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

March 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]

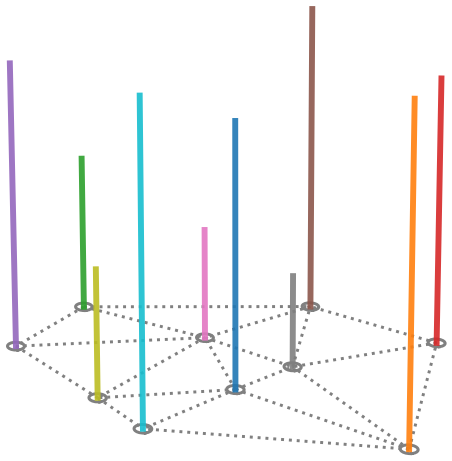


FIGURE 1.1: A graphical depiction of a graph signal. Here, the nodes are represented by circles, the edges as dotted lines, and the value of the signal at each node is represented by the height of its associated bar.

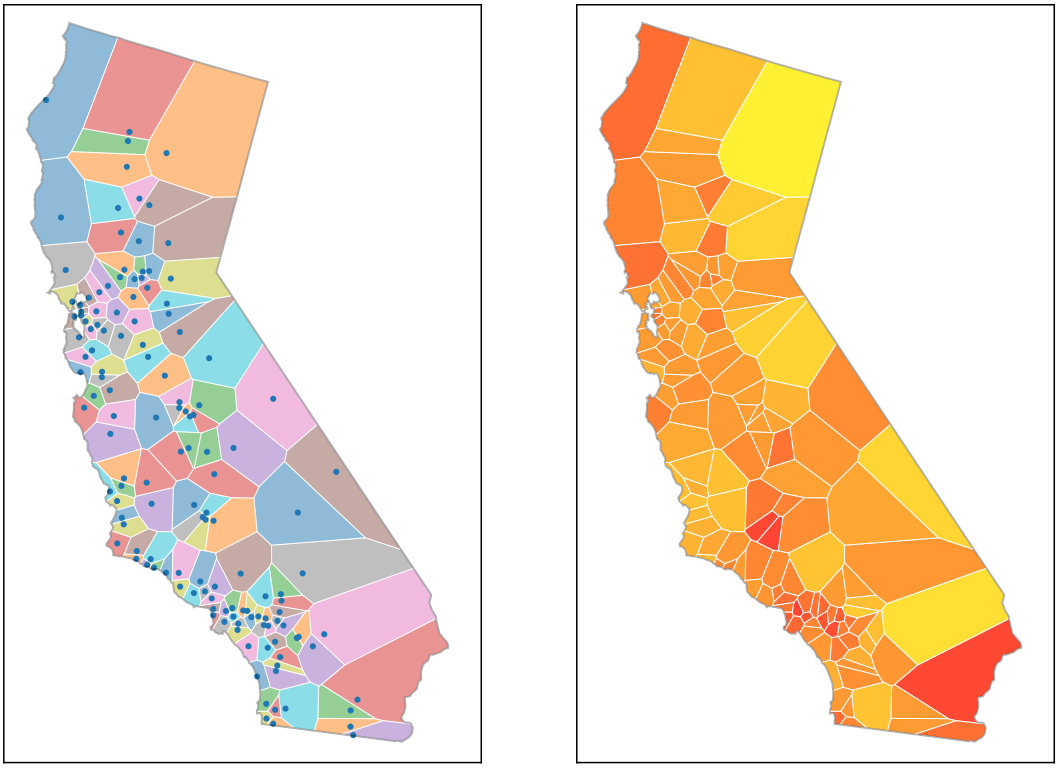


FIGURE 1.2: Air pollution monitors in California

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

2.2.2 Kernel Graph Regression

2.2.2.1 Gaussian Processes on Graphs

2.2.3 Regression with Network Cohesion

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

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 explore a class of reconstruction algorithms as applied to signals defined on the nodes of a Cartesian product graph. In particular, we pose the reconstruction task in terms of Bayesian inference of an underlying signal given a noisy partial observation, and investigate scalable methods for obtaining the posterior mean. We begin in section 4.1 by reviewing the concept of a graph product, and explain why we choose to look specifically at the Cartesian product. We also review how concepts from standard one-dimensional graph signal processing such as the GFT and spectral filtering can be extended to the two dimensional case. In section 4.2 we introduce the statistical model defining GSR on a Cartesian product graph, and derive two alternative methods for solving for the posterior mean. These comprise a Stationary Iterative Method (SIM) and a Conjugate Gradient Method (CGM). In each case, we show how graph spectral considerations can be leveraged to increase their convergence rate, and make use of the properties of the Kronecker product to complete each iteration efficiently. Finally, in section 4.3, we analyse the convergence properties of each method in depth and derive how the rate of convergence is affected by the hyperparameters. In particular, we show how the optimal choice of method depends on the value of said hyperparameters and offer some rules-of-thumb for selecting a method in practice.

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.

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

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

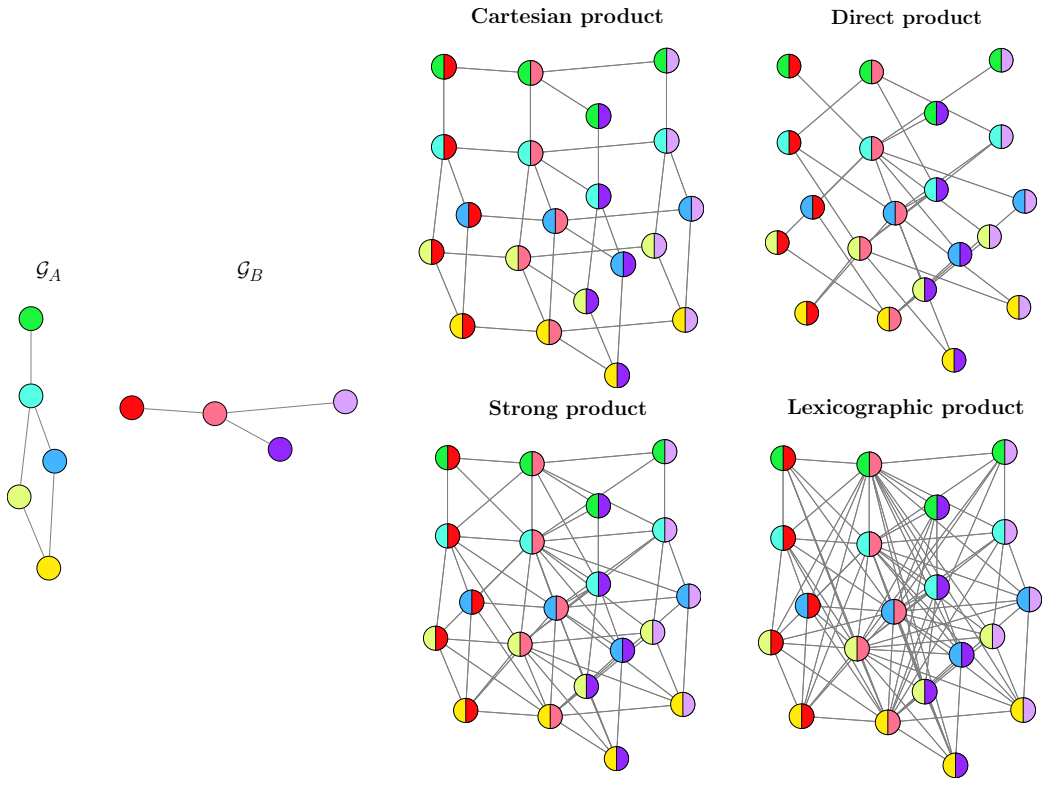


FIGURE 4.1: A graphical depiction of the four standard graph products

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.

	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.

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 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 $\mathbf{L}_B$.

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

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

$$\mathbf{\Lambda}_A = \begin{bmatrix} \lambda_1^{(A)}, & & & \\ & \lambda_2^{(A)} & & \\ & & \ddots & \\ & & & \lambda_A^{(A)} \end{bmatrix} \quad \text{and} \quad \mathbf{\Lambda}_B = \begin{bmatrix} \lambda_1^{(B)}, & & & \\ & \lambda_2^{(B)} & & \\ & & \ddots & \\ & & & \lambda_B^{(B)} \end{bmatrix} \tag{4.4}$$

Given these definitions, table 4.2 gives information about the spectral decomposition of the standard graph products.

	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.

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.

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \quad \text{where} \quad \mathbf{U} = \mathbf{U}_A \otimes \mathbf{U}_B \quad \text{and} \quad \mathbf{\Lambda} = \mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B \tag{4.5}$$

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.

$$\mathrm{GFT}(\mathbf{Y}) = \mathrm{mat}\left(\left(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\right)\mathrm{vec}\left(\mathbf{Y}\right)\right) = \mathbf{U}_B^\top\mathbf{Y}\mathbf{U}_A \tag{4.6}$$

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

$$\mathrm{IGFT}(\mathbf{Z}) = \mathrm{mat}\left(\left(\mathbf{U}_A \otimes \mathbf{U}_B\right)\mathrm{vec}\left(\mathbf{Z}\right)\right) = \mathbf{U}_B\mathbf{Z}\mathbf{U}_A^\top \tag{4.7}$$

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 $\mathrm{vec}\,(\cdot)$ function, along with its reverse operator $\mathrm{mat}\,(\cdot)$. However, this is somewhat problematic since $\mathrm{vec}\,(\cdot)$ is defined to act in *column-major* order, that is

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

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 $\mathrm{vec}\,(\cdot)$ 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.

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

$$\begin{aligned}\mathbf{H} &= g(\mathbf{L}_A \oplus \mathbf{L}_B) \\ &= \big(\mathbf{U}_A \otimes \mathbf{U}_B\big)\, g\big(\mathbf{\Lambda}_A \oplus \mathbf{\Lambda}_B\big)\, \big(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\big) \\ &= \big(\mathbf{U}_A \otimes \mathbf{U}_B\big)\, \mathrm{diag}\,(\mathrm{vec}\,(\mathbf{G}))\, \big(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\big) \\ &= \mathbf{U}\mathbf{D}_{\mathbf{G}}\mathbf{U}^\top \end{aligned} \tag{4.8}$$

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

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

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.

$$\mathbf{G}_{ba} = g\left(\lambda_a^{(A)}, \lambda_b^{(B)};\, \beta_a, \beta_b\right) \tag{4.10}$$

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.

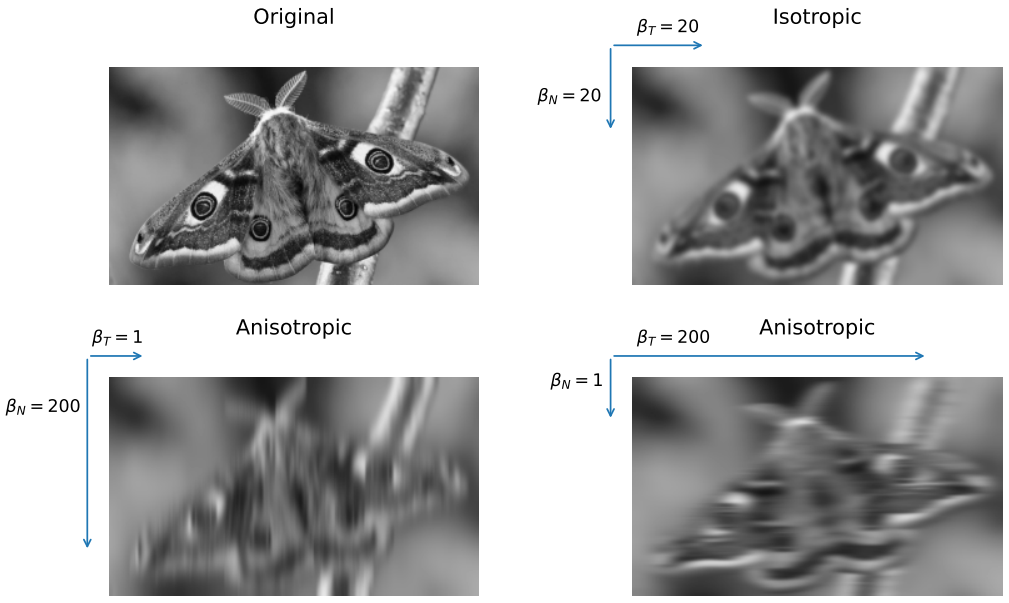


FIGURE 4.2: A graphical depiction of a time-vertex Cartesian product graph

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.

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

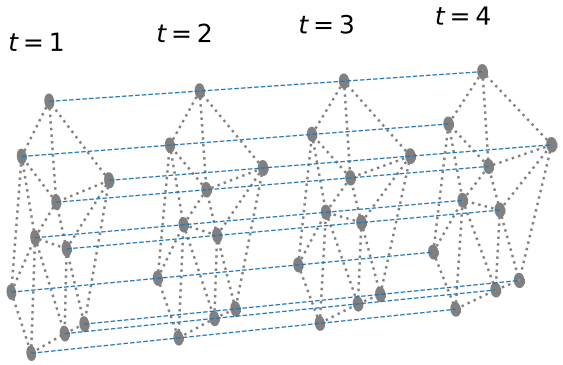


FIGURE 4.3: A graphical depiction of a time-vertex Cartesian product graph

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

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

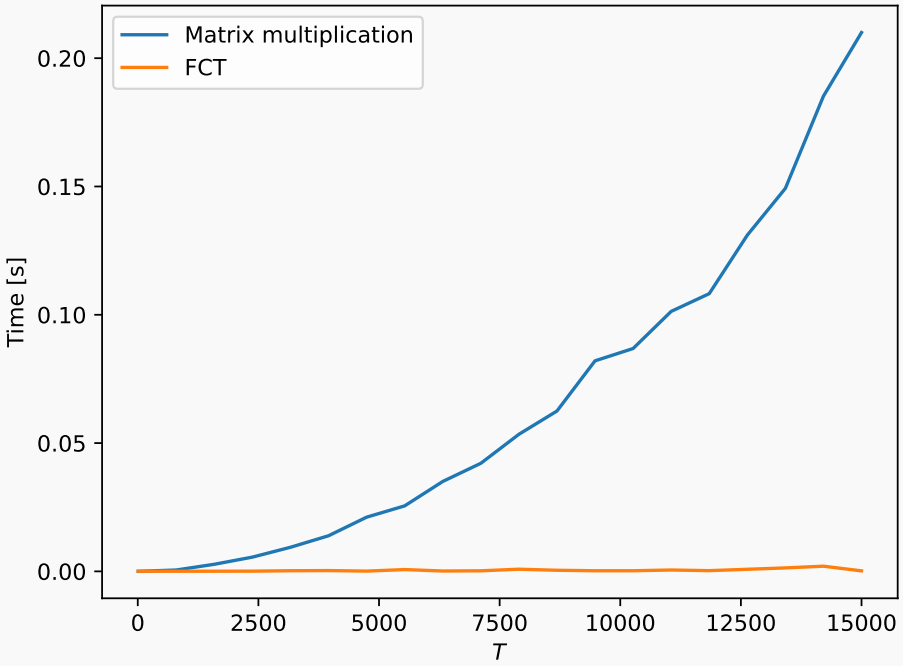
$$(\mathbf{U}_T)_{ij} = \cos\left(\frac{j-1}{T}(i-\frac{1}{2})\pi\right)$$

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

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

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 $\mathbf{U}^\top \mathbf{y}$ 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.

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{F} + \mathbf{E}\big) \tag{4.11}$$

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

$$\mathbf{S}_{nt} = \begin{cases} 1 & \text{if } \ (n,t) \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

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

$$\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F} \sim \mathcal{N}\Big(\operatorname{vec}(\mathbf{S} \circ \mathbf{F}),\ \operatorname{diag}(\operatorname{vec}(\mathbf{S}))\Big) \tag{4.13}$$

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 (4.8). For now, in the absence of any further information, we assume that the prior mean for $\mathbf{F}$ is zero across all elements.

$$\operatorname{vec}(\mathbf{F}) \sim \mathcal{N}\big(\mathbf{0},\ \gamma^{-1}\mathbf{H}^2\big) \tag{4.14}$$

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

$$\pi\big(\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{\pi\big(\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F}\big)\pi(\mathbf{F})}{\pi(\mathbf{Y})}. \tag{4.15}$$

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

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

$$\operatorname{vec}(\mathbf{F}) \mid \mathbf{Y} \sim \mathcal{N}\left(\mathbf{P}^{-1} \operatorname{vec}(\mathbf{Y}),\ \mathbf{P}^{-1}\right) \tag{4.16}$$

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

$$\mathbf{P} = \operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma \mathbf{H}^{-2} \tag{4.17}$$

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.

$$\operatorname{vec}(\mathbf{F}) = \left(\operatorname{diag}(\operatorname{vec}(\mathbf{S})) + \gamma \mathbf{H}^{-2}\right)^{-1} \operatorname{vec}(\mathbf{Y}) \tag{4.18}$$

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. (4.18) 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

$$\mathrm{vec}\,(\mathbf{F}) = (\mathbf{M} - \mathbf{N})^{-1}\mathrm{vec}\,(\mathbf{Y}) \tag{4.19}$$

By noting that

$$\mathbf{M}\mathrm{vec}\,(\mathbf{F}) = \mathbf{N}\mathrm{vec}\,(\mathbf{F}) + \mathrm{vec}\,(\mathbf{Y}) \tag{4.20}$$

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

we devise an iterative scheme given by

$$\mathrm{vec}\,(\mathbf{F}_{k+1}) = \mathbf{M}^{-1}\mathbf{N}\mathrm{vec}\,(\mathbf{F}_k) + \mathbf{M}^{-1}\mathrm{vec}\,(\mathbf{Y}) \tag{4.22}$$

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. (4.22) 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 folowing, we set

$$\mathbf{M} = \gamma\mathbf{H}^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \mathrm{diag}\left(\mathrm{vec}\left(\mathbf{S}'\right)\right). \tag{4.23}$$

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

$$\mathbf{S}'_{nt} = \begin{cases} 1 & \text{if } (n,t) \notin \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

In this way, the update equation is given by

$$\operatorname{vec}(\mathbf{F}_{k+1}) = \left(\gamma\mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}'\right)\right) \operatorname{vec}(\mathbf{F}_k) + \left(\gamma\mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \operatorname{vec}(\mathbf{Y}) \tag{4.25}$$

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. (4.8), this can be written as

$$\begin{aligned} \mathbf{M}^{-1} &= \left(\gamma\mathbf{H}^{-2} + \mathbf{I}\right)^{-1} \\ &= \left(\gamma\left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}(\mathbf{G})\right)^{-2} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) + \mathbf{I}\right)^{-1} \\ &= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \left(\gamma \operatorname{diag}\left(\operatorname{vec}(\mathbf{G})\right)^{-2} + \mathbf{I}\right)^{-1} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \\ &= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}(\mathbf{J})\right) \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \end{aligned} \tag{4.26}$$

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

$$\mathbf{J}_{nt} = \frac{\mathbf{G}_{nt}^2}{\mathbf{G}_{nt}^2 + \gamma}. \tag{4.27}$$

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

$$\mathbf{F}_{k+1} = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top + \mathbf{F}_0 \tag{4.28}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{4.29}$$

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. (4.26), 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. (4.27). 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

$$\rho(\mathbf{M}^{-1}) = \frac{1}{1+\gamma} \tag{4.30}$$

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

$$\rho(\mathbf{AB}) \leq \rho(\mathbf{A})\,\rho(\mathbf{B}) \tag{4.31}$$

[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 4.3.1.

Finally, the update formulas given in eqs. (4.28) and (4.29) can be written equivalently as

$$\Delta\mathbf{F}_0 = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{4.32}$$

$$\Delta\mathbf{F}_{k+1} = \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \Delta\mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \tag{4.33}$$

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**.

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: $\mathbf{F}$

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

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

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 (4.8). 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.

$$\mathbf{F} = \mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top \tag{4.34}$$

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 (4.13), and the prior of equation (4.14), 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

$$\operatorname{vec}(\mathbf{Z}) = \left(\mathbf{C} + \gamma \mathbf{I}_T \otimes \mathbf{I}_N\right)^{-1} \operatorname{vec}\left(\mathbf{G} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T)\right) \tag{4.35}$$

where $\mathbf{C}$ is the symmetric PSD matrix

$$\mathbf{C} = \mathbf{D}_\mathbf{G} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_\mathbf{S} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_\mathbf{G} \tag{4.36}$$

where we have abbreviated $\operatorname{diag}(\operatorname{vec}(\mathbf{G}))$ and $\operatorname{diag}(\operatorname{vec}(\mathbf{S}))$ as $\mathbf{D}_\mathbf{G}$ and $\mathbf{D}_\mathbf{S}$ respectively. Note that the conditioning of the coefficient matrix $\mathbf{C} + \gamma \mathbf{I}$ 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 $\operatorname{vec}(\mathbf{R})$ by the coefficient matrix can be computed efficiently as

$$\operatorname{mat}\left(\left(\mathbf{C} + \gamma \mathbf{I}\right) \operatorname{vec}(\mathbf{R})\right) = \gamma \mathbf{R} + \mathbf{G} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S} \circ \left(\mathbf{U}_N (\mathbf{G} \circ \mathbf{R}) \mathbf{U}_T^\top\right)\right) \mathbf{U}_T\right)$$

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 $O\left(NT \log NT\right)$ for data residing on a grid (see section 4.2).

The linear system defined eq. (4.35) can be understood as a two-sided symmetrically preconditioned version of the original linear system given in eq. (4.18). In particular, the new expression can be constructed by modifying the original system in the following way.

$$\left(\mathbf{\Psi}^\top \left(\mathbf{D}_\mathbf{S} + \gamma \mathbf{H}^{-2}\right) \mathbf{\Psi}\right) \left(\mathbf{\Psi}^{-1} \operatorname{vec}(\mathbf{F})\right) = \mathbf{\Psi}^\top \operatorname{vec}(\mathbf{Y}), \tag{4.37}$$

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

where

$$\boldsymbol{\Psi} = \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_{\mathbf{G}}. \tag{4.38}$$

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 Verifying basic properties

In this subsection, we seek to verify several basic properties of the CGM and SIM. First, we want to show that both methods converge to the same prediction for the smooth underlying signal $\mathbf{F}$ an check that the solution is

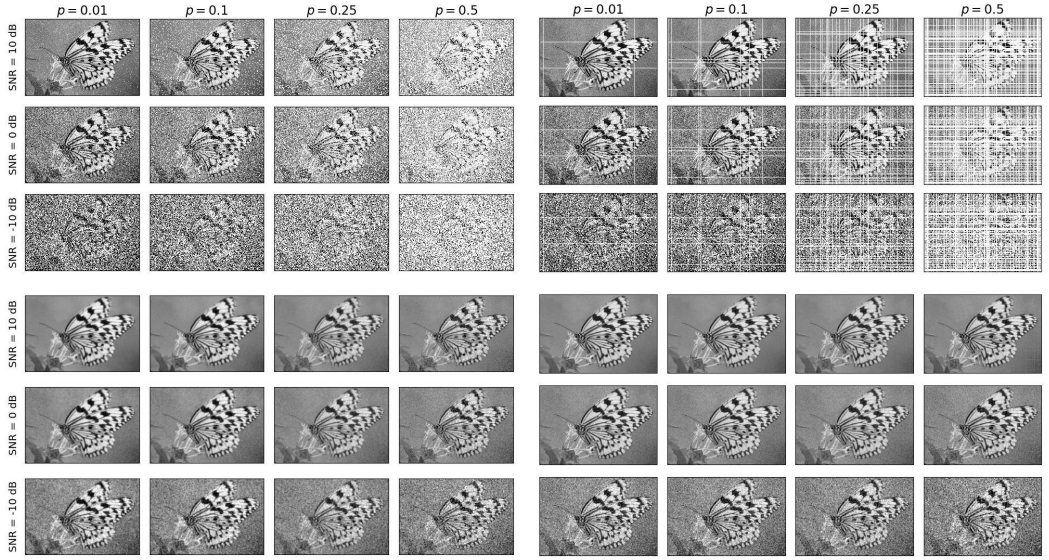


FIGURE 4.4: The output from the first experiment is depicted. In the top left quadrant, the input images are shown across a range of noise levels and missing pixel percentages, with missing pixels chosen uniformly at random. Below that, in the lower left quadrant, the corresponding reconstructed images are shown. The right half of the plot is the same, except here entire columns and rows of pixels are removed at random.

4.3 Convergence properties

In this section, we take a closer look at the convergence properties of both the SIM and the CGM. As we will show, the method with the best convergence rate depends on the value of the hyperparameters β, which describes the strength of the graph filter, γ, which determines the regularisation strength, and $m = |\mathcal{S}'|/NT$, which is the fraction of values that are missing from the original graph signal. For both methods, we provide bounds on the convergence rate and show how each of these three parameters can be expected to effect convergence in practice.

We begin first with a brief review the generic convergence properties of the SIM and CGM. Next, we analyse their behaviour in the specific case of graph signal reconstruction as described in our model. We demonstrate that, for both models, the convergence rate is bounded by two edge cases; one in the limit of a weak graph filter and the other in the limit of a strong graph filter. Furthermore we show that in the weak filter limit, where $\beta \to 0$, the CGM has better convergence behaviour. On the other hand, in the strong filter limit where $\beta \to \infty$, the SIM has better convergence behaviour. We also provide intuition for selecting a method for intermediate values of β.

4.3.1 Convergence of the SIM

In the SIM, we have that

$$\mathbf{M}\text{vec}\left(\mathbf{F}\right) = \mathbf{N}\text{vec}\left(\mathbf{F}\right) + \text{vec}\left(\mathbf{Y}\right) \tag{4.39}$$

where $\text{vec}\left(\mathbf{F}\right)$ represents the true solution to the linear system. This leads directly to an update equation given by

$$\mathbf{M}\text{vec}\left(\mathbf{F}_k\right) = \mathbf{N}\text{vec}\left(\mathbf{F}_{k-1}\right) + \text{vec}\left(\mathbf{Y}\right) \tag{4.40}$$

Subtracting eq. (4.39) from eq. (4.40) gives

$$\begin{aligned} \mathbf{M}\text{vec}\left(\mathbf{F}_k\right) - \mathbf{M}\text{vec}\left(\mathbf{F}\right) &= \mathbf{N}\text{vec}\left(\mathbf{F}_{k-1}\right) - \mathbf{N}\text{vec}\left(\mathbf{F}\right) \\ \text{vec}\left(\mathbf{E}_k\right) &= \mathbf{M}^{-1}\mathbf{N}\text{vec}\left(\mathbf{E}_{k-1}\right) \\ &= \left(\mathbf{M}^{-1}\mathbf{N}\right)^k \text{vec}\left(\mathbf{E}_0\right) \end{aligned} \tag{4.41}$$

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,

$$\lim_{k\to\infty} \text{vec}\left(\mathbf{E}_k\right) = \lim_{k\to\infty} \left(\mathbf{M}^{-1}\mathbf{N}\right)^k \text{vec}\left(\mathbf{E}_0\right) = \mathbf{0}. \tag{4.42}$$

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 [Demmel, 1997]. Therefore, given that

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

where we use the shorthands

$$\mathbf{U} = \mathbf{U}_T \otimes \mathbf{U}_N, \quad \mathbf{D}_\mathbf{J} = \text{diag}\left(\text{vec}\left(\mathbf{J}\right)\right), \quad \text{and} \quad \mathbf{D}_{\mathbf{S}'} = \text{diag}\left(\text{vec}\left(\mathbf{S}'\right)\right).$$

the complexity of the SIM in our context follows

$$n_{\text{SIM}} \propto -\frac{1}{\log \rho \left(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'} \right)} \tag{4.43}$$

The matrix $\mathbf{J}$ [see eq. (4.27)] has entries that depend on both the the regularisation parameter γ and the spectral scaling matrix $\mathbf{G}$ [see eqs. (4.9) and (4.10)], which is itself a function of the graph filter parameter(s) β. The matrix $\mathbf{D}_{\mathbf{S}'}$ has entries that depend on the structure of the missing data in the graph signal. Therefore we expect that ρ, and consequently the number of steps required for convergence n_{SIM}, may be affected by all three of these variables.

4.3.2 Convergence of the CGM

In the conjugate gradient method, by contrast, the number of steps required to achieve some termination condition is well-known to follow $O(\sqrt{\kappa})$, where κ is the condition number of the coefficient matrix Kelley [1995]. In our case, the coefficient matrix is given in equation (4.35) as $\mathbf{C} + \gamma\mathbf{I}$.

Therefore, given the definition for $\mathbf{C}$ given in equation (4.36), we expect that the number of iterations required for convergence of the CGM will scale as

$$n_{\text{CGM}} \propto \sqrt{\kappa \left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I} \right)} \tag{4.44}$$

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 missingness structure, and the parameter γ. Therefore, we should expect that, in general, convergence is affected by all three of these variables.

4.3.3 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. (4.8)] applied to any graph signal $\text{vec}\,(\mathbf{Y})$ returns the same signal back

$$\mathbf{H}\text{vec}\,(\mathbf{Y}) = \text{vec}\,(\mathbf{Y})$$

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). In this limit, every element of the spectral scaling matrix $\mathbf{G}$ will be equal to one. Given eq. (4.27), this further implies the every entry in the matrix $\mathbf{J}$ becomes $1/(1+\gamma)$. Therefore,

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

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

$$\begin{aligned} \lim_{\beta \to 0} \rho\Big(\mathbf{U}\mathbf{D}_{\mathbf{J}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}\Big) &= \frac{1}{1+\gamma}\rho\Big(\mathbf{U}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}'}\Big) \\ &= \frac{1}{1+\gamma}\rho\Big(\mathbf{D}_{\mathbf{S}'}\Big) \\ &= \frac{1}{1+\gamma} \end{aligned} \tag{4.45}$$

Next, consider the condition number κ of the coefficient matrix in the CGM. Again, this can be directly evaluated as

$$\begin{aligned} \lim_{\beta \to 0} \kappa\left(\mathbf{D}_{\mathbf{G}}\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U}\mathbf{D}_{\mathbf{G}} + \gamma\mathbf{I}\right) &= \kappa\left(\mathbf{U}^{\top}\mathbf{D}_{\mathbf{S}}\mathbf{U} + \gamma\mathbf{I}\right) \\ &= \kappa\left(\mathbf{U}^{\top}\left(\mathbf{D}_{\mathbf{S}} + \gamma\mathbf{I}\right)\mathbf{U}\right) \\ &= \frac{1+\gamma}{\gamma} \end{aligned} \tag{4.46}$$

Given eqs. (4.43) and (4.44), 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

$$\lim_{\beta \to 0} n_{\text{SIM}} \propto \frac{1}{\log(1+\gamma)} \tag{4.47}$$

$$\lim_{\beta \to 0} n_{\text{CGM}} \propto \sqrt{\frac{1}{\gamma} + 1} \tag{4.48}$$

Several similarities exist between these two expressions. In both cases, the fraction of unobserved values, m, has no effect on the convergence rate. When γ is high, 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 $\gamma = 0$.

$$\lim_{\beta \to 0} n_{\text{SIM}} \propto \gamma^{-1} + \frac{1}{2} - \frac{\gamma}{12} + O(\gamma^2) \tag{4.49}$$

$$\lim_{\beta \to 0} n_{\text{CGM}} \propto \gamma^{-1/2} + \frac{\gamma^{1/2}}{2} + O\left(\gamma^{3/2}\right) \tag{4.50}$$

The dominant behaviour for small γ follows $O(\gamma^{-1})$ for the SIM and $O(\gamma^{-1/2})$ for the CGM. This implies that, in the limit of a weak filter, the CGM will be generally preferable, especially when γ is small.

4.3.4 Lower bound on convergence: the strong filter limit

Consider now the limiting case of a strong filter, where every spectral component is filtered out except the the first Laplacian frequency component $\mathbf{u}_1 \propto \mathbf{1}$ (also known as the bias), with eigenvalue $\lambda_1 = 0$, which passes through the filter unaffected. 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 graph filter operates on a generic graph signal $\text{vec}(\mathbf{Y})$ as follows.

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

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

$$\lim_{\beta \to \infty} \mathbf{D}_{\mathbf{G}} = \mathbf{\Delta}, \quad \text{and} \quad \lim_{\beta \to \infty} \mathbf{D}_{\mathbf{J}} = \frac{1}{1+\gamma}\mathbf{\Delta}, \quad \text{where} \quad \mathbf{\Delta} = \begin{bmatrix} 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & \\ \vdots & & & \ddots \end{bmatrix}$$

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

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

Note that

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

where $\mathbf{O}$ is a matrix of ones. Therefore the spectral radius is given by

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

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 $\sum \mathrm{vec}\,(\mathbf{S}')_i = |\mathcal{S}'|$. Denoting $m = |\mathcal{S}'|/NT$, this can be expressed as

$$\lim_{\beta\to\infty} \rho(\mathbf{U}\mathbf{D}_\mathbf{J}\mathbf{U}^\top\mathbf{D}_{\mathbf{S}'}) = \frac{1}{1+\gamma}\frac{|\mathcal{S}'|}{NT} = \frac{m}{1+\gamma} \tag{4.51}$$

In the case of the CGM, we have that

$$\begin{aligned}
&\lim_{\beta\to\infty} \quad \kappa\left(\mathbf{D}_\mathbf{G}\mathbf{U}^\top\mathbf{D}_\mathbf{S}\mathbf{U}\mathbf{D}_\mathbf{G} + \gamma\mathbf{I}\right) \\
&\qquad = \kappa\left(\mathbf{D}_\boldsymbol{\Delta}\mathbf{U}^\top\mathbf{D}_\mathbf{S}\mathbf{U}\mathbf{D}_\boldsymbol{\Delta} + \gamma\mathbf{I}\right) \\
&\qquad = \kappa\left(\begin{bmatrix} \mathbf{u}_1^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{bmatrix} \mathbf{D}_\mathbf{S}\Big[\mathbf{u}_1, \mathbf{0}, \ldots, \mathbf{0}\Big] + \gamma\mathbf{I}\right) \\
&\qquad = \kappa\left(\frac{1}{NT}\begin{bmatrix} |\mathcal{S}| & 0 & 0 & \cdots \\ 0 & 0 & 0 & \\ \vdots & & & \ddots \end{bmatrix} + \gamma\mathbf{I}\right) \\
&\qquad = \frac{1-m+\gamma}{\gamma}
\end{aligned} \tag{4.52}$$

Given eqs. (4.43) and (4.44), we can write the scaling rate for the number of iterations in the SIM and CGM respectively.

$$\lim_{\beta\to\infty} n_{\text{SIM}} \propto \frac{1}{\log(1+\gamma) - \log m} \tag{4.53}$$

$$\lim_{\beta\to\infty} n_{\text{CGM}} \propto \sqrt{\frac{1-m+\gamma}{\gamma}} \tag{4.54}$$

Note 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.

$$\lim_{\beta\to\infty} n_{\text{SIM}} \propto -\frac{1}{\log m} - \frac{\gamma}{\log^2 m} + O\left(\gamma^2\right) \tag{4.55}$$

$$\lim_{\beta\to\infty} n_{\text{CGM}} \propto \left(\frac{\gamma}{1-m}\right)^{-1/2} + \frac{1}{2}\left(\frac{\gamma}{1-m}\right)^{1/2} + O\left(\gamma^{3/2}\right) \tag{4.56}$$

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. This implies that, in the strong filter limit, the SIM will be preferable when γ is small. Note that m cannot scale arbitrarily close to zero or one, since it will surely be between $1/NT$ and $1 - 1/NT$.

4.3.5 Choosing a method in practice

We have shown that the CGM will be preferable in the limit of a weak filter, when $\beta \to 0$, and that the SIM will be preferable in the limit of a strong filter, when $\beta \to \infty$. For filters that are neither asymptotically strong or weak, i.e. positive finite values of β, it is natural to ask which method can be expected to perform better. This decision will be more significant when γ is small since, when it is large, both methods converge quickly. A visual overview of the convergence behaviour is given in fig. 4.5.

In order to test the true convergence behaviour in practice, we ran a small experiment on synthetic data residing on a grid. In particular, we use a 16×16 grid of pixel data, and

generated an observation matrix $\mathbf{Y}$ via white Gaussian noise. Next, for three different values of m, we chose pixels to be removed uniformly at random. We then counted the total number of iterations required to reconstruct the signal to a tolerance of 10^{-8} for both the SIM and the CGM, over a range of values of γ. This was done for four values of β, with a diffusion graph filter (see table 2.1). The results are shown in fig. 4.6.

As is visible, the optimal choice of method will depend strongly on the choice of hyperparameters. Given these experiments, we give some "rules-of-thumb" for making this choice under various hyperparameter settings. This is summarised in table 4.4.

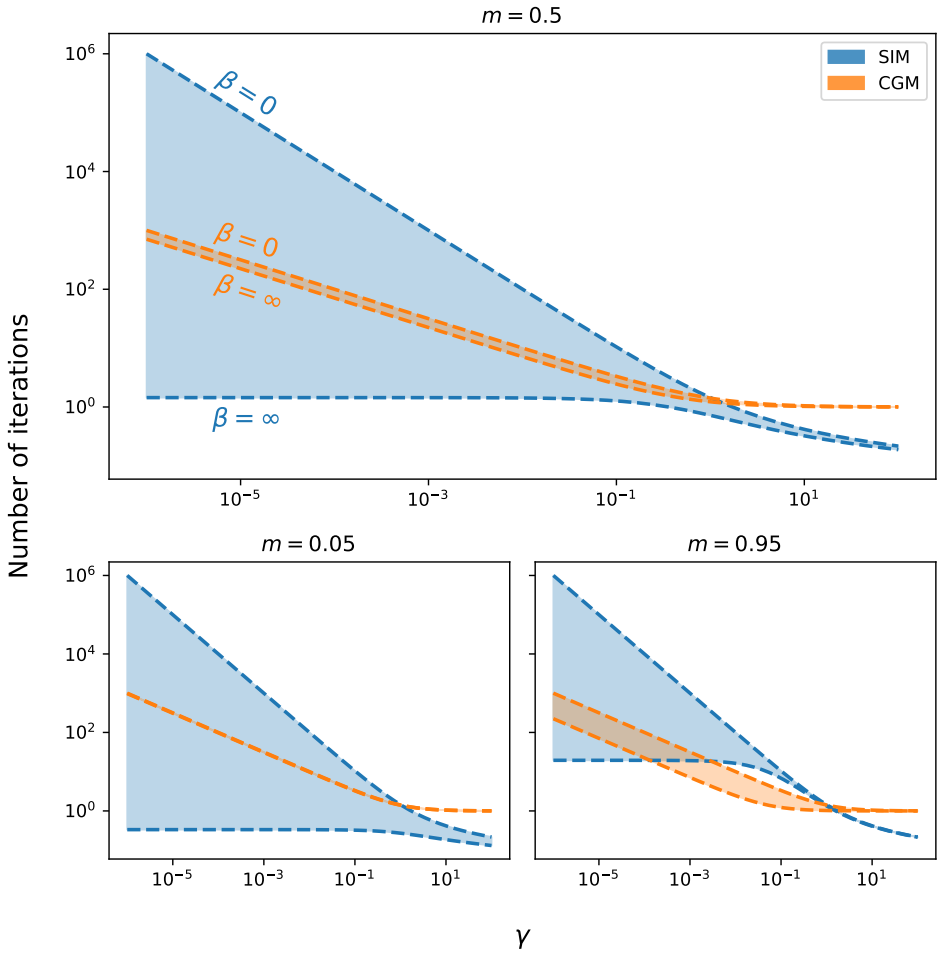


FIGURE 4.5: Here, we have plotted the functions given in eqs. (4.47) and (4.48) and eqs. (4.53) and (4.54) to give a visual sense of the scaling rate of each method. Note that, since these functions are *proportional* to the true number of iterations, the real values may be shifted up or down in this log-log plot.

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. (4.18). While a naive approach to computing this would have time complexity of $O(N^3T^3)$, we can utilise classic iterative methods in conjunction with the properties of the Kronecker product to solve the linear system 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

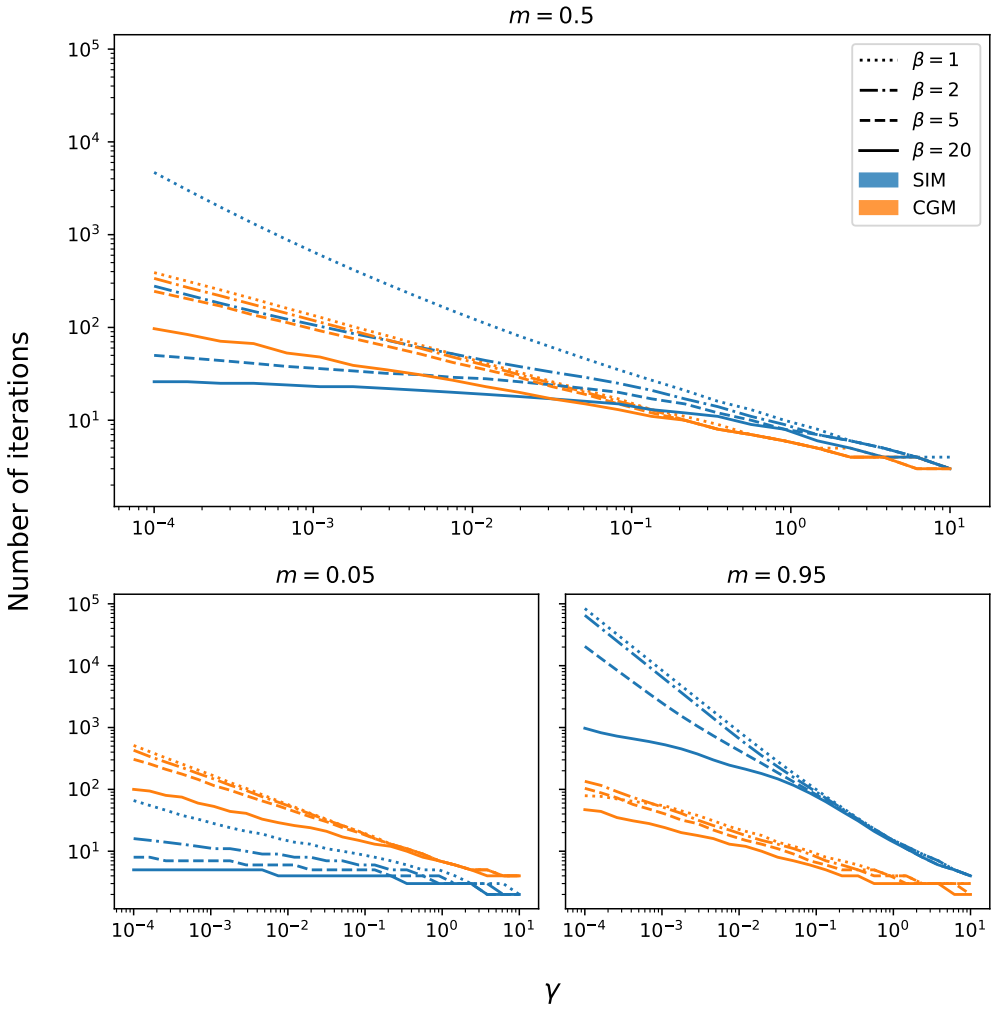


FIGURE 4.6: The emprical number of steps required to reach some comparable convergence criteria is shown for a range of values of β and γ for both the SIM and CGM. Note that in all cases, $m = 0.5$.

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

(SIM) and the Conjugate Gradient Method (CGM), tailored for the task of graph signal reconstruction. These were designed by making use of domain knowledge to produce a matrix splitting for the SIM and a preconditioner for the CGM that both guarantee bounded convergence. By analysing the spectral properties of the matrices involved in each algorithm, we provided a detailed overview 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 hyperparameters β, γ and m. Given these results, we have provided some rules-of-thumb for making a choice of iterative method in practive, which is summaries in table 4.4. This decision is of less importance when γ is large, as both methods converge quickly in this domain, however it may be of great significance when γ is small.

	SIM		**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 $\gamma \to 0$.

Chapter 5

Multivariate Regression Models for Network-Structured Data

In this chapter, we build on the results from chapter 4 to solve several Bayesian regression models for graph signals.

5.1 Kernel Graph Regression with Unrestricted Missing Data Patterns

5.1.1 Model description

Consider a sequence of graph signals $\mathbf{y}_t$ measured on a static N-node graph, each with an associated vector of explanatory variables $\mathbf{x}_t$, containing M distinct features. Each graph signal may have unique and arbitrary missing data, specified by a binary vector $\mathbf{s}_t$ which contains ones where data was successfully collected and zeros elsewhere. Any entry in $\mathbf{y}_t$ where no data was collected should be filled with a zero. As such, the input data for this problem can be summarised as follows.

$$\text{input data} = \left\{ \left(\mathbf{x}_t \in \mathbb{R}^M,\ \mathbf{y}_t \in \mathbb{R}^N,\ \mathbf{s}_t \in [0,1]^N \right) \right\}_{t=1}^{T} \tag{5.1}$$

In the following, we assume that the explanatory variables not not contain any missing values. If missing values are present here, they can be filled in using standard techniques such as those described in Little and Rubin [2019]. Note that, under this model specification, there is no hard distinction between training data and data for which we would

like to make a prediction. To indicate a particular value of $\mathbf{x}_t$ for which a full prediction should be made, one can just 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 $\boldsymbol{\phi}(\mathbf{x}_t) \in \mathbb{R}^P$.

$$\boldsymbol{\phi}(\mathbf{x}_t) = \begin{bmatrix} \phi_1(\mathbf{x}_t) & \phi_2(\mathbf{x}_t) & \dots & \phi_P(\mathbf{x}_t) \end{bmatrix}^\top \tag{5.2}$$

In the following, we will 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.

$$\mathbf{y}_t = \mathbf{s}_t \circ \big(\mathbf{W}\boldsymbol{\phi}(\mathbf{x}_t) + \mathbf{e}_t\big), \qquad \text{for } \; t = 1, 2, ..., T \tag{5.3}$$

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 iid Gaussian noise with zero mean and unit variance. The basis function vectors can be horizontally stacked together to form a design matrix $\boldsymbol{\Phi}$.

$$\boldsymbol{\Phi} \in \mathbb{R}^{P \times T} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \dots & \phi_1(\mathbf{x}_T) \\ \phi_2(\mathbf{x}_1) & \phi_2(\mathbf{x}_2) & \dots & \phi_2(\mathbf{x}_T) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_P(\mathbf{x}_1) & \phi_P(\mathbf{x}_2) & \dots & \phi_P(\mathbf{x}_T) \end{bmatrix} \tag{5.4}$$

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

$$\mathbf{Y} = \mathbf{S} \circ \big(\mathbf{W}\boldsymbol{\Phi} + \mathbf{E}\big) \tag{5.5}$$

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 iid Gaussian noise with zero mean and unit variance. This implies that the probability distribution for $\mathrm{vec}\,(\mathbf{Y}) \mid \mathbf{W}$ is given by

$$\mathrm{vec}\,(\mathbf{Y}) \mid \mathbf{W} \sim \mathcal{N}\big(\mathrm{vec}\,(\mathbf{S} \circ (\mathbf{W}\boldsymbol{\Phi})),\ \mathbf{I}\big) \tag{5.6}$$

In order to find the posterior distribution over the model coefficients $\mathbf{W}$, we must specify a prior. This should capture the assumption that predicted signals are expected to be smooth with respect to the topology of the graph. We assert that an appropriate prior for $\mathbf{W}$ is

$$\operatorname{vec}(\mathbf{W}) \sim \mathcal{N}\left(\mathbf{0},\ \gamma^{-1}\mathbf{I}_P \otimes \mathbf{H}_N^2\right) \tag{5.7}$$

where $\mathbf{H}_N \in \mathbb{R}^N$ is a graph filter constructed according to one of the univariate filter functions defined in table 2.1, and γ is a hyperparameter representing the prior precision. The justification for this prior is as follows. Consider a random graph signal which is constructed as $\mathbf{W}\boldsymbol{\phi}$ where both $\mathbf{W}$ and $\boldsymbol{\phi}$ have Gaussian iid entries with zero mean and unit variance. The probability distribution of their product will also be an iid multivariate Gaussian with zero mean. Consider now smoothing this signal by applying a graph filter $\mathbf{H}_N$. The resultant signal will have the same probability distribution as $\mathbf{W}\boldsymbol{\phi}$ if instead $\mathbf{W}$ was drawn from the distribution given in eq. (5.7). Therefore, the effect of this prior can be understood as promoting the probability 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. (5.7), we can ask what the implied prior over $\mathbf{F}$ 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.

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

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

$$\operatorname{vec}(\mathbf{Y}) \mid \mathbf{F} \sim \mathcal{N}\big(\operatorname{vec}(\mathbf{S} \circ \mathbf{F}),\ \mathbf{I}\big) \tag{5.8}$$

$$\operatorname{vec}(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\ \gamma^{-1}\big(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\big) \otimes \mathbf{H}_N^2\Big) \tag{5.9}$$

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

$$\mathbf{\Phi}^\top\mathbf{\Phi} \in \mathbb{R}^{T\times T} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_1)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\ \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_2)^\top\boldsymbol{\phi}(\mathbf{x}_T) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_2) & \dots & \boldsymbol{\phi}(\mathbf{x}_T)^\top\boldsymbol{\phi}(\mathbf{x}_T) \end{bmatrix} \tag{5.10}$$

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 $\mathbf{\Phi}^\top\mathbf{\Phi}$ with a so-called kernel matrix $\mathbf{K} \in \mathbb{R}^{T\times T}$, which has entries defined by

$$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{5.11}$$

$\kappa(\cdot, \cdot)$ can be any valid Mercer kernel. A common example is the Gaussian kernel given below.

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{5.12}$$

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

$$\text{vec}\,(\mathbf{F}) \sim \mathcal{N}\Big(\mathbf{0},\ \gamma^{-1}\mathbf{K} \otimes \mathbf{H}_N^2\Big) \tag{5.13}$$

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

$$\text{vec}\,(\mathbf{F})\ |\ \mathbf{Y} \sim \mathcal{N}\left(\bar{\mathbf{P}}^{-1}\text{vec}\,(\mathbf{Y}),\ \bar{\mathbf{P}}^{-1}\right) \tag{5.14}$$

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

$$\bar{\mathbf{P}} = \mathbf{D}_\mathbf{S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} \tag{5.15}$$

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. To see this, compare the KGR posterior described in eqs. (5.14) and (5.15) to the GSR posterior described in eqs. (4.16) and (4.17). In the original GSR model, the posterior precision matrix is given by $\mathbf{D_S} + \gamma\mathbf{H}^{-2}$, and in the case of KGR it is given by $\mathbf{D_S} + \gamma\mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}$. In each case, it is the sum of $\mathbf{D_S}$ and a Hermitian matrix. To make this analogy clearer, we can expand the second term of each expression in terms of its respective eigendecomposition. For GSR, this is

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

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

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

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

$$\mathbf{K} = \mathbf{V}\boldsymbol{\Lambda}_K\mathbf{V}^\top, \quad \text{and} \quad \mathbf{H}_N = \mathbf{U}_N\, g\left(\boldsymbol{\Lambda}_N\right)\mathbf{U}_N^\top \tag{5.16}$$

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 $\mathbf{D}_{\bar{\mathbf{G}}} = \mathrm{diag}\left(\mathrm{vec}\left(\bar{\mathbf{G}}\right)\right)$, and $\bar{\mathbf{G}}$ has elements given by

$$\bar{\mathbf{G}}_{nt} = g\left(\lambda_n^{(N)}\right)\sqrt{\lambda_t^{(K)}} \tag{5.17}$$

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

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

As such, the iterative algorithms developed in chapter 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. The goal is to solve the following linear system

$$\operatorname{vec}(\mathbf{F}) = \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2}\right)^{-1} \operatorname{vec}(\mathbf{Y}) \tag{5.18}$$

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

$$\mathbf{M} = \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \mathbf{D}_{\mathbf{S}'}. \tag{5.19}$$

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

$$\begin{aligned}
\mathbf{M}^{-1} &= \left(\gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\gamma \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{G}}}^{-2}\, \bar{\mathbf{U}}^{\top} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right) \bar{\mathbf{U}}^{\top}\right)^{-1} \\
&= \bar{\mathbf{U}} \left(\gamma \mathbf{D}_{\bar{\mathbf{G}}}^{-2} + \mathbf{I}_{NT}\right)^{-1} \bar{\mathbf{U}}^{\top} \\
&= \bar{\mathbf{U}}\, \mathbf{D}_{\bar{\mathbf{J}}}\, \bar{\mathbf{U}}^{\top}
\end{aligned} \tag{5.20}$$

where $\mathbf{D}_{\bar{\mathbf{J}}} = \operatorname{diag}\left(\operatorname{vec}\left(\bar{\mathbf{J}}\right)\right)$, and $\bar{\mathbf{J}}$ has entries given by

$$\bar{\mathbf{J}}_{nt} = \frac{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right)}{\lambda_t^{(K)} g^2\left(\lambda_n^{(N)}\right) + \gamma} = \frac{\bar{\mathbf{G}}_{nt}^2}{\bar{\mathbf{G}}_{nt}^2 + \gamma} \tag{5.21}$$

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

$$\mathbf{F}_{k+1} = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k \right) \mathbf{V} \right) \right) \mathbf{V}^\top + \mathbf{F}_0 \tag{5.22}$$

$$\text{with} \qquad \mathbf{F}_0 = \mathbf{U}_N \left(\bar{\mathbf{J}} \circ \left(\mathbf{U}_N^\top \mathbf{Y} \, \mathbf{V} \right) \right) \mathbf{V}^\top \tag{5.23}$$

Next, let us return to the CGM. Recall that the strategy for solving the linear system in eq. (5.18) is to utilise a symmetric a preconditioner $\boldsymbol{\Psi}$ such that the new system is given by

$$\left(\boldsymbol{\Psi}^\top \left(\mathbf{D}_\mathbf{S} + \gamma \mathbf{K}^{-1} \otimes \mathbf{H}_N^{-2} \right) \boldsymbol{\Psi} \right) \left(\boldsymbol{\Psi}^{-1} \operatorname{vec}(\mathbf{F}) \right) = \boldsymbol{\Psi}^\top \operatorname{vec}(\mathbf{Y}), \tag{5.24}$$

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

$$\boldsymbol{\Psi} = \left(\mathbf{V} \otimes \mathbf{U}_N \right) \operatorname{diag}\left(\operatorname{vec}\left(\bar{\mathbf{G}} \right) \right) = \bar{\mathbf{U}} \mathbf{D}_{\bar{\mathbf{G}}}. \tag{5.25}$$

This preconditioner transforms the coefficient matrix into

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

5.2 Regression with Network Cohesion

5.2.1 Model description

Consider a sequence of graph signals

5.2.2 Regression with node-level covariates

Hello

5.2.3 Convergence properties

Hello

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.

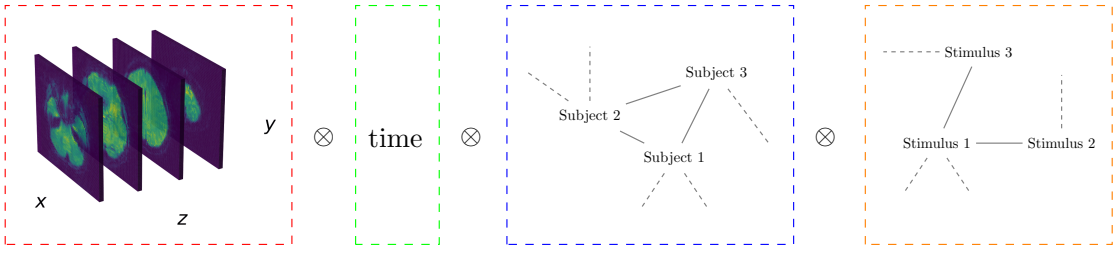


FIGURE 6.1: Graphical depiction of a six-way graph signal originating from a hypothetical fMRI experiment.

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.

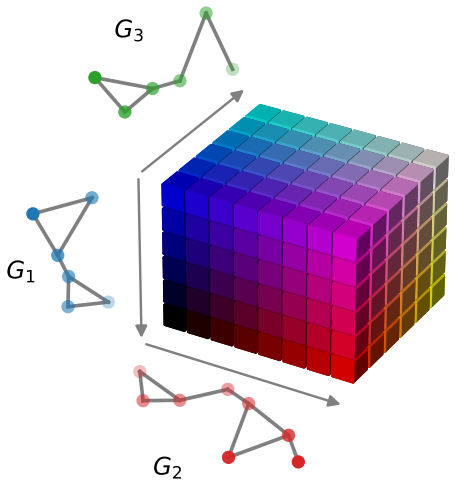


FIGURE 6.2: Graphical depiction of an order-3 tensor signal with graphs underlying each axis, which are linked together to form a Cartesian product graph.

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.

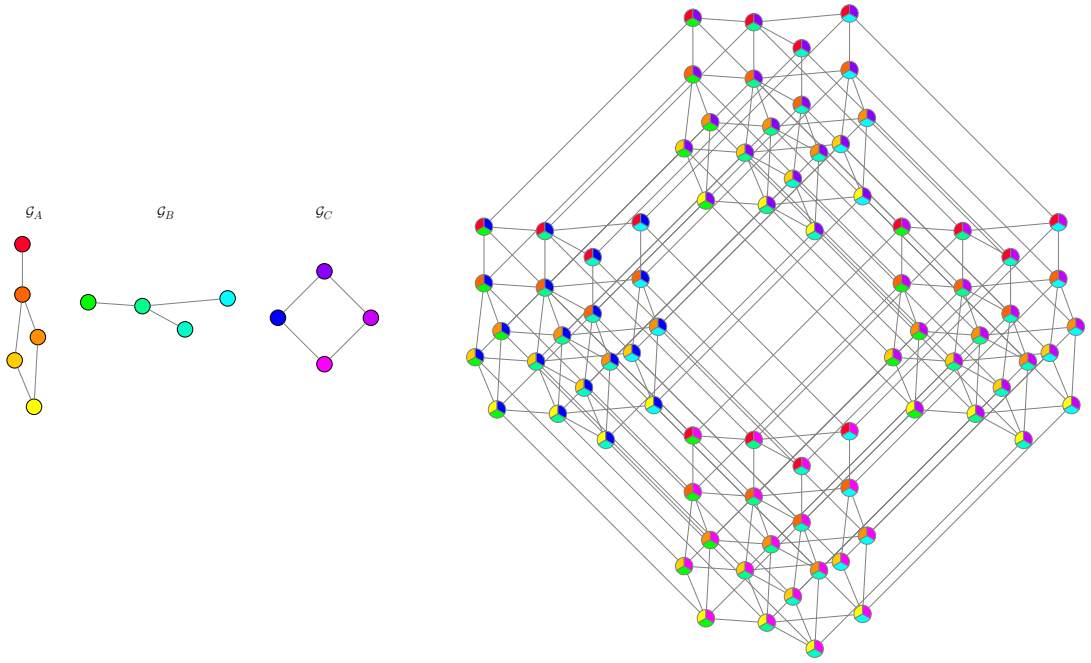


FIGURE 6.3: Graphical depiction of a 3D Cartesian product graph

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

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

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

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

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

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.

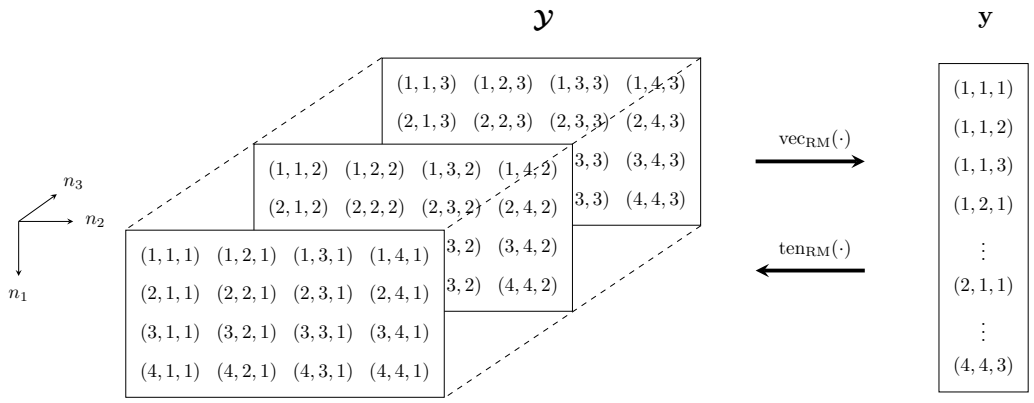


FIGURE 6.4: A graphical depiction of the process of converting an order-3 tensor between its multidimensional array and vector form in row-major order. Note that the elements in the vectorised signal are lexicographically ordered.

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

(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. (4.6) and (4.7), 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. (4.22), 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

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

or, equivalently,

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

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

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

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

7.1 Introduction

7.2 Posterior Estimation

7.2.1 Log-variance prediction

7.2.2 Estimation models

7.2.3 Query strategies

7.2.4 Comparison and analysis

7.3 Posterior Sampling

7.3.1 Perturbation optimization

7.4 Estimation vs Sampling

7.4.1 Experiments

Chapter 8

Working with Binary-Valued Graph Signals

8.1 Logistic Graph Signal Reconstruction

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 4.13 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

$$-\log \pi\big(\mathrm{vec}\,(\mathbf{F}) \mid \mathbf{Y}\big) = \frac{1}{2}\Big(\mathrm{vec}\,(\mathbf{F})^\top \,\big(\mathrm{diag}\,(\mathrm{vec}\,(\mathbf{S})) + \gamma \mathbf{H}^{-2}\big)\mathrm{vec}\,(\mathbf{F}) - 2\,\mathrm{vec}\,(\mathbf{Y})^\top \,\mathbf{F}\Big) \tag{A.9}$$

Using the conjugacy of the normal distribution, by direct inspection we can conclude that the posterior covariance is given by

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

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

□

Theorem A.2. Consider the random matrix $\mathbf{Z}$ which is related to the random matrix $\mathbf{F}$ as follows.

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

or, equivalently,

$$\operatorname{vec}\left(\mathbf{F}\right) = \big(\mathbf{U}_T \otimes \mathbf{U}_N\big)\mathbf{D}_{\mathbf{G}} \operatorname{vec}\left(\mathbf{Z}\right)$$

Then the posterior mean for $\mathbf{Z}|\mathbf{Y}$ is given by

$$\mathrm{E}\big[\mathbf{Z}|\mathbf{Y}\big] = \left(\mathbf{C} + \gamma \mathbf{I}_T \otimes \mathbf{I}_N\right)^{-1} \operatorname{vec}\Big(\mathbf{G} \circ \left(\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T\right)\Big)$$

where

$$\mathbf{C} = \mathbf{D}_{\mathbf{G}} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_{\mathbf{S}} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_{\mathbf{G}}$$

(Here we have abbreviated $\operatorname{diag}\left(\operatorname{vec}\left(\mathbf{G}\right)\right)$ and $\operatorname{diag}\left(\operatorname{vec}\left(\mathbf{S}\right)\right)$ as $\mathbf{D}_{\mathbf{G}}$ and $\mathbf{D}_{\mathbf{S}}$ respectively.)

Proof. The conditional distribution of $\mathbf{Y}|\mathbf{Z}$ is obtained by substituting in the definition of $\mathbf{F}$ in terms of $\mathbf{Z}$ into the original conditional likelihood expression.

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

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

$$\left(\mathbf{C} + \gamma\mathbf{I}_T \otimes \mathbf{I}_N\right)^{-1}\mathrm{vec}\left(\mathbf{G}\circ(\mathbf{U}_N^\top\mathbf{Y}\mathbf{U}_T)\right)$$

□

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.

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.

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.

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.

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.

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.

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

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.

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.

Makhoul, J. (1980). A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34.

Mieghem, P. v. (2010). *Graph Spectra for Complex Networks*. Cambridge University Press.

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

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M. F., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828.

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

Rao, K. and Yip, P. (1990). *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Elsevier Science & Technology Books.

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.

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.

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.