph regularisation parameter $\gamma \in \mathbb{R}^{+}$

Input: Graph filter function $g(\,\cdot\,;\boldsymbol{\beta})$

Input: Feautre regularisation parameter $\lambda \in \mathbb{R}^{+}$

$\mathbf{X} \leftarrow \text{reshape}\big(\boldsymbol{\mathcal{X}}, (N, M)\big)$

$\mathbf{y} \leftarrow \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)$

$\mathbf{s} \leftarrow \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)$

Decompose each $\mathbf{L}^{(i)}$ into $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

$\mathbf{U} \leftarrow \bigotimes \mathbf{U}^{(i)}$

Compute $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ as $\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n}); \boldsymbol{\beta}\big)$ (see eqs. (5.17) and (5.18))

$\mathbf{D}_{\boldsymbol{\mathcal{G}}} \leftarrow \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right)$

$\mathbf{D}_{\boldsymbol{\mathcal{S}}} \leftarrow \text{diag}\left(\mathbf{s}\right)$

Initialise $\boldsymbol{\theta} \in \mathbb{R}^{N+M}$ randomly

while $|\Delta\boldsymbol{\theta}| > \text{tol}$ **do**

$\boldsymbol{\mu} \leftarrow \exp\left(\left[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C\right]\boldsymbol{\theta}\right) / \Big(\left(\mathbf{I}_N \otimes \mathbf{1}_C\right)\exp\left(\left[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C\right]\boldsymbol{\theta}\right)\Big) \otimes \mathbf{1}_C$

$\mathbf{M} \leftarrow \text{reshape}\big(\boldsymbol{\mu}, (N, C)\big)$

$\mathbf{R}_{\boldsymbol{\mu}} \leftarrow \left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} \otimes \mathbf{I}_C\right)\Big(\text{diag}\big(\boldsymbol{\mu}\big) - \sum_{n=1}^{N} \boldsymbol{\Delta}_n \otimes \mathbf{m}_n\mathbf{m}_n^{\top}\Big)$ ($\mathbf{m}_n$ is the n-th row of $\mathbf{M}$)

Decompose $\mathbf{X}^{\top}\mathbf{R}_{\boldsymbol{\mu}}\mathbf{X}$ into $\mathbf{U}_M\boldsymbol{\Lambda}_M\mathbf{U}_M^{\top}$

$\mathbf{D}_M \leftarrow \left(\boldsymbol{\Lambda}_M + \lambda\mathbf{I}_M\right)^{-1/2}$

$\boldsymbol{\Psi} \leftarrow \begin{bmatrix} \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M \end{bmatrix}$

$\mathbf{t} \leftarrow \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{S}}} \\ \mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}} \end{bmatrix}\left(\mathbf{y} - \boldsymbol{\mu} + \mathbf{R}_{\boldsymbol{\mu}}\left[\mathbf{I}_N \;\; \mathbf{X}\right]\boldsymbol{\theta}\right)$

$\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{R}_{\boldsymbol{\mu}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{NT} & \mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{R}_{\boldsymbol{\mu}}\mathbf{X}\mathbf{U}_M\mathbf{D}_M \\ \mathbf{D}_M\mathbf{U}_M^{\top}\mathbf{X}^{\top}\mathbf{R}_{\boldsymbol{\mu}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} & \mathbf{I}_M \end{bmatrix}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Psi}\mathbf{Q}^{-1}\boldsymbol{\Psi}^{\top}\mathbf{t}$ solve with the CGM

end while

$\boldsymbol{\mu} \leftarrow \exp\left(\left[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C\right]\boldsymbol{\theta}\right) / \Big(\left(\mathbf{I}_N \otimes \mathbf{1}_C\right)\exp\left(\left[\mathbf{I}_N \;\; \mathbf{X} \otimes \mathbf{I}_C\right]\boldsymbol{\theta}\right)\Big) \otimes \mathbf{1}_C$

Output: $\text{reshape}\big(\boldsymbol{\mu},\, (N_1, \ldots, N_d)\big)$

7.4 Image segmentation

In this section, we examine the behaviour of the binary and multiclass algorithms developed in this chapter using an image segmentation task. The objective is to estimate the true class label at unlabelled pixels within an image, where a subset of the pixels have been classified into two or more groups. As briefly discussed in section 3.1.3, we can interpret image data as a special case of a 2D graph signal, with underlying graphs being simple chains reflecting the image's dimensions. By performing a Cartesian product of these factor graphs, we generate a lattice structure where each node corresponds to a pixel.

In the following sections we analyse the properties of the L-GSR and L-RNC models introduced earlier in this chapter, in both their binary and multiclass form. For the binary case, we consider a foreground/background separation task on standard RGB images sourced from the Berkeley Segmentation Dataset [Martin et al., 2001]. For the multiclass case, we consider a hyperspectral image segmentation task, with data captured by an airborne sensor of farmland that has been divided into seventeen classes based on land use [Baumgardner et al., 2015].

Many sophisticated techniques, most notably utilising deep neural networks, exist for image segmentation tasks (refer to Minaee et al. [2022], Wang et al. [2022b] for a comprehensive review). Whilst the classical statistical models introduced in this section are unlikely to be competitive with such methods on image segmentation tasks, they bring the advantage of versatility, enabling classification over generic Cartesian product graphs. However, since image segmentation datasets are readily accessible and easy to interpret visually, they serve as a useful test bed to investigate the properties of our more general graph-based algorithms. As such, it is worth emphasising that the goal of this section is not to produce a state-of-the-art image segmentation algorithm, but only to verify and analyse the properties of the techniques presented in this chapter.

7.4.1 Background/foreground separation

The first task we attempted was to separate the background from the foreground using four 481×321 pixel images taken from the Berkeley Segmentation Dataset [Martin et al., 2001]. These images had been manually labelled on a per-pixel basis into multiple objects, which we manually divided into background and foreground. We then randomly selected a certain fraction of pixels to serve as the labelled set $\mathcal{S}$, generating the input graph signals $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{S}}$, both of shape (481, 321). Then, we attempted to estimate the class probabilities of the remaining pixels using both the L-GSR and L-RNC algorithms

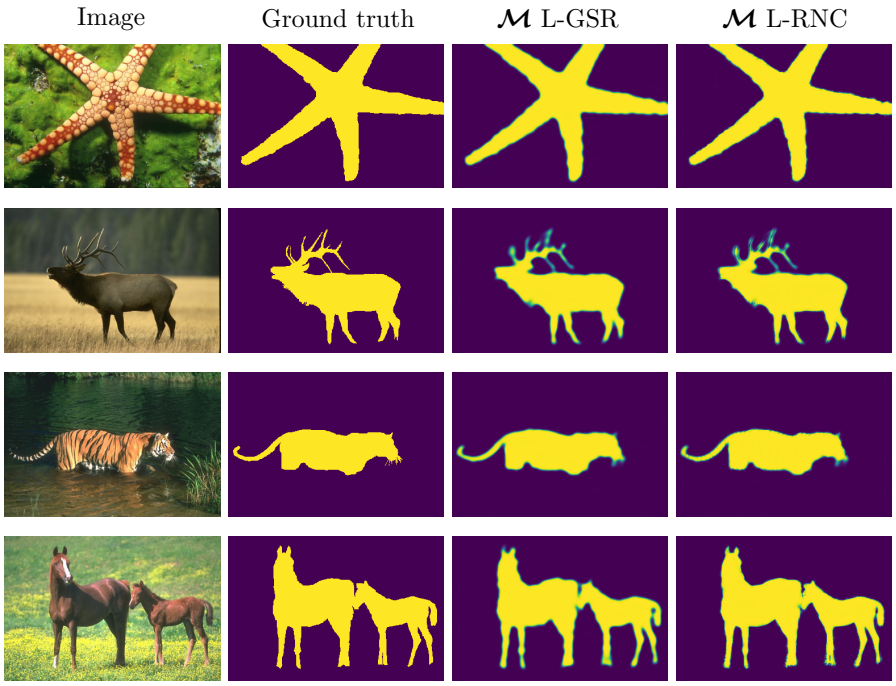


FIGURE 7.5: A color-map of the output probabilities, $\mathcal{M}$, from the L-GSR and L-RNC are shown, along with the original image and the ground truth for the four examples. These were generated with 10% of the pixels labelled, using an isotropic diffusion filter with $\beta = 50$, and $\gamma = 5 \times 10^{-5}$. For L-RNC, λ was set to 10^4.

on a lattice graph using an isotropic diffusion graph filter. For L-RNC, the algorithm also had access to the the RGB pixel data, creating a tensor of explanatory variables, $\mathcal{X}$, with shape (481, 321, 3). To ensure that the algorithms generated sensible outputs, we ran preliminary tests with 10% of the pixels labelled. The results can be seen in fig. 7.5, where the output, $\mathcal{M}$, is depicted as a colour-map. As can be seen, both algorithms broadly succeed at the separating the background from the foreground.

7.4.1.1 Assessing accuracy as a function of label percentage

In the first experiment, we examined the accuracy of each method as a function of the fraction of labelled pixels. Figure 7.6 provides a visual depiction of the estimated probability tensor $\mathcal{M}$ over a range of fractions. As expected, both methods progressively improve as the labelled fraction increases. The L-RNC model appears to outperform the L-GSR, particularly at lower label fractions, likely due to its access to the RGB colour profile in the explanatory tensor $\mathcal{X}$. L-GSR, on the other hand, only has access to the lattice graph structure and the location of the labelled pixels. This effect is visible in the first column, where the L-GSR model can only identify the approximate shape of the object, whereas the L-RNC model shows artifacts from the original image, such as the texture of the starfish body.

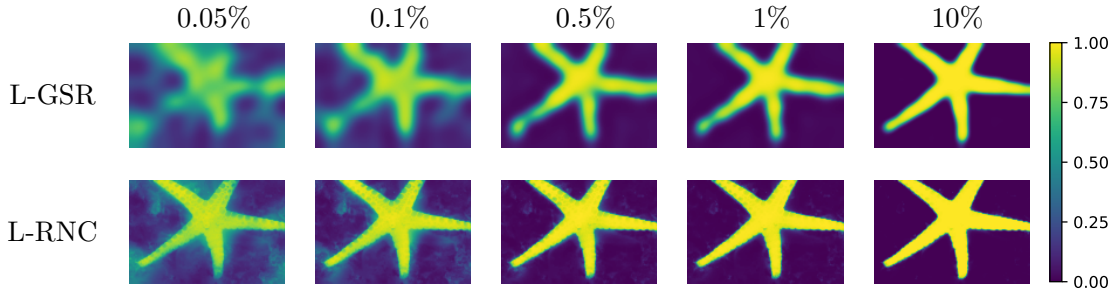


FIGURE 7.6: The estimated probability tensor, $\boldsymbol{\mathcal{M}}$, output from the L-GSR and L-RNC models, is shown for several fractions of observed data.

To further scrutinise this, we measured each model's accuracy across a range of label fractions for each image. Here, accuracy is gauged by counting the rate of correct predictions for all pixels, by determining whether the probability is less than or greater than 1/2. Simultaneously, we also measured each method's runtime for each label fraction. The results, shown in fig. 7.7, reveal both methods steadily improving in accuracy as the label fraction increases. L-RNC generally outperforms L-GSR, especially at lower label percentages, confirming our visual intuition from fig. 7.6. However, this comes at the cost of a longer runtime, likely due to additional matrix and memory management operations when running the CGM with block matrices.

It is clear that the runtime of both methods increases as the label fraction grows, consistent with the features of the CGM algorithm, which, as discussed in section 3.3.3, is expected to converge quicker when the fraction of missing data is higher. We also observe sporadic spikes in runtime above the general trend, which can likely be attributed to Jax's JIT compiler, which periodically retraces functions to ascertain their impact on inputs of a specific shape and data type [Bradbury et al., 2018].

7.4.1.2 Qualitative effects of varying γ and β

The second experiment sought to assess the effects on the probability output of the L-GSR and L-RNC algorithms when γ and β are varied independently. After randomly selecting 0.05% of the pixels to have their true label revealed, we evaluated the output of the L-GSR and L-RNC algorithms over a range of β values while maintaining γ fixed at 5×10^{-5} and, for L-RNC, λ fixed at 100. Subsequently, we performed the experiment again keeping β fixed at 100 and varying γ. In both cases, an isotropic diffusion filter was used. The results from these experiments are depicted in figs. 7.8 and 7.9, respectively.

As seen in fig. 7.8, an increase in beta tends to extend the range over which each labelled pixel can influence nearby probability. When β is low, at 10, the labelled pixels only

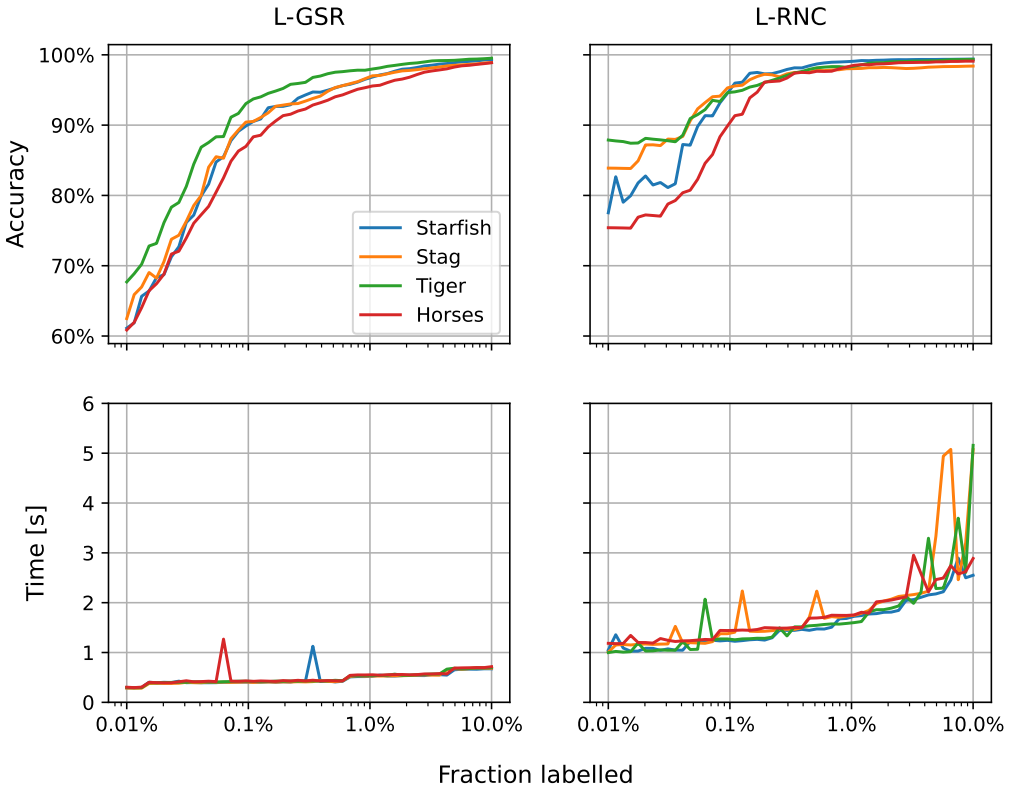


FIGURE 7.7: The accuracy on a per-pixel basis is shown for the L-GSR and L-RNC models for each of the four images as a function of the fraction of randomly selected pixels that were labelled

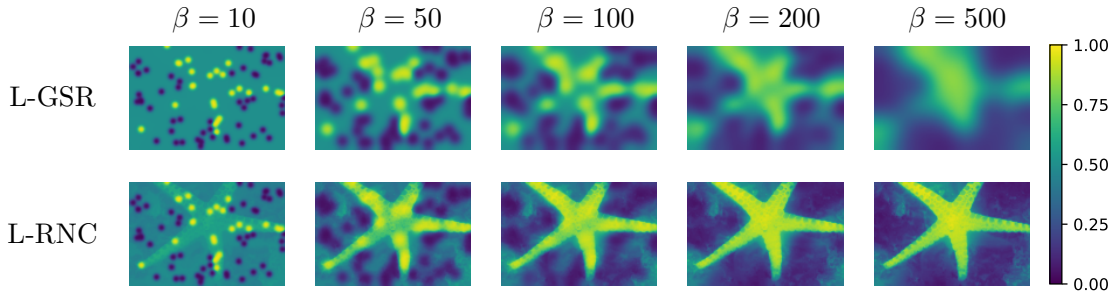


FIGURE 7.8: A color-map of the probability output of the L-GSR and L-RNC algorithms are shown for various increasing values of β. Here, γ is set to 5×10^{-5} and for L-RNC, λ is set to 10^4.

affect other pixels in the immediate vicinity, with the probability quickly reverting to 0.5 (i.e. unknown) for more distant unlabelled pixels. As β increases, so does this influence range, which is most visible in the L-GSR model where, when $\beta = 500$, the probability prediction becomes quite blurred. Furthermore, these results suggest that increasing β assists the L-RNC model in utilising the RGB data, as the outline of the foreground object becomes more defined with the increase in β.

Figure 7.9 illustrates the impact of altering γ. As visible, a lower γ tends to accentuate

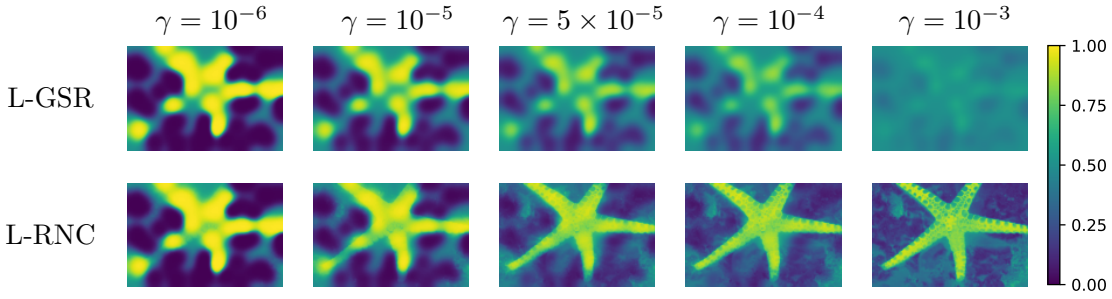


FIGURE 7.9: A color-map of the probability output of the L-GSR and L-RNC algorithms are shown for various increasing values of γ. Here, β is set to 100 and for L-RNC, λ is set to 10^4.

the transition between class probabilities, with the model yielding probabilities that cluster around zero or one. Interestingly, in this low γ region, the graph intercept predominates over the explanatory variables within the L-RNC model. On the other hand, with higher γ values, we observe the L-GSR probabilities leaning towards $1/2$ across the entire lattice. For the L-RNC model in this scenario, the influence of the graph intercept term wanes, allowing the explanatory variables to take precedence in the prediction process.

7.4.2 Multiclass segmentation with hyper-spectral images

In this section, we test the multiclass L-GSR and L-RNC algorithms on a segmentation task using a hyperspectral image of a patch of farmland in North-western Indiana [Baumgardner et al., 2015]. The image has been manually segmented into seventeen distinct classes representing different crop types and land uses. The image is 145×145 pixels in shape, and has 200 channels representing the response over a wavelength range of $0.4 - 2.5 \times 10^{-6}$m, with some bands covering the region of water absorption removed. Figure 7.10 gives an overview of this dataset, with the image on the left an example reading from one of the wavelength channels, and the image on the right a color-representation of the different land types.

In order to verify the algorithms behave as expected, we tested the output when 20% of the pixels had the correct class labelled. This generated the binary input tensor $\boldsymbol{\mathcal{Y}}$ of shape $(145, 145, 17)$, with a one-hot encoding along the class dimension, and the sensing tensor $\boldsymbol{\mathcal{S}}$ of shape $(144, 144)$. For the L-RNC algorithm, we allowed access to the pixel data as a tensor of explanatory variables $\boldsymbol{\mathcal{X}}$, of shape $(144, 144, 200)$. We then ran the L-GSR and L-RNC algorithms using an isotropic diffusion filter with parameter $\beta = 20$, and set $\gamma = 5 \times 10^{-3}$. For the L-RNC algorithm, we used a regularisation parameter of $\lambda = 100$.

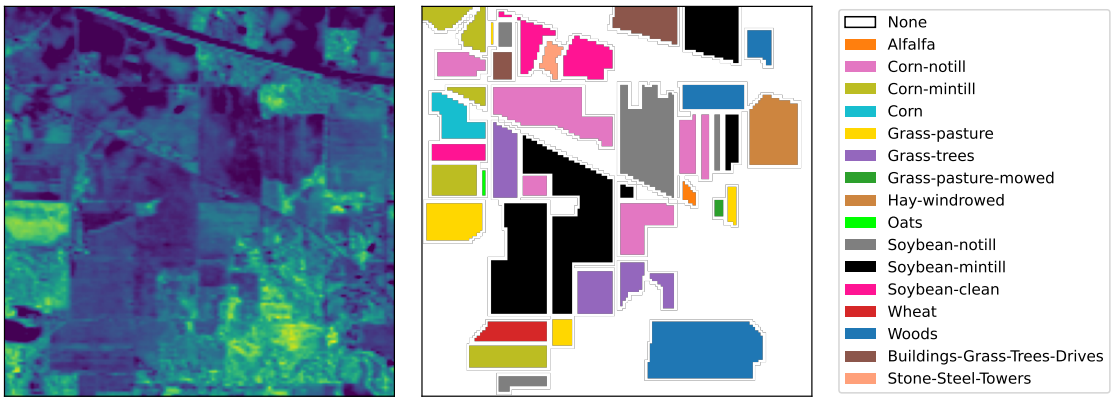


FIGURE 7.10: An overview of the Indian Pines hyperspectral image dataset. On the left is an example reading of the intensity response form the channel representing a wavelength of 0.9×10^{-6}m. On the right is the ground truth for the land use types, along with the associated color key.

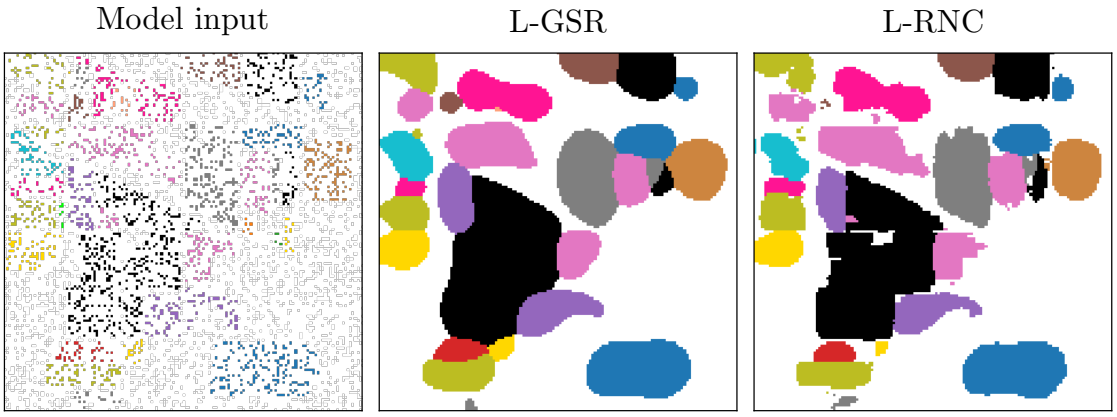


FIGURE 7.11: A color representation of the predicted classes from the multiclass L-GSR an L-RNC algorithms as applied to the Indian Pines dataset. On the left, we show the represent the input $\boldsymbol{\mathcal{Y}}$ available to the models, which contains 20% of the labelled pixels randomly selected.

Both these algorithms produced tensors representing the class probabilities at each pixel in the form of a tensor $\boldsymbol{\mathcal{M}}$ of shape $(144, 144, 17)$. From this, we obtained the most likely class by taking the maximum element along the final dimension. The results can be seen in fig. 7.11. The L-GSR model achieved an accuracy of 84%, and the L-RNC algorithm achieved 89%.

7.5 Discussion

7.5.1 Convergence of the IRLS algorithm

In general, algorithms based on Newton's method, including Iteratively Reweighted Least Squares (IRLS), frequently encounter convergence difficulties [Kelley, 2003]. In the course of our experimental work, we encountered two distinct problems, both of which we managed to rectify through relatively straightforward solutions.

Initially, we employed 32-bit floating point numbers for all arrays, with a termination criterion for the IRLS iterations defined as the point at which the absolute change in the target vector or tensor had an average value less than 10^{-6}. Nevertheless, we often noted an issue where the algorithm failed to converge, with the average change in each element stagnating around the order of 10^{-5}. To mitigate this issue, we upgraded to 64-bit floating point arrays. Although this change resulted in marginally slower computational speeds and a somewhat larger memory footprint, it effectively addressed the convergence problem.

A second challenge we faced manifested as either divergence or cyclic behaviour, where the target vector would oscillate through a finite series of points without ever achieving convergence. Broadly speaking, we found that these difficulties could be circumvented by selecting a more precise initial value for the iterative process. Our initial strategy was to simply set the initial estimate to a vector of zeros, but this approach led to inconsistent convergence outcomes.

Instead, we found that using an initial estimate derived from the solution to the real-valued GSR/KGR/RNC problem proved to be more effective. To illustrate, for a binary L-GSR problem with inputs $\boldsymbol{\mathcal{Y}}$, $\boldsymbol{\mathcal{S}}$, γ, and β, our initial estimate was the solution to the real-valued GSR problem with inputs $2\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{S}}$, γ, and β. The transformation $\boldsymbol{\mathcal{Y}} \to 2\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{S}}$ preserves all values equal to one, while all observed values that were zero are converted to negative one, with all unobserved values remaining at zero. This method provided a generally superior initial estimate for the underlying signal, and once implemented, eliminated all previously observed convergence issues. In the case of multiclass problems, we essentially solved the real-valued problem C times, corresponding to each dimension of the class label. This strategy ensured a smoother and more consistent convergence across various hyperparameter settings.

7.5.2 Efficient computation in the multiclass L-RNC algorithm

It is straightforward to see how the binary L-GSR and L-RNC algorithms can be executed efficiently, using the same ideas used consistently in this thesis. The CGM primarily entails repeated multiplications of the coefficient matrix by an arbitrary vector, while the IRLS algorithm involves repetitive execution of the CGM. In the L-GSR model, the coefficient matrix $\mathbf{Q}$ is constructed solely of diagonal and Kronecker-structured operators, enabling accelerated multiplication onto tensors. This is achieved by efficiently applying section 5.1.4, as elaborated in section 5.1.4. In the L-RNC scenario, the coefficient matrix $\widetilde{\mathbf{Q}}$ is a block matrix. The blocks aligned along the diagonal consist only of diagonal and Kronecker operators, while the off-diagonal blocks can be directly computed.

As we progress to the multiclass L-GSR algorithm, a complication emerges due to the presence of a non-diagonal, non-Kronecker-structured operator, $\mathbf{R}_{\boldsymbol{\mu}}$, within the coefficient matrix. Nevertheless, as detailed in section 7.2 and explicitly stated in eq. (7.22), the action of $\mathbf{R}_{\boldsymbol{\mu}}$ on a vector or tensor can be computed efficiently by leveraging its special structure, with a time-complexity of $O(NC)$.

The multiclass L-RNC, however, presents two additional challenges. First, it becomes necessary to compute the matrix $(\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{X} \otimes \mathbf{I}_C)$, so that it can then be decomposed into $\mathbf{U}_M\boldsymbol{\Lambda}_M(\mathbf{U}_M)^\top$. We assume that this decomposition step, which entails solving the eigenvalue problem for a dense symmetric matrix of dimensions $MC \times MC$, is feasible. This assumption is likely reasonable for most problem scenarios. For instance, in our case study involving 200 explanatory variables and 17 classes, the process required the decomposition of a 3400×3400 matrix. Nevertheless, efficient computation of the matrix $(\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{X} \otimes \mathbf{I}_C)$ is non-trivial. The straightforward strategy of instantiating the $NC \times MC$ matrix $\mathbf{X} \otimes \mathbf{I}_C$ directly, and then sequentially multiplying $\mathbf{R}_{\boldsymbol{\mu}}$ onto each column demands C^2 copies of $\mathbf{X}$, which may be challenging for large problems. In our relatively modest example, this matrix would be of size $(4.2 \times 10^6) \times 3400$, amounting to almost 10GB of 64-bit floating point memory. With a memory complexity of $O(NMC^2)$, this is potentially intractable.

Fortunately, a more efficient approach is possible. The matrix $(\mathbf{X}^\top \otimes \mathbf{I}_C)\mathbf{R}_{\boldsymbol{\mu}}(\mathbf{X} \otimes \mathbf{I}_C) \in \mathbb{R}^{MC \times MC}$ can be viewed as a block matrix comprising M^2 blocks, each of dimensions $C \times C$. The computation of Block (i, j) can be conducted as follows.

$$\text{Block}_{ij} = \text{Block}_{ji} = \text{diag}\big((\mathbf{X}_i \circ \mathbf{X}_j \circ \mathbf{M})^\top \mathbf{1}_C\big) - (\mathbf{X}_i \circ \mathbf{M})^\top (\mathbf{X}_j \circ \mathbf{M})$$

where, in this context, $\mathbf{X}_i \in \mathbb{R}^{N\times C}$ is the i-th column of $\mathbf{X}$ repeated C times, and $\mathbf{M}$ is $\boldsymbol{\mathcal{M}}$ reshaped to have dimensions (N, C). Thus the memory complexity of this approach is reduced to $O(NM + NC)$.

The second complication that arises is in the computation of the block matrix $\widetilde{\mathbf{Q}}$. Here, the upper right block is given by

$$(\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C)^\top \mathbf{R}_{\boldsymbol{\mu}} (\mathbf{X} \otimes \mathbf{I}_C) \mathbf{U}_M \mathbf{D}_M \in \mathbb{R}^{NC\times MC}$$

and the lower left block is the transpose of this. Again, since this object has NMC^2 elements, instantiating it in memory may be problematic. Instead, we can efficiently compute its action on a vector $\mathbf{v}$ of length MC by taking the following steps.

1. $\mathbf{v} \in \mathbb{R}^{MC} \leftarrow \mathbf{U}_M \mathbf{D}_M \mathbf{v}$ *(compute this as usual)*
2. $\mathbf{v} \in \mathbb{R}^{NC} \leftarrow \mathrm{vec}_{\mathrm{RM}}\big(\mathbf{X}\,\mathrm{reshape}\big(\mathbf{v}, (M, C)\big)\big)$ *(compute using the standard row-major vec trick)*
3. $\mathbf{v} \in \mathbb{R}^{NC} \leftarrow \mathbf{R}_{\boldsymbol{\mu}} \mathbf{v}$ *(compute efficiently using eq. (7.22))*
4. $\mathbf{v} \in \mathbb{R}^{NC} \leftarrow (\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \otimes \mathbf{I}_C)^\top \mathbf{v}$ *(compute efficiently using fast Kronecker product)*

The reverse process, of multiplying the transpose of this block onto a vector of length NC can be achieved by reversing these steps with each operation transposed. In both cases, this reduces the computational and memory footprint substantially.

7.6 Conclusions

Classification tasks over networks can be effectively represented by partially labelled binary graph signals. Although the Graph Signal Processing (GSP) community has made significant strides in the study of real-valued signals, binary signals have received comparatively less attention. In this chapter, we have taken a graph spectral approach to network-based classification tasks, examining several statistical models for regression and reconstruction that incorporate binary-valued graph signals.

In particular, we have developed several three novel graph signal processing models by generalising the Graph Signal Reconstruction (GSR), Kernel Graph Regression (KGR), and Regression with Network Cohesion (RNC) models, which were developed earlier in this thesis for real-valued signals, to accommodate binary graph signals. This led to

the creation of three "logistic" models: L-GSR, L-KGR, and L-RNC. Similar to their real-valued counterparts introduced in chapter 5, all three models are devised to handle signals residing on the nodes of general Cartesian product graphs, making them relevant for applications such as network time series modelling. Additionally, we introduced multiclass versions of these three algorithms to manage scenarios where nodes need to be classified into one of several groups or classes.

Chapter 8

Conclusions

In this thesis, we have developed several novel Graph Signal Processing (GSP) models designed to estimate arbitrary missing values within multivariate graph signals. This final chapter gives an overview of all the models presented in this thesis, followed by a discussion of some of their key characteristics. In addition, we highlight areas for improvement and give an outlook on possible future work in this area.

8.1 Summary of models

Graph Signal Reconstruction (GSR)

In chapter 3, we addressed the topic of Graph Signal Reconstruction on Cartesian product graphs.

Kernel Graph Regression (KGR)

Regression with Network Cohesion (RNC)

Kernel Graph Regression with Network Cohesion (KG-RNC)

Logistic Graph Signal Reconstruction (L-GSR)

Logistic Kernel Graph Regression (L-KGR)

Logistic Regression with Network Cohesion (L-RNC)

Logistic Kernel Graph Regression with Network Cohesion (L-KGRNC)

8.2 Discussion

Scalability

Distributed methods, incomplete eigendecomposition of $\mathbf{L}$, not possible for kernel models (common problem with GPs).

Hyperparameter selection

8.3 Future work

Directed graphs

Appendix A

Proofs

Theorem A.1. *The posterior distribution for* $\mathbf{F}$ *is given by*

$$\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\left(\boldsymbol{\Sigma} \operatorname{vec}(\mathbf{Y}),\ \boldsymbol{\Sigma}\right) \tag{A.1}$$

where

$$\boldsymbol{\Sigma} = \left(\operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma \mathbf{H}^{-2}\right)^{-1} \tag{A.2}$$

Proof. Consider the matrix $\mathbf{S}_\epsilon$ defined in the following manner.

$$(\mathbf{S}_\epsilon)_{nt} = \begin{cases} 1 & \text{if } (n,t) \in \mathcal{S} \\ \epsilon & \text{otherwise} \end{cases} \tag{A.3}$$

We can use this definition to rewrite equation 3.12 for the probability distribution of $\mathbf{Y}|\mathbf{F}$.

$$\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F} \sim \lim_{\epsilon \to 0} \left[\mathcal{N}\Big(\operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F}),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S}_\epsilon))\Big) \right] \tag{A.4}$$

In this way, the negative log-likelihood of an observation $\mathbf{Y}|\mathbf{F}$ is given by

$$-\log \pi(\mathbf{Y}|\mathbf{F}) = \lim_{\epsilon \to 0} \left[\frac{1}{2} \operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y})^\top \operatorname{diag}(\operatorname{vec}(\mathbf{S}_\epsilon))^{-1} \operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y}) \right] \tag{A.5}$$

up to an additive constant which does not depend on $\mathbf{F}$. Note that, since $\mathbf{Y} = \mathbf{S}_\epsilon \circ \mathbf{Y}$, we can rewrite $\mathrm{vec}\,(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y})$ as

$$\begin{aligned}\mathrm{vec}\,(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y}) &= \mathrm{vec}\,(\mathbf{S}_\epsilon \circ (\mathbf{F} - \mathbf{Y})) \\ &= \mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S}_\epsilon))\,\mathrm{vec}\,(\mathbf{F} - \mathbf{Y})\end{aligned} \tag{A.6}$$

Therefore, equation A.5 can be rewritten as

$$\begin{aligned}-\log \pi(\mathbf{Y}|\mathbf{F}) &= \lim_{\epsilon \to 0} \left[\frac{1}{2} \mathrm{vec}\,(\mathbf{F} - \mathbf{Y})^\top \,\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S}_\epsilon))\; \mathrm{vec}\,(\mathbf{F} - \mathbf{Y}) \right] \\ &= \frac{1}{2} \mathrm{vec}\,(\mathbf{F} - \mathbf{Y})^\top \,\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S}))\; \mathrm{vec}\,(\mathbf{F} - \mathbf{Y})\end{aligned} \tag{A.7}$$

Now consider the full log-posterior. Using Bayes rule, this can be written as

$$\begin{aligned}-\log \pi\big(\mathrm{vec}\,(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{1}{2} \mathrm{vec}\,(\mathbf{F} - \mathbf{Y})^\top \,\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S}))\; \mathrm{vec}\,(\mathbf{F} - \mathbf{Y}) \;+ \\ \frac{\gamma}{2} \mathrm{vec}\,(\mathbf{F})^\top \,\mathbf{H}^{-2}\,\mathrm{vec}\,(\mathbf{F})\end{aligned} \tag{A.8}$$

Up to an additive constant not dependent $\mathbf{F}$, this can be written as

$$-\log \pi\big(\mathrm{vec}\,(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{1}{2}\Big(\mathrm{vec}\,(\mathbf{F})^\top \left(\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S})) + \gamma \mathbf{H}^{-2}\right) \mathrm{vec}\,(\mathbf{F}) - 2\,\mathrm{vec}\,(\mathbf{Y})^\top \,\mathbf{F}\Big) \tag{A.9}$$

Using the conjugacy of the normal distribution, by direct inspection we can conclude that the posterior covariance is given by

$$\mathbf{\Sigma} = \Big(\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S})) + \gamma \mathbf{H}^{-2}\Big)^{-1} \tag{A.10}$$

and that the posterior mean is given by $\mathbf{\Sigma}\,\mathrm{vec}\,(\mathbf{Y})$.

□

Theorem A.2. Consider the random matrix $\mathbf{Z}$ which is related to the random matrix $\mathbf{F}$ as follows.

$$\mathbf{F} = \mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top$$

or, equivalently,

$$\operatorname{vec}\left(\mathbf{F}\right) = \big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\mathbf{D}_{\mathbf{G}} \operatorname{vec}\left(\mathbf{Z}\right)$$

Then the posterior mean for $\mathbf{Z}|\mathbf{Y}$ is given by

$$\mathrm{E}\big[\mathbf{Z}|\mathbf{Y}\big] = \left(\mathbf{C} + \gamma \mathbf{I}_T \otimes \mathbf{I}_N\right)^{-1} \operatorname{vec}\Big(\mathbf{G} \circ \left(\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T\right)\Big)$$

where

$$\mathbf{C} = \mathbf{D}_{\mathbf{G}} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_{\mathbf{S}} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_{\mathbf{G}}$$

(Here we have abbreviated $\operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right)$ and $\operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}\right)\right)$ as $\mathbf{D}_{\mathbf{G}}$ and $\mathbf{D}_{\mathbf{S}}$ respectively.)

Proof. The conditional distribution of $\mathbf{Y}|\mathbf{Z}$ is obtained by substituting in the definition of $\mathbf{F}$ in terms of $\mathbf{Z}$ into the original conditional likelihood expression.

$$\operatorname{vec}\left(\mathbf{Y}\right) \,|\, \mathbf{Z} \sim \mathcal{N}\Big(\operatorname{vec}\Big(\mathbf{S} \circ \left(\mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top\right)\Big),\ \mathbf{D}_{\mathbf{S}}\Big)$$

Similarly, since the prior specified for $\mathbf{F}$ is $\mathcal{N}\big(\mathbf{0},\, \gamma^{-1}\mathbf{H}^2\big)$, this implies that the prior over $\mathbf{Z}$ is simply

$$\operatorname{vec}\left(\mathbf{Z}\right) \sim \mathcal{N}\big(\mathbf{0},\, \gamma^{-1}\mathbf{I}_{NT}\big)$$

To see this, consider the following

$$\begin{aligned}
\operatorname{Cov}\big[\operatorname{vec}\left(\mathbf{F}\right)\big] &= \operatorname{Cov}\big[\big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\mathbf{D}_{\mathbf{G}} \operatorname{vec}\left(\mathbf{Z}\right)\big] \\
&= \big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\mathbf{D}_{\mathbf{G}} \operatorname{Cov}\big[\operatorname{vec}\left(\mathbf{Z}\right)\big]\mathbf{D}_{\mathbf{G}}\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big)
\end{aligned}$$

If $\operatorname{vec}\left(\mathbf{Z}\right)$ has covariance $\gamma^{-1}\mathbf{I}$, then $\operatorname{vec}\left(\mathbf{F}\right)$ has covariance given by

$$\begin{aligned}\mathrm{Cov}\big[\mathrm{vec}\,(\mathbf{F})\big] &= \gamma^{-1}\big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\mathbf{D}_{\mathbf{G}}^2\big(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\big)\\ &= \gamma^{-1}\mathbf{H}^2\end{aligned}$$

by the definition of $\mathbf{H}$.

Now consider the transformed posterior

$$\begin{aligned}-\log p(\mathbf{Z}|\mathbf{Y}) &= -\log p(\mathbf{Y}|\mathbf{Z}) - \log p(\mathbf{Z})\\ &= \frac{1}{2}\mathrm{vec}\left(\mathbf{U}_N\left(\mathbf{G}\circ\mathbf{Z}\right)\mathbf{U}_T^\top - \mathbf{Y}\right)^\top \times\\ &\quad \mathbf{D}_{\mathbf{S}}\,\mathrm{vec}\left(\mathbf{U}_N\left(\mathbf{G}\circ\mathbf{Z}\right)\mathbf{U}_T^\top - \mathbf{Y}\right)\\ &\quad + \frac{\gamma}{2}\,\mathrm{vec}\,(\mathbf{Z})^\top\,\mathrm{vec}\,(\mathbf{Z})\end{aligned}$$

Up to an additive constant, this is equal to

$$\begin{aligned}-\log p(\mathbf{Z}|\mathbf{Y}) &= \frac{1}{2}\mathrm{vec}\,(\mathbf{Z})^\top\left(\mathbf{C} + \gamma\mathbf{I}_{NT}\right)\mathrm{vec}\,(\mathbf{Z})\\ &\quad - \mathrm{vec}\left(\mathbf{U}_N\left(\mathbf{G}\circ\mathbf{Z}\right)\mathbf{U}_T\right)^\top\,\mathrm{vec}\,(\mathbf{Y})\\ &= \frac{1}{2}\mathrm{vec}\,(\mathbf{Z})^\top\left(\mathbf{C} + \gamma\mathbf{I}_{NT}\right)\mathrm{vec}\,(\mathbf{Z})\\ &\quad - \mathrm{vec}\,(\mathbf{Z})^\top\,\mathrm{vec}\left(\mathbf{G}\circ(\mathbf{U}_N^\top\mathbf{Y}\mathbf{U}_T)\right)\end{aligned}$$

By inspection, again, we can see that the posterior mean for $\mathbf{Z}$ is

$$\left(\mathbf{C} + \gamma\mathbf{I}_T \otimes \mathbf{I}_N\right)^{-1}\mathrm{vec}\left(\mathbf{G}\circ(\mathbf{U}_N^\top\mathbf{Y}\mathbf{U}_T)\right)$$

□

Theorem A.3. *The number of steps required to reach a given level of precision for matrix splitting methods follows*

$$n_{SIM} \propto -\frac{1}{\log\rho(\mathbf{M}^{-1}\mathbf{N})} \tag{A.11}$$

where the coefficient matrix is split as $\mathbf{M} - \mathbf{N}$.

Proof. In the SIM, we have that

$$\mathbf{M}\text{vec}\left(\mathbf{F}\right) = \mathbf{N}\text{vec}\left(\mathbf{F}\right) + \text{vec}\left(\mathbf{Y}\right) \tag{A.12}$$

where $\text{vec}\left(\mathbf{F}\right)$ represents the true solution to the linear system. This leads directly to an update equation given by

$$\mathbf{M}\text{vec}\left(\mathbf{F}_k\right) = \mathbf{N}\text{vec}\left(\mathbf{F}_{k-1}\right) + \text{vec}\left(\mathbf{Y}\right) \tag{A.13}$$

Subtracting eq. (A.12) from eq. (A.13) gives

$$\begin{aligned} \mathbf{M}\text{vec}\left(\mathbf{F}_k\right) - \mathbf{M}\text{vec}\left(\mathbf{F}\right) &= \mathbf{N}\text{vec}\left(\mathbf{F}_{k-1}\right) - \mathbf{N}\text{vec}\left(\mathbf{F}\right) \\ \text{vec}\left(\mathbf{E}_k\right) &= \mathbf{M}^{-1}\mathbf{N}\text{vec}\left(\mathbf{E}_{k-1}\right) \\ &= \left(\mathbf{M}^{-1}\mathbf{N}\right)^k \text{vec}\left(\mathbf{E}_0\right) \end{aligned} \tag{A.14}$$

where we denote the error at the k-th iteration as $\text{vec}\left(\mathbf{E}_k\right) = \text{vec}\left(\mathbf{F}_k\right) - \text{vec}\left(\mathbf{F}\right)$. From this it is clear to see that convergence will be achieved so long as the spectral radius $\rho(\mathbf{M}^{-1}\mathbf{N})$ is less than one. If this condition holds then,

$$\lim_{k\to\infty} \text{vec}\left(\mathbf{E}_k\right) = \lim_{k\to\infty} \left(\mathbf{M}^{-1}\mathbf{N}\right)^k \text{vec}\left(\mathbf{E}_0\right) = \mathbf{0}. \tag{A.15}$$

In general, the number of iterations required to achieve some specified reduction in the magnitude of the error is proportional to one over the logarithm of the spectral radius □

Theorem A.4. *The values of* n_{SIM} *and* n_{CGM} *always increase when* γ*, within its valid range, decreases in both the strong a weak filter limits.*

Proof. Consider each of the following expressions.

$$n_{\text{SIM}}(\gamma;\ \beta \to 0) = \frac{1}{\log(1+\gamma)}$$

$$n_{\mathrm{CGM}}(\gamma;\ \beta \to 0) = \sqrt{\frac{1}{\gamma} + 1}$$

$$n_{\mathrm{SIM}}(\gamma, m;\ \beta \to \infty) = \frac{1}{\log(1+\gamma) - \log m}$$

$$n_{\mathrm{CGM}}(\gamma, m;\ \beta \to \infty) = \sqrt{\frac{1-m+\gamma}{\gamma}}$$

In order to prove the theorem, we must show that the partial derivative of each expression with respect to γ is strictly negative over the domain $0 \leq \gamma \leq \infty$. Let us compute this for each expression in turn.

$$\frac{\partial\, n_{\mathrm{SIM}}(\gamma;\ \beta \to 0)}{\partial\, \gamma} = -\frac{1}{(1+\gamma)\log^2(1+\gamma)}$$

$$\frac{\partial\, n_{\mathrm{CGM}}(\gamma;\ \beta \to 0)}{\partial\, \gamma} = -\frac{1}{2\gamma^2\sqrt{\frac{1}{\gamma}+1}}$$

$$\frac{\partial\, n_{\mathrm{SIM}}(\gamma, m;\ \beta \to \infty)}{\partial\, \gamma} = -\frac{1}{(1+\gamma)\big(\log(1+\gamma) - \log(m)\big)^2}$$

$$\frac{\partial\, n_{\mathrm{CGM}}(\gamma, m;\ \beta \to \infty)}{\partial\, \gamma} = -\frac{1-m}{2\gamma^2\sqrt{\frac{1-m+\gamma}{\gamma}}}$$

In each of these expressions we have a fraction for which both the numerator and denominator can easily be shown to be strictly positive over the valid ranges of γ and m. Each expression also includes a negative sign in front. As such, every expression is strictly negative.

□

Theorem A.5. *In the limit of a strong filter, for any fixed value of γ, the number of iterations for convergence will always increase as the proportion of missing data m increases for the SIM, whereas it the number will always decrease for the CGM.*

Proof. To prove this theorem it suffices to show that the partial derivative of n_{SIM} with respect to m is always positive, whereas the partial derivative of n_{CGM} with respect to m is always negative. They are respectively given by

$$\frac{\partial\, n_{\text{SIM}}(\gamma, m;\ \beta \to \infty)}{\partial\, m} = \frac{1}{m\big(\log(1+\gamma) - \log(m)\big)^2}$$

$$\frac{\partial\, n_{\text{CGM}}(\gamma, m;\ \beta \to \infty)}{\partial\, m} = -\frac{1}{2\gamma\sqrt{\frac{1-m+\gamma}{\gamma}}}$$

Since γ is strictly positive and m must be in the range $0 \leq m \leq 1$, clearly the first expression is always positive whereas the second is always negative.

□

Theorem A.6. *In the RNC model, the posterior distribution over the combined parameter vector* $\boldsymbol{\theta}$ *is given by*

$$\boldsymbol{\theta} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\text{vec}\,(\mathbf{Y}) \\ \mathbf{X}^\top \text{vec}\,(\mathbf{Y})\end{bmatrix},\ \widetilde{\mathbf{P}}^{-1}\right) \tag{A.16}$$

where

$$\widetilde{\mathbf{P}} \in \mathbb{R}^{(NT+M)\times(NT+M)} = \begin{bmatrix}\mathbf{D_S} + \gamma\mathbf{H}^{-2} & \mathbf{D_S X} \\ \mathbf{X}^\top\mathbf{D_S} & \mathbf{X}^\top\mathbf{D_S X} + \lambda\mathbf{I}_M\end{bmatrix} \tag{A.17}$$

Proof. The distribution of $\mathbf{Y}$ given $\boldsymbol{\theta}$ is given in eq. (4.29). This can also be written as

$$\text{vec}\,(\mathbf{Y})\ |\,\boldsymbol{\theta} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta},\ \mathbf{D_S}\right) \tag{A.18}$$

The prior for $\boldsymbol{\theta}$ is given in equation eq. (4.30). Therefore, using Bayes rule, we can write

$$-2\log\pi(\boldsymbol{\theta}|\mathbf{Y}) \propto \Big(\text{vec}\,(\mathbf{Y}) - \begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta}\Big)^\top \mathbf{D_S}^{-1}\Big(\text{vec}\,(\mathbf{Y}) - \begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta}\Big)$$
$$+\,\boldsymbol{\theta}^\top\begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}^{-1}\boldsymbol{\theta}$$

Using the same trick as in theorem A.1, where we parametrise $\mathbf{S}$ as $\mathbf{S} = \lim_{\epsilon\to 0}\mathbf{S}_\epsilon$. Given this, we can rewrite the above expression as

$$-2\log\pi(\boldsymbol{\theta}|\mathbf{Y}) \propto \Big(\mathrm{vec}\,(\mathbf{Y}) - \big[\mathbf{I}_{NT} \;\; \mathbf{X}\big]\boldsymbol{\theta}\Big)^{\top}\mathbf{D}_{\mathbf{S}}\Big(\mathrm{vec}\,(\mathbf{Y}) - \big[\mathbf{I}_{NT} \;\; \mathbf{X}\big]\boldsymbol{\theta}\Big) + \boldsymbol{\theta}^{\top}\begin{bmatrix}\gamma\mathbf{H}^{-2} & \mathbf{0}\\ \mathbf{0} & \lambda\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta}$$

Collecting like terms, and dropping a constant not dependent on $\boldsymbol{\theta}$, this can be written as

$$-2\log\pi(\boldsymbol{\theta}|\mathbf{Y}) \propto -2\boldsymbol{\theta}^{\top}\begin{bmatrix}\mathrm{vec}\,(\mathbf{Y})\\ \mathbf{X}^{\top}\mathrm{vec}\,(\mathbf{Y})\end{bmatrix} + \boldsymbol{\theta}^{\top}\begin{bmatrix}\mathbf{D}_{\mathbf{S}}+\gamma\mathbf{H}^{-2} & \mathbf{D}_{\mathbf{S}}\mathbf{X}\\ \mathbf{X}^{\top}\mathbf{D}_{\mathbf{S}} & \mathbf{X}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{X}+\lambda\mathbf{I}_M\end{bmatrix}\boldsymbol{\theta}$$

Using the conjugacy of the normal distribution, we can conclude by direct inspection that the posterior covariance matrix is given by $\widetilde{\mathbf{P}}^{-1}$, and the posterior mean is given by

$$\widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathrm{vec}\,(\mathbf{Y})\\ \mathbf{X}^{\top}\mathrm{vec}\,(\mathbf{Y})\end{bmatrix}$$

where

$$\widetilde{\mathbf{P}} = \begin{bmatrix}\mathbf{D}_{\mathbf{S}}+\gamma\mathbf{H}^{-2} & \mathbf{D}_{\mathbf{S}}\mathbf{X}\\ \mathbf{X}^{\top}\mathbf{D}_{\mathbf{S}} & \mathbf{X}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{X}+\lambda\mathbf{I}_M\end{bmatrix}$$

□

Theorem A.7. *This diagonal of the matrices* $\mathbf{H}$ *and* $\mathbf{H}^2$ *can be efficiently calculated.*

Proof. Hi □

Theorem A.8. *Consider the following function,* $\xi(\mathbf{f})$*, which maps a vector* $\mathbf{f} \in \mathbb{R}^N$ *to a scalar.*

$$\xi(\mathbf{f}) = -\mathbf{s}^{\top}\big(\mathbf{y}\circ\log\boldsymbol{\mu}(\mathbf{f}) + (\mathbf{1}-\mathbf{y})\circ\log\left(\mathbf{1}-\boldsymbol{\mu}(\mathbf{f})\right)\big) + \frac{\gamma}{2}\mathbf{f}^{\top}\mathbf{H}^{-2}\mathbf{f} \tag{A.19}$$

The derivative of this function with respect to $\mathbf{f}$ *is given by*

$$\mathbf{g}(\mathbf{f}) = \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}\big) + \gamma\mathbf{H}^{-2}\mathbf{f} \tag{A.20}$$

and the Hessian is given by

$$\mathbf{P} = \mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}) + \gamma\mathbf{H}^{-2} \tag{A.21}$$

where

$$\mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}) = \operatorname{diag}\big(\mathbf{s} \circ \boldsymbol{\mu}(\mathbf{f}) \circ (\mathbf{1} - \boldsymbol{\mu}(\mathbf{f}))\big)$$

Proof. First, recall that the formula for $\boldsymbol{\mu}(\mathbf{f})$ is given by

$$\boldsymbol{\mu}(\mathbf{f}) = \frac{\mathbf{1}}{\mathbf{1} + \exp(-\mathbf{f})} \tag{A.22}$$

This further implies that

$$\mathbf{1} - \boldsymbol{\mu} = \frac{\exp(-\mathbf{f})}{\mathbf{1} + \exp(-\mathbf{f})} \tag{A.23}$$

$$\log \boldsymbol{\mu} = -\log\big(\mathbf{1} + \exp(-\mathbf{f})\big) \tag{A.24}$$

$$\log(\mathbf{1} - \boldsymbol{\mu}) = -\mathbf{f} - \log\big(\mathbf{1} + \exp(-\mathbf{f})\big) \tag{A.25}$$

Note that all operations in these expressions, including division, logarithms, and exponentiation should be understood as element-wise. Using these equations, we can rewrite expression for $\xi(\mathbf{f})$.

$$\begin{aligned}
\xi(\mathbf{f}) &= -\log \boldsymbol{\mu}^\top(\mathbf{s} \circ \mathbf{y}) - \log(\mathbf{1} - \boldsymbol{\mu})^\top(\mathbf{s} \circ (\mathbf{1} - \mathbf{y})) + \frac{\gamma}{2}\mathbf{f}^\top\mathbf{H}^{-2}\mathbf{f} \\
&= \log\big(\mathbf{1} + \exp(-\mathbf{f})\big)^\top(\mathbf{s} \circ \mathbf{y}) + \big(\mathbf{f} + \log\big(\mathbf{1} + \exp(-\mathbf{f})\big)\big)^\top(\mathbf{s} \circ (\mathbf{1} - \mathbf{y})) + \frac{\gamma}{2}\mathbf{f}^\top\mathbf{H}^{-2}\mathbf{f} \\
&= \mathbf{f}^\top(\mathbf{s} \circ (\mathbf{1} - \mathbf{y})) + \log\big(\mathbf{1} + \exp(-\mathbf{f})\big)^\top\mathbf{s} + \frac{\gamma}{2}\mathbf{f}^\top\mathbf{H}^{-2}\mathbf{f}
\end{aligned}$$

Now, take the derivate of this expression with respect to $\mathbf{f}$, using the chain rule for the middle term.

$$\begin{aligned}\frac{\partial \xi(\mathbf{f})}{\partial \mathbf{f}} &= \mathbf{s} \circ (\mathbf{1} - \mathbf{y}) - \mathbf{s} \circ \frac{\exp(-\mathbf{f})}{\mathbf{1} + \exp(-\mathbf{f})} + \gamma \mathbf{H}^{-2}\mathbf{f} \\ &= \mathbf{s} \circ (\mathbf{1} - \mathbf{y}) + \mathbf{s} \circ (\boldsymbol{\mu} - \mathbf{1}) + \gamma \mathbf{H}^{-2}\mathbf{f} \\ &= \mathbf{s} \circ (\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}) + \gamma \mathbf{H}^{-2}\mathbf{f} \\ &= \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big(\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}\big) + \gamma \mathbf{H}^{-2}\mathbf{f}\end{aligned}$$

This is the given expression for the gradient, $\mathbf{g}(\mathbf{f})$. The Hessian can be found by taking the derivative of the gradient with respect to $\mathbf{f}$.

$$\begin{aligned}\frac{\partial \mathbf{g}}{\partial \mathbf{f}} &= \frac{\partial}{\partial \mathbf{f}}\Big[\mathbf{s} \circ \big(\boldsymbol{\mu}(\mathbf{f}) - \mathbf{y}\big) + \gamma \mathbf{H}^{-2}\mathbf{f}\Big] \\ &= \frac{\partial}{\partial \mathbf{f}}\left[\frac{\mathbf{s}}{\mathbf{1} + \exp(-\mathbf{f})} + \gamma \mathbf{H}^{-2}\mathbf{f}\right] \\ &= \operatorname{diag}\left(\exp(-\mathbf{f}) \circ \frac{\mathbf{s}}{\big(\mathbf{1} + \exp(-\mathbf{f})\big)^2}\right) + \gamma \mathbf{H}^{-2} \\ &= \operatorname{diag}\left(\mathbf{s} \circ \frac{\mathbf{1}}{\mathbf{1} + \exp(-\mathbf{f})} \circ \frac{\exp(-\mathbf{f})}{\mathbf{1} + \exp(-\mathbf{f})}\right) + \gamma \mathbf{H}^{-2} \\ &= \operatorname{diag}\Big(\mathbf{s} \circ \boldsymbol{\mu}(\mathbf{f}) \circ \big(\mathbf{1} - \boldsymbol{\mu}(\mathbf{f})\big)\Big) + \gamma \mathbf{H}^{-2} \\ &= \mathbf{D}_{\boldsymbol{\mu}}(\mathbf{f}) + \gamma \mathbf{H}^{-2}\end{aligned}$$

□

Bibliography

Abraham, R., Marsden, J. E., and Rațiu, T. S. (1988). *Manifolds, tensor analysis, and applications*. Springer-Verlag, New York, 2nd ed edition.

Ahmed, H. B., Dare, D., and Boudraa, A. (2017). Graph signals classification using total variation and graph energy informations. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 667–671.

Ahmed, N., Natarajan, T., and Rao, K. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93.

Anis, A., Gadde, A., and Ortega, A. (2016). Efficient sampling set selection for bandlimited graph signals using graph spectral proxies. *IEEE Transactions on Signal Processing*, 64(14):3775–3789.

Antonian, E., Peters, G. W., and Chantler, M. (2023). Pykronecker: A python library for the efficient manipulation of kronecker products and related structures. *Journal of Open Source Software*, 8(81):4900.

Arroyo, A., Scalzo, B., Stanković, L., and Mandic, D. P. (2022). Dynamic portfolio cuts: A spectral approach to graph-theoretic diversification. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5468–5472.

Atasoy, S., Donnelly, I., and Pearson, J. (2016). Human brain networks function in connectome-specific harmonic waves. *Nature Communications*, 7(1).

Atchinson, J. and Shen, S. (1980). Logistic-normal distributions:Some properties and uses. *Biometrika*, 67(2):261–272.

Aubert, G. and Kornprobst, P. (2006). *Mathematical problems in image processing*. Applied mathematical sciences. Springer, New York, NY, 2 edition.

Barik, S., Bapat, R. B., and Pati, S. (2015). On the laplacian spectra of product graphs. *Applicable Analysis and Discrete Mathematics*, 9:39–58.

Barik, S., Kalita, D., Pati, S., and Sahoo, G. (2018). Spectra of graphs resulting from various graph operations and products: a survey. *Special Matrices*, 6:323 – 342.

Baumgardner, M. F., Biehl, L. L., and Landgrebe, D. A. (2015). 220 band aviris hyperspectral image data set: June 12, 1992 indian pine test site 3.

Bekas, C., Kokiopoulou, E., and Saad, Y. (2007). An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57(11):1214–1229.

Belkin, M., Matveeva, I., and Niyogi, P. (2004). Regularization and semi-supervised learning on large graphs. In Shawe-Taylor, J. and Singer, Y., editors, *Learning Theory*, pages 624–638, Berlin, Heidelberg. Springer Berlin Heidelberg.

Belkin, M. and Niyogi, P. (2002). Using manifold structure for partially labelled classification. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 953–960, Cambridge, MA, USA. MIT Press.

Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396.

Bhatia, R. (1997). *Matrix analysis*. Number 169 in Graduate texts in mathematics. Springer, New York.

Box, G. E. P. and Muller, M. E. (1958). A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29(2):610 – 611.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brenner, S. C., Scott, L. R., and Scott, L. R. (2008). *The mathematical theory of finite element methods*, volume 3. Springer.

Burden, R. and Faires, J. (2010). *Numerical Analysis*. Cengage Learning.

CALFIRE (2023). The department of forestry and fire protection: Incident data. https://www.fire.ca.gov/incidents.

Chakraborty, P. (2017). Trend Filtering in Network Time Series with Applications to Traffic Incident Detection. *Time Series Workshop, Neural Information Processing Systems*, pages 1–4.

Chakraborty, P., Hegde, C., and Sharma, A. (2019). Data-driven parallelizable traffic incident detection using spatio- temporally denoised robust thresholds. *Transportation Research Part C*, 105(June):81–99.

Cheeger, J. (1971). *A Lower Bound for the Smallest Eigenvalue of the Laplacian*, pages 195–200. Princeton University Press, Princeton.

Cheng, R. and Li, Q. (2021). Modeling the momentum spillover effect for stock prediction via attribute-driven graph attention networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):55–62.

Chi, Y., Jiang, J., Zhou, F., and Xu, S. (2022). A distributed algorithm for reconstructing time-varying graph signals. *Circuits, Systems, and Signal Processing*, 41(6):3624–3641.

Chong, Y., Ding, Y., Yan, Q., and Pan, S. (2020). Graph-based semi-supervised learning: A review. *Neurocomputing*, 408:216–230.

Chung, F. (1997). *Spectral Graph Theory*. Conference Board of Mathematical Sciences. American Mathematical Society.

Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., and Phan, H. A. (2015). Tensor Decompositions for Signal Processing Applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163.

Cleland, S. E., West, J. J., Jia, Y., Reid, S., Raffuse, S., O'Neill, S., and Serre, M. L. (2020). Estimating wildfire smoke concentrations during the october 2017 california fires through BME space/time data fusion of observed, modeled, and satellite-derived PM2.5. *Environ. Sci. Technol.*, 54(21):13439–13447.

Collatz, L. and Sinogowitz, U. (1957). Spektren endlicher grafen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 21:63–77.

Colonnese, S., Petti, M., Farina, L., Scarano, G., and Cuomo, F. (2021). Protein-protein interaction prediction via graph signal processing. *IEEE Access*, 9:142681–142692.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301.

Cvetkovic, D., Doob, M., and Sachs, H. (1980). *Spectra of Graphs: Theory and Application*. Pure and applied mathematics : a series of monographs and textbooks. Academic Press.

DeBoor, C. (1979). Efficient computer manipulation of tensor products. *ACM Transactions on Mathematmal Software*, 5:173–182.

Dees, B. S., Stanković, L., Constantinides, A. G., and Mandic, D. P. (2020). Portfolio cuts: A graph-theoretic framework to diversification. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8454–8458.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2017). Convolutional neural networks on graphs with fast localized spectral filtering.

Demmel, J. W. (1997). *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia.

DeResende, B. M. and Costa, L. d. (2020). Characterization and comparison of large directed networks through the spectra of the magnetic Laplacian. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(7):073141.

Dong, X., Mavroeidis, D., Calabrese, F., and Frossard, P. (2015). Multiscale event detection in social media. *Data Min. Knowl. Discov.*, 29(5):1374–1405.

Dong, X., Thanou, D., Frossard, P., and Vandergheynst, P. (2016). Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173.

Dong, X., Thanou, D., Rabbat, M., and Frossard, P. (2019). Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63.

Donoho, D. L. and Grimes, C. (2003). Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596.

Duhamel, P. and Vetterli, M. (1990). Fast fourier transforms: A tutorial review and a state of the art. *Signal Processing*, 19(4):259–299.

Elias, V. R. M., Gogineni, V. C., Martins, W. A., and Werner, S. (2022). Kernel regression over graphs using random fourier features. *IEEE Transactions on Signal Processing*, 70:936–949.

Elman, H. C. (1982). *Iterative methods for large, sparse, nonsymmetric systems of linear equations*. PhD thesis, Yale University.

EPA (2023). Us environmental protection agency: Air quality system data mart. https://www.epa.gov/outdoor-air-quality-data. Accessed: April 10, 2023.

Fackler, P. L. (2019). Algorithm 993: Efficient computation with kronecker products. *ACM Trans. Math. Softw.*, 45(2).

Fan, J., Li, R., Zhang, C., and Zou, H. (2020). *Statistical Foundations of Data Science*. Chapman & Hall/CRC Data Science Series. CRC Press.

Fanuel, M., Alaíz, C. M., and Suykens, J. A. K. (2017). Magnetic eigenmaps for community detection in directed networks. *Physical Review E*, 95(2).

Federal Reserve Bank of St. Louis (2023). Fred® (federal reserve economic data) online api. accessed june 30th, 2023. https://fred.stlouisfed.org.

Fiedler, M. (1973). Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305.

Fletcher, R. (2013). *Practical Methods of Optimization*. Wiley.

Fortune, S. (1986). A sweepline algorithm for voronoi diagrams. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, page 313–322, New York, NY, USA. Association for Computing Machinery.

Friedman, J., Hastie, T., and Tibshirani, R. (2007). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441.

Furutani, S., Shibahara, T., Akiyama, M., Hato, K., and Aida, M. (2020). Graph signal processing for directed graphs based on the hermitian laplacian. In Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M., and Robardet, C., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 447–463, Cham. Springer International Publishing.

Gadde, A. and Ortega, A. (2015). A probabilistic interpretation of sampling theory of graph signals. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3257–3261.

Gama, F., Isufi, E., Leus, G., and Ribeiro, A. (2020). Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Processing Magazine*, 37(6):128–138.

Gao, F. and Han, L. (2012). Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Comput. Optim. Appl.*, 51(1):259–277.

Gao, J., Ying, X., Xu, C., Wang, J., Zhang, S., and Li, Z. (2021). Graph-based stock recommendation by time-aware relational attention network. *ACM Trans. Knowl. Discov. Data*, 16(1).

Gerschgorin, S. (1931). Uber die abgrenzung der eigenwerte einer matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika*, 7(3):749–754.

Giraldo, J. H., Mahmood, A., Garcia-Garcia, B., Thanou, D., and Bouwmans, T. (2022). Reconstruction of time-varying graph signals via sobolev smoothness. *IEEE Transactions on Signal and Information Processing over Networks*, 8:201–214.

Goldsberry, L., Huang, W., Wymbs, N. F., Grafton, S. T., Bassett, D. S., and Ribeiro, A. (2017). Brain signal analytics from graph signal processing perspective. In *2017*

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 851–855.

Granata, J., Conner, M., and Tolimieri, R. (1992). Recursive Fast Algorithms and the Role of the Tensor Product. *IEEE Transactions on Signal Processing*, 40(12):2921–2930.

Grassi, F., Loukas, A., Perraudin, N., and Ricaud, B. (2018). A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs. *IEEE Transactions on Signal Processing*, 66(3):817–829.

Grote, M. J. and Huckle, T. (1997). Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853.

Guestrin, C., Bodik, P., Thibaux, R., Paskin, M., and Madden, S. (2004). Distributed regression: an efficient framework for modeling sensor network data. In *Third International Symposium on Information Processing in Sensor Networks, 2004. IPSN 2004*, pages 1–10.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hartigan, J. A. and Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

Harzheim, E. (2005). Chapter 4 - products of orders. In *Ordered Sets*, volume 7 of *Advances in Mathematics*. Springer-Verlag, New York.

Hasanzadeh, A., Liu, X., Duffield, N. G., Narayanan, K. R., and Chigoy, B. T. (2017). A graph signal processing approach for real-time traffic prediction in transportation networks. *arXiv: Signal Processing*.

He, K., Stankovic, L., Liao, J., and Stankovic, V. (2018). Non-intrusive load disaggregation using graph signal processing. *IEEE Transactions on Smart Grid*, 9(3):1739–1747.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–435.

Hoffman, A. J. (1969). The change in the least eigenvalued of the adjacency matrix of a graph under imbedding. *SIAM Journal on Applied Mathematics*, 17(4):664–671.

Holmes, D. E. and Jain, L. C. (2008). Introduction to bayesian networks. In *Innovations in Bayesian Networks*.

Horn, R. A. and Johnson, C. R. (2012). *Matrix Analysis*. Cambridge University Press, 2 edition.

Hu, C., Cheng, L., Sepulcre, J., Johnson, K. A., Fakhri, G. E., Lu, Y. M., and Li, Q. (2015). A spectral graph regression model for learning brain connectivity of alzheimer's disease. *PLOS ONE*, 10(5):1–24.

Huang, W., Goldsberry, L., Wymbs, N. F., Grafton, S. T., Bassett, D. S., and Ribeiro, A. (2016). Graph frequency analysis of brain signals. *IEEE Journal of Selected Topics in Signal Processing*, 10(7):1189–1203.

Huckel, E. (1931). Quantentheoretische beiträge zum benzolproblem. *Zeitschrift für Physik*, 70(3-4):204–286.

Hull, J. (2009). *Options, Futures and Other Derivatives*. Eastern economy edition. Pearson/Prentice Hall.

Hutchinson, M. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450.

Imrich, W. and Klavžar, S. (2000). *Product Graphs: Structure and Recognition*. A Wiley-Interscience publication. Wiley.

Ioannidis, V. N., Romero, D., and Giannakis, G. B. (2016). Kernel-based reconstruction of space-time functions via extended graphs. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 1829–1833.

Ioannidis, V. N., Romero, D., and Giannakis, G. B. (2018). Inference of spatio-temporal functions over graphs via multikernel kriged kalman filtering. *IEEE Transactions on Signal Processing*, 66(12):3228–3239.

Isufi, E., Gama, F., Shuman, D. I., and Segarra, S. (2022). Graph filters for signal processing and machine learning on graphs.

Isufi, E., Loukas, A., Simonetto, A., and Leus, G. (2017). Autoregressive moving average graph filtering. *IEEE Transactions on Signal Processing*, 65(2):274–288.

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

Jablonski, I. (2017). Graph signal processing in applications to sensor networks, smart grids, and smart cities. *IEEE Sensors Journal*, 17(23):7659–7666.

Jaffe, D. A., O'Neill, S. M., Larkin, N. K., Holder, A. L., Peterson, D. L., Halofsky, J. E., and Rappold, A. G. (2020). Wildfire and prescribed burning impacts on air quality in the united states. *Journal of the Air & Waste Management Association*, 70(6):583–615.

Jha, K., Saha, S., and Singh, H. (2022). Prediction of protein–protein interaction using graph neural networks. *Scientific Reports*, 12(1).

Jiang, J. (2012). Introduction to spectral graph theory.

Kalofolias, V. (2016). How to learn a graph from smooth signals.

Kaveh, A. and Alinejad, B. (2011). Laplacian matrices of product graphs: applications in structural mechanics. *Acta Mechanica*, 222:331–350.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *J. Comput. Aided Mol. Des.*, 30(8):595–608.

Kelley, C. T. (1995). *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics.

Kelley, C. T. (2003). *Solving Nonlinear Equations with Newton's Method*. Society for Industrial and Applied Mathematics.

Kiers, H. A. L. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks.

Kolaczyk, E. D. (2009). *Statistical Analysis of Network Data*. Springer New York.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Kondor, R. I. and Lafferty, J. (2002). Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322.

Kriege, N. M., Johansson, F. D., and Morris, C. (2020). A survey on graph kernels. *Applied Network Science*, 5(1):1–42.

Lake, B. and Tenenbaum, J. B. (2010). Discovering structure by learning sparse graphs. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*.

Le, C. M. and Li, T. (2022). Linear Regression and Its Inference on Noisy Network-Linked Data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(5):1851–1885.

LeMagoarou, L. and Gribonval, R. (2016). Are there approximate fast fourier transforms on graphs? In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4811–4815, Shanghai. IEEE.

Leus, G., Marques, A. G., Moura, J. M., Ortega, A., and Shuman, D. I. (2023). Graph signal processing: History, development, impact, and outlook. *IEEE Signal Processing Magazine*, 40(4):49–60.

Li, J., Bioucas-Dias, J. M., and Plaza, A. (2012). Spectral–spatial hyperspectral image segmentation using subspace multinomial logistic regression and markov random fields. *IEEE Transactions on Geoscience and Remote Sensing*, 50(3):809–823.

Li, R., Yuan, X., Radfar, M., Marendy, P., Ni, W., O'Brien, T. J., and Casillas-Espinosa, P. M. (2023). Graph signal processing, graph neural network and graph learning on biological data: A systematic review. *IEEE Reviews in Biomedical Engineering*, 16:109–135.

Li, T., Levina, E., and Zhu, J. (2019). Prediction models for network-linked data. *The Annals of Applied Statistics*, 13(1):132 – 164.

Little, R. and Rubin, D. (2019). *Statistical Analysis with Missing Data*. Wiley Series in Probability and Statistics. Wiley.

Loukas, A. and Foucard, D. (2016). Frequency analysis of time-varying graph signals. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 346–350.

MacQueen, J. (1967). Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA.

Makhoul, J. (1980). A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34.

Marques, A. G., Segarra, S., and Mateos, G. (2020). Signal processing on directed graphs: The role of edge directionality when processing and learning from network data. *IEEE Signal Processing Magazine*, 37(6):99–116.

Marti, G., Nielsen, F., Bińkowski, M., and Donnat, P. (2021). *A Review of Two Decades of Correlations, Hierarchies, Networks and Clustering in Financial Markets*, pages 245–274. Springer International Publishing, Cham.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423.

Mateos, G., Segarra, S., Marques, A. G., and Ribeiro, A. (2019). Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Processing Magazine*, 36(3):16–43.

Menoret, M., Farrugia, N., Pasdeloup, B., and Gripon, V. (2017). Evaluating graph signal processing for neuroimaging through classification and dimensionality reduction. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 618–622.

Miao, X., Jiang, A., Zhu, Y., and Kwan, H. K. (2022). A joint learning framework for gaussian processes regression and graph learning. *Signal Processing*, 201:108708.

Mieghem, P. v. (2010). *Graph Spectra for Complex Networks*. Cambridge University Press.

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2022). Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, MA.

Narang, S. K., Gadde, A., and Ortega, A. (2013a). Signal processing techniques for interpolation in graph structured data. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5445–5449.

Narang, S. K., Gadde, A., and Ortega, A. (2013b). Signal processing techniques for interpolation in graph structured data. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5445–5449.

Nelder, J. A. and Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384.

ONS (2019). Local authority districts (december 2019) boundaries uk buc. https://www.data.gov.uk/dataset/e73c1990-ad5d-4184-8fb1-b474b94918ba/local-authority-districts-december-2019-boundaries-uk-buc. Accessed: 2023-05-30.

Orieux, F., Feron, O., and Giovannelli, J.-F. (2012). Sampling high-dimensional gaussian distributions for general linear inverse problems. *IEEE Signal Processing Letters*, 19(5):251–254.

Ortega, A. (2022). *Introduction to Graph Signal Processing*. Cambridge University Press.

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M. F., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828.

Papandreou, G. and Yuille, A. L. (2010). Gaussian sampling by local perturbations. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Pare, P. E., Beck, C. L., and Basar, T. (2020). Modeling, estimation, and analysis of epidemics over networks: An overview. *Annual Reviews in Control*, 50:345–360.

Pereyra, V. and Scherer, G. (1973). Efficient computer manipulation of tensor products with applications to multidimensional approximation. *Mathematics of Computation*, 27:595–605.

Perraudin, N. and Vandergheynst, P. (2017). Stationary signal processing on graphs. *IEEE Transactions on Signal Processing*, 65(13):3462–3477.

Peters, G., Zhu, R., Tzougas, G., Rabitti, G., and Yusuf, I. (2022). The role and significance of green bonds in funding transition to a low carbon economy: A case study forecasting portfolios of green bond instrument returns. *SSRN Electron. J.*

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

Puschel, M. and Moura, J. M. F. (2003). The algebraic approach to the discrete cosine and sine transforms and their fast algorithms. *SIAM Journal on Computing*, 32(5):1280–1316.

Puschel, M. and Moura, J. M. F. (2006). Algebraic signal processing theory.

Puschel, M. and Moura, J. M. F. (2008). Algebraic signal processing theory: Foundation and 1-d time. *IEEE Transactions on Signal Processing*, 56(8):3572–3585.

Qiao, L., Zhang, L., Chen, S., and Shen, D. (2018). Data-driven graph construction and graph learning: A review. *Neurocomputing*, 312:336–351.

Qiu, K., Mao, X., Shen, X., Wang, X., Li, T., and Gu, Y. (2017). Time-varying graph signal reconstruction. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):870–883.

Rabiner, L. and Gold, B. (1975). *Theory and Application of Digital Signal Processing*. Prentice-Hall signal processing series. Prentice-Hall.

Ramakrishna, R. and Scaglione, A. (2021). Grid-graph signal processing (grid-gsp): A graph signal processing framework for the power grid. *IEEE Transactions on Signal Processing*, 69:2725–2739.

Rao, K. and Yip, P. (1990). *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Elsevier Science & Technology Books.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning*. The MIT Press.

Renteln, P. (2013). *Manifolds, Tensors, and Forms: An Introduction for Mathematicians and Physicists*. Cambridge University Press.

Rimleanscaia, O. and Isufi, E. (2020). Rational chebyshev graph filters. In *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pages 736–740.

Rivlin, T. (2020). *Chebyshev Polynomials*. Dover Books on Mathematics. Dover Publications.

Romero, D., Ioannidis, V. N., and Giannakis, G. B. (2017a). Kernel-based reconstruction of space-time functions on dynamic graphs. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):856–869.

Romero, D., Ma, M., and Giannakis, G. B. (2017b). Kernel-based reconstruction of graph signals. *IEEE Transactions on Signal Processing*, 65(3):764–778.

Roth, W. E. (1934). On direct product matrices. *Bulletin of the American Mathematical Society*.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition.

Saad, Y. and Schultz, M. H. (1986). Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *Siam Journal on Scientific and Statistical Computing*, 7:856–869.

Sandryhaila, A. and Moura, J. M. F. (2013a). Classification via regularization on graphs. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 495–498.

Sandryhaila, A. and Moura, J. M. F. (2013b). Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656.

Sandryhaila, A. and Moura, J. M. F. (2013c). Discrete signal processing on graphs: Frequency analysis.

Sardellitti, S., Barbarossa, S., and Lorenzo, P. D. (2017). On the graph fourier transform for directed graphs. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):796–811.

Sayama, H. (2016). Estimation of laplacian spectra of direct and strong product graphs. *Discrete Applied Mathematics*, 205:160–170.

Scholkopf, B. and Smola, A. J. (2018). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press.

Shafipour, R., Khodabakhsh, A., Mateos, G., and Nikolova, E. (2019). A directed graph fourier transform with spread frequency components. *IEEE Transactions on Signal Processing*, 67(4):946–960.

Shubin, M. A. (1994). Discrete magnetic laplacian. *Communications in Mathematical Physics*, 164(2):259–275.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.

Shuman, D. I., Vandergheynst, P., Kressner, D., and Frossard, P. (2018). Distributed signal processing via chebyshev polynomial approximation. *IEEE Transactions on Signal and Information Processing over Networks*, 4(4):736–751.

Smith, W. and Smith, J. (1995). *Handbook of Real-Time Fast Fourier Transforms: Algorithms to Product Testing*. Wiley.

Smola, A. J. and Kondor, R. (2003). Kernels and regularization on graphs. In Schölkopf, B. and Warmuth, M. K., editors, *Learning Theory and Kernel Machines*, pages 144–158, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sneddon, I. (1995). *Fourier Transforms*. Dover books on mathematics. Dover Publications.

Spielman, D. A. (2019). *Spectral and Algebraic Graph Theory*. Upcoming.

Srivastava, D., Bagler, G., and Kumar, V. (2023). Graph signal processing on protein residue networks helps in studying its biophysical properties. *Physica A: Statistical Mechanics and its Applications*, 615:128603.

Stanley, J. S., Chi, E. C., and Mishne, G. (2020). Multiway Graph Signal Processing on Tensors: Integrative Analysis of Irregular Geometries. *IEEE Signal Processing Magazine*, 37(6):160–173.

Tanaka, Y., Eldar, Y. C., Ortega, A., and Cheung, G. (2020). Sampling signals on graphs: From theory to applications. *IEEE Signal Processing Magazine*, 37(6):14–30.

Tang, J. M. and Saad, Y. (2012). A probing method for computing the diagonal of a matrix inverse. *Numerical Linear Algebra with Applications*, 19(3):485–501.

Tay, D. B. (2021). Sensor network data denoising via recursive graph median filters. *Signal Processing*, 189:108302.

Tolimieri, R., An, M., Lu, C., and Burrus, C. S. (2013). *Algorithms for discrete Fourier transform and convolution*. Signal Processing and Digital Filtering. Springer, New York, NY, 1989 edition.

Tran, N., Ambos, H., and Jung, A. (2020). Classifying partially labeled networked data via logistic network lasso. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3832–3836.

Tseng, C.-C. (2020). Rational graph filter design using spectral transformation and iir digital filter. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 247–250.

Venkitaraman, A., Chatterjee, S., and Handel, P. (2020). Gaussian processes over graphs. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5640–5644.

Venkitaraman, A., Chatterjee, S., and Händel, P. (2019). Predicting graph signals using kernel regression where the input signal is agnostic to a graph. *IEEE Transactions on Signal and Information Processing over Networks*, 5(4):698–710.

Vinicius, J. d. M. C., Ying, J., and Palomar, D. P. (2020). Algorithms for learning graphs in financial markets.

Vono, M., Dobigeon, N., and Chainais, P. (2022). High-dimensional gaussian sampling: A review and a unifying approach based on a stochastic proximal point algorithm. *SIAM Review*, 64(1):3–56.

Wagner, R., Baraniuk, R., Du, S., Johnson, D., and Cohen, A. (2006). An architecture for distributed wavelet analysis and processing in sensor networks. In *2006 5th International Conference on Information Processing in Sensor Networks*, pages 243–250.

Wagner, R., Choi, H., Baraniuk, R., and Delouille, V. (2005). Distributed wavelet transform for irregular sensor network grids. In *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, pages 1196–1201.

Wang, J., Zhang, S., Xiao, Y., and Song, R. (2022a). A review on graph neural network methods in financial applications.

Wang, X., Liu, P., and Gu, Y. (2015a). Local-set-based graph signal reconstruction. *IEEE Transactions on Signal Processing*, 63(9):2432–2444.

Wang, X., Wang, M., and Gu, Y. (2015b). A distributed tracking algorithm for reconstruction of graph signals. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):728–740.

Wang, Y., Ahsan, U., Li, H., and Hagen, M. (2022b). A comprehensive review of modern object segmentation approaches. *Foundations and Trends in Computer Graphics and Vision*, 13(2-3):111–283.

Wang, Y., Zhao, B., Li, X., Luan, W., and Liu, B. (2022c). Abnormal battery identification via graph signal processing method. In *2022 7th Asia Conference on Power and Electrical Engineering (ACPEE)*, pages 208–212.

Wang, Y.-X., Sharpnack, J., Smola, A. J., and Tibshirani, R. J. (2016). Trend filtering on graphs. *J. Mach. Learn. Res.*, 17(1):3651–3691.

Xiao, Z., Fang, H., and Wang, X. (2021). Anomalous iot sensor data detection: An efficient approach enabled by nonlinear frequency-domain graph analysis. *IEEE Internet of Things Journal*, 8(5):3812–3821.

Xiu, C., Sun, Y., and Peng, Q. (2022). Modelling traffic as multi-graph signals: Using domain knowledge to enhance the network-level passenger flow prediction in metro systems. *Journal of Rail Transport Planning & Management*, 24:100342.

Ying, W., Rui, X., Xiaoyang, M., Qiang, F., Jinshuai, Z., and Runze, Z. (2022). Spectral graph theory-based recovery method for missing harmonic data. *IEEE Transactions on Power Delivery*, 37(5):3688–3697.

Zhang, C., Florencio, D., and Chou, P. A. (2015). Graph signal processing - a probabilistic framework. Technical Report MSR-TR-2015-31, Microsoft Research.

Zhang, S., Ma, X., Fang, Z., Pan, H., Yang, G., and Arce, G. R. (2023). Financial time series forecasting based on momentum-driven graph signal processing. *Applied Intelligence*.

Zhang, X., He, Y., Brugnone, N., Perlmutter, M., and Hirn, M. (2021). Magnet: A neural network for directed graphs.

Zhang, Z.-W., Jing, X.-Y., and Wang, T.-J. (2017). Label propagation based semi-supervised learning for software defect prediction. *Autom. Softw. Eng.*, 24(1):47–69.

Zhao, X., Dai, Q., Wu, J., Peng, H., Liu, M., Bai, X., Tan, J., Wang, S., and Yu, P. S. (2023). Multi-view tensor graph neural networks through reinforced aggregation. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4077–4091.

Zheng, D., Ma, X., Wang, Y., Wang, Y., and Luo, H. (2022). Non-intrusive load monitoring based on the graph least squares reconstruction method. *IEEE Transactions on Power Delivery*, 37(4):2562–2570.

Zhi, Y.-C., Ng, Y. C., and Dong, X. (2023). Gaussian processes on graphs via spectral kernel learning. *IEEE Transactions on Signal and Information Processing over Networks*, 9:304–314.

Zhou, D. and Schölkopf, B. (2004). A regularization framework for learning from graph data. In *ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields*.

Zhou, F., Jiang, J., and Tay, D. B. (2022a). Distributed reconstruction of time-varying graph signals via a modified newton's method. *Journal of the Franklin Institute*, 359(16):9401–9421.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

Zhou, X., Liu, S., Xu, W., Xin, K., Wu, Y., and Meng, F. (2022b). Bridging hydraulics and graph signal processing: A new perspective to estimate water distribution network pressures. *Water Research*, 217:118416.

Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 912–919. AAAI Press.

Zhu, X. and Rabbat, M. (2012). Graph spectral compressed sensing for sensor networks. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2865–2868.