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

February 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

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
$\mathbf{J}_N$	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
$\lvert\lvert \cdot \rvert\rvert_\mathrm{F}$	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	$\mathrm{vec}(\mathbf{AXB})$	$(\mathbf{B}^\top \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{X})$
2	$\mathrm{tr}\big(\mathbf{A}^\top\mathbf{B}\big)$	$\mathrm{vec}(\mathbf{A})^\top\mathrm{vec}(\mathbf{B})$
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	$\mathrm{tr}\big(\mathbf{X}^\top\mathbf{AYB}\big)$	$\mathrm{vec}(\mathbf{X})^\top(\mathbf{B}^\top \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{Y})$
6	$\mathrm{vec}(\mathbf{J} \circ \mathbf{Y})$	$\mathrm{diag}\big(\mathrm{vec}(\mathbf{J})\big)\mathrm{vec}(\mathbf{Y})$
9	$\mathrm{diag}^{-1}\big(\mathbf{A}\,\mathrm{diag}(\mathbf{x})\,\mathbf{B}\big)$	$(\mathbf{B}^\top \circ \mathbf{A})\,\mathbf{x}$

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 1.1 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]

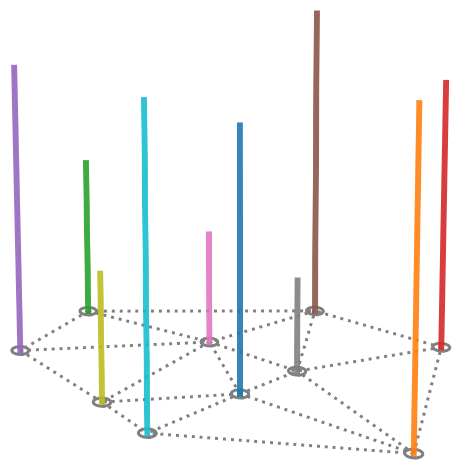


FIGURE 1.1: A graphical depiction of a graph signal. Here, the nodes are represented by circles, the edges as dotted lines, and the value of the signal at each node is represented by the height of its associated bar.

1.2 Thesis overview

Chapter 2

Outline and Fundamentals

2.1 Graph Signal Processing

2.1.1 A broad overview of the field

2.1.2 The graph Laplacian

2.1.3 Graph filters

2.2 Regression and Reconstruction

2.2.1 Graph Signal Reconstruction

Introduce the known work on GSR

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

2.2.2 Kernel Graph Regression

Introduce the known work on KGR and GPoG

2.2.3 Regression with Network Cohesion

Introduce the known work on RNC

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

Regression and Reconstruction on Cartesian Product Graphs

4.1 Graph Products

In this chapter, we turn our attention to the topic of 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 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	$\mathbf{I}_A \otimes \mathbf{A}_B + \mathbf{A}_A \otimes \mathbf{J}_A$	$\mathbf{I}_A \otimes \mathbf{L}_B + \mathbf{L}_A \otimes \mathbf{J}_B + \mathbf{D}_A \otimes (\lvert\mathcal{V}_B\rvert \mathbf{I}_B - \mathbf{J}_B)$

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 $\mathbf{J}_A$ are the $(A \times A)$ identity matrix and matrix of ones respectively.

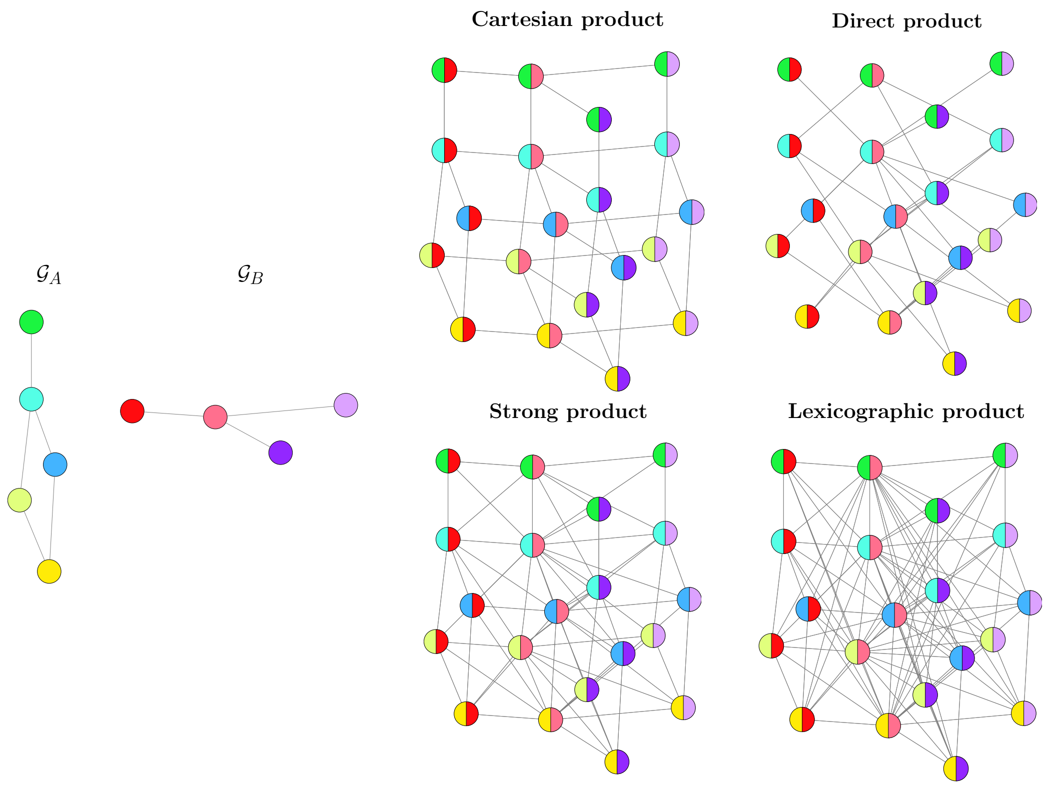


FIGURE 4.1: A graphical depiction of the four standard graph products

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 $\mathbf{L}_B$.

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

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

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

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.

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

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.

$$\text{GFT}(\mathbf{Y}) = \text{mat}\Big(\big(\mathbf{U}_A^\top \otimes \mathbf{U}_B^\top\big)\,\text{vec}(\mathbf{Y})\Big) = \mathbf{U}_B^\top \mathbf{Y} \mathbf{U}_A \tag{4.6}$$

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

$$\text{IGFT}(\mathbf{Z}) = \text{mat}\Big(\big(\mathbf{U}_A \otimes \mathbf{U}_B\big)\,\text{vec}(\mathbf{Z})\Big) = \mathbf{U}_B \mathbf{Z} \mathbf{U}_A^\top \tag{4.7}$$

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

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

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

Given these definitions, we can define a spectral operator (usually a low-pass filter) $\mathbf{H}$ which acts on graph signals according to a spectral scaling function $g(\lambda\,;\,\beta)$ such as one of those defined in table 2.1. As with regular non-product graphs, the action of this operator can be understood as first transforming a signal into the frequency domain via the GFT, then scaling the spectral components according to some function, and finally transforming back into the vertex domain via the IGFT.

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

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

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

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.

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

where now $g(\boldsymbol{\lambda}\,;\,\boldsymbol{\beta})$ 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 a vector of parameters $\boldsymbol{\beta}$ which control the filter intensity in each dimension.

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 4.3: Anisotropic graph filter functions

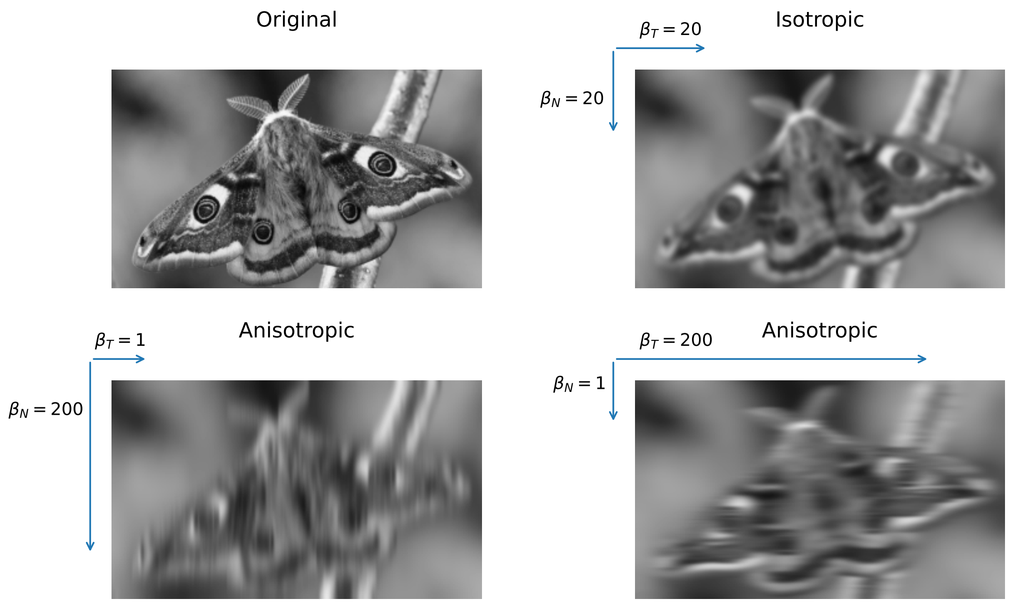


FIGURE 4.2: A graphical depiction of a time-vertex Cartesian product graph

4.2 Graph Signal Reconstruction on Cartesian Product Graphs

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

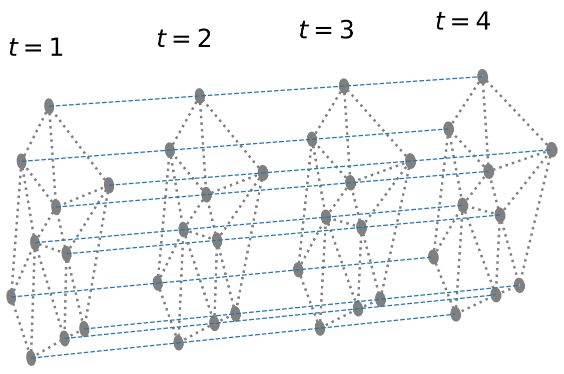


FIGURE 4.3: A graphical depiction of a time-vertex Cartesian product graph

On the Laplacian spectrum of the path graph

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

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

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

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

and

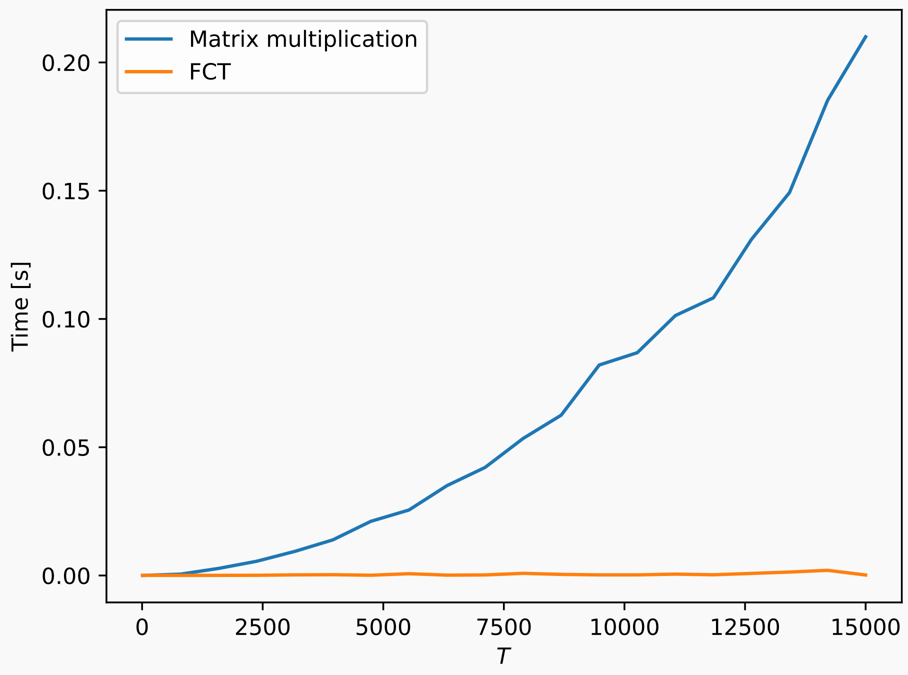
$$(\mathbf{U}_T)_{ij} = \cos\left(\frac{j-1}{T}(i - \frac{1}{2})\pi\right)$$

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

In particular, this reduces both of the following procedures

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

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

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

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

$$\text{vec}(\mathbf{Y}) \,|\, \mathbf{F} \sim \mathcal{N}\Big(\text{vec}(\mathbf{S} \circ \mathbf{F}), \; \text{diag}\big(\text{vec}(\mathbf{S})\big)\Big) \tag{4.13}$$

Note that the covariance matrix $\text{diag}\big(\text{vec}(\mathbf{S})\big)$ 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.

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

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

$$\pi\big(\text{vec}(\mathbf{F}) \,|\, \mathbf{Y}\big) = \frac{\pi\big(\text{vec}(\mathbf{Y}) \,|\, \mathbf{F}\big)\pi(\mathbf{F})}{\pi(\mathbf{Y})}. \tag{4.15}$$

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

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

$$\text{vec}\big(\mathbf{F}\big) \,|\, \mathbf{Y} \sim \mathcal{N}\big(\mathbf{\Sigma}\,\text{vec}\big(\mathbf{Y}\big),\ \mathbf{\Sigma}\big) \tag{4.16}$$

where

$$\mathbf{\Sigma} = \Big(\text{diag}\big(\text{vec}(\mathbf{S})\big) + \gamma \mathbf{H}^{-2}\Big)^{-1} \tag{4.17}$$

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.

$$\text{vec}(\mathbf{F}) = \Big(\text{diag}\big(\text{vec}(\mathbf{S})\big) + \gamma \mathbf{H}^{-2}\Big)^{-1} \text{vec}(\mathbf{Y}) \tag{4.18}$$

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

Two significant computational challenges arise when working with non-trivial graph signal reconstruction problems, where the number of vertices in the product graph is large. First, although the posterior mean point estimator given in eq. (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. 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. 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 $\mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is a non-singular matrix that is easy to solve in the context of a linear system. Commonly used examples of this are 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 this case. This is because, for each of these methods, the coefficient matrix must be split according to its diagonal and off-diagonal elements. Consequently, this would require the evaluation of $\mathbf{H}^{-2}$ directly which, as we have discussed, may be severely ill-conditioned or undefined.

Instead, we must construct a custom splitting that avoids direct evaluation of $\mathbf{H}^{-2}$, and such that $\mathbf{M}^{-1}$ can be computed easily by taking advantage of the properties of the Kronecker product. The main novelty of this subsection is the selection of an appropriate splitting and an investigation of the consequences of that choice. For the system defined in equation (4.18), the coefficient matrix can be split up in the following way:

$$\text{diag}\big(\text{vec}(\mathbf{S})\big) + \gamma\mathbf{H}^{-2} = \mathbf{M} - \mathbf{N} \tag{4.19}$$

where

$$\mathbf{M} = \gamma\mathbf{H}^{-2} + \mathbf{I}_{NT}, \quad \text{and} \quad \mathbf{N} = \text{diag}\big(\text{vec}(\mathbf{S}')\big). \tag{4.20}$$

Here, $\mathbf{S}'$ denotes a binary matrix representing the complement to the set of selected nodes.

$$\mathbf{S}'_{nt} = \begin{cases} 1 & \text{if } \ (n,t) \notin \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

This gives a valid splitting since $\mathbf{M}^{-1}$ is guaranteed to exist. This can also easily be computed since 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

$$\begin{aligned}
\mathbf{M}^{-1} &= \left(\mathbf{H}^{-2} + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}(\mathbf{G})\right)^{-1} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) + \mathbf{I}_{NT}\right)^{-1} \\
&= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \left(\operatorname{diag}\left(\operatorname{vec}(\mathbf{G})\right)^{-1} + \mathbf{I}_{NT}\right)^{-1} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \\
&= \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \operatorname{diag}\left(\operatorname{vec}(\mathbf{J})\right) \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right)
\end{aligned} \tag{4.22}$$

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

$$\mathbf{J}_{nt} = \frac{\mathbf{G}_{nt}^2}{\mathbf{G}_{nt}^2 + \gamma}. \tag{4.23}$$

Therefore, the above splitting leads to a consistent linear stationary iterative method of first degree, with an update equation given by

$$\operatorname{vec}(\mathbf{F}_{k+1}) = \mathbf{M}^{-1}\mathbf{N} \operatorname{vec}(\mathbf{F}_k) + \mathbf{M}^{-1} \operatorname{vec}(\mathbf{Y}) \tag{4.24}$$

One calls a given splitting of the coefficient matrix convergent if the iteration matrix has a spectral radius satisfying $\rho\left(\mathbf{M}^{-1}\mathbf{N}\right) < 1$. Furthermore, the error propagation can be defined as follows, where this contraction condition can be explicitly shown to achieve convergence. Denoting the difference between the estimate at the k-th iteration and the true solution as $\mathbf{E}_k$, we can see that the error is updated according to the following equation.

$$\begin{aligned}
\operatorname{vec}(\mathbf{E}_{k+1}) &= \operatorname{vec}(\mathbf{F}_{k+1}) - \operatorname{vec}(\mathbf{F}_k) \\
&= \mathbf{M}^{-1}\mathbf{N} \operatorname{vec}(\mathbf{F}_k) + \mathbf{M}^{-1} \operatorname{vec}(\mathbf{Y}) - \left(\mathbf{M}^{-1}\mathbf{N} \operatorname{vec}(\mathbf{F}_{k-1}) + \mathbf{M}^{-1} \operatorname{vec}(\mathbf{Y})\right) \\
&= \mathbf{M}^{-1}\mathbf{N}\left(\operatorname{vec}(\mathbf{F}_k) - \operatorname{vec}(\mathbf{F}_{k-1})\right) \\
&= \mathbf{M}^{-1}\mathbf{N} \operatorname{vec}(\mathbf{E}_k)
\end{aligned} \tag{4.25}$$

From this it is clear to see that convergence will be achieved so long as the maximum eigenvalue of $\mathbf{M}^{-1}\mathbf{N}$ is less than 1, since, if this condition holds then,

$$\lim_{k\to\infty} \text{vec}(\mathbf{E}_k) = \lim_{k\to\infty} \left(\mathbf{M}^{-1}\mathbf{N}\right)^k \text{vec}(\mathbf{E}_0) = \mathbf{0}. \tag{4.26}$$

By directly inspecting equation (4.22), it is clear that the spectral radius of $\mathbf{M}^{-1}$ will be the maximum entry in the matrix $\mathbf{J}$. 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

$$\rho(\mathbf{M}^{-1}) = \frac{1}{1+\gamma} \tag{4.27}$$

The spectral radius of $\mathbf{N}$ 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

$$\rho(\mathbf{A}\mathbf{B}) \leq \rho(\mathbf{A})\,\rho(\mathbf{B}) \tag{4.28}$$

[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.

Finally, we can leverage properties of the Kronecker product to write an efficient matrix-update equation. By substituting in the expression for $\mathbf{M}^{-1}$ given in equation (4.22), and applying it to the update formula given in equation (4.24), an update procedure with per-step time complexity is $O(N^2T + NT^2)$ is generated. This is given by

$$\begin{aligned}
\mathbf{F}_0 &= \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \\
\mathbf{F}_{k+1} &= \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top + \mathbf{F}_0
\end{aligned} \tag{4.29}$$

or, equivalently,

$$\begin{aligned}
\Delta\mathbf{F}_0 &= \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \mathbf{Y}\, \mathbf{U}_T\right)\right) \mathbf{U}_T^\top \\
\Delta\mathbf{F}_{k+1} &= \mathbf{U}_N \left(\mathbf{J} \circ \left(\mathbf{U}_N^\top \left(\mathbf{S}' \circ \Delta\mathbf{F}_k\right) \mathbf{U}_T\right)\right) \mathbf{U}_T^\top
\end{aligned} \tag{4.30}$$

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 $\mathbf{G}_{nt} = g\left(\begin{bmatrix} \lambda_t^{(T)} \\ \lambda_n^{(N)} \end{bmatrix}, \boldsymbol{\beta}\right)$
  Compute $\mathbf{J} \in \mathbb{R}^{N \times T}$ as $\mathbf{J}_{nt} = \mathbf{G}_{nt}^2 / (\mathbf{G}_{nt}^2 + \gamma)$
  $\mathbf{S}' \leftarrow \mathbf{1} \in \mathbb{R}^{N \times T} - \mathbf{S}$
  $\Delta\mathbf{F} \leftarrow \mathbf{U}_N\big(\mathbf{J} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T)\big)\mathbf{U}_T^\top$
  $\mathbf{F} \leftarrow \Delta\mathbf{F}$
  while $|\Delta\mathbf{F}| >$ tol do
    $\Delta\mathbf{F} \leftarrow \mathbf{U}_N\Big(\mathbf{J} \circ \big(\mathbf{U}_N^\top (\mathbf{S}' \circ \Delta\mathbf{F})\, \mathbf{U}_T\big)\Big)\mathbf{U}_T^\top$
    $\mathbf{F} \leftarrow \mathbf{F} + \Delta\mathbf{F}$
  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, Kelley, 1995]. The CGM works best when the coefficient matrix has a low condition number (that is, the ratio between the largest and smallest eigenvalue is small) and, as such, a preconditioning step is often necessary. In our case, preconditioning will be essential for convergence. To see why, consider the matrix $\mathbf{H}^{-2}$ within the linear system of equation (4.18), along with 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-frequency eigenvalues of the graph Laplacian, which can result in a severely ill-conditioned system. 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 contribution of this subsection is to find a preconditioner that is both efficient to compute and 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 methods 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 order to derive an effective preconditioner, first consider the transformed variable $\mathbf{Z}$, related to $\mathbf{F}$ in the following way.

$$\mathbf{F} = \mathbf{U}_N \left(\mathbf{G} \circ \mathbf{Z}\right) \mathbf{U}_T^\top \tag{4.31}$$

Here, $\mathbf{Z}$ can be interpreted as set of 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-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 given by the solution to the following linear system

$$\big(\mathbf{C} + \gamma \mathbf{I}_T \otimes \mathbf{I}_N\big)\, \text{vec}(\mathbf{Z}) = \text{vec}\big(\mathbf{G} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T)\big) \tag{4.32}$$

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

$$\mathbf{C} = \mathbf{D}_{\mathbf{G}} \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_{\mathbf{S}} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right) \mathbf{D}_{\mathbf{G}} \tag{4.33}$$

where we have abbreviated $\text{diag}\big(\text{vec}(\mathbf{G})\big)$ and $\text{diag}\big(\text{vec}(\mathbf{S})\big)$ as $\mathbf{D}_{\mathbf{G}}$ and $\mathbf{D}_{\mathbf{S}}$ respectively. For a full derivation of this, consult the supplementary materials. The conditioning of the matrix $\mathbf{C} + \gamma \mathbf{I}$ is greatly improved from the untransformed problem.

Another way of interpreting the effect of the transformation defined in equation (4.31) is as a two-sided symmetric preconditioner to the original linear system. The transformed problem defined in equation (4.32) can be recovered by preconditioning the system in equation (4.18) in the following way.

$$\Big(\mathbf{\Phi}^\top \left(\mathbf{D}_{\mathbf{S}} + \gamma \mathbf{H}^{-2}\right) \mathbf{\Phi}\Big) \Big(\mathbf{\Phi}^{-1}\, \text{vec}(\mathbf{F})\Big) = \mathbf{\Phi}^\top\, \text{vec}(\mathbf{Y}), \tag{4.34}$$

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 $\mathbf{U}_N \Lambda_L \mathbf{U}_N^\top$ 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 $\mathbf{G}_{nt} = g\left(\begin{bmatrix} \lambda_t^{(T)} \\ \lambda_n^{(N)} \end{bmatrix}, \boldsymbol{\beta}\right)$
  Initialise $\mathbf{Z} \in \mathbb{R}^{N \times T}$ randomly
  $\mathbf{R} \leftarrow \mathbf{G} \circ (\mathbf{U}_N^\top \mathbf{Y} \mathbf{U}_T) - \gamma \mathbf{Z} - \mathbf{G} \circ \Big(\mathbf{U}_N^\top \big(\mathbf{S} \circ (\mathbf{U}_N (\mathbf{G} \circ \mathbf{Z}) \mathbf{U}_T^\top)\big) \mathbf{U}_T\Big)$
  $\mathbf{D} \leftarrow \mathbf{R}$
  while $|\Delta \mathbf{R}| >$ tol do
    $\mathbf{A}_D \leftarrow \gamma \mathbf{D} + \mathbf{G} \circ \Big(\mathbf{U}_N^\top \big(\mathbf{S} \circ (\mathbf{U}_N (\mathbf{G} \circ \mathbf{D}) \mathbf{U}_T^\top)\big) \mathbf{U}_T\Big)$
    $\alpha \leftarrow \mathrm{tr}\big(\mathbf{R}^\top \mathbf{R}\big) \,/\, \mathrm{tr}\big(\mathbf{R}^\top \mathbf{A}_D \mathbf{R}\big)$
    $\mathbf{Z} \leftarrow \mathbf{Z} + \alpha \mathbf{D}$
    $\mathbf{R} \leftarrow \mathbf{R} - \alpha \mathbf{A}_D$
    $\delta \leftarrow \mathrm{tr}\big(\mathbf{R}^\top \mathbf{R}\big) \,/\, \mathrm{tr}\big((\mathbf{R} + \alpha \mathbf{A}_D)^\top (\mathbf{R} + \alpha \mathbf{A}_D)\big)$
    $\mathbf{D} \leftarrow \mathbf{R} + \delta \mathbf{D}$
  end while
Output: $\mathbf{U}_N (\mathbf{G} \circ \mathbf{Z}) \mathbf{U}_T^\top$
```

where

$$\boldsymbol{\Phi} = \big(\mathbf{U}_T \otimes \mathbf{U}_N\big) \mathbf{D}_{\mathbf{G}}. \tag{4.35}$$

Since preconditioning of the coefficient matrix on the left is achieved with $\boldsymbol{\Phi}^\top$ and on the right with $\boldsymbol{\Phi}$, 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. Just as in section 4.2.2, this process takes advantage of several identities involving Kronecker products and vectorization, to give $O(N^2T + NT^2)$ complexity per iteration.

4.2.4 Convergence properties

In this section, we contribute theoretical upper bounds for the time-complexity of both methods, and then analyse their practical performance in numerical case studies.

For the stationary iterative method, equation (4.25) demonstrates that the magnitude of the error $\text{vec}(\mathbf{E}_k)$ is reduced at each iteration. In particular, the smallest possible rate of error reduction occurs when $\text{vec}(\mathbf{E}_0) = k\mathbf{1}$, i.e. it is proportional to the first eigenvector of the graph Laplacian, which is constant over all nodes. In this scenario, the worst-case error reduction follows

$$\text{vec}(\mathbf{E}_k) = \frac{1}{(1+\gamma)^k} \text{vec}(\mathbf{E}_0) \tag{4.36}$$

This implies that the number of iterations, n, required to reduce the error to a fraction ϵ follows

$$\epsilon = \frac{1}{(1+\gamma)^n} \quad \to \quad n = \frac{-\log \epsilon}{\log{(1+\gamma)}}$$

Therefore, for a given requirement in error reduction, the complexity of **algorithm 1** is bounded by

$$\frac{m}{\log{(1+\gamma)}} = m\left(\frac{1}{\gamma} + \frac{1}{2} - \frac{\gamma}{12} + \frac{\gamma^2}{24} - ...\right)$$

where m is the number of multiplications required to complete each iteration. From the Taylor series expansion, we can see that the dominant behaviour for small γ is $O(\gamma^{-1})$. Therefore, for small γ, the overall run-time complexity of the SIM is given by

$$O\left(\frac{N^2T + NT^2}{\log{(1+\gamma)}}\right) \approx O\left(\frac{N^2T + NT^2}{\gamma}\right) \tag{4.37}$$

The conjugate gradient method, by contrast, is known to have a theoretical time complexity of $O(\sqrt{\kappa})$, where κ is the condition number of the coefficient matrix Kelley [1995]. In our case, the coefficient matrix is given in equation (4.32) as $\mathbf{C}+\gamma\mathbf{I}$, and its associated condition number can be bounded.

Consider the definition for $\mathbf{C}$ given in equation (4.33). Note that it can be described as the product of matrices, $\mathbf{C} = \mathbf{C}_1\mathbf{C}_2\mathbf{C}_1$ where

$$\mathbf{C}_1 = \mathbf{D}_\mathbf{G}, \quad \mathbf{C}_2 = \left(\mathbf{U}_T^\top \otimes \mathbf{U}_N^\top\right) \mathbf{D}_\mathbf{S} \left(\mathbf{U}_T \otimes \mathbf{U}_N\right)$$

The spectral radius $\rho(\mathbf{C}_1)$, is clearly 1, since the maximum output of $g(\cdot)$ on the domain $\mathbb{R}^+$, and correspondingly the maximum element contained in the matrix $\mathbf{G}$, is 1. Similarly, $\rho(\mathbf{C}_2)$ is also 1. This is clear to see since it is already diagonalised in the basis $\mathbf{U}_T \otimes \mathbf{U}_N$, with the matrix $\mathbf{S}$ being binary. Taking these facts together, and making use of equation (4.28), we see that the maximum eigenvalue of the matrix $\mathbf{C}$ is 1. On the other hand, the minimum possible eigenvalue of $\mathbf{C}$ is zero. This could occur if, say, $\mathbf{G}$ represented a band-limited frequency response, i.e. $\mathbf{G}$ contains zero elements. Clearly, this means $\mathbf{C}$ transforms non-trivial vectors to zero.

We know that the maximum possible eigenvalue of the matrix $\mathbf{C}$ is 1, and the minimum possible eigenvalue is 0. Accordingly, the maximum eigenvalue of the coefficient matrix $\mathbf{C} + \gamma\mathbf{I}$ is $1 + \gamma$ and the minimum is γ. This gives an upper bound for the condition number.

$$\kappa \leq \frac{1+\gamma}{\gamma} \tag{4.38}$$

This limits the time complexity of the whole procedure stated in **algorithm 2** to an upper bound of

$$m\sqrt{\frac{1+\gamma}{\gamma}} = m\left(\frac{1}{\sqrt{\gamma}} + \frac{\gamma}{2\sqrt{\gamma}} - \frac{\gamma^3}{8\sqrt{\gamma}} + ...\right)$$

From this expansion, we can see that the dominant behaviour for small γ is $O(\gamma^{-1/2})$. Therefore, for small γ, the overall run-time complexity of the CGM is given by

$$O\left(\sqrt{\frac{1+\gamma}{\gamma}}(N^2T + NT^2)\right) \approx O\left(\frac{N^2T + NT^2}{\sqrt{\gamma}}\right) \tag{4.39}$$

4.3 Image processing experiments

In this section, we run several small experiments to corroborate the properties of the CGM and SIM. In particular, we verify a) that both algorithms result in the same output for a given problem; b) that the runtime of the SIM and CGM increases at a significantly lower rate than naive Gaussian elimination as nodes are added; and c) that the number of iterations required for convergence as a function of γ is less than or equal to the upper bounds derived in section 4.2.4.

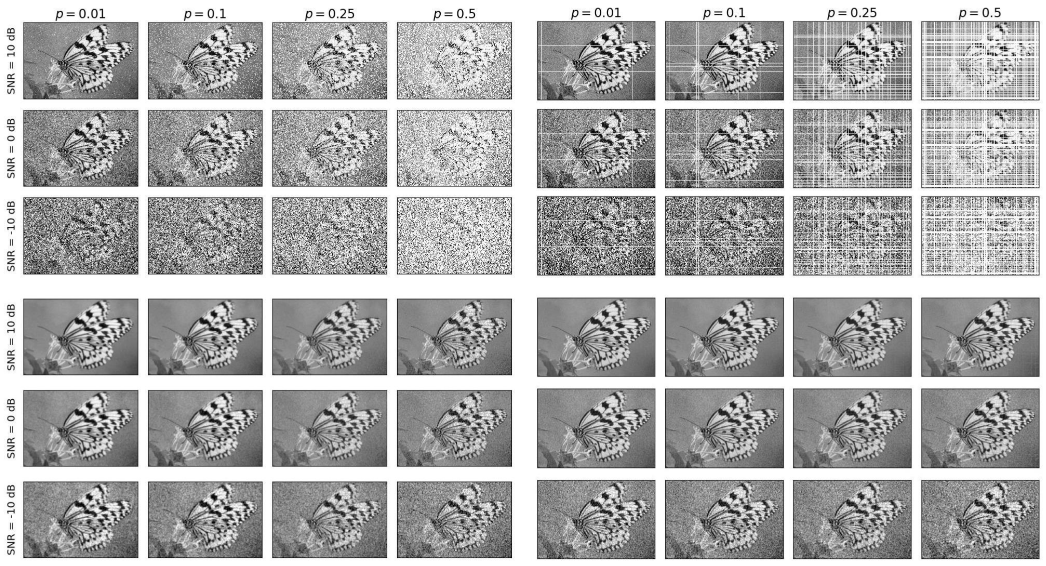


FIGURE 4.4: The output from the first experiment is depicted. In the top left quadrant, the input images are shown across a range of noise levels and missing pixel percentages, with missing pixels chosen uniformly at random. Below that, in the lower left quadrant, the corresponding reconstructed images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

In order to perform these checks, we use a small image denoising/reconstruction task. In particular, we use a grey-scale image of width 571 and height of 856 pixels which has been zero-centred and normalised. Note that an image can be considered a graph signal, with each pixel site representing a node connected to the other pixels immediately adjacent. Furthermore, the data itself lies on a two-dimensional lattice which is a special case of a product graph, with each sub-graph in the product being a simple chain, in this case with $N = 856$ and $T = 571$ respectively. Therefore, images can serve as a simple test case for checking the behaviour of product graph algorithms. Note that the purpose of this section is not to compare graph signal reconstruction to existing algorithms specialised for image denoising/reconstruction, but to examine the computational properties of GSR.

First, we simulate a graph signal reconstruction task across a range of noise levels and missing data distributions. In particular, we add white Gaussian noise to the raw image with a Signal to Noise Ratio (SNR) of -10, 0 and 10 dB. We also remove pixels according to two rules. First, pixels are removed uniformly at random, and second, we remove entire rows and columns uniformly at random such that a fixed total percentage p of the pixels is missing. We do this for p equal to 0.01, 0.1, 0.25 and 0.5 giving 24 unique trials which are shown in **figure 2**. We find that both the SIM and the CGM converge quickly and result in identical predicted outputs, as expected. Furthermore, the statistical model seems relatively robust to noise and missing data of both types, visually performing

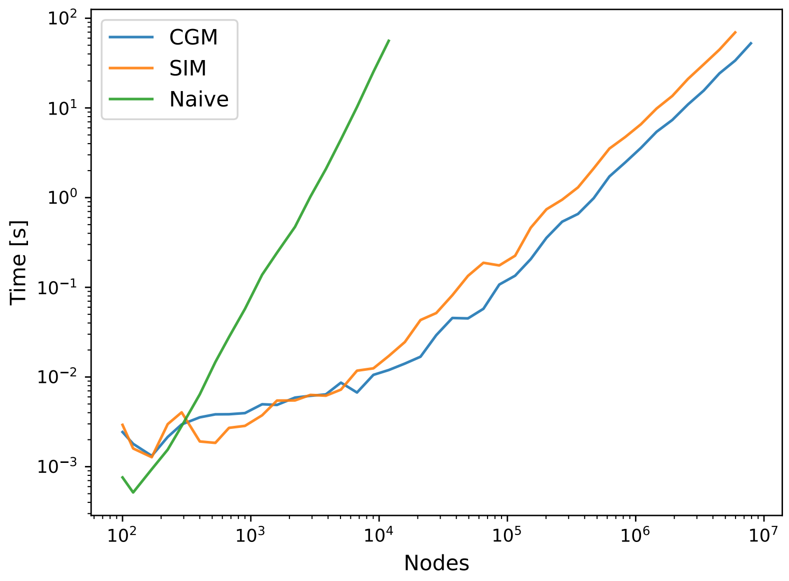


FIGURE 4.5: The total runtime in seconds for the SIM and CGM compared to a naive Gaussian elimination approach is shown as a function of the total number of nodes using a quad-core Intel i7-7700HQ CPU.

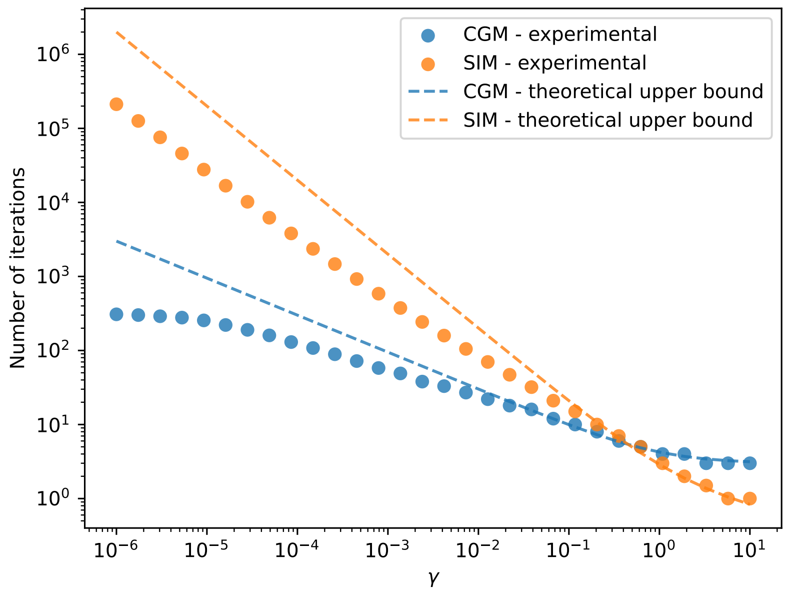


FIGURE 4.6: The number of iterations required to reach a certain level of precision is shown experimentally, along with the theoretical upper bound, for the SIM and CGM

adequately at the level of $p = 0.5$ with SNR=-10dB, which is typically considered a challenging image reconstruction task.

Next, we measured the total runtime of the SIM and the CGM as the number of nodes was increased, and compared this to a more naive approach to solving equation (4.18) which is direct Gaussian elimination. In particular, we fixed the SNR at 0dB and the missingness to $p = 0.5$, with missing pixels chosen uniformly at random, and created square images of increasing size from 10^2 to 10^7 total pixels. The results are shown in **figure 3**. As is visible, the runtime of the SIM and CGM scales at a significantly slower rate that a naive approach, remaining tractable at well above 10^7 nodes.

Finally, we test the effect that varying the hyperparameter γ has on the number of iterations required for convergence. To do this, we fixed the SNR at 0dB and the missingness to $p = 0.5$, with missing pixels chosen uniformly at random. Next, we varied γ in logarithmically spaced increments from 10^{-6} to 10^1. For each unique value of γ, we ran both the SIM and the CGM and counted the number of steps required for each algorithm to reach a specific level of precision (10^{-8} across all elements). The results are shown in **figure 4**. We also plot the theoretical upper bound derived in the previous subsection for each method.

As is visible, both methods converge within the bounds of their theoretical worst-case complexity. As expected, the CGM takes significantly fewer iterations than the SIM to converge at small γ. It is also visible that the CGM seems to outperform the theoretical worst-case scaling of $\gamma^{-0.5}$, converging with a rate closer to $\gamma^{-0.3}$ over the majority of the range, and even showing signs of reducing further at very small γ. The SIM on the other hand empirically converges slightly faster but close to the theoretical worst-case rate of γ^{-1}. Interestingly, the SIM converges faster than the CGM at relatively high γ, with the cross-over occurring at around $\gamma = 0.5$. Although the absolute number of iterations required for convergence in this domain is low (between 1 and 10), this could still be significant for very large problems, meaning it still has some value as a solution.

4.4 Kernel Graph Regression with Unrestricted Missing Data Patterns

Hello

4.4.1 Cartesian product graphs and KGR

Hello

4.4.2 Convergence properties

Hello

4.5 Regression with Network Cohesion

Hello

4.5.1 Regression with node-level covariates

Hello

4.5.2 Convergence properties

Hello

Chapter 5

Multi-Dimensional Cartesian Product Graphs

Hello

5.1 Fast computation with *d*-dimensional Kronecker products

Hello

5.2 Signal reconstruction

Hello

5.3 Kernel Graph Regression

Hello

5.4 Regression with Network Cohesion

Chapter 6

Signal Uncertainty: Estimation and Sampling

6.1 Introduction

6.2 Posterior Estimation

6.2.1 Log-variance prediction

6.2.2 Estimation models

6.2.3 Query strategies

6.2.4 Comparison and analysis

6.3 Posterior Sampling

6.3.1 Perturbation optimization

6.4 Estimation vs Sampling

6.4.1 Experiments

Chapter 7

Working with Binary-Valued Graph Signals

7.1 Logistic Graph Signal Reconstruction

7.2 Logistic Kernel Graph Regression

7.3 Logistic Regression with Network Cohesion

7.4 Approximate Sampling via the Laplace Approximation

Chapter 8

Conclusions

8.1 Main Section 1

Appendix A

Proofs

Theorem A.1. *The posterior distribution for* $\mathbf{F}$ *is given by*

$$\operatorname{vec}\big(\mathbf{F}\big) \mid \mathbf{Y} \sim \mathcal{N}\big(\boldsymbol{\Sigma}\operatorname{vec}\big(\mathbf{Y}\big),\ \boldsymbol{\Sigma}\big) \tag{A.1}$$

where

$$\boldsymbol{\Sigma} = \Big(\operatorname{diag}\big(\operatorname{vec}(\mathbf{S})\big) + \gamma\mathbf{H}^{-2}\Big)^{-1} \tag{A.2}$$

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 $\mathbf{Y}|\mathbf{F}$.

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

In this way, the negative log-likelihood of an observation $\mathbf{Y}|\mathbf{F}$ is given by

$$-\log \pi(\mathbf{Y}|\mathbf{F}) = \lim_{\epsilon\to 0}\left[\frac{1}{2}\operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y})^\top \operatorname{diag}\big(\operatorname{vec}(\mathbf{S}_\epsilon)\big)^{-1}\operatorname{vec}(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y})\right] \tag{A.5}$$

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

$$\begin{aligned} \mathrm{vec}(\mathbf{S}_\epsilon \circ \mathbf{F} - \mathbf{Y}) &= \mathrm{vec}\big(\mathbf{S}_\epsilon \circ (\mathbf{F} - \mathbf{Y})\big) \\ &= \mathrm{diag}\big(\mathrm{vec}(\mathbf{S}_\epsilon)\big)\mathrm{vec}(\mathbf{F} - \mathbf{Y}) \end{aligned} \tag{A.6}$$

Therefore, equation A.5 can be rewritten as

$$\begin{aligned} -\log \pi(\mathbf{Y}|\mathbf{F}) &= \lim_{\epsilon \to 0} \left[\frac{1}{2}\mathrm{vec}(\mathbf{F} - \mathbf{Y})^\top \mathrm{diag}\big(\mathrm{vec}(\mathbf{S}_\epsilon)\big)\, \mathrm{vec}(\mathbf{F} - \mathbf{Y}) \right] \\ &= \frac{1}{2}\mathrm{vec}(\mathbf{F} - \mathbf{Y})^\top \mathrm{diag}\big(\mathrm{vec}(\mathbf{S})\big)\, \mathrm{vec}(\mathbf{F} - \mathbf{Y}) \end{aligned} \tag{A.7}$$

Now consider the full log-posterior. Using Bayes rule, this can be written as

$$\begin{aligned} -\log \pi\big(\mathrm{vec}(\mathbf{F}) \,|\, \mathbf{Y}\big) = \frac{1}{2}\mathrm{vec}(\mathbf{F} - \mathbf{Y})^\top \mathrm{diag}\big(\mathrm{vec}(\mathbf{S})\big)\, \mathrm{vec}(\mathbf{F} - \mathbf{Y}) + \\ \frac{\gamma}{2}\mathrm{vec}(\mathbf{F})^\top \mathbf{H}^{-2}\, \mathrm{vec}(\mathbf{F}) \end{aligned} \tag{A.8}$$

Up to an additive constant not dependent $\mathbf{F}$, this can be written as

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

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

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

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

$\square$

Bibliography

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

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.

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

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

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.

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.

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.

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.

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

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, second edition.

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.