Heriot-Watt University

Masters Thesis

Bayesian Reconsruction and Regression over Networks

Author:
John Smith

Supervisor:
Dr. James Smith

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

in the

School of Mathematical and Computer Sciences

May 2023

Declaration of Authorship

I, John Smith, declare that this thesis titled, 'Bayesian Reconsruction and Regression over Networks' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: _____

Date: _____

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

Abstract

The Thesis Abstract is written here (and usually kept to just this page).

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor :)

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

1.1 Background and Definitions

Graph Signal Processing (GSP) is a rapidly evolving field that sits at the intersection between spectral graph theory, statistics and data science [Shuman et al., 2013]. In this context, a graph is an abstract collection of objects in which any pair may be, in some sense, "related". These objects are referred to as vertices (or nodes) and their connections as edges [Newman, 2018]. GSP is concerned with the mathematical analysis of signals that are defined over the nodes of a graph, referred to simply as *graph signals*.

A graph signal can be thought of as a value that is measured simultaneously at each node in a graph. In practice, it is represented as a vector where each element corresponds to a single node. For example, consider a social network where each node represents an individual and presence of an edge between two nodes indicates that the two individuals have met. An example of a graph signal in this context could be the age of each person in the network. Figure 1.2 shows a graphical depiction of a signal defined over a network.

Graphs and graph signals have proven a useful way to describe data across a broad range of applications owing to their flexibility and relative simplicity. They are able to summarise the of properties large, complex systems within a single easily-digestible structure. Much of the data

The GSP community, in particular, is focused on generalising tools designed for traditional signal processing tasks to irregular graph-structured domains.

[Ortega et al., 2018]

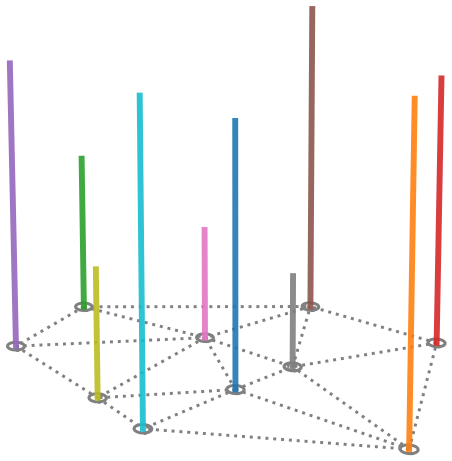


FIGURE 1.1: A graphical depiction of a graph signal. Here, the nodes are represented by circles, the edges as dotted lines, and the value of the signal at each node is represented by the height of its associated bar.

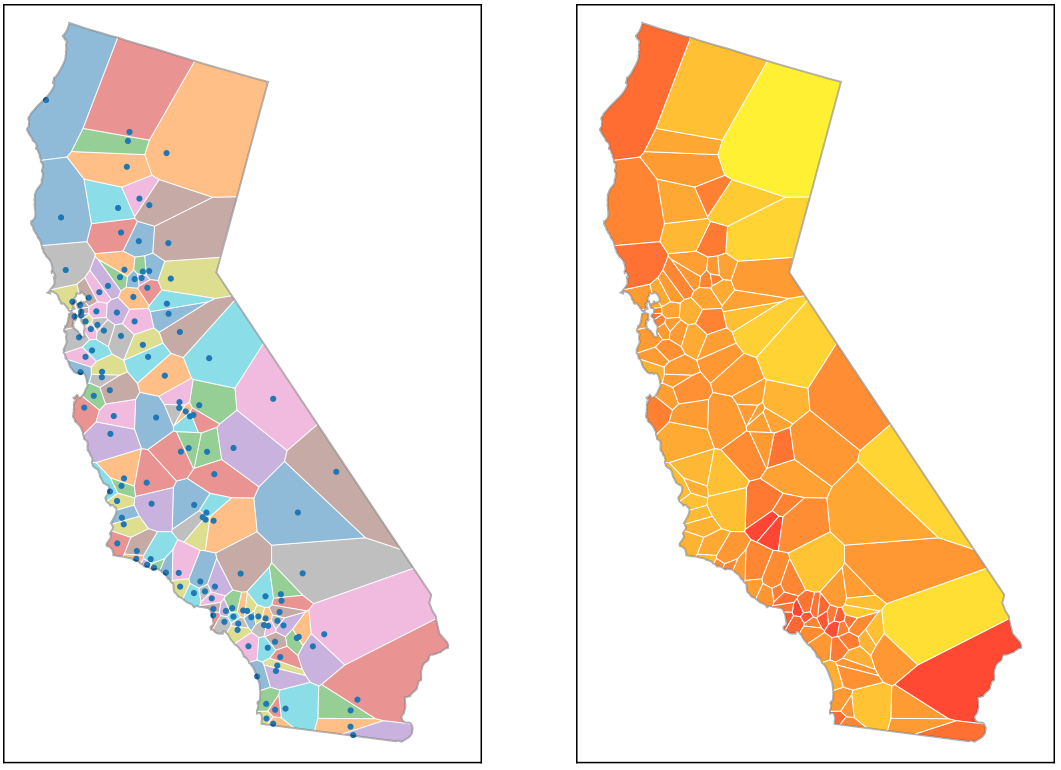


FIGURE 1.2: Air pollution monitors in California

1.2 Thesis overview

Chapter 2

Literature Review and Thesis Outline

2.1 Graph Signal Processing

2.1.1 A broad overview of the field

2.1.2 The graph Laplacian

LeMagoarou and Gribonval [2016]

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

TABLE 2.1: Isotropic graph filter functions

2.1.3 Graph filters

2.1.3.1 Graph Kernels

2.2 Regression and Reconstruction

2.2.1 Graph Signal Reconstruction

Qiu et al. [2017] Time varying signal reconstruction.

2.2.2 Kernel Graph Regression

Takeda et al. [2007]

Elias et al. [2022]

Venkitaraman et al. [2019]

2.2.2.1 Gaussian Processes on Graphs

Venkitaraman et al. [2020]

Miao et al. [2022]: GPoG + graph learning

2.2.3 Regression with Network Cohesion

Le and Li [2022]

Li et al. [2019]

2.3 GSP on higher order graphs

Multiway data processing

Smilde et al. [2004] Kroonenberg [2008]

Ji and Tay [2019]

Cammoun et al. [2009]

2.3.1 Multi-Layer Graph Signal Processing

Zhang et al. [2022] describe M-GSP

Zhang et al. [2018] extend M-GSP to multiple layers with different number of nodes by adding fake nodes in where they are missing from layers.

2.3.2 Multi-Way Graph Signal Processing

Zhao et al. [2023]

Li et al. [2012]

Chapter 3

Kernel Generalized Least Squares Regression for Network Data

3.1 Kernel Graph Regression with Missing Values

3.2 GLS Kernel Graph Regression

3.2.1 A Gauss-Markov estimator

3.2.2 AR(1) processes

3.2.3 Experiments

Chapter 4

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 4.1 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 4.2, 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 4.3, 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.

4.1 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.

4.1.1 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.

$$\mathcal{G} = \mathcal{G}_A \diamond \mathcal{G}_B = (\mathcal{V}, \mathcal{E}) \tag{4.1}$$

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

$$\mathcal{V} = \mathcal{V}_A \times \mathcal{V}_B = \{(a, b) \in \mathbb{N}^2 \,|\, a \leq A \text{ and } b \leq B\} \tag{4.2}$$

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 4.1. 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 4.1.

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 4.1: 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.

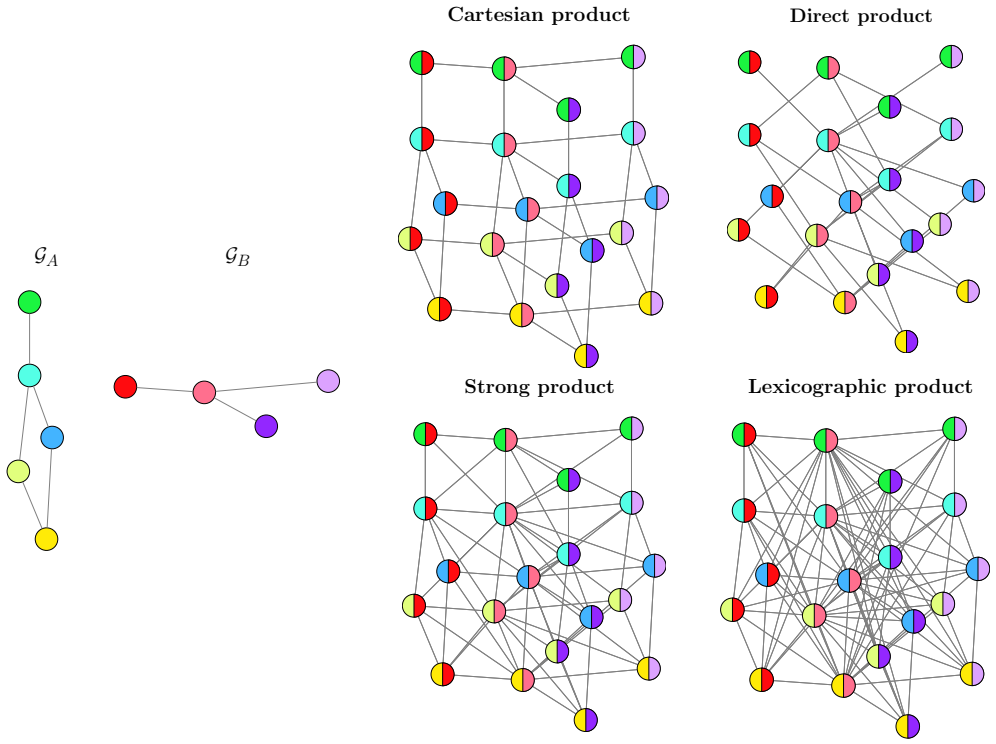


FIGURE 4.1: A graphical depiction of the four standard graph products

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

4.1.2 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 4.2: 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.

$$\mathbf{L}_A = \mathbf{U}_A\mathbf{\Lambda}_A\mathbf{U}_A^\top, \quad \text{and} \quad \mathbf{L}_B = \mathbf{U}_B\mathbf{\Lambda}_B\mathbf{U}_B^\top \tag{4.3}$$

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 4.2 gives information about the spectral decomposition of the standard graph products.

4.1.3 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 4.2, we can write the Laplacian eigendecomposition in matrix form as follows.

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \quad \text{where} \quad \mathbf{U} = \mathbf{U}_A \otimes \mathbf{U}_B \quad \text{and} \quad \mathbf{\Lambda} = \mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B \tag{4.4}$$

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.

$$\text{GFT}(\mathbf{Y}) = \text{mat}\left(\left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \text{vec}\left(\mathbf{Y}\right)\right) = \mathbf{U}_B^\top \mathbf{Y} \mathbf{U}_A \tag{4.5}$$

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

$$\text{IGFT}(\mathbf{Z}) = \text{mat}\left(\left(\mathbf{U}_A \otimes \mathbf{U}_B\right) \text{vec}\left(\mathbf{Z}\right)\right) = \mathbf{U}_B \mathbf{Z} \mathbf{U}_A^\top \tag{4.6}$$

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 2.1. 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.

$$\begin{aligned}\mathbf{H} &= g(\mathbf{L}_A \oplus \mathbf{L}_B) \\ &= \left(\mathbf{U}_A \otimes \mathbf{U}_B\right) g\left(\mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B\right) \left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \\ &= \left(\mathbf{U}_A \otimes \mathbf{U}_B\right) \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right) \left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right) \\ &= \mathbf{U}\mathbf{D}_{\mathbf{G}}\mathbf{U}^\top \end{aligned} \tag{4.7}$$

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

$$\mathbf{G}_{ba} = g\left(\lambda_a^{(A)} + \lambda_b^{(B)}; \beta\right) \tag{4.8}$$

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.

$$\mathbf{G}_{ba} = g\left(\lambda_a^{(A)}, \lambda_b^{(B)};\, \beta_a, \beta_b\right) \tag{4.9}$$

where now g is chosen to be an anisotropic graph filter such as one of those listed in table 4.3. 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 4.2 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 4.3: Anisotropic graph filter functions in two dimensions

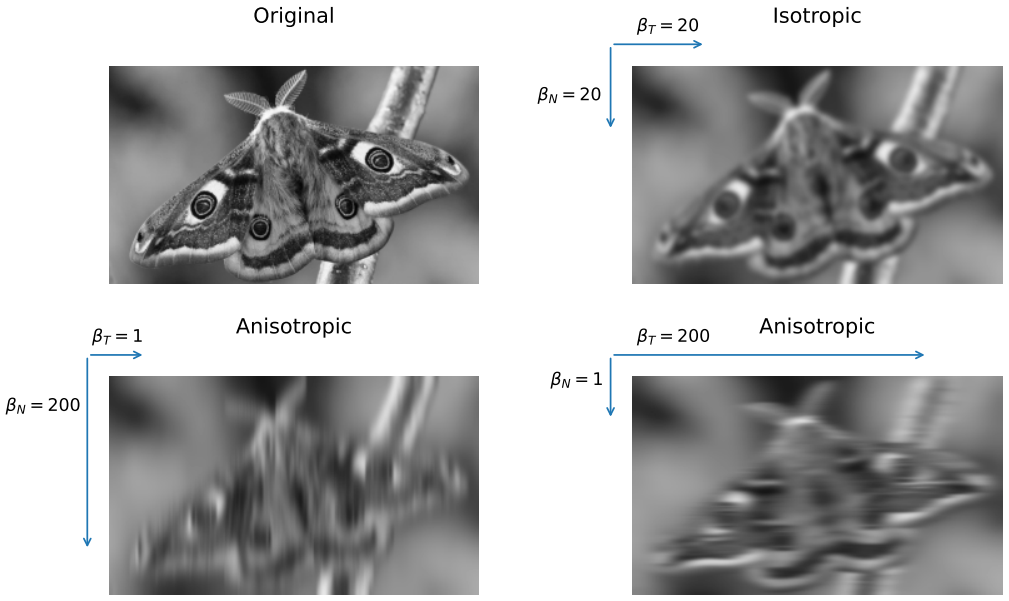


FIGURE 4.2: A graphical depiction of a time-vertex Cartesian product graph

4.2 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 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.

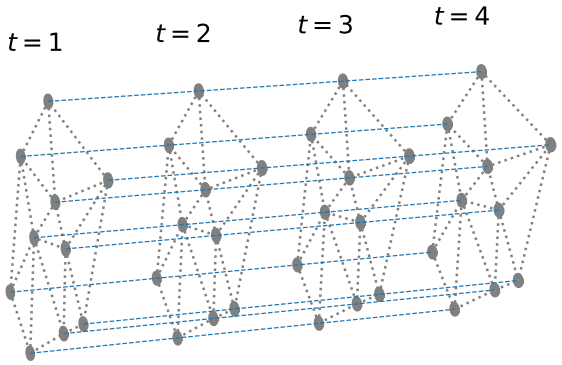


FIGURE 4.3: A graphical depiction of a time-vertex Cartesian product graph

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

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

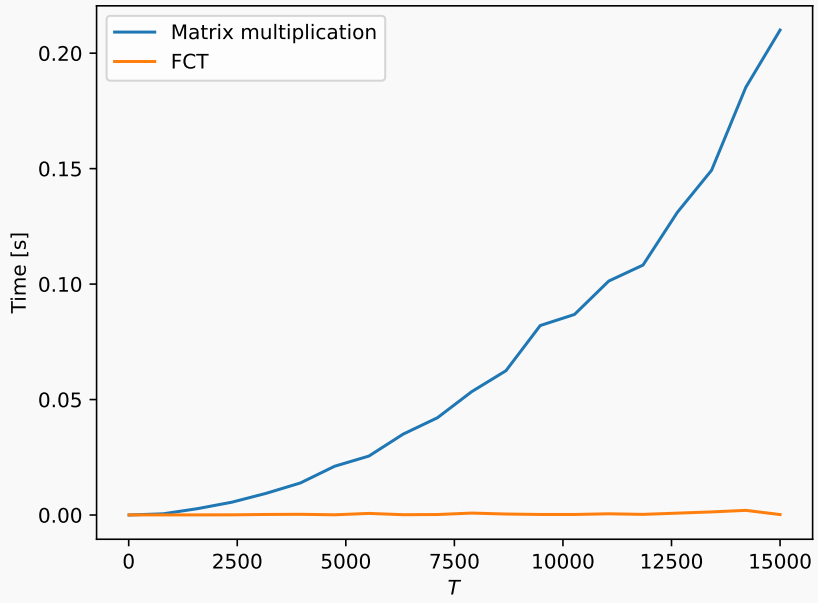
where the columns of $\mathbf{U}$ are appropriately normalised such that the magnitude of each eigenvector is one. Furthermore, this implies that that the graph Fourier transform of a signal $\mathbf{y} \in \mathbb{R}^T$ is given by the orthogonal type-II Discrete Cosine Transform (DCT) [Ahmed et al., 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 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.

4.2.1 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. 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 itself assumed to be smooth with respect to the graph topology.

We define the statistical model for the generation of an observation matrix $\mathbf{Y}$ as follows.

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{F} + \mathbf{E}\big) \tag{4.10}$$

where $\mathbf{S} \in [0, 1]^{N \times T}$ is referred to as the sensing matrix, and has entries given by

$$\mathbf{S}_{nt} = \begin{cases} 1 & \text{if } (n, t) \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

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

$$\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F} \sim \mathcal{N}\Big(\operatorname{vec}(\mathbf{S} \circ \mathbf{F}),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S}))\Big) \tag{4.12}$$

Note that the covariance matrix $\operatorname{diag}(\operatorname{vec}(\mathbf{S}))$ 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 (4.7). For now, in the absence of any further information, we assume that the prior mean for $\mathbf{F}$ is zero across all elements.

$$\operatorname{vec}(\mathbf{F}) \sim \mathcal{N}\big(\mathbf{0},\ \gamma^{-1}\mathbf{H}^2\big) \tag{4.13}$$

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

$$\pi\big(\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{\pi\big(\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F}\big)\pi(\mathbf{F})}{\pi(\mathbf{Y})}. \tag{4.14}$$

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

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

$$\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\left(\mathbf{P}^{-1} \operatorname{vec}(\mathbf{Y}),\ \mathbf{P}^{-1}\right) \tag{4.15}$$

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

$$\mathbf{P} = \operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma \mathbf{H}^{-2} \tag{4.16}$$

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.

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

We return to the question of sampling from the posterior and estimating the posterior covariance directly in chapter 7.

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. (4.17) 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.

4.2.2 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

$$\text{vec}\,(\mathbf{F}) = (\mathbf{M} - \mathbf{N})^{-1}\text{vec}\,(\mathbf{Y}) \tag{4.18}$$

By noting that

$$\mathbf{M}\text{vec}\,(\mathbf{F}) = \mathbf{N}\text{vec}\,(\mathbf{F}) + \text{vec}\,(\mathbf{Y}) \tag{4.19}$$

$$\text{vec}\,(\mathbf{F}) = \mathbf{M}^{-1}\mathbf{N}\text{vec}\,(\mathbf{F}) + \mathbf{M}^{-1}\text{vec}\,(\mathbf{Y}) \tag{4.20}$$

we devise an iterative scheme given by

$$\text{vec}\,(\mathbf{F}_{k+1}) = \mathbf{M}^{-1}\mathbf{N}\text{vec}\,(\mathbf{F}_k) + \mathbf{M}^{-1}\text{vec}\,(\mathbf{Y}) \tag{4.21}$$

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. (4.21) 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

$$\mathbf{M} = \gamma \mathbf{H}^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}'\right)\right). \tag{4.22}$$

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

$$\mathbf{S}'_{nt} = \begin{cases} 1 & \text{if } (n,t) \notin \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4.23}$$

In this way, the update equation is given by

$$\operatorname{vec}\left(\mathbf{F}_{k+1}\right) = \left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}'\right)\right) \operatorname{vec}\left(\mathbf{F}_k\right) + \left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{vec}\left(\mathbf{Y}\right) \tag{4.24}$$

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. (4.7), this can be written as

$$\begin{aligned} \mathbf{M}^{-1} &= \left(\gamma \mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \\ &= \left(\gamma \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right)^{-2} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) + \mathbf{I}\right)^{-1} \\ &= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \left(\gamma \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right)^{-2} + \mathbf{I}\right)^{-1} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \\ &= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{J}\right)\right) \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \end{aligned} \tag{4.25}$$

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

$$\mathbf{J}_{nt} = \frac{\mathbf{G}_{nt}^2}{\mathbf{G}_{nt}^2 + \gamma}. \tag{4.26}$$

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

$$\mathbf{F}_{k+1} = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top + \mathbf{F}_0 \tag{4.27}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{4.28}$$

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 4.2).

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. (4.25), 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. (4.26). 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

$$\rho(\mathbf{M}^{-1}) = \frac{1}{1+\gamma} \tag{4.29}$$

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

$$\rho(\mathbf{AB}) \leq \rho(\mathbf{A})\,\rho(\mathbf{B}) \tag{4.30}$$

[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 4.3.

Finally, the update formulas given in eqs. (4.27) and (4.28) can be written equivalently as

$$\Delta\mathbf{F}_0 = \mathbf{U}_N\left(\mathbf{J} \circ \left(\mathbf{U}_N^\top\,\mathbf{Y}\,\mathbf{U}_T\right)\right)\mathbf{U}_T^\top \tag{4.31}$$

$$\Delta\mathbf{F}_{k+1} = \mathbf{U}_N\left(\mathbf{J} \circ \left(\mathbf{U}_N^\top\left(\mathbf{S}' \circ \Delta\mathbf{F}_k\right)\mathbf{U}_T\right)\right)\mathbf{U}_T^\top \tag{4.32}$$

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 $\mathbf{G}_{nt} = g\left(\lambda_a^{(A)}, \lambda_b^{(B)};\, \beta_a, \beta_b\right)$
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**
  $\Delta\mathbf{F} \leftarrow \mathbf{U}_N\Big(\mathbf{J} \circ \big(\mathbf{U}_N^\top\,(\mathbf{S}' \circ \Delta\mathbf{F})\,\mathbf{U}_T\big)\Big)\mathbf{U}_T^\top$
  $\mathbf{F} \leftarrow \mathbf{F} + \Delta\mathbf{F}$
end while

Output: $\mathbf{F}$

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.

4.2.3 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 (4.7). 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.

$$\mathbf{F} = \mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top \tag{4.33}$$

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 (4.12), and the prior of equation (4.13), 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

$$\operatorname{vec}\left(\mathbf{Z}\right) = \left(\mathbf{D}_\mathbf{G}\left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_\mathbf{S} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_\mathbf{G} + \gamma \mathbf{I}_{NT}\right)^{-1} \operatorname{vec}\left(\mathbf{G} \circ \left(\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T\right)\right) \tag{4.34}$$

where 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. Note that the conditioning of the coefficient matrix is greatly improved from the untransformed problem, as we will discuss in greater detail in section 4.3. Note also that multiplication of a vector $\operatorname{vec}\left(\mathbf{R}\right)$ by the coefficient matrix can be computed efficiently as

$$\begin{aligned}\operatorname{mat}\Big(\Big(\mathbf{D}_\mathbf{G}\left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_\mathbf{S} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_\mathbf{G} + \gamma \mathbf{I}_{NT}\Big) \operatorname{vec}\left(\mathbf{R}\right)\Big) \\ = \gamma \mathbf{R} + \mathbf{G} \circ \Big(\mathbf{U}_N^\top \Big(\mathbf{S} \circ \left(\mathbf{U}_N (\mathbf{G} \circ \mathbf{R}) \mathbf{U}_T^\top\right)\Big) \mathbf{U}_T\Big)\end{aligned} \tag{4.35}$$

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 4.2).

The linear system defined eq. (4.34) can be understood as a two-sided symmetrically preconditioned version of the original linear system given in eq. (4.17). In particular,

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

the new expression can be constructed by modifying the original system in the following way.

$$\left(\boldsymbol{\Psi}^\top \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{H}^{-2}\right) \boldsymbol{\Psi}\right) \left(\boldsymbol{\Psi}^{-1} \operatorname{vec}\left(\mathbf{F}\right)\right) = \boldsymbol{\Psi}^\top \operatorname{vec}\left(\mathbf{Y}\right), \tag{4.36}$$

where

$$\boldsymbol{\Psi} = \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_{\mathbf{G}}. \tag{4.37}$$

Since preconditioning of the coefficient matrix on the left is achieved with $\boldsymbol{\Psi}^\top$ and on the right with $\boldsymbol{\Psi}$, 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.

4.2.4 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" 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 4.4 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

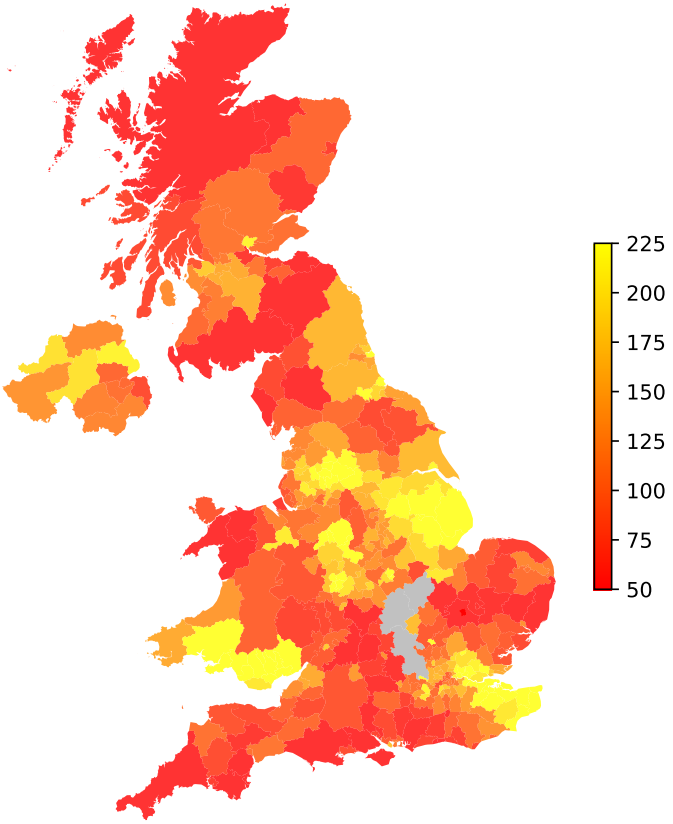


FIGURE 4.4: The seven day rolling rate of new covid cases reported per 100,000 residents in each lower tier local reporting authority in the United Kingdom on the 1st of December 2020. Missing data is indicated in gray.

[1]See https://coronavirus.data.gov.uk/details/download
[2]See https://data.gov.uk/dataset/local-authority-districts-december-2019-boundaries-uk-buc

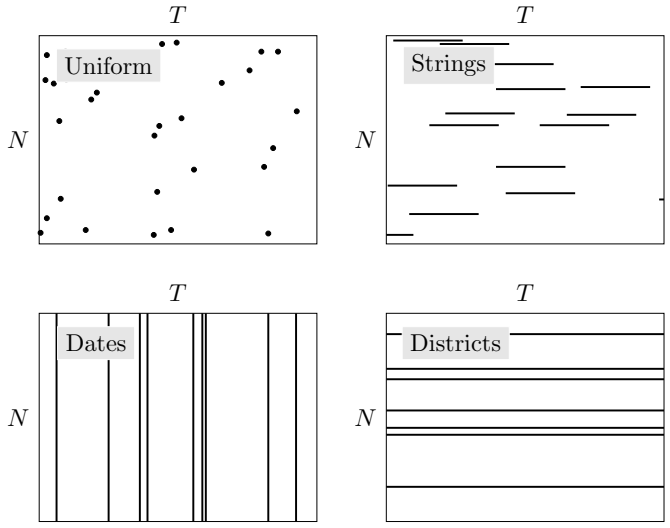


FIGURE 4.5: A visual depiction of the four ways we removed data. Black lines/dots indicate data that was removed.

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).

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. 4.5. 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 measured the Root Mean Square Error (RMSE) across the reconstructed entries, over four increasing values of m. The results are shown in table 4.4.

	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	0.049	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 4.4: 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.

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.

4.3 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 4.2.2, $\mathbf{M}$ 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

$$n_{\text{SIM}} \propto -\frac{1}{\log \rho\left(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}\right)} \tag{4.38}$$

Note that the matrix $\mathbf{J}$ [see eq. (4.26) 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. (4.8) and (4.9)]. 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. (4.34). Therefore, we should expect that the number of iterations required for convergence of the CGM will scale as

$$n_{\text{CGM}} \propto \sqrt{\kappa\left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I}_{NT}\right)} \tag{4.39}$$

where $\mathbf{D}_{\mathbf{G}} = \operatorname{diag}(\operatorname{vec}(\mathbf{G}))$. 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).

4.3.1 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. (4.7)], which appears in the prior distribution for $\mathbf{F}$ [see eq. (4.13)], 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 tables 2.1 and 4.3, 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. (4.26), 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

$$\lim_{\beta \to 0} \rho\Big(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}\Big) = \frac{1}{1+\gamma}\rho\Big(\mathbf{D}_{\mathbf{S}'}\Big) = \frac{1}{1+\gamma} \tag{4.40}$$

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

$$\lim_{\beta \to 0} \kappa\left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I}\right) = \kappa\left(\mathbf{U}^{\top}\left(\mathbf{D}_{\mathbf{S}} + \gamma\mathbf{I}\right)\mathbf{U}\right) = \frac{1+\gamma}{\gamma} \tag{4.41}$$

Given eqs. (4.38) and (4.39), 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

$$\lim_{\beta\to 0} n_{\text{SIM}} \propto \frac{1}{\log(1+\gamma)}, \quad \text{and} \quad \lim_{\beta\to 0} n_{\text{CGM}} \propto \sqrt{\frac{1}{\gamma}+1} \tag{4.42}$$

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 $\gamma = 0$.

$$\lim_{\beta\to 0} n_{\text{SIM}} \propto \gamma^{-1} + O(\gamma), \quad \text{and} \quad \lim_{\beta\to 0} n_{\text{CGM}} \propto \gamma^{-1/2} + O\left(\gamma^{1/2}\right) \tag{4.43}$$

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

4.3.2 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 tables 2.1 and 4.3, we can associate this with the limit as $\beta \to \infty$. Here, the effect of applying the graph filter to a generic graph signal $\text{vec}(\mathbf{Y})$ is to extract the mean, that is

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

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

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

Note that

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

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

$$\lim_{\beta\to\infty} \rho(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}) = \frac{1}{1+\gamma}\frac{|\mathcal{S}'|}{NT} = \frac{m}{1+\gamma} \tag{4.44}$$

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

$$\begin{aligned}
\lim_{\beta\to\infty} \quad \kappa\left(\mathbf{D_G U}^\top \mathbf{D_S U D_G} + \gamma\mathbf{I}\right) &= \kappa\left(\mathbf{\Delta U}^\top \mathbf{D_S U \Delta} + \gamma\mathbf{I}\right) \\
&= \kappa\left(\begin{bmatrix} \mathbf{u}_1^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{bmatrix} \mathbf{D_S}\Big[\mathbf{u}_1, \mathbf{0}, \ldots, \mathbf{0}\Big] + \gamma\mathbf{I}\right) \\
&= \kappa\left(\frac{1}{NT}\begin{bmatrix} |\mathcal{S}| & 0 & 0 & \ldots \\ 0 & 0 & 0 & \\ \vdots & & & \ddots \end{bmatrix} + \gamma\mathbf{I}\right) \\
&= \frac{1-m+\gamma}{\gamma} \qquad (4.45)
\end{aligned}$$

Given eqs. (4.38) and (4.39), we can write the scaling rate for the number of iterations in the SIM and CGM respectively.

$$\lim_{\beta\to\infty} n_{\text{SIM}} \propto \frac{1}{\log(1+\gamma) - \log m}, \quad \text{and} \quad \lim_{\beta\to\infty} n_{\text{CGM}} \propto \sqrt{\frac{1-m+\gamma}{\gamma}} \qquad (4.46)$$

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.

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

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 $1 - 1/NT$.

	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 4.5: 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 $\gamma \to 0$.

4.3.3 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 4.5. 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. (4.13), 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. (4.13). 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.

4.3.4 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 table 2.1) 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 4.3.

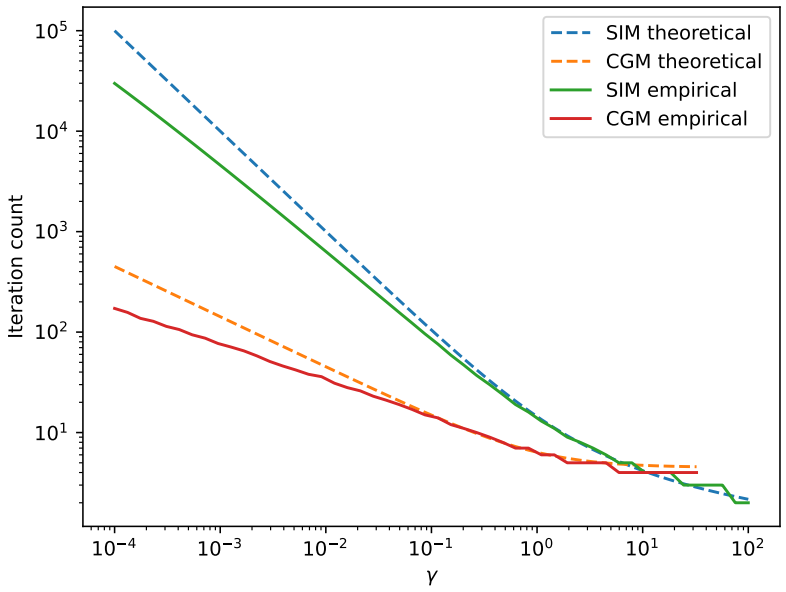


FIGURE 4.6: The number of iterations required for convergence in the weak filter limit for the SIM an CGM both theoretically and empirically, as a function of γ.

4.3.4.1 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 4.5. 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. 4.6. 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 4.5. 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

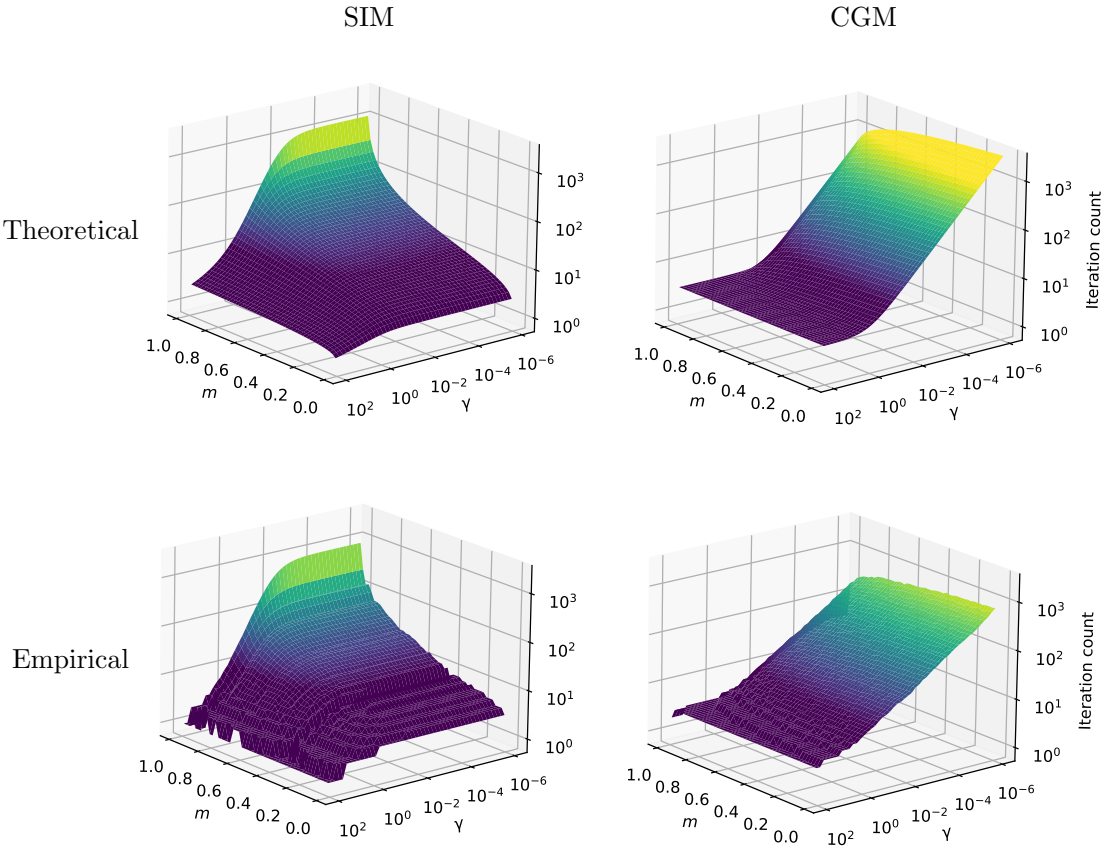


FIGURE 4.7: The number of iterations required for convergence is shown both theoretically and empirically for the SIM and CGM in the strong filter limit, where $\beta = \infty$. In this case, each plot is a function of both m and γ. The colormap corresponds to vertical height and is nomalised across each plot to aid comparison.

for convergence and compared this to the theoretical predictions. The results are shown in fig. 4.7. Again, the empirical results broadly follow the theoretical functions which, 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.

4.3.4.2 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 4.3 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. 4.8.

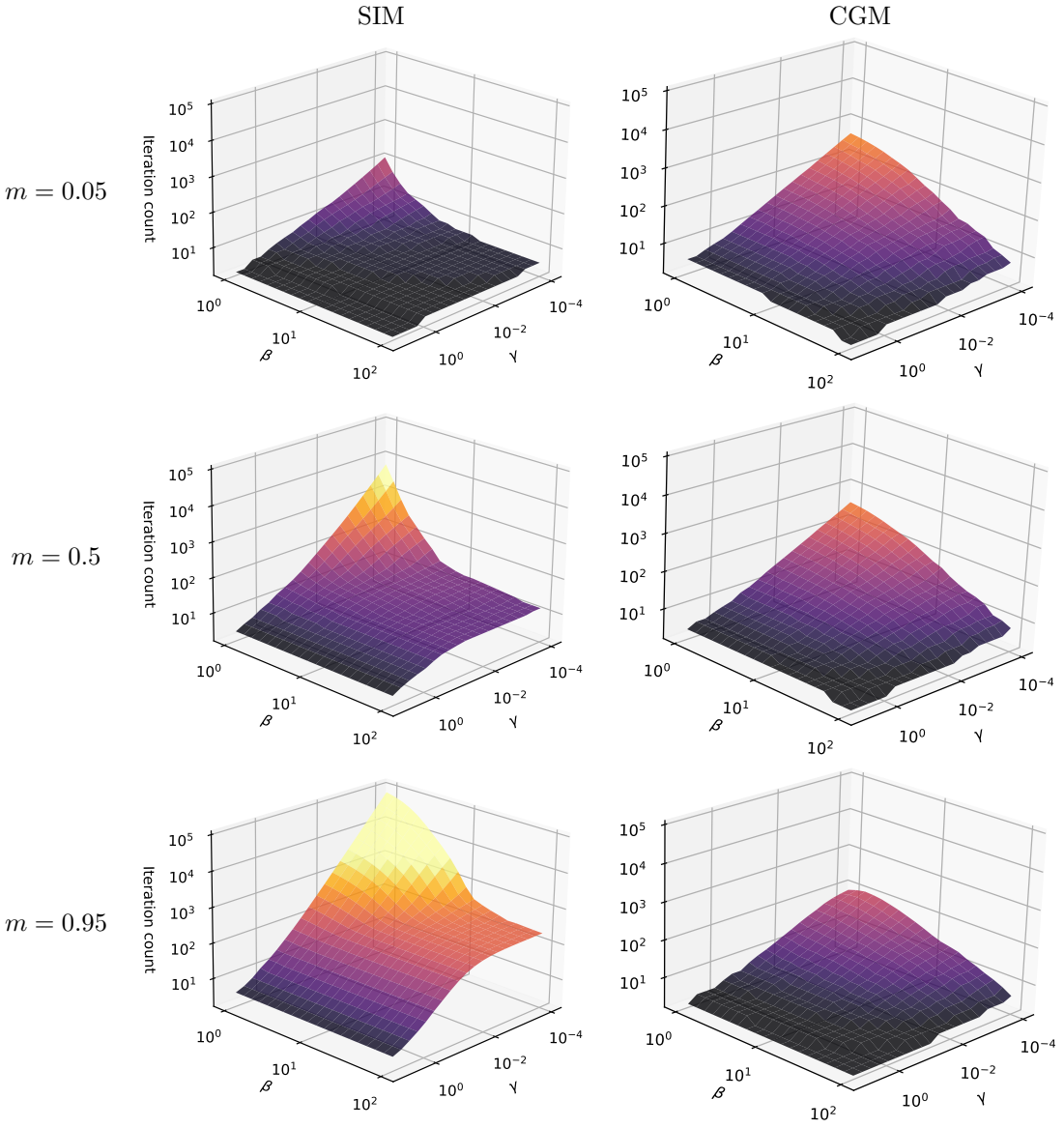


FIGURE 4.8: Six 3D surfaces are shown representing the number of iterations required for convergence for three values of m for both the SIM and CGM algorithms. In each plot, the x-axis represents the filter parameter β, and the y-axis represents the regularisation parameter γ. The colormap, which is normalsied equally across all plots, tracks the vertical height.

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.

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.

4.4 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. (4.17). 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 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 (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 4.5, which gives a factor proportional to the number of iterations required for convergence in terms of the hyper-parameters 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 4.6. 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.

	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 4.6: 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.

Chapter 5

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. 5.1.

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 5.2 offers a visual depiction of the data encountered in this particular scenario.

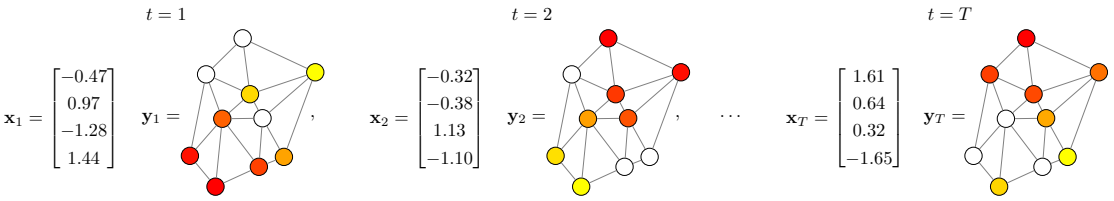


FIGURE 5.1: A visual representation of a time-series graph signal regression problem with exogenous variables. Here, there are T graph signals measured over a static graph, each with independent missing data (indicated in white). Each signal $\mathbf{y}_t$ is accompanied by a unique vector of global explanatory variables $\mathbf{x}_t$.

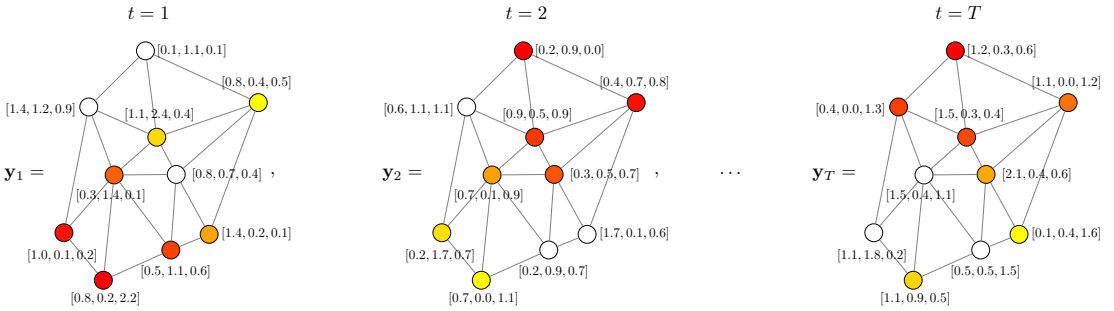


FIGURE 5.2: A visual representation of a time-series graph signal regression problem with local explanatory variables. As before, there are T graph signals measured over a static graph, each with independent missing data (indicated in white). In this case, each signal $\mathbf{y}_t$ has a vector of explanatory 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 4, 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.

5.1 Kernel Graph Regression with Unrestricted Missing Data Patterns

5.1.1 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 4.2.1, 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 4.1.3.

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

$$\boldsymbol{\phi}(\mathbf{x}_t) = \begin{bmatrix} \phi_1(\mathbf{x}_t) & \phi_2(\mathbf{x}_t) & \dots & \phi_P(\mathbf{x}_t) \end{bmatrix}^\top \tag{5.1}$$

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.

$$\mathbf{y}_t = \mathbf{s}_t \circ \big(\mathbf{W}\boldsymbol{\phi}(\mathbf{x}_t) + \mathbf{e}_t\big), \quad \text{for } t = 1, 2, ..., T \tag{5.2}$$

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 $\boldsymbol{\Phi}$.

$$\boldsymbol{\Phi} \in \mathbb{R}^{P \times T} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \dots & \phi_1(\mathbf{x}_T) \\ \phi_2(\mathbf{x}_1) & \phi_2(\mathbf{x}_2) & \dots & \phi_2(\mathbf{x}_T) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_P(\mathbf{x}_1) & \phi_P(\mathbf{x}_2) & \dots & \phi_P(\mathbf{x}_T) \end{bmatrix} \tag{5.3}$$

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

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{W}\boldsymbol{\Phi} + \mathbf{E}\big) \tag{5.4}$$

where $\mathbf{S} \in [0, 1]^{N \times T}$ is the binary sensing matrix as defined in section 4.2.1, 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

$$\text{vec}\,(\mathbf{Y}) \mid \mathbf{W} \sim \mathcal{N}\big(\text{vec}\,(\mathbf{S} \circ (\mathbf{W}\boldsymbol{\Phi}))\,,\ \mathbf{I}_{NT}\big) \tag{5.5}$$

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

$$\text{vec}\,(\mathbf{W}) \sim \mathcal{N}\left(\mathbf{0},\ \gamma^{-1}\mathbf{I}_P \otimes \mathbf{H}_N^2\right) \tag{5.6}$$

Here, $\mathbf{H}_N \in \mathbb{R}^N$ represents a graph filter constructed in accordance with one of the univariate filter functions defined in table 2.1, 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. (5.6). 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. (5.6), 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.

$$\mathrm{vec}\,(\mathbf{Y})\;|\,\mathbf{F} \sim \mathcal{N}\big(\,\mathrm{vec}\,(\mathbf{S} \circ \mathbf{F})\,,\,\mathbf{I}_{NT}\,\big) \tag{5.7}$$

$$\mathrm{vec}\,(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\;\gamma^{-1}\big(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\big) \otimes \mathbf{H}_N^2\Big) \tag{5.8}$$

The final step to convert this into a non-parametric regression model is to apply the 'kernel-trick'. 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 $\mathbf{x}_j$.

$$\boldsymbol{\Phi}^\top\boldsymbol{\Phi} \in \mathbb{R}^{T\times T} = \begin{bmatrix}
\boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\
\boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\
\vdots & \vdots & \ddots & \vdots \\
\boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_T)
\end{bmatrix} \tag{5.9}$$

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

$$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{5.10}$$

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

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{5.11}$$

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

$$\text{vec}\,(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\ \gamma^{-1}\mathbf{K} \otimes \mathbf{H}_N^2\Big) \tag{5.12}$$

In a similar manner to section 4.2.1, the posterior distribution for $\mathbf{F}|\mathbf{Y}$ is given by

$$\text{vec}\,(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\left(\bar{\mathbf{P}}^{-1}\text{vec}\,(\mathbf{Y}),\ \bar{\mathbf{P}}^{-1}\right) \tag{5.13}$$

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

$$\bar{\mathbf{P}} = \mathbf{D}_\mathbf{S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} \tag{5.14}$$

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

5.1.2 Relation to graph signal reconstruction

Despite arising from a different set of modelling assumptions, Kernel Graph Regression as described in section 5.1.1 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. (4.13)], and in KGR it has covariance $\mathbf{K} \otimes \mathbf{H}_N^2$ [see eq. (5.12)]. 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

$$\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top, \quad \text{and} \quad \mathbf{H}_N = \mathbf{U}_N\,g\,(\boldsymbol{\Lambda}_N)\,\mathbf{U}_N^\top \tag{5.15}$$

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

$$\bar{\mathbf{G}}_{nt} = g\left(\lambda_n^{(N)}\right)\sqrt{\lambda_t^{(K)}} \quad \rightarrow \quad \mathbf{D}_{\bar{\mathbf{G}}} = \boldsymbol{\Lambda}_K^{1/2} \otimes g\,(\boldsymbol{\Lambda}_N) \tag{5.16}$$

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

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

As such, the iterative algorithms developed in chapter 4 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.

5.1.3 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

$$\operatorname{vec}(\mathbf{F}) = \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right)^{-1} \operatorname{vec}(\mathbf{Y}) \tag{5.17}$$

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

$$\mathbf{M} = \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\mathbf{S}'}. \tag{5.18}$$

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

$$\begin{aligned}
\mathbf{M}^{-1} &= \left(\gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\gamma \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{G}}}^{-2}\, \bar{\mathbf{U}}^{\top} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right) \bar{\mathbf{U}}^{\top}\right)^{-1} \\
&= \bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right)^{-1} \bar{\mathbf{U}}^{\top} \\
&= \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{J}}}\, \bar{\mathbf{U}}^{\top}
\end{aligned} \tag{5.19}$$

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

$$\bar{\mathbf{J}}_{nt} = \frac{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right)}{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right) + \gamma} = \frac{\bar{\mathbf{G}}_{nt}^2}{\bar{\mathbf{G}}_{nt}^2 + \gamma} \tag{5.20}$$

The SIM algorithm then proceeds in much the same way as described in section 4.2.2. 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

$$\mathbf{F}_{k+1} = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k\right) \mathbf{V}\right)\right) \mathbf{V}^\top + \mathbf{F}_0 \tag{5.21}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{V}\right)\right) \mathbf{V}^\top \tag{5.22}$$

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. (5.17) is to utilise a symmetric a preconditioner $\mathbf{\Psi}$ such that the new system is given by

$$\left(\bar{\mathbf{\Psi}}^\top \left(\mathbf{D_S} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right) \bar{\mathbf{\Psi}}\right)\left(\bar{\mathbf{\Psi}}^{-1} \operatorname{vec}\left(\mathbf{F}\right)\right) = \bar{\mathbf{\Psi}}^\top \operatorname{vec}\left(\mathbf{Y}\right), \tag{5.23}$$

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

$$\bar{\mathbf{\Psi}} = \left(\mathbf{V} \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}\left(\bar{\mathbf{G}}\right)\right) = \bar{\mathbf{U}} \mathbf{D}_{\bar{\mathbf{G}}}. \tag{5.24}$$

This preconditioner transforms the coefficient matrix into

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

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.

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

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

5.2 Regression with Network Cohesion

5.2.1 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. 5.2.

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

Here, we have introduced the notation $\boldsymbol{\mathcal{X}}$, which is an order-3 tensor, to

with 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 5.3 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 $(NT \times M)$.

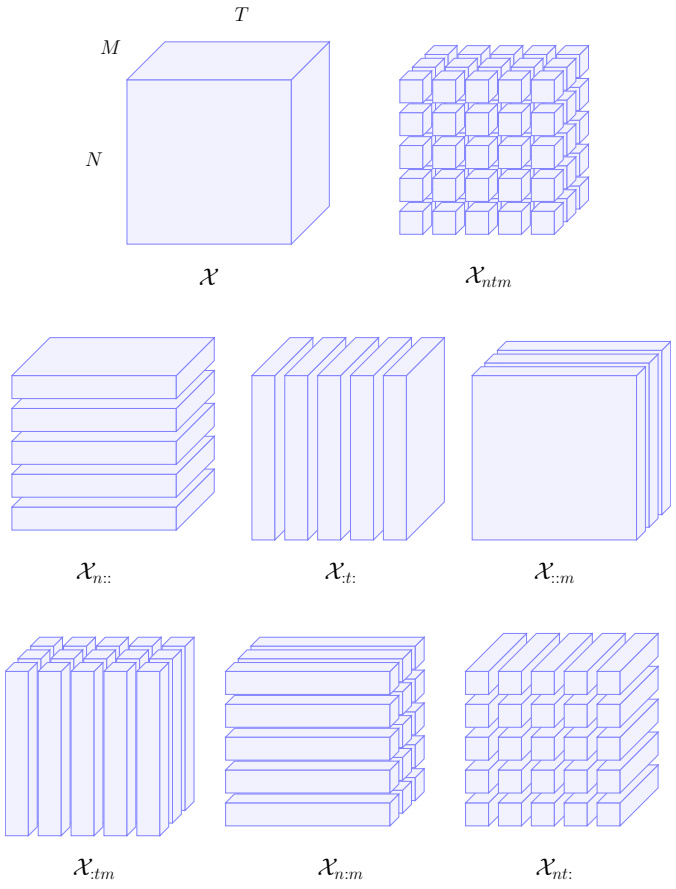


FIGURE 5.3: A visual representation of a several different ways of indexing into an order-3 tensor, with the corresponding notation indicated.

$$\mathbf{X} \in \mathbb{R}^{NT \times M} = \Big[\text{vec}\left(\boldsymbol{\mathcal{X}}_{::1}\right) \quad \text{vec}\left(\boldsymbol{\mathcal{X}}_{::2}\right) \quad \dots \quad \text{vec}\left(\boldsymbol{\mathcal{X}}_{::M}\right) \Big] \tag{5.25}$$

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.

$$\text{vec}\left(\mathbf{Y}\right) = \text{vec}\left(\mathbf{S}\right) \circ \big(\text{vec}\left(\mathbf{C}\right) + \mathbf{X}\mathbf{w} + \text{vec}\left(\mathbf{E}\right) \big) \tag{5.26}$$

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. (5.25), as follows.

$$\operatorname{vec}(\mathbf{Y}) = \operatorname{vec}(\mathbf{S}) \circ \Big(\begin{bmatrix} \mathbf{I}_{NT} & \mathbf{X} \end{bmatrix} \boldsymbol{\theta} + \operatorname{vec}(\mathbf{E}) \Big) \tag{5.27}$$

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,

$$\boldsymbol{\theta} \in \mathbb{R}^{NT+M} = \begin{bmatrix} \operatorname{vec}(\mathbf{C}) \\ \mathbf{w} \end{bmatrix} \tag{5.28}$$

Given eq. (5.27), we can write the probability distribution for $\mathbf{Y} \,|\, \boldsymbol{\theta}$ as follows.

$$\operatorname{vec}(\mathbf{Y}) \,|\, \boldsymbol{\theta} \sim \mathcal{N}\Big(\operatorname{vec}(\mathbf{S}) \circ \Big(\begin{bmatrix} \mathbf{I}_{NT} & \mathbf{X} \end{bmatrix} \boldsymbol{\theta} \Big),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S})) \Big) \tag{5.29}$$

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.

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0},\ \begin{bmatrix} \gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M \end{bmatrix} \right) \tag{5.30}$$

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

$$\boldsymbol{\theta} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1} \begin{bmatrix} \operatorname{vec}(\mathbf{Y}) \\ \mathbf{X}^\top \operatorname{vec}(\mathbf{Y}) \end{bmatrix},\ \widetilde{\mathbf{P}}^{-1} \right) \tag{5.31}$$

where

$$\widetilde{\mathbf{P}} \in \mathbb{R}^{(NT+M) \times (NT+M)} = \begin{bmatrix} \mathbf{D}_\mathbf{S} + \gamma \mathbf{H}^{-2} & \mathbf{D}_\mathbf{S}\mathbf{X} \\ \mathbf{X}^\top \mathbf{D}_\mathbf{S} & \mathbf{X}^\top \mathbf{D}_\mathbf{S} \mathbf{X} + \lambda \mathbf{I}_M \end{bmatrix} \tag{5.32}$$

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

5.2.2 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

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

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.

$$\left(\widetilde{\mathbf{\Psi}}^\top \begin{bmatrix} \mathbf{D_S} + \gamma \mathbf{H}^{-2} & \mathbf{D_S X} \\ \mathbf{X}^\top \mathbf{D_S} & \mathbf{X}^\top \mathbf{D_S X} + \lambda \mathbf{I}_M \end{bmatrix} \widetilde{\mathbf{\Psi}} \right) \left(\widetilde{\mathbf{\Psi}}^{-1} \boldsymbol{\theta} \right) = \widetilde{\mathbf{\Psi}}^\top \begin{bmatrix} \operatorname{vec}(\mathbf{Y}) \\ \mathbf{X}^\top \operatorname{vec}(\mathbf{Y}) \end{bmatrix} \tag{5.33}$$

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

$$\mathbf{X}^\top \mathbf{D_S X} = \mathbf{U}_M \mathbf{\Lambda}_M \mathbf{U}_M^\top \tag{5.34}$$

Since $\mathbf{X}^\top \mathbf{D_S X}$ 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

$$\mathbf{D}_M = \left(\mathbf{\Lambda}_M + \lambda \mathbf{I}_M \right)^{-1/2} \tag{5.35}$$

We propose that an effective preconditioner is given by

$$\widetilde{\mathbf{\Psi}} = \begin{bmatrix} \mathbf{U D_G} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_M \mathbf{D}_M \end{bmatrix} \tag{5.36}$$

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 $O(N^2T + NT \log T)$.

5.3 Network regression with both global and local explanatory variables

In section 5.1, 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. 5.1. On the other hand, Regression with Network Cohesion (RNC), discussed in section 5.2, is suitable for situations where 'local' explanatory variables are associated with individual nodes over time, as depicted in fig. 5.2.

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. (5.25).

Leveraging the methods developed for KGR and RNC in sections 5.1 and 5.2, 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 5.1.2, 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. (5.10).

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.

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0},\ \begin{bmatrix} \gamma^{-1}\mathbf{K}\otimes\mathbf{H}_N^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M \end{bmatrix}\right) \tag{5.37}$$

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

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

Given the prior for $\boldsymbol{\theta}$ in eq. (5.37), and the likelihood which is identical to that of the RNC model given in eq. (5.29), the posterior distribution is given by

$$\boldsymbol{\theta} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\hat{\mathbf{P}}^{-1}\begin{bmatrix}\text{vec}\left(\mathbf{Y}\right)\\ \mathbf{X}^\top\text{vec}\left(\mathbf{Y}\right)\end{bmatrix}, \hat{\mathbf{P}}^{-1}\right) \tag{5.38}$$

where

$$\hat{\mathbf{P}} \in \mathbb{R}^{(NT+M)\times(NT+M)} = \begin{bmatrix}\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} & \mathbf{D_S}\mathbf{X}\\ \mathbf{X}^\top\mathbf{D_S} & \mathbf{X}^\top\mathbf{D_S}\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix} \tag{5.39}$$

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

$$\left(\hat{\boldsymbol{\Psi}}^\top\begin{bmatrix}\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} & \mathbf{D_S}\mathbf{X}\\ \mathbf{X}^\top\mathbf{D_S} & \mathbf{X}^\top\mathbf{D_S}\mathbf{X} + \lambda\mathbf{I}_M\end{bmatrix}\hat{\boldsymbol{\Psi}}\right)\left(\hat{\boldsymbol{\Psi}}^{-1}\boldsymbol{\theta}\right) = \hat{\boldsymbol{\Psi}}^\top\begin{bmatrix}\text{vec}\left(\mathbf{Y}\right)\\ \mathbf{X}^\top\text{vec}\left(\mathbf{Y}\right)\end{bmatrix} \tag{5.40}$$

where

$$\hat{\boldsymbol{\Psi}} = \begin{bmatrix}\bar{\mathbf{U}}\mathbf{D}_{\bar{\mathbf{G}}} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}_M\mathbf{D}_M\end{bmatrix} \tag{5.41}$$

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.

Chapter 6

Regression and Reconstruction with Tensor-Valued Muliway 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. 6.1.

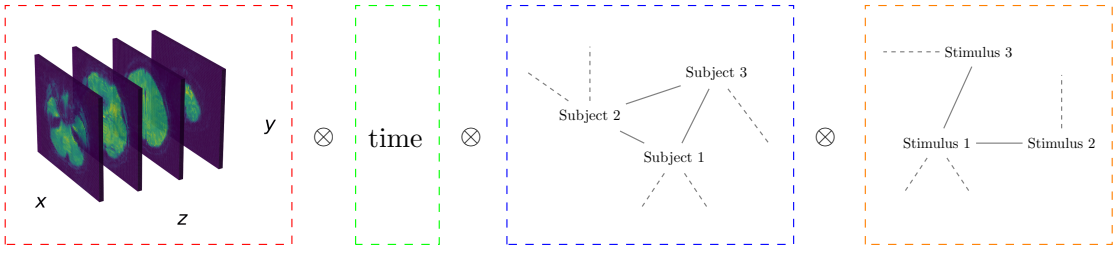


FIGURE 6.1: Graphical depiction of a six-way graph signal originating from a hypothetical fMRI experiment.

The goal of this chapter is to extend the methods developed in chapters 4 and 5 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 6.1 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 6.2, we begin the core contributions of this chapter by generalising Bayesian GSR as defined in chapter 4 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 6.2.1 and 6.2.2. Next, in section 6.3, we generalise KGR for the non-parametric prediction of multiway graph signals as a function of exogenous variables. Finally, in section 6.4, we the generalise RNC as defined in section 5.2 in much the same way.

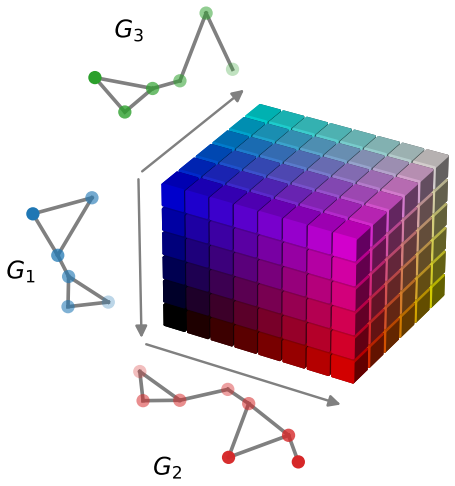


FIGURE 6.2: Graphical depiction of an order-3 tensor signal with graphs underlying each axis, which are linked together to form a Cartesian product graph.

6.1 Multiway Graph Signal Processing

6.1.1 The Cartesian product of more than two graphs

In section 4.1.1 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

$$\mathcal{G} \;=\; \mathcal{G}_A \,\square\, \mathcal{G}_B \,\square\, \mathcal{G}_C \;=\; \{\mathcal{V},\, \mathcal{E}\} \tag{6.1}$$

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

$$\mathcal{V} = \mathcal{V}_A \times \mathcal{V}_B \times \mathcal{V}_C = \{(a,\, b,\, c) \in \mathbb{N}^3 \,|\, a \leq A,\; b \leq B, \text{and } c \leq C\} \tag{6.2}$$

The new edge set, $\mathcal{E}$, is given by recursively applying conditions 1 and 7 from, section 4.1.1 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 6.3 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.

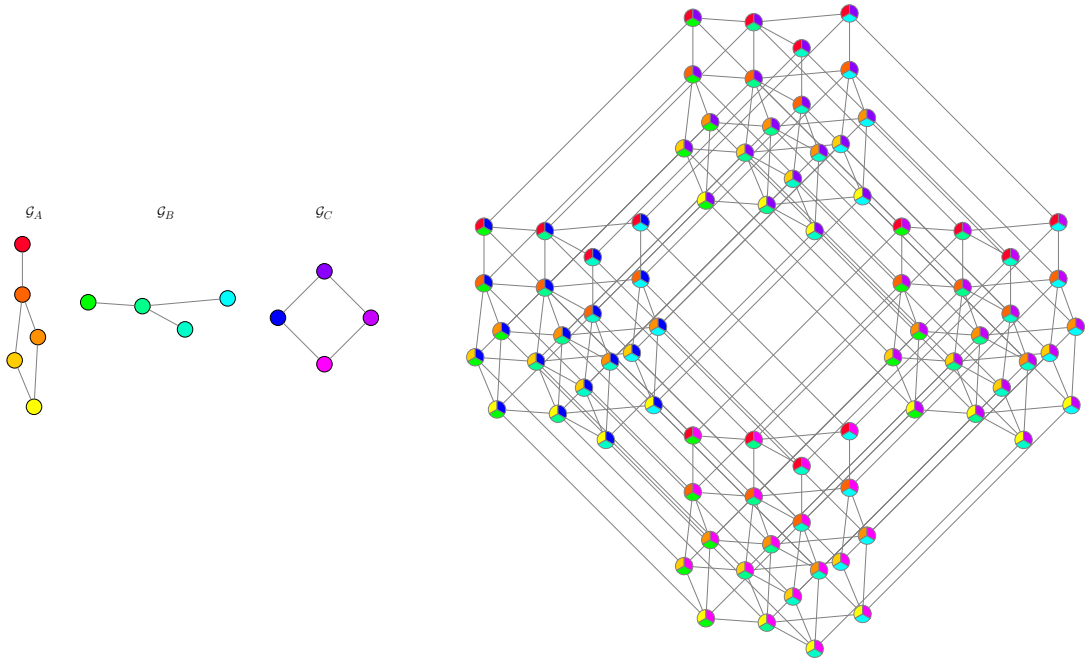


FIGURE 6.3: Graphical depiction of a 3D Cartesian product graph

$$\begin{aligned}\mathbf{A} &= \mathbf{A}_A \oplus \mathbf{A}_B \oplus \mathbf{A}_C \\ &= \mathbf{A}_A \otimes \mathbf{I}_B \otimes \mathbf{I}_C + \mathbf{I}_A \otimes \mathbf{A}_B \otimes \mathbf{I}_C + \mathbf{I}_A \otimes \mathbf{I}_B \otimes \mathbf{A}_C \end{aligned} \tag{6.3}$$

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

$$\begin{aligned}\mathbf{A} = \mathbf{A}^{(1)} &\oplus \mathbf{A}^{(2)} \oplus \ \dots \ \oplus \mathbf{A}^{(d)} \\ = \mathbf{A}^{(1)} &\otimes \mathbf{I}_{N_2} \otimes \ \dots \ \otimes \mathbf{I}_{N_d} + \\ \mathbf{I}_{N_1} &\otimes \mathbf{A}^{(2)} \otimes \ \dots \ \otimes \mathbf{I}_{N_d} + \ \dots \ + \\ \mathbf{I}_{N_1} &\otimes \mathbf{I}_{N_2} \otimes \ \dots \ \otimes \mathbf{A}^{(d)} \end{aligned} \tag{6.4}$$

This can be written compactly as

$$\mathbf{A} = \bigoplus_{i=1}^{d} \mathbf{A}^{(i)} \tag{6.5}$$

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

$$\mathbf{L} = \bigoplus_{i=1}^{d} \mathbf{L}^{(i)} \tag{6.6}$$

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

$$\mathbf{L}^{(i)} = \mathbf{U}^{(i)} \mathbf{\Lambda}^{(i)} (\mathbf{U}^{(i)})^{\top} \tag{6.7}$$

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.

$$\begin{aligned} \mathbf{L} &= \bigoplus_{i=1}^{d} \mathbf{U}^{(i)} \mathbf{\Lambda}^{(i)} (\mathbf{U}^{(i)})^{\top} \\ &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right) \left(\bigoplus_{i=1}^{d} \mathbf{\Lambda}^{(i)} \right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right)^{\top} \\ &= \mathbf{U} \mathbf{\Lambda} \mathbf{U}^{\top} \end{aligned} \tag{6.8}$$

where

$$\mathbf{U} = \bigotimes_{i=1}^{d} \mathbf{U}^{(i)}, \quad \text{and} \quad \mathbf{\Lambda} = \bigoplus_{i=1}^{d} \mathbf{\Lambda}^{(i)} \tag{6.9}$$

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 $\{\mathbf{U}^{(i)}\}$.

6.1.2 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. (6.9) 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,

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

Figure 6.4 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.

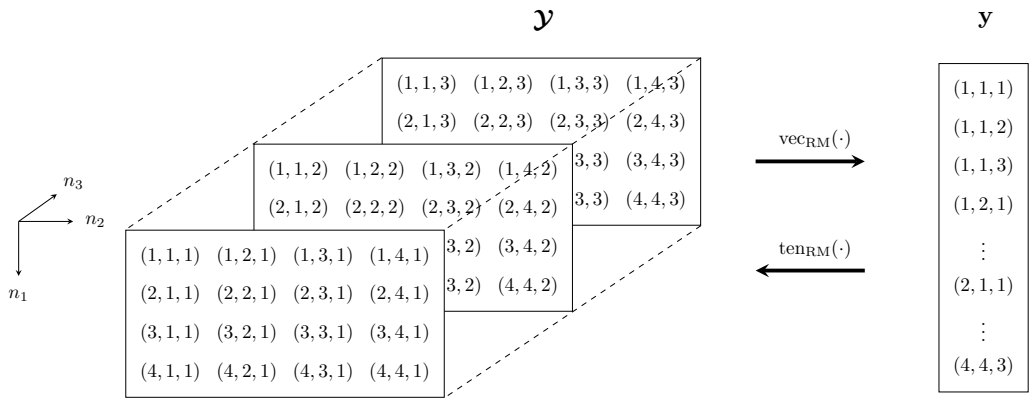


FIGURE 6.4: A graphical depiction of the process of converting an order-3 tensor between its multidimensional array and vector form in row-major order. Note that the elements in the vectorised signal are lexicographically ordered.

$$k = 1 + \sum_{i=1}^{d} \Big(\prod_{j=i+1}^{d} N_j \Big) (n_i - 1) \tag{6.10}$$

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

6.1.3 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. (4.5) and (4.6), we can define the Graph Fourier Transform (GFT) and its corresponding inverse (IGFT) of this signal as follows.

$$\text{GFT}(\boldsymbol{\mathcal{Y}}) = \text{ten}_{\text{RM}}\left(\mathbf{U}^\top \mathbf{y}\right) = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right)^\top \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)\right) \tag{6.11}$$

$$\text{IGFT}(\boldsymbol{\mathcal{Y}}) = \text{ten}_{\text{RM}}\left(\mathbf{U}\mathbf{y}\right) \quad = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right)\right) \tag{6.12}$$

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 table 2.1. A filter $\mathbf{H}$, defined to act on a vectorised graph signal, can be represented as an $N \times N$ matrix, constructed as follows.

$$\begin{aligned}\mathbf{H} &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) g\left(\bigoplus_{i=1}^{d} \boldsymbol{\Lambda}^{(i)}; \beta\right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right)^\top \\ &= \mathbf{U}\,\text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right)\,\mathbf{U}^\top \end{aligned} \tag{6.13}$$

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

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\left(\sum_{i=1}^{d} \lambda_{n_i}^{(i)};\, \beta\right) \tag{6.14}$$

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

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = \prod_{i=1}^{d} g\left(\lambda_{n_i}^{(i)};\, \beta\right) \tag{6.15}$$

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 6.1: Anisotropic graph filter functions in an arbitrary number of dimensions

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

$$\mathbf{H} = \bigotimes_{i=1}^{d} \mathbf{H}^{(i)}, \quad \text{where} \quad \mathbf{H}^{(i)} = \mathbf{U}^{(i)} g\big(\boldsymbol{\Lambda}^{(i)}\big) \left(\mathbf{U}^{(i)}\right)^\top \tag{6.16}$$

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 6.1 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

$$\boldsymbol{\mathcal{G}}_{\mathbf{n}} = g\big(\boldsymbol{\lambda}(\mathbf{n});\, \boldsymbol{\beta}\big) \tag{6.17}$$

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.

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

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

$$\boldsymbol{\mathcal{Y}}' = \mathrm{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \mathrm{GFT}(\boldsymbol{\mathcal{Y}})\right) \tag{6.18}$$

6.1.4 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. (6.11) and (6.12). 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

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

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. 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 $\mathcal{Z}$. In our notation, this is written as

$$\mathcal{Y} = \text{ten}_{\text{RM}}\left(\left(\bigotimes_{i=1}^{d}\mathbf{U}^{(i)}\right)\text{vec}_{\text{RM}}\left(\mathcal{Z}\right)\right) \tag{6.19}$$

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.

$$\mathcal{Y}^{i_1,i_2,\ldots,i_d} = \left(\mathbf{U}^{(1)}\right)^{i_1}_{j_1}\left(\mathbf{U}^{(2)}\right)^{i_2}_{j_2}\ldots\left(\mathbf{U}^{(d)}\right)^{i_d}_{j_d}\mathcal{Z}^{j_1,j_2,\ldots,j_d} \tag{6.20}$$

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 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 6.1.4, the full $N \times N$ matrix implied by eq. (6.19) is never actually instantiated in memory (see algorithm 4).

6.2 Tensor Graph Signal reconstruction

The model we use to describe graph signal reconstruction in the multi-dimensional setting is as follows. Consider a tensor signal $\mathcal{Y}$ of shape $\left(N_1, N_2, \ldots N_d\right)$ with elements interpreted as existing on the nodes of a d-dimensional Cartesian product graph. Only a partial set $\mathcal{S} = \{\mathbf{n}_1, \mathbf{n}_2, \ldots\}$ of the vector elements of $\mathcal{Y}$ are available at observation

time, with unobserved values set to zero. The goal is to estimate the signal value at these unobserved entries.

The binary sensing tensor $\boldsymbol{\mathcal{S}}$, of the same shape as $\boldsymbol{\mathcal{Y}}$, indicates which elements of $\boldsymbol{\mathcal{Y}}$ were observed in the following way

$$\boldsymbol{\mathcal{S}}_{\mathbf{n}} = \begin{cases} 1 & \text{if } \mathbf{n} \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{6.21}$$

In analogy with the two dimensional case, (see section 4.2.1), 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 in the following statistical model.

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{S}} \circ \left(\boldsymbol{\mathcal{F}} + \boldsymbol{\mathcal{E}}\right) \tag{6.22}$$

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,

$$\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{E}}\right) \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{6.23}$$

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

$$\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{Y}}\right) \,|\, \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{F}}\right) \sim \mathcal{N}\Big(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}} \circ \boldsymbol{\mathcal{F}}\right),\ \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)\right)\Big) \tag{6.24}$$

Or, more concisely,

$$\mathbf{y} \,|\, \mathbf{f} \sim \mathcal{N}\big(\mathbf{s} \circ \mathbf{f},\ \mathbf{D}_{\boldsymbol{\mathcal{S}}}\big) \tag{6.25}$$

where $\mathbf{D}_{\boldsymbol{\mathcal{S}}} = \text{diag}\left(\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{S}}\right)\right)$. 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.

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0},\ \gamma^{-1}\mathbf{H}^2\right) \tag{6.26}$$

where $\mathbf{H}$ is constructed from a graph filter function in the manner specified in eq. (6.13). 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 $\mathbf{y}$.

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right) \tag{6.27}$$

where

$$\mathbf{P} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2} \tag{6.28}$$

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

$$\mathbf{f}^{\star} = \left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2}\right)^{-1} \mathbf{y} \tag{6.29}$$

Once again, we are faced with a similar set of problems to those outlined in section 4.2.1. 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.

6.2.1 Tensor SIM

Generalising the SIM, which we describe in section 4.2.2, to the tensor setting is straightforward. In particular, eq. (6.29) 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

$$\mathbf{M} = \gamma \mathbf{H}^{-2} + \mathbf{I}, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\boldsymbol{\mathcal{S}}'} \tag{6.30}$$

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

$$\begin{aligned} \mathbf{M}^{-1} &= \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right) \operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}\left(\boldsymbol{\mathcal{J}}\right)\right) \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)}\right)^{\top} \\ &= \mathbf{U}\, \mathbf{D}_{\boldsymbol{\mathcal{J}}}\, \mathbf{U}^{\top} \end{aligned} \tag{6.31}$$

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

$$\boldsymbol{\mathcal{J}}_{\mathbf{n}} = \frac{\boldsymbol{\mathcal{G}}_{\mathbf{n}}^2}{\boldsymbol{\mathcal{G}}_{\mathbf{n}}^2 + \gamma}. \tag{6.32}$$

Here, the entries of $\boldsymbol{\mathcal{G}}$ are given by either eq. (6.14) or eq. (6.17), which correspond to an isotropic or anisotropic filter function respectively. In a very similar manner to eq. (4.21), the SIM update equation is given by

$$\mathbf{f}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{f}_k + \mathbf{M}^{-1}\mathbf{y} \tag{6.33}$$

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 6.1.4. To be explicit, this update formula can be computed efficiently as

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

or, equivalently,

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

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 4.3). As before, the true scaling rate will depend on the strength of the graph filter.

Algorithm 5 The tensor SIM for GSR

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

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 $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

Initialise $\boldsymbol{\mathcal{G}}, \boldsymbol{\mathcal{J}}, \boldsymbol{\mathcal{S}}'$

for n_1 **from** 1 **to** N_1 **do**
  for n_2 **from** 1 **to** N_2 **do**
    $\ddots$
      for n_d **from** 1 **to** N_d **do**

$\mathbf{n} \leftarrow [n_1, n_2, \ldots, n_d]$

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

$\boldsymbol{\mathcal{J}}_{\mathbf{n}} \leftarrow \boldsymbol{\mathcal{G}}_{\mathbf{n}}^{2} \,/\, (\boldsymbol{\mathcal{G}}_{\mathbf{n}}^{2} + \gamma)$

$\boldsymbol{\mathcal{S}}'_{\mathbf{n}} \leftarrow 1 - \boldsymbol{\mathcal{S}}_{\mathbf{n}}$

      end for
  end for
$\iddots$
end for

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

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

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

6.2.2 Tensor CGM

The tensor version of the CGM also follows naturally from the two-dimensional case outlined in section 4.2.3. In particular, eq. (6.29) can be transformed into the following equivalent preconditioned linear system.

$$\mathbf{f}^{\star} = \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^{\top}\left(\mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma\mathbf{H}^{-2}\right)\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\top}\mathbf{y} \tag{6.34}$$

where, in the tensor case,

$$\mathbf{\Phi} = \left(\bigotimes_{i=1}^{d} \mathbf{U}^{(i)} \right) \operatorname{diag}\left(\operatorname{vec}_{\mathrm{RM}}\left(\boldsymbol{\mathcal{G}}\right)\right) = \mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} \tag{6.35}$$

This means eq. (6.34) can be expressed as

$$\mathbf{f}^{\star} = \mathbf{\Phi}\left(\mathbf{D}_{\boldsymbol{\mathcal{G}}}\mathbf{U}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{S}}}\mathbf{U}\mathbf{D}_{\boldsymbol{\mathcal{G}}} + \gamma\mathbf{I}_{N}\right)^{-1}\mathbf{\Phi}^{\top}\mathbf{y} \tag{6.36}$$

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,

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

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

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

6.3 Kernel Graph Tensor Regression

Hello

6.4 Tensor Regression with Network Cohesion

Ahhhh

6.5 Application

6.6 Conclusions

Algorithm 6 The tensor CGM for GSR

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

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 $\mathbf{U}^{(i)}\boldsymbol{\Lambda}^{(i)}\left(\mathbf{U}^{(i)}\right)^{\top}$

Initialise $\boldsymbol{\mathcal{G}}'$

for n_1 **from** 1 **to** N_1 **do**
  for n_2 **from** 1 **to** N_2 **do**
    $\ddots$
      for n_d **from** 1 **to** N_d **do**
        $\mathbf{n} \leftarrow [n_1, n_2, ..., n_d]$
        $\boldsymbol{\mathcal{G}}_{\mathbf{n}} \leftarrow g\big(\boldsymbol{\lambda}(\mathbf{n}); \boldsymbol{\beta}\big)$
      end for
  end for
$\iddots$
end for

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 $|\Delta\mathbf{R}| >$ 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: $\mathrm{IGFT}\left(\boldsymbol{\mathcal{G}} \circ \boldsymbol{\mathcal{Z}}\right)$

Chapter 7

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 4.2. 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 6, where the covariance matrices have $O(N^2)$ elements, where $N = \prod N_i$. Table 7.1 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

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 7.1, 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, our method significantly outperforms the aforementioned stochastic technique on a test dataset. Next, in section 7.2, we make use of a recently proposed technique for direct sampling known as Perturbation Optimisation (PO) [Vono et al., 2022]. PO is relevant

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 7.1: The posterior covariance matrix appearing in the tensor GSR, KGR, RNC and KR-RNC models.

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 6, 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 6. However, since the two-dimensional methods covered in chapters 4 and 5 can be considered special cases of the more general MWGSP format, these too are included under the same framework.

7.1 Estimating the posterior marginal variance

Consider the multiway Graph Signal Reconstruction problem as defined in chapter 6. 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 6.2, 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

$$\mathbf{f} \,|\, \mathbf{y} \sim \mathcal{N}\left(\mathbf{P}^{-1}\mathbf{y}, \mathbf{P}^{-1}\right) \tag{7.1}$$

where

$$\mathbf{P} \in \mathbb{R}^{N \times N} = \mathbf{D}_{\boldsymbol{\mathcal{S}}} + \gamma \mathbf{H}^{-2} \tag{7.2}$$

The goal of this section is to estimate the diagonal of the covariance matrix $\mathbf{P}^{-1}$ without directly evaluating the matrix in full. For the sake of brevity, we focus on estimating the posterior marginal variance for the GSR model alone in this section. However, the methods discussed can easily be generalised to KGR, RNC and KG-RNC by making suitable modifications.

7.1.1 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 has lower time and memory 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 not scalable. A natural question, then, is whether the full marginal variance can be *estimated* for a lower computational cost.

7.1.2 Matrix diagonal estimation

One notable recent 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.

$$\text{diag}\left(\mathbf{P}^{-1}\right) \approx \left(\sum_{r=1}^{R} \mathbf{v}_r \circ \mathbf{P}^{-1}\mathbf{v}_r\right) \oslash \left(\sum_{k=1}^{s} \mathbf{v}_r \circ \mathbf{v}_r\right) \tag{7.3}$$

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 7.1.1, we must reach an acceptable level of accuracy for $R \ll N$.

7.1.3 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 7.1.1, 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.

$$\boldsymbol{\Omega} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d} = \text{ten}_{\text{RM}} \left(\log \left(\text{diag}^{-1} \left(\mathbf{P}^{-1} \right) \right) \right) \tag{7.4}$$

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.

$$\boldsymbol{\mathcal{Q}}_{\mathbf{n}} = \begin{cases} 1 & \text{if } \mathbf{n} \in \mathcal{Q} \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

For each index $\mathbf{n}$, the corresponding element of $\boldsymbol{\Omega}$ can be computed as

$$\boldsymbol{\Omega}_{\mathbf{n}} = \text{ten}_{\text{RM}} \left(\log \left(\mathbf{P}^{-1} \mathbf{e}_n \right) \right)_{\mathbf{n}} \tag{7.6}$$

For a vector index $\mathbf{n} = [n_1, n_2, ..., n_d]$, the corresponding unit vector $\mathbf{e}_n$ is

$$\mathbf{e}_n = \mathbf{e}_{n_1} \otimes \mathbf{e}_{n_2} \otimes ... \otimes \mathbf{e}_{n_d} \tag{7.7}$$

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 6.2.1 and 6.2.2. We refer to the process of computing

	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 7.2: 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

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 $\boldsymbol{\Omega}_Q$.

$$\boldsymbol{\Omega}_Q = \boldsymbol{\mathcal{Q}} \circ \boldsymbol{\Omega} \tag{7.8}$$

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 the tensor $\boldsymbol{\mathcal{X}}$ such that it is likely to explain the posterior marginal variance well.

By inspecting eq. (7.2), 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 7.2. 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 6.4, we can combine these tensor variables together into a single matrix $\mathbf{X}$ as follows.

$$\mathbf{X} \in \mathbb{R}^{N\times 7} = \left[\text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,1}\right) \quad \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,2}\right) \quad \dots \quad \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{X}}_{:,7}\right)\right] \tag{7.9}$$

7.1.3.1 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 7.2, 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).

7.1.3.2 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 6.4. 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

$$\boldsymbol{\Omega}^\star = \boldsymbol{\mathcal{C}}^\star + \text{ten}_{\text{RM}}\left(\mathbf{X}\mathbf{w}^\star\right) \tag{7.10}$$

where

$$\begin{bmatrix} \text{vec}_{\text{RM}}\left(\boldsymbol{\mathcal{C}}^{\star}\right) \\ \mathbf{w}^{\star} \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{\boldsymbol{\mathcal{Q}}} + \gamma \mathbf{H}^{-2} & \mathbf{D}_{\boldsymbol{\mathcal{Q}}}\mathbf{X} \\ \mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{Q}}} & \mathbf{X}^{\top}\mathbf{D}_{\boldsymbol{\mathcal{Q}}}\mathbf{X} + \lambda \mathbf{I}_{7} \end{bmatrix}^{-1} \begin{bmatrix} \text{vec}_{\text{RM}}\left(\boldsymbol{\Omega}_{Q}\right) \\ \mathbf{X}^{\top}\text{vec}_{\text{RM}}\left(\boldsymbol{\Omega}_{Q}\right) \end{bmatrix} \tag{7.11}$$

Once again, this linear system can be solved efficiently using the SIM or CGM as discussed in section 6.4.

7.1.3.3 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 7.2, 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.

$$\boldsymbol{\Omega}^{\star} = \text{ten}_{\text{RM}}\left(\mathbf{X}(\boldsymbol{\beta}^{\star})\mathbf{w}^{\star}\right) \tag{7.12}$$

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.

$$\xi(\mathbf{w}, \boldsymbol{\beta}) = ||\boldsymbol{\Omega}_{Q} - \boldsymbol{\mathcal{Q}} \circ \text{ten}_{\text{RM}}\left(\mathbf{X}(\boldsymbol{\beta})\mathbf{w}\right)||_{F}^{2} + \lambda\left(||\mathbf{w}||^{2} + ||\boldsymbol{\beta} - \boldsymbol{\beta}_{0}||^{2}\right) \tag{7.13}$$

where $\boldsymbol{\beta}_{0}$ represents the parameters characterising the original graph filter in the base GSR problem. Minimising the expression in eq. (7.13) 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].

7.1.4 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.

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 $\mathcal{Q}$

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.

7.1.5 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 7.1.1 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 7.1.2, Ridge Regression (RR), as outlined in section 7.1.3.1, Regression with Network Cohesion (RNC) as outlined in section 7.1.3.2, and finally ridge regression with Learned Filter Parameters (LFP), as outlined in section 7.1.3.3. 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.

$$\text{Square error} = ||\exp(\boldsymbol{\Omega}) - \exp(\boldsymbol{\Omega}^\star)||_F^2 \tag{7.14}$$

$$R^2 = 1 - \frac{||\exp(\boldsymbol{\Omega}) - \exp(\boldsymbol{\Omega}^\star)||_F^2}{||\exp(\boldsymbol{\Omega}) - \overline{\exp(\boldsymbol{\Omega})}||_F^2} \tag{7.15}$$

where $\overline{\exp(\boldsymbol{\Omega})}$ 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. (7.3)]. 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. 7.1. 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 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.

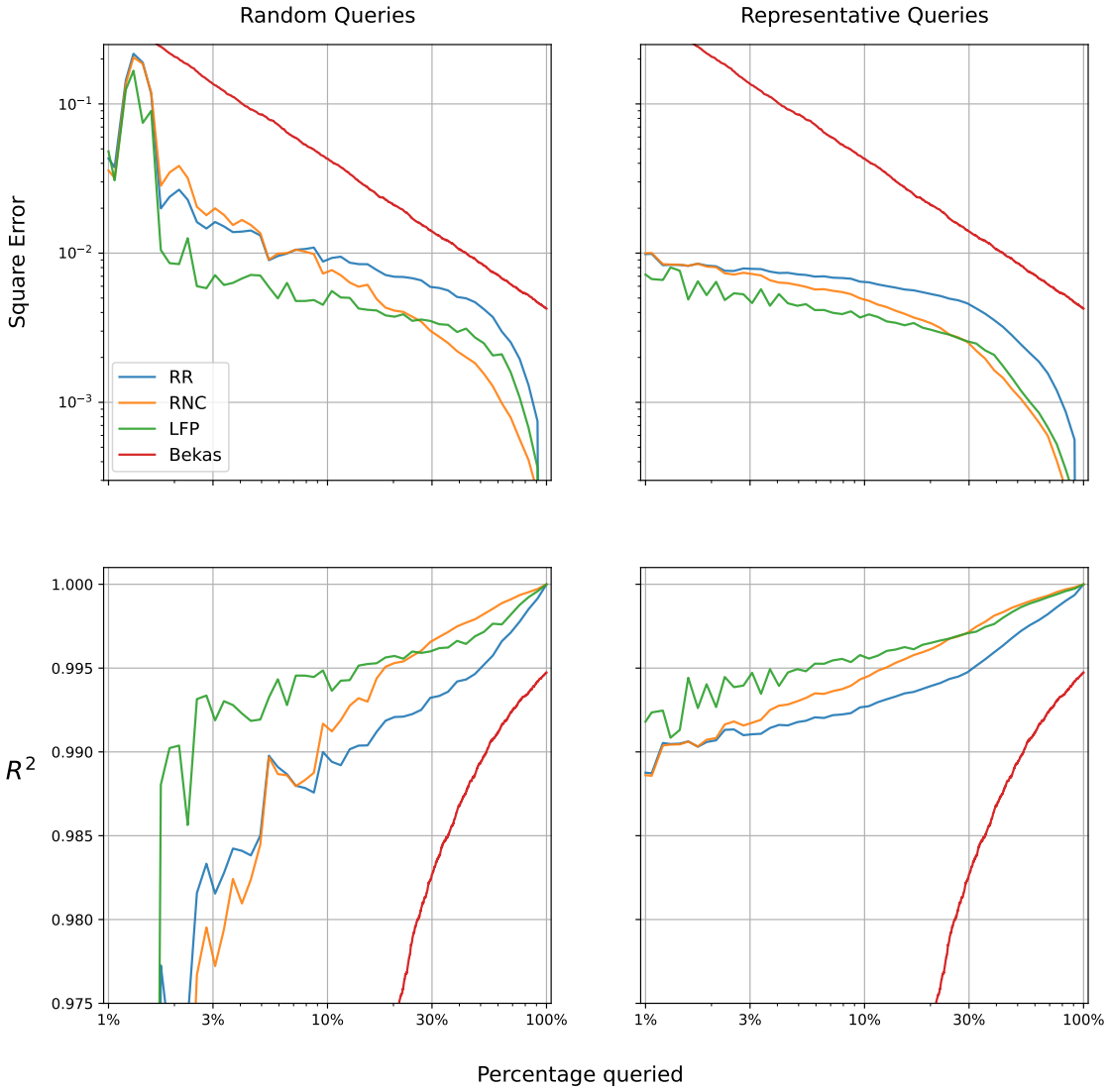


FIGURE 7.1: The performance of the various algorithms for predicting the posterior log-variance is shown as a function of the percentage of $\boldsymbol{\Omega}$ that has been queried. The top row shows the total square error of the posterior variance, and the bottom row shows the R^2 statistic. The 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 not depend on this algorithm, it is repeated in the left and right columns for reference.

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 this particular graph

signal processing application, the supervised learning strategy is clearly more effective than the more general technique of Bekas et al. [2007] on this dataset.

7.2 Posterior Sampling

In this section we move to the second aim of this chapter which is to produce an algorithm for direct sampling from the posterior.

This technique has been dubbed Perturbation Optimisation (PO) since it draws perturbed versions of the mean vectors before using them to define the linear system to solve [Orieux et al., 2012, Papandreou and Yuille, 2010].

7.2.1 Perturbation optimization

7.3 Estimation vs Sampling

7.3.1 Experiments

Chapter 8

Working with Binary-Valued Graph Signals

8.1 Logistic Graph Signal Reconstruction

Li et al. [2012]

8.2 Logistic Kernel Graph Regression

8.3 Logistic Regression with Network Cohesion

8.4 Approximate Sampling via the Laplace Approximation

Chapter 9

Conclusions

9.1 Main Section 1

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 4.12 for the probability distribution of $\mathbf{Y}|\mathbf{F}$.

$$\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F} \sim \lim_{\epsilon \to 0} \left[\mathcal{N}\left(\operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F}),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S}_\epsilon))\right) \right] \tag{A.4}$$

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

$$\boldsymbol{\Sigma} = \left(\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S})) + \gamma \mathbf{H}^{-2}\right)^{-1} \tag{A.10}$$

and that the posterior mean is given by $\boldsymbol{\Sigma}\,\mathrm{vec}\,(\mathbf{Y})$.

□

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.

$$\operatorname{vec}\left(\mathbf{Y}\right) \mid \mathbf{Z} \sim \mathcal{N}\Big(\operatorname{vec}\Big(\mathbf{S} \circ \left(\mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top\right)\Big),\ \mathbf{D}_{\mathbf{S}}\Big)$$

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

$$\frac{\partial\, n_{\mathrm{SIM}}(\gamma, m;\ \beta \to \infty)}{\partial\, m} = \frac{1}{m\big(\log(1+\gamma) - \log(m)\big)^2}$$

$$\frac{\partial\, n_{\mathrm{CGM}}(\gamma, m;\ \beta \to \infty)}{\partial\, m} = -\frac{1}{2\gamma\sqrt{\frac{1-m+\gamma}{\gamma}}}$$

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*

$$\boldsymbol{\theta} \,|\, \mathbf{Y} \sim \mathcal{N}\left(\widetilde{\mathbf{P}}^{-1}\begin{bmatrix}\mathrm{vec}\,(\mathbf{Y}) \\ \mathbf{X}^\top \mathrm{vec}\,(\mathbf{Y})\end{bmatrix},\ \widetilde{\mathbf{P}}^{-1}\right) \tag{A.16}$$

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. (5.29). This can also be written as

$$\mathrm{vec}\,(\mathbf{Y})\ |\,\boldsymbol{\theta} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta},\ \mathbf{D_S}\right) \tag{A.18}$$

The prior for $\boldsymbol{\theta}$ is given in equation eq. (5.30). Therefore, using Bayes rule, we can write

$$-2\log\pi(\boldsymbol{\theta}|\mathbf{Y}) \propto \Big(\mathrm{vec}\,(\mathbf{Y}) - \begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta}\Big)^\top \mathbf{D_S}^{-1}\Big(\mathrm{vec}\,(\mathbf{Y}) - \begin{bmatrix}\mathbf{D_S} & \mathbf{D_S X}\end{bmatrix}\boldsymbol{\theta}\Big)$$
$$+ \boldsymbol{\theta}^\top \begin{bmatrix}\gamma^{-1}\mathbf{H}^2 & \mathbf{0} \\ \mathbf{0} & \lambda^{-1}\mathbf{I}_M\end{bmatrix}^{-1}\boldsymbol{\theta}$$

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 □

Bibliography

Abraham, R., Marsden, J. E., and Rațiu, T. S. (1988). *Manifolds, tensor analysis, and applications*. Springer-Verlag, New York, 2nd ed edition.

Ahmed, N., Natarajan, T., and Rao, K. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93.

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.

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.

Bekas, C., Kokiopoulou, E., and Saad, Y. (2007). An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57(11):1214–1229.

Bhatia, R. (1997). *Matrix analysis*. Number 169 in Graduate texts in mathematics. Springer, New York.

Brenner, S. C., Scott, L. R., and Scott, L. R. (2008). *The mathematical theory of finite element methods*, volume 3. Springer.

Cammoun, L., Castaño-Moraga, C., Muñoz-Moreno, E., Sosa-Cabrera, D., Acar, B., Rodriguez-Florido, M., Brun, A., Knutsson, H., and Thiran, J. P. (2009). *A Review of Tensors and Tensor Signal Processing*, pages 1–32. Springer London, London.

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.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301.

DeBoor, C. (1979). Efficient computer manipulation of tensor products. *ACM Transactions on Mathematmal Software*, 5:173–182.

Demmel, J. W. (1997). *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia.

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.

Fackler, P. L. (2019). Algorithm 993: Efficient computation with kronecker products. *ACM Trans. Math. Softw.*, 45(2).

Fiedler, M. (1973). Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305.

Fletcher, R. (2013). *Practical Methods of Optimization*. Wiley.

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.

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.

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.

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.

Isufi, E., Loukas, A., Simonetto, A., and Leus, G. (2017). Autoregressive moving average graph filtering. *IEEE Transactions on Signal Processing*, 65(2):274–288.

Ji, F. and Tay, W. P. (2019). A hilbert space theory of generalized graph signal processing. *IEEE Transactions on Signal Processing*, 67(24):6188–6203.

Jiang, J. (2012). Introduction to spectral graph theory.

Kaveh, A. and Alinejad, B. (2011). Laplacian matrices of product graphs: applications in structural mechanics. *Acta Mechanica*, 222:331–350.

Kelley, C. T. (1995). *Iterative Methods for Linear and Nonlinear Equations*. 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.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Kroonenberg, P. M. (2008). *Applied Multiway Data Analysis*. Wiley Series in Probability and Statistics. Wiley-Blackwell, Hoboken, NJ.

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.

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, 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.

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.

Newman, M. (2018). *Networks*. Oxford University Press.

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., 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.

Pereyra, V. and Scherer, G. (1973). Efficient computer manipulation of tensor products with applications to multidimensional approximation. *Mathematics of Computation*, 27:595–605.

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.

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.

Roth, W. E. (1934). On direct product matrices. *Bulletin of the American Mathematical Society*.

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.

Sayama, H. (2016). Estimation of laplacian spectra of direct and strong product graphs. *Discrete Applied Mathematics*, 205:160–170.

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.

Smilde, A., Bro, R., and Geladi, P. (2004). *Multi-way analysis*. John Wiley & Sons, Nashville, TN.

Smith, W. and Smith, J. (1995). *Handbook of Real-Time Fast Fourier Transforms: Algorithms to Product Testing*. Wiley.

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.

Takeda, H., Farsiu, S., and Milanfar, P. (2007). Kernel regression for image processing and reconstruction. *IEEE Transactions on Image Processing*, 16(2):349–366.

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.

Tolimieri, R., An, M., and Lu, C. (2013). *Algorithms for discrete Fourier transform and convolution*. Signal Processing and Digital Filtering. Springer, New York, NY, 1989 edition.

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.

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.

Zhang, S., Deng, Q., and Ding, Z. (2022). Signal Processing over Multilayer Graphs: Theoretical Foundations and Practical Applications.

Zhang, S., Zhang, H., Li, H., and Cui, S. (2018). Tensor-based spectral analysis of cascading failures over multilayer complex systems. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE.

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.